Strictly as per Revised Syllabus of

ANNA UNIVERSITY

Choice Based Credit System (CBCS)
Semester-IV (CSE/IT)

OPERATING SYSTEMS

Iresh A. Dhotre

M.E. (Information Technology)
Ex-Faculty, Sinhgad College of Engineering
Pune

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

Price : ₹ 325/-
ISBN 978-93-332-2115-3

SYLLABUS

Operating Systems - CS8493

Unit - I Operating System Overview

Computer System Overview-Basic Elements, Instruction Execution, Interrupts, Memory Hierarchy, Cache Memory, Direct Memory Access, Multiprocessor and Multicore Organization. Operating system overview-objectives and functions, Evolution of Operating System.- Computer System Organization Operating System Structure and Operations- System Calls, System Programs, OS Generation and System Boot. **(Chapter - 1)**

UNIT - II Process Management

Processes - Process Concept, Process Scheduling, Operations on Processes, Inter-process Communication; CPU Scheduling - Scheduling criteria, Scheduling algorithms, Multiple-processor scheduling, Real time scheduling; Threads- Overview, Multithreading models, Threading issues; Process Synchronization - The critical-section problem, Synchronization hardware, Mutex locks, Semaphores, Classic problems of synchronization, Critical regions, Monitors; Deadlock - System model, Deadlock characterization, Methods for handling deadlocks, Deadlock prevention, Deadlock avoidance, Deadlock detection, Recovery from deadlock. **(Chapters - 2, 3)**

UNIT - III Storage Management

Main Memory - Background, Swapping, Contiguous Memory Allocation, Paging, Segmentation, Segmentation with paging, 32 and 64 bit architecture Examples; Virtual Memory - Background, Demand Paging, Page Replacement, Allocation, Thrashing; Allocating Kernel Memory, OS Examples. **(Chapter - 4)**

UNIT - IV File Systems and I/O Systems

Mass Storage system - Overview of Mass Storage Structure, Disk Structure, Disk Scheduling and Management, swap space management; File-System Interface - File concept, Access methods, Directory Structure, Directory organization, File system mounting, File Sharing and Protection; File System Implementation- File System Structure, Directory implementation, Allocation Methods, Free Space Management, Efficiency and Performance, Recovery; I/O Systems - I/O Hardware, Application I/O interface, Kernel I/O subsystem, Streams, Performance. **(Chapters - 5, 6)**

UNIT - V Case Study

Linux System - Design Principles, Kernel Modules, Process Management, Scheduling, Memory Management, Input-Output Management, File System, Inter-process Communication; Mobile OS - iOS and Android - Architecture and SDK Framework, Media Layer, Services Layer, Core OS Layer, File System. **(Chapters - 7, 8)**

TABLE OF CONTENTS

Unit - I

Unit - II

Unit - III

Unit - IV

Unit - V

UNIT - I

1 Operating System Overview

Syllabus :

Computer system overview - Basic elements, Instruction execution, Interrupts, Memory hierarchy, Cache memory, Direct memory access, Multiprocessor and multicore organization. Operating system overview - Objectives and functions, Evolution of operating system - Computer system organization - Operating system structure and operations - System calls, System programs, OS generation and system boot.

Contents

1.1 Computer System Overview

AU : May-15, 17, 18, Dec.-15, 17

- Computer system consists of hardware device and software that are combined to provide a tool to user for solving problems.
- Fig. 1.1.1 shows modern computer system.

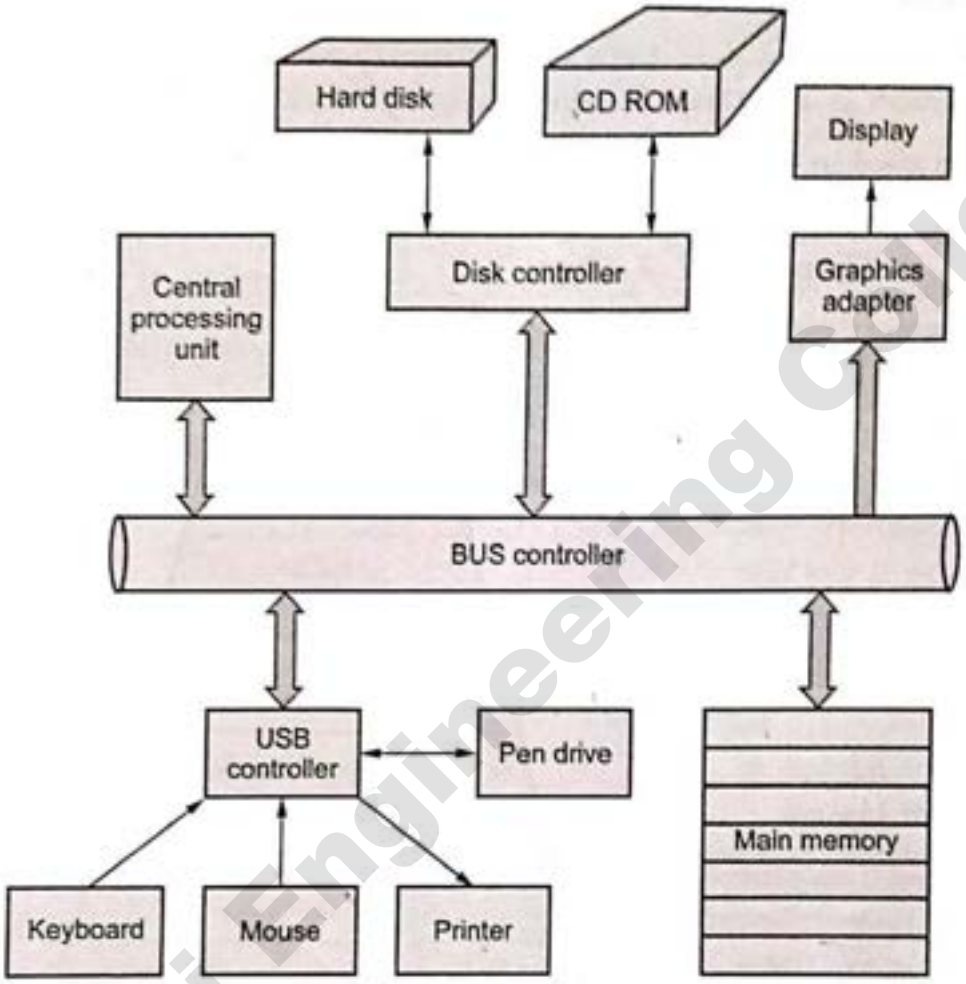


Fig. 1.1.1 Modern computer system

- Modern computer consists of one or two CPU with main memory and various I/O devices. Common bus is used for communication between these devices. Each device has its own device controller.
- CPU and device controller uses memory cycle for execution purposes. But memory cycle is only available to one device at a time.
- Bootstrap program is loaded when user start the computer. It initialies all the device connected to the computer system and then loads required device drivers.
- After this, operating system loads in the computer system. In UNIX OS, an 'init' is the first process which execute by OS.
- Interrupt is software and hardware. It is used to send signal to CPU. Software interrupt is sometime called system call.

- When interrupt is trigger, the CPU stops executing the instruction and control is transfer to the fixed location. Starting address is stored at fixed location where the service routine executes.
- Interrupts do not alter the control flow of the process executing on the processor.

Storage structure

- Processor access the data from main memory before executing any instruction. Main memory is also called Random Access Memory (RAM).
- DRAM is used in main memory. Fig. 1.1.2 shows hierarchy of storage device.
- At the top of the hierarchy, we have storage on the CPU registers. For accessing the CPU, it is fastest form of storage.
- Cache memory capacity is less than 1 MB.
- User program and data are stored in the main memory. Main memory is volatile, so it can not stored permanently.
- Storage system is classified as temporary storage or permanent storage.
- Top level storage devices are low capacity with faster CPU access and bottom level storage devices having very large capacity with slow CPU access speed.

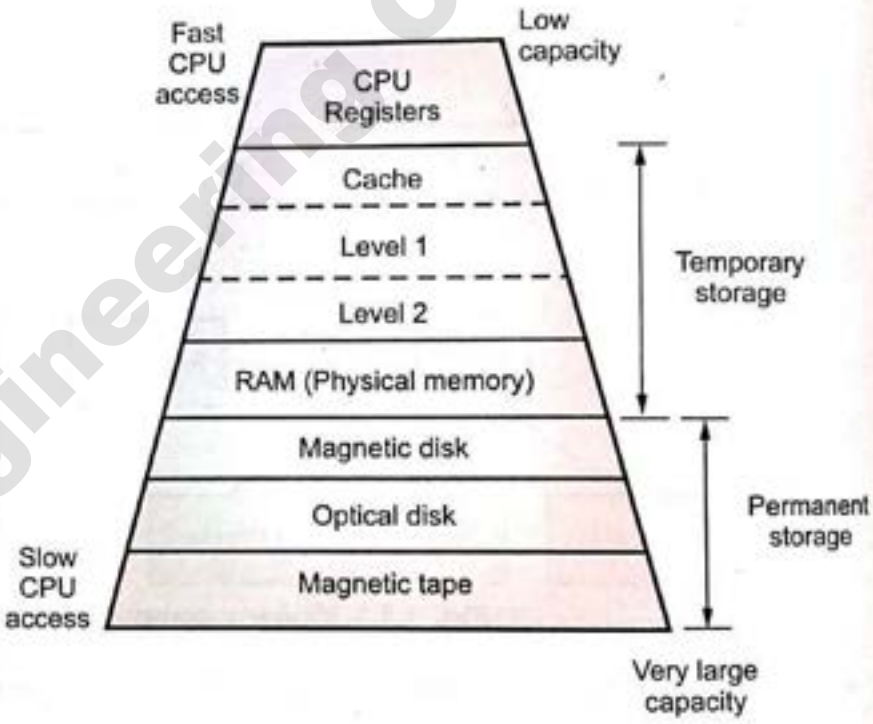


Fig. 1.1.2 Hierarchy of storage device

I/O structure

- Every device uses a device controller to connect it to the computer's address and data bus. Devices can be classified as a block oriented or character oriented, depending on the number of bytes transferred on an individual operation.
- Storage devices are used to store data while the computer is off.

- All the I/O devices are connected to each other by using common bus. CPU and main memory is also connected with this bus.
- Various types of controller is used in the computer system. Small Computer System Interface (SCSI) controller can handle upto seven devices. Each device controller have its own buffer.
- Device controller manage the data transfer between peripheral device and its controller. Device driver is handled by device controller.

I/O operation steps

1. Device driver loads the registers within the device controller.
2. Device controller takes action according to the data loaded into the register.
3. Data is transfer from device to its local buffer with the help of device controller.
4. After completion of data transfer, the device controller sends an interrupt signal to device driver about data transfer completion operation.
5. Then control goes to operating system.

- The device driver is the operating system entity that controls CPU-I/O parallelism. The software that communicates with device controller is called device driver.
- A device can be started by the device driver, and the application program can continue operation in parallel with the device operation.

1.1.1 Instruction Execution

- The program which is to be executed is a set of instructions which are stored in memory. The Central Processing Unit (CPU) executes the instructions of the program to complete a task.
- The major responsibility of the instruction execution is with the CPU. The instruction execution takes place in the CPU registers.
- A special register contains the address of the instruction. The CPU "fetches" the instruction from memory at that address.
- The CPU "decodes" the instruction to figure out what to do. The CPU "fetches" any data (operands) needed by the instruction, from memory or registers.

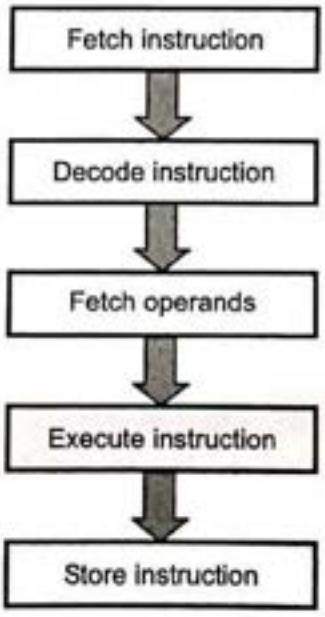


Fig. 1.1.3 Instruction execution cycle

- The CPU "executes" the operation specified by the instruction on this data. Th CPU "stores" any results into a register or memory.
- Fig. 1.1.3 shows instruction execution cycle.
- The register used for instruction execution are as follows :
  1. **Memory Address Register (MAR) :** It specifies the address of memory location from which data or instruction is to be accessed (for read operation) or to which the data is to be stored (for write operation).
  2. **Program Counter (PC) :** It keeps track of the instruction which is to be executed next, after the execution of an on-going instruction.
  3. **Instruction Register (IR) :** Here the instructions are loaded before their execution.

1.1.2 Interrupts

- **Definition :** It is an event external to the currently executing process that causes a change in the normal flow of instruction execution; usually generated by hardware devices external to the CPU. They tell the CPU to stop its current activities and execute the appropriate part of the operating system.
- Fig. 1.1.4 shows interrupts.

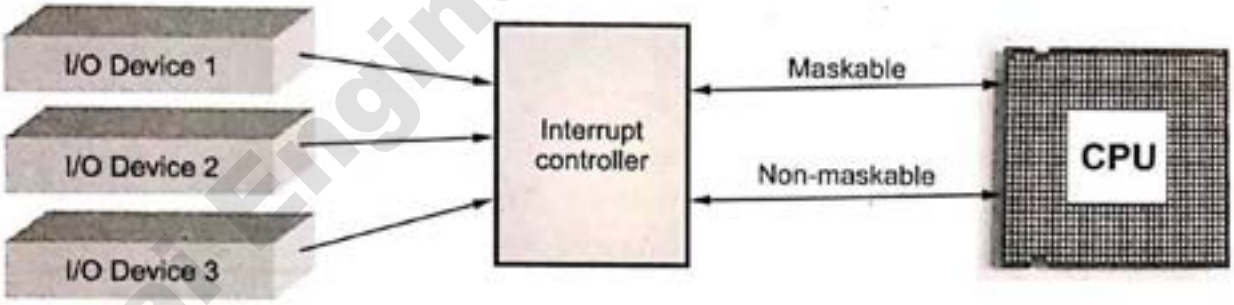


Fig. 1.1.4 Interrupts

- The CPU uses a table and the interrupt vector to find OS the code to execute in response to interrupts. When interrupt signaled, processor executes a routine called an interrupt handler to deal with the interrupt.
- Operating system may specify a set of instructions, called an interrupt handler, to be executed in response to each type of interrupt.
- Interrupt Service Routine (ISR) is the software code that is executed when the hardware requests interrupt. The design of the interrupt service routine requires careful consideration of many factors. Although interrupt handlers can create and use local variables, parameter passing between threads must be implemented using shared global memory variables.

- A private global variable can be used if an interrupt thread wishes to pass information to itself, e.g., from one interrupt instance to another. The execution of the main program is called the foreground thread, and the executions of the various interrupt service routines are called background threads.
- Interrupts are of three types : **Hardware, software and trap.**
- **Hardware Interrupts** are generated by hardware devices to signal that they need some attention from the OS. **Software Interrupts** are generated by programs when they want to request a system call to be performed by the operating system.
- **Traps** are generated by the CPU itself to indicate that some error or condition occurred for which assistance from the operating system is needed.
- Interrupt and trap numbers are defined by the hardware which is also responsible for calling the procedure in the kernel space. An interrupt handler is called in response to a signal from another device while a trap handler is called in response to an instruction executed within the CPU.
- Synchronous interrupts occur when a process attempts to perform an illegal action, such as dividing by zero or referencing a protected memory location. Synchronous interrupt occurs due to software execution.

1.1.3 Cache Memory and Hierarchy

- Cache is small high speed memory. It is derived from SRAM.
- Caches are useful when two ore more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components.
- Caches are useful because they can increase the speed of the average memory access and they do so without taking up as much physical space as the lower elements of the memory hierachy.
- Data is normally kept in storage system. when it required, it is copied into a faster storage system i.e. cache memory for temporary basics.
- When user required a particular data, system check whether it is in the cache. If data found then, we use the data directly from the cache. It is not found then, use the data from the source.
- Internal programmable registers, such as index registers, provide a high-speed cache for main memory. The programmer implements the registor-allocation and register-replacement algorithms to decide which information to keep in registers and which to keep in main memory.

- If the fast device finds the data it needs in the cache, it need not wait for the slower device. The data in the cache must be kept consistent with the data in the components.
- If a component has data value change, and the datum is also in the cache, the cache must also be updated. This is especially a prolem on multiprocessor systems where more than one process may be accessing a datum.
- A component may be eliminated by an equal sized cache, but only if the cache and the component have equivalent state-saving capacity and the cache is affordable, because aster storage tends to be more expensive.
- Unfortunately, cache also introduce an additional level of complexity (coherency and consistency assurance).We also incur an economic and space penalty when we add a cache.
- Making a cache as large as a disk would be ineffective because it would be too costly, the immense size would slow it down and a cache is generally a volatile memory, while we want data on disks to be persistent..
- It holds data for temporary purpose to reduce the time required to service I/O requests from the host. Fig. 1.1.5 shows memory hierarchy.

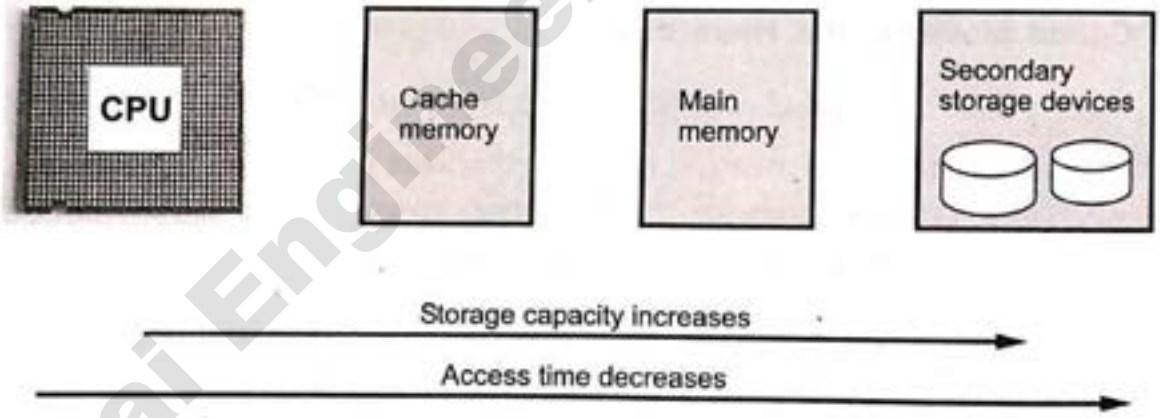


Fig. 1.1.5 Memory hierarchy

- Cache in CPU unit is referred as Level1 cache (L1 cache) and cache stored in a chip next to CPU is termed as Level2 cache (L2 cache) and it resides on the motherboard.
- The function of cache is to act as a buffer between a relatively fast device and a relatively slow one.
- Small unit of allocation in cache is page. Cache is arranged into slots or pages. It uses two components data store and tag RAM. The actual data is stored in a different part of the cache, called the data store. The values stored in the tag RAM determine whether a cache lookup results in a hit or a miss.

- The size of the tag RAM determines what range of main memory can be cached. Tag RAM is a small piece of SRAM that stores the addresses of the data that is stored in the SRAM
- A cache address can be specified simply by index and offset. The tag is kept to allow the cache to translate from a cache address to a unique CPU address.
- A cache hit means that the CPU tried to access an address, and a matching cache block was available in cache. So, the cache did not need to access RAM. In a cache miss, the CPU tries to access an address, and there is no matching cache block. So, the cache is forced to access RAM.
- Cache includes tags to identify which block of main memory is in each cache slot.
- Every cache block has associated with it at least the modify and valid bits, and a tag address. The valid bit says if the cache block is used or is unused. The modify bit makes sense only if the valid bit is set. The modify bit says whether the data in the cache block is different from RAM or is the same as RAM.
- The size of a page is dependent on the size of the cache and how the cache is organized. A cache page is broken into smaller pieces, each called a cache line. The size of a cache line is determined by both the processor and the cache design.
- When the processor starts a read cycle, the cache checks to see if that address is a cache hit.
- **Cache Hit :** If the cache contains the memory location, then the cache will respond to the read cycle and terminate the bus cycle.
- **Cache Miss :** It is reference to item that is not resident in cache, but is resident in main memory. For cache misses, the fast memory is cache and the slow memory is main memory.

1.1.4 Direct Memory Access Structure

- Direct Memory Access (DMA) is one of several methods for coordinating the timing of data transfers between an input/output (I/O) device and the core processing unit or memory in a computer. DMA is one of the faster types of synchronization mechanisms.
- Without DMA, the processor is responsible for the physical movement of data between main memory and a device. A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called Direct Memory Access. Fig. 1.1.6 shows simplified diagram of DMA controller.

- The DMA controller uses hold request (HOLD) and hold acknowledge (HOLD A) signals to ask the CPU to stop driving the address, data and control buses so that the DMA controller can drive them to carry out a transfer.
- A DMA controller implements direct memory access in a computer system. It connects directly to the I/O device at one end and to the system buses at the other end. It also interacts with the CPU, both via the system buses and two new direct connections.

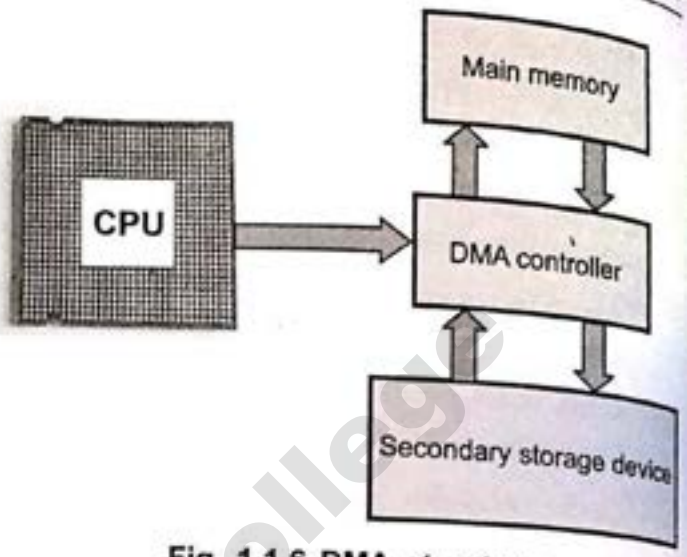


Fig. 1.1.6 DMA structure

- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention. Only one interrupt is generated per block, rather than the one interrupt per byte.
- The DMA controller may either stop the CPU and access the memory or use the bus while the CPU is not using it

University Questions

1. *Sketch the structure of direct memory access in detail.* **AU : May-15, Marks 10**
2. *With neat sketch discuss computer system overview.* **AU : Dec.-15, Marks 8**
3. *State the basic functions of OS and DMA.* **AU : Dec.-15, Marks 6**
4. *Discuss about direct memory access.* **AU : May-17, Marks 6**
5. *Explain cache memory and its mapping.* **AU : Dec.-17, Marks 13**
6. *Give reason why caches are useful. What problems do they solve ? What problems do they cause ? If a cache can be made as large as the device for which it is caching why not make it that large and eliminated the device ?* **AU : May-18, Marks 8**

1.2 Objective and Functions of Operating Systems **AU : Dec.-15**

- **OS definition :** *Operating System is a program that controls the execution of application programs. It is an interface between applications and hardware.*
- OS provides different types of view. For user, it is abstract view because it provides features which are important for users. OS is intermediary between user and the computer system.

- The major design goals/ functions of an operating system are :

1. Efficient use of a computer system 2. User convenience 3. Ability to evolve

- An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.
- Efficiency is the one of the parameter for use of computer system. Operating system consumes some resources during its own operation. For example it uses CPU and memory. CPU is busy with scheduling and memory is occupied by instruction and required data.
- This is one type of overhead and because of this lesser resources are available to the user.
- An OS makes a computer more convenient to use. If the operating system can not allocates the free available resources to program or it over allocates the resources then efficiency is poor.

Efficient use

- For efficient use of resources, it must be monitored by operating system. Proper scheduling of resources is also required.
- Computer contains different type's resources like CPU, memory and I/O device etc. Proper monitoring is required on these resources to avoid the overhead. As per the resource, scheduling is required.
- Special attention to be given for CPU and memory. If memory is not free then user can not load new program into the memory. Then CPU will be busy with memory management.

User convenience

- User convenience is affected by computing environment of the computer system. The computing environment is comprised of computer system, its interfaces with other systems and nature of computations performed by its users.
- Computer architecture and use change the computing environment of the system. Following factors are considered while considering user convenience :

1. Good service 2. Ease of use 3. New programming model
4. Evolution 5. User friendly OS

Ability to evolve

- An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions without at the same time interfering with service.

- Task performed by operating systems :
  1. Maintain the list of resources in the system
  2. Maintain the list of authorized users
  3. Initiate the execution of programs and process
  4. Maintain resource usage list
  5. Maintain the resource allocated list
  6. Scheduling of resources (CPU, Secondary storage etc.)
  7. Also maintain the protection information.

1.2.1 Operating System as a User Interface

- Computer system consists of software and hardware to solve specific problems. User, application program, operating system and the hardware are the components of the computer systems.
- Application program used to solve specific program. Student attendance monitoring is the example of application program.
- Operating system is a subset of the system software. OS interacts directly with the hardware to provide an interface to other system software.

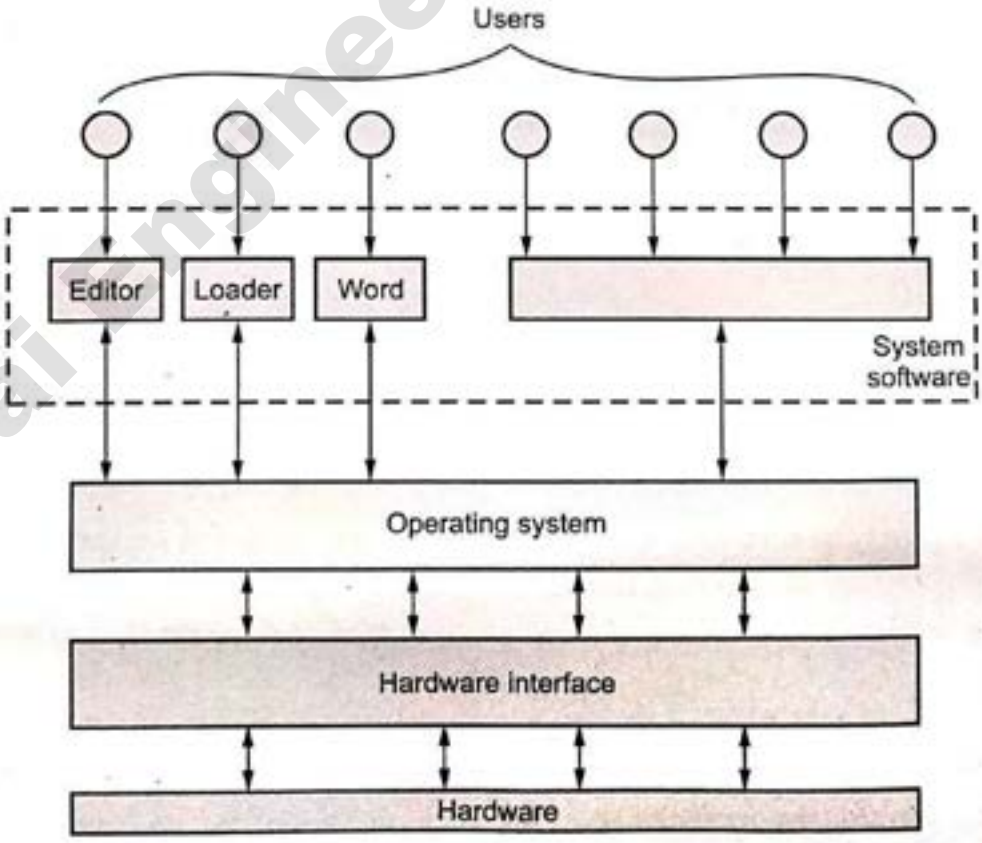


Fig. 1.2.1 Computer system conceptual view

- System software and hardware exist to support the creation and effective use application software.
- Fig. 1.2.1 shows conceptual view of a computer system.
- Resource sharing and resource abstraction are two key aspects of the operating system.
- Purposes of the operating system :
  1. OS provides an interface between the computer hardware and computer user (programmer). It simplifies the programmer job like editing, coding, creation.
  2. Allocation and use of computer resources among the programmer is controlled by OS.

1.2.2 Operating System as Resource Manager

- A computer is a set of resources. These resource provides various functions to the user. Functions like data movement, storing of data and program, operation on data are control by an operating system.
- Fig. 1.2.2 shows OS as a resource manager.

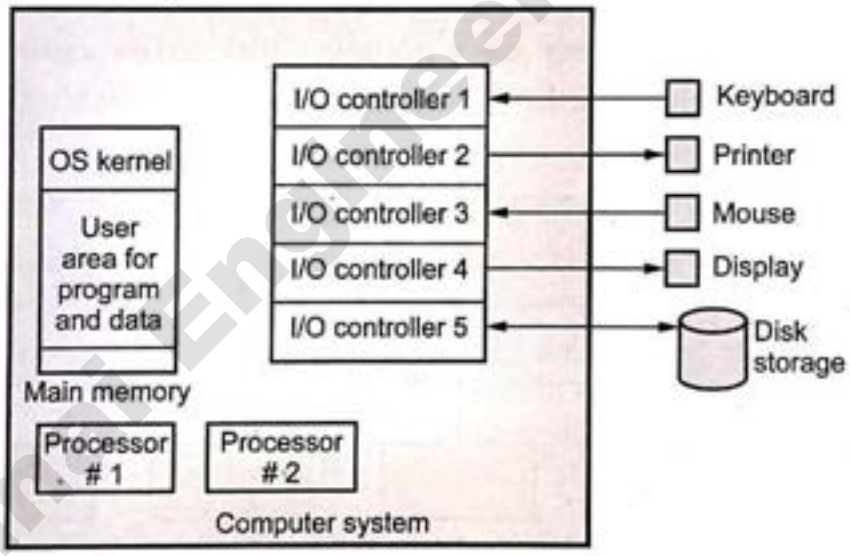


Fig. 1.2.2 OS as a resource manager

- The operating system is responsible for managing the all resources. A portion of the OS is in main memory. This portion of the OS is called kernel.
- User program and data is also stored in remaining parts of the memory. Allocation of main memory is controlled by operating system with the help of memory management hardware.

- I/O device is controlled by OS and it decides when an I/O device can be used by program in execution. Processor is one type of resource and OS control the execution of user program on the processor.
- Modern OS allows multiple programs to run at the same time. If multiple users are using computer then there is need of managing and protecting the memory, I/O devices and other resources.
- Resource management includes sharing resources in different ways. Time and space are the two concept for resource sharing.

1. **Time :** Time slot is allocated to each program first one gets to use the resource then another and so on.
2. **Space :** Consider the example of main memory. Main memory is normally divided up among several running programs, so each one can be resident at the same time.

University Question

1. *State the basic functions of OS and DMA.* **AU : Dec.-15, Marks 6**

1.3 Evolution of Operating System

AU : Dec.-15, 17, May-18

- Types of an operating system are a grouping that differentiates or identifies the operating system based on how it works the type of hardware it controls and the applications it supports.

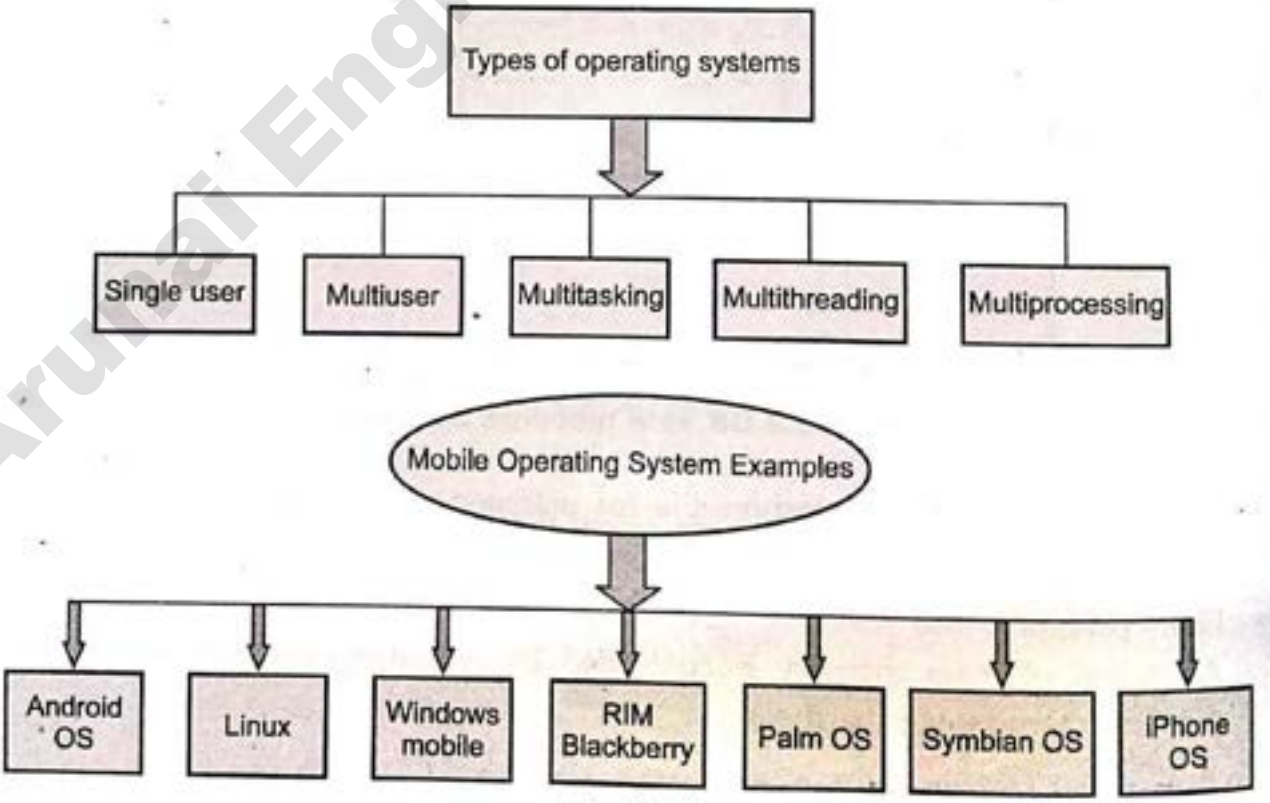


Fig. 1.3.1

1.3.1 Batch System

- Batch system process a collection of jobs, called a batch. Batch is a sequence of user jobs.
- Job is a predefined sequence of commands, programs and data that are combined into a single unit.
- Each job in the batch is independent of other jobs in the batch. A user can define a job control specification by constructing a file with a sequence of commands.
- Jobs with similar needs were batched together to speed up processing. Card readers and tape drives are the input device in batch systems. Output devices are tape drives, card punches and line printers.
- Primary function of the batch system is to service the jobs in a batch one after another without requiring the operator's intervention. There is no need for human/user interaction with the job when it runs, since all the information required to complete job is kept in files.
- Some computer systems only did one thing at a time. They had a list of instructions to carry out and these would be carried out one after the other. This is called a **serial system.** The mechanics of development and preparation of programs in such environments are quite slow and numerous manual operations involved in the process.
- **Batch monitor** is used to implement batch processing system. Batch monitor is also called **kernel**. Kernel resides in one part of the computer main memory.
- The memory allocator managed the main memory space. Batch monitor controls the sequence of events. Main memory store the batch monitor and users program and data (jobs). Fig. 1.3.2 shows memory layout for a batch system.

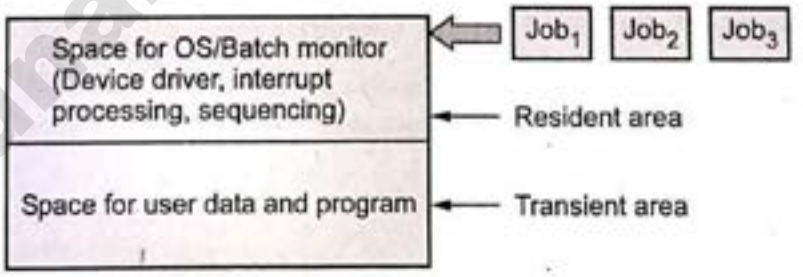


Fig. 1.3.2 Memory layout for batch system

- Computer operator gives a command to start the processing of a batch, the kernel sets up the processing of the first job. Job was selected from the job queue and loaded into main memory. When a job completed execution, its memory was released and the output for the job was copied.

- When a job is completed, it returns control to the monitor, which immediately reads in the next job.
- Fig. 1.3.3 shows concept of batch system.

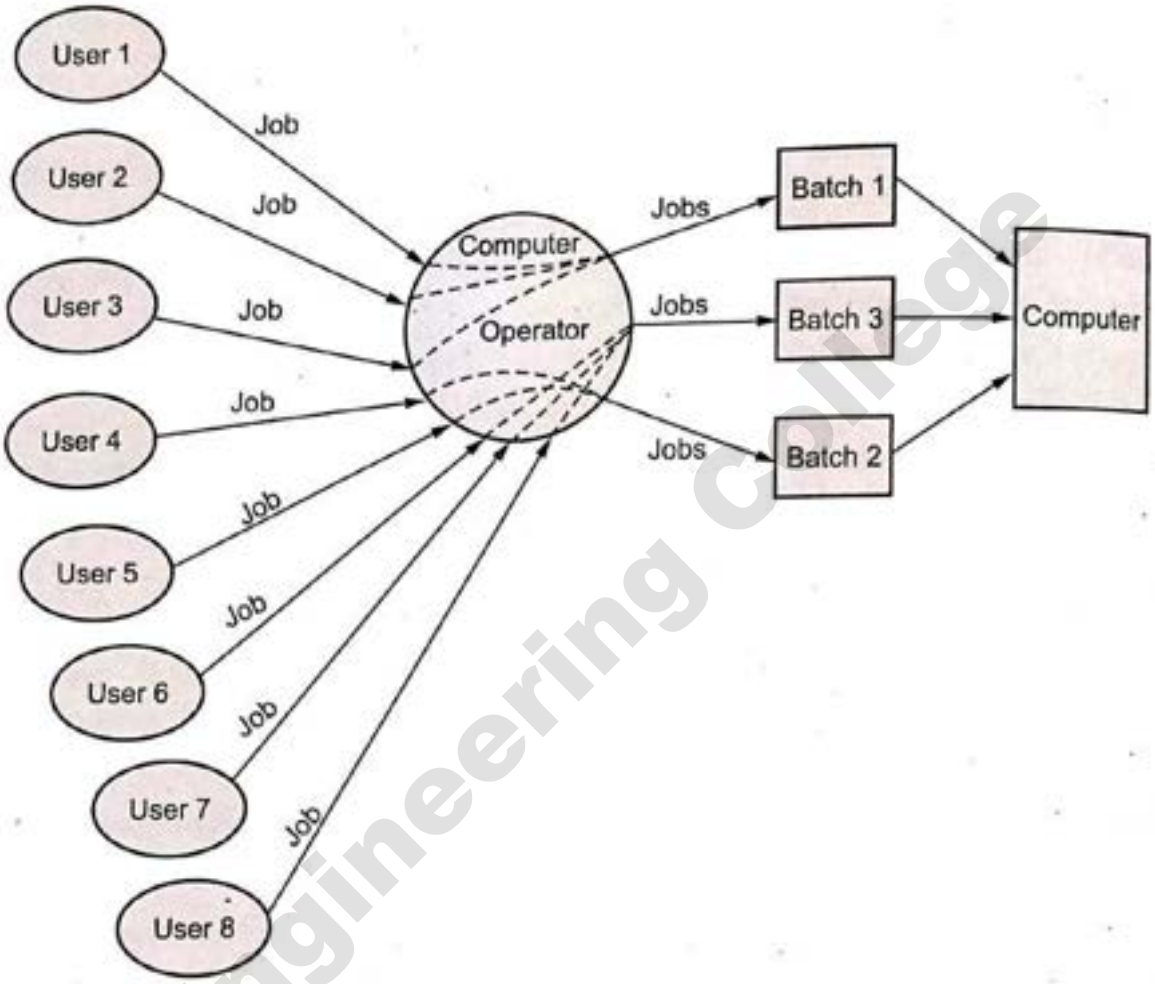


Fig. 1.3.3 Concept of batch system

- Scheduling is also simple in batch system. Jobs are processed in the order of submission i.e. first come first served fashion.
- When a job completes execution, its memory is released and the output for the job gets copied into an output **spool** for later printing.

1.3.1.1 Spooling

- Spooling an acronym for **simultaneous peripheral operation on line.** Spooling uses the disk as a large buffer for outputting data to printers and other devices. It can also be used for input, but is generally used for output.
- Its main use is to prevent two users from alternating printing lines to the line printer on the same page, getting their output completely mixed together. It also helps in reducing idle time and overlapped I/O and CPU.

- Batch system often provides simple forms of file management. Access to file is serial. Batch systems do not require any time critical device management.
- Batch systems are inconvenient for users because users cannot interact with their jobs to fix problems. There may also be long turnaround times. Example of this system is generating monthly bank statement.
- An optimization used to minimize the discrepancy between CPU and I/O speeds is spooling. It overlaps of one job with computation of other job.
- The spooler for instance could be reading the input of one job while printing the output of different job.
- Spooling refers to putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready.
- Spooling is useful because device access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up.
- Computer can perform I/O in parallel with computation, it becomes possible to have the computer read a deck of cards to a tape, drum or disk and to write out to a tape printer while it was computing. This process is called **spooling.**
- The most common spooling application is print spooling.
- Spooling batch system were the first and are the simplest of the multiprogramming systems.

Advantages of spooling

1. The spooling operation uses a disk as a very large buffer.
2. Spooling is however capable of overlapping I/O operation for one job with processor operations for another job.

Advantages of batch system

1. Move much of the work of the operator to the computer.
2. Increased performance since it was possible for job to start as soon as the previous job finished.

Disadvantages of batch system

1. Turn around time can be large from user standpoint.
2. Program debugging is difficult.
3. There was possibility of entering jobs in infinite loop.
4. A job could corrupt the monitor, thus affecting pending jobs.
5. Due to lack of protection scheme, one batch job can affect pending jobs.

1.3.2 Multiprogramming Operating System

- CPU remains idle in batch system. At any time either CPU or I/O device was idle in batch system. To keep CPU busy, more than one program/job must be loaded for execution. It increases the CPU utilizations. So multiprogramming increases the CPU utilization.
- Resource management is the main aim of multiprogramming operating system. File system, command processor, I/O control system and transient area are the essential components of a single user operating system. Multiprogramming operating system divides the transient area to store the multiple programs and provides resource management to the operating system.
- The concurrent execution of programs improves the utilization of system resources. A program in execution is called a "Process", "Job" or a "Task".
- When two or more programs are in the memory at the same time, sharing the processor is referred to the multiprogramming operating system.

Fig. 1.3.4 Multiprogramming OS memory layout

- Fig. 1.3.4 shows the memory layout for a multiprogramming operating system.
- Operating system keeps number of programs into the memory. It selects one program from the memory and executes it. All the programs that enter the system are kept in the job pool.
- Job pool consists of all processes residing on disk and waiting to allocate primary memory. Job scheduling concept is used if there is no space for process in the primary memory.
- Memory management is also required to manage the memory for process. CPU scheduling is applied for selecting process from memory.
- When computer loads more than one program in to the memory, CPU executes one program and I/O system is busy with other programs.
- Multiprogramming operating system do not provide user interaction with the program.
- Fig. 1.3.5 Shows working of multiprogramming OS.

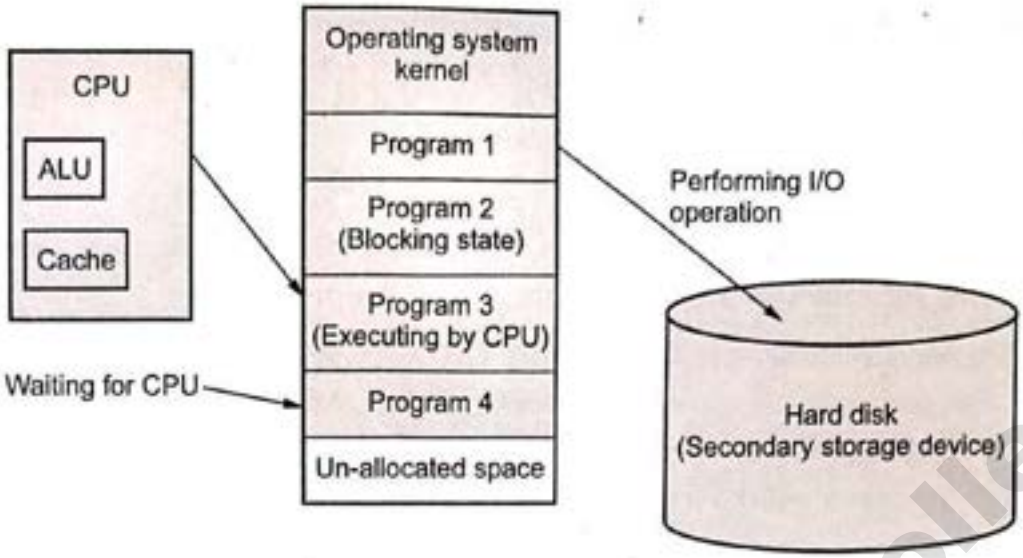


Fig. 1.3.5 Working of multiprogramming OS

- In multiprogramming operating system, programs are competing for resources. A function of the multiprogramming operating system is CPU scheduling, memory management and I/O management.
- Suppose there are four programs for execution. All four programs are loaded into the memory. CPU select first program for execution. Normally programs contain instruction for CPU and I/O operation.
- CPU bound instructions : c = a + b
- I/O bound instructions : printf, scanf etc.
- When any I/O instruction is encounter in the program, CPU select next program for processing. CPU select second program for execution and I/O system select first program for performing I/O operation. Multiprogramming operating system monitors the state of all active programs and system resources.

Advantages :

1. CPU utilization is high.
2. It increases the degree of multiprogramming.

Disadvantages :

1. CPU scheduling is required.
2. Memory management is also required.

1.3.3 Time Sharing System

- Time sharing is also called multitasking operating system. It is logical extension of the multiprogramming operating systems.
- User interaction with program is possible in time sharing operating system. During execution of the program, user interacts directly with the program, supplying information to the program.
- Multi-tasking means that the computer can work with more than one program at a time. For example, user could be working with information from one database on the screen analyzing data, while the computer is sorting information from another database, while a excel sheet is performing calculations on a separate worksheet.
- Many users share the computer system simultaneously in time sharing operating system. Time sharing system uses multiprogramming and CPU scheduling. Each user has at least one separate program in memory.
- In time sharing system, each user is given a time slice for executing his/her job in round robin fashion. Job continues until the time slice ends.
- Fig. 1.3.6 shows multitasking OS.

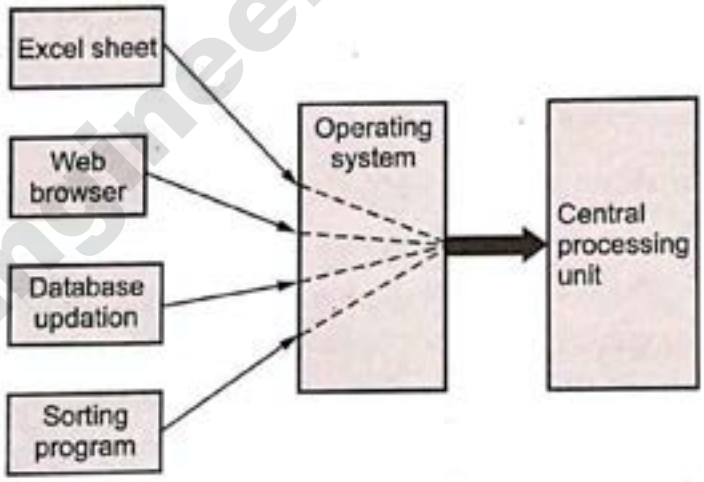


Fig. 1.3.6 Multitasking OS

- Concept of virtual machine is used in time sharing system. It creates virtual machine one per user. User interaction with system by using virtual machine. User enters the command for virtual machine and result will received back to user.
- Time sharing system is more complex than multiprogramming operating system. It also takes help of file system. File system is stored on the disk so disk management is also required.
- Major problem with time sharing system is protection and security of data.

- Time sharing system uses medium term scheduling such as round robin for the foreground. Background process uses can use a different scheduling method.
- Difference between multiprogramming and multitasking operating system is context switching. In multiprogramming system a context switching occurs only when the currently executing process stalls for some reasons. Time sharing system gives each user the impression that the entire system is dedicated to his use. Context switching simply allows several applications to be open, but only one is working at a time.
- Truly speaking, even in true multi-tasking, only one application program is ever running at anyone instant. Because the computer automatically switches from one program to the next program so quickly, all the programs seem to run simultaneously.

University Question

1. *Explain system calls system programs and OS generation.* **AU : Dec.-15, Marks 6**
2. *Describe evolution of operating system.* **AU : Dec.-17, Marks 13**
3. *What optimization where used to minimize the discrepancy between CPU and I/O speeds on early computer systems.* **AU : May - 18, Marks 8**

1.4 Multiprocessor System

AU : May-16

- Multiprocessor system means more than one processor in close communication. All the processor share common bus, clock, memory and peripheral devices.
- Multiprocessor system is also called parallel system or tightly coupled systems. Multiprocessor operating systems are just regular operating systems. They handle system calls, do memory management, provide a file system, and manage I/O devices.
- Multiprocessor systems is also known as *parallel systems* or *multicore systems.*
- **Features of Multiprocessor Systems :**
  1. The processor should support efficient context switching operation.
  2. It supports large physical address space and larger virtual address space.
  3. If one processor fails, then other processors should retrieve the interrupted process state so that execution of the process can continue.
  4. The inter-process communication mechanism should be provided and implemented in hardware as it becomes efficient and easy.

- **Multiprocessors can be used in different ways :**
  1. Uniprossesors (single-instruction, single-data or SISD)
  2. Within a single system to execute multiple, independent sequences of instructions in multiple contexts (multiple-instruction, multiple-data or MIMD);
  3. A single sequence of instructions in multiple contexts (single-instruction, multiple-data or SIMD, often used in vector processing);
  4. Multiple sequences of instructions in a single context (multiple-instruction, single-data or MISD, used for redundancy in fail-safe systems and sometimes applied to describe pipelined processors or hyper threading).

1.4.1 Advantages and Disadvantages of Multiprocessor Systems

- **Advantages of Multiprocessor Systems :**
  1. **Throughput :** Throughput increases because number of processor is increases.
  2. **Cost :** Multiprocessor system is cheaper than the multiple single processor system.
  3. **Reliability :** If one processor fails, it has no effect on whole system operation.
  4. **Response time :** Response time is less because of increased number of processor
- **Disadvantages of Multiprocessor Systems**
  1. Multiprocessor systems are more complex in both hardware and software.
  2. Additional CPU cycles are required to manage the cooperation, so per-CPU efficiency goes down.
- Multiprocessor systems are of two types: symmetric multiprocessing and asymmetric multiprocessing.

1.4.2 Symmetric Multiprocessing

- The simplest multiprocessors are based on a single bus. In symmetric multiprocessing, number of homogeneous processor are running independently without affecting other programs.
- Systems that treat all CPUs equally are called symmetric multiprocessing (SMP) systems.
- Each processor uses different data and program but sharing some common resources like memory, I/O device etc. Fig. 1.4.1 shows symmetric multiprogramming system.
- In SMP, all processor are at the same level. They work in peers. There is no master-slave relationship in between the processor. Each processor shares a single copy of operating system.

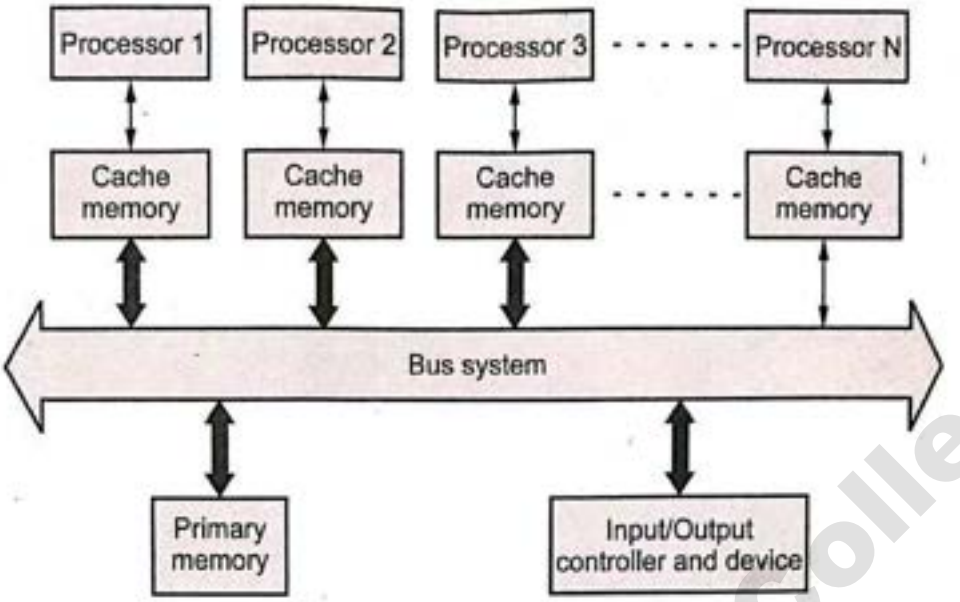


Fig. 1.4.1 Symmetric multiprogramming system

- Symmetric multiprocessing is easier to implement in operating systems. Examples of symmetric multiprocessing operating systems are Windows NT 4.0, Novell Netware 2.1 etc.

1.4.3 Asymmetric Multiprocessor

- If all CPUs are not equal, system resources may be divided in a number of ways, including asymmetric multiprocessing (ASMP), non-uniform memory access (NUMA) multiprocessing, and clustered multiprocessing.
- All CPUs are not equal. There is one master processor and remaining is the slave processor. Each processor has its own memory address space.
- A master processor controls the whole operations. It gives the instruction to the slave processor or slave processor uses some predefined set of tasks.
- Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The Master distributes takes among the slaves and I/O is usually done by the master only.
- Each processor has own task which assign by master processor. Asymmetric multiprocessing is difficult to implement.

1.4.4 Difference between Symmetric and Asymmetric Multiprocessor

Symmetric Multiprocessor	Asymmetric Multiprocessor
Symmetric multiprocessing treats all processors are equals, an I/O can be processed on any CPU.	Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves.
Symmetric multiprocessing is easier to implement in operating systems	Asymmetric multiprocessing is difficult to implement.
Single OS manages all processor cores simultaneously.	Separate OS, or separate copy of same OS manage each core
Master-slave concept is not used.	It uses concept of master-slave concept.
The processors communicate with each other through shared memory.	Shared memory is not used for communication.

University Question

1. *Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems ?* **AU : May-16, Marks 8**

1.5 Clustered System

AU : Dec.-16

- Clustered system is a group of computer system connected with a high speed communication link. Each computer system has its own memory and peripheral devices. Clustering provides high availability.
- Clustering systems are integrated with hardware cluster and software cluster. Hardware cluster support sharing of high performance disks. Software cluster is in the form of unified control of the computer system in a cluster.
- Cluster node contains layer of cluster software and it run on the node. Each node monitors the network whether the cluster is working properly or not. If the monitored computer fails, then the monitoring computer takes the control and ownership of its storage and other resources. It restarts the application of failed computer. The users and other clients will get the resources. The failed computer can remain down until it repair.
- Clustered systems are divided into two groups:
  1. Asymmetric clustering 2. Symmetric clustering
- **Asymmetric clustering :** In this type of clustering, one machine is always in the hot standby mode and other machines are running its applications. The hot

standby machine only monitors the active server. When active server fails, then hot standby machine takes the control and becomes the active server.

- **Symmetric clustering :** more than one host runs the applications. It uses all of the available hardware for operation. Host monitors working of other host for smooth operation of the cluster.
- Fig. 1.5.1 shows structure of cluster system.

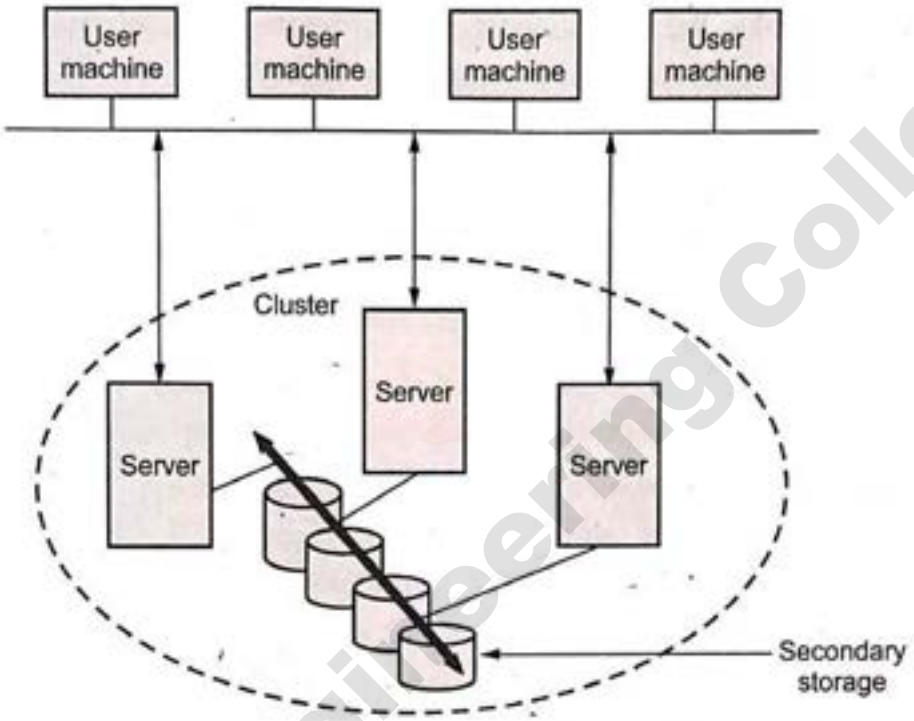


Fig. 1.5.1 Cluster system

- Clustered system and symmetric multiprocessor provide a support for high demand applications.
- Clustered system is different than parallel system.
- Other type of clustered system is parallel clusters. It uses shared storage of data. Parallel cluster requires some special purpose software and special type of applications. Example of parallel cluster is Oracle Parallel Server.
- Cluster technology changes rapidly. New developments are added to this cluster technology. Now a day, global cluster concept is used for large organizations.

University Question

1. *Discuss the different multiprocessor organizations with block diagrams.*

AU : Dec.-16, Marks 8

1.6 Distributed System

AU : May-16

- Definition : A distributed system is a collection of autonomous hosts that are connected through a computer network.
- A distributed system is a collection of independent computers that appears to its users as a single coherent system. Each host executes components and operates a distribution middleware.
- Middleware enables the components to coordinate their activities. Users perceive the system as a single, integrated computing facility.
- A distributed computer system consists of multiple software components that are on multiple computers, but run as a single system. The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network.
- A distributed system can consist of any number of possible configurations, such as mainframes, personal computers, workstations, minicomputers and so on.
- Distributed operating systems depend on networking for their operation. Distributed OS runs on and controls the resources of multiple machines. It provides resource sharing across the boundaries of a single computer system. It looks to users like a single machine OS.
- Distributing OS owns the whole network and makes it look like a virtual uni-processor or may be a virtual multiprocessor.
- Definition : A distributed operating system is one that looks to its users like an ordinary operating system but runs on multiple, independent CPU.
- Distributed systems depend on networking for their functionality. Fig. 1.6.1 shows the distributed system.
- Examples of distributed operating system are Amoeba, chrous, mach and v-system
- A **Local Area Network** (LAN) is a network that is confined to a relatively small area. It is generally limited to a geographic area such as a college, lab or building.

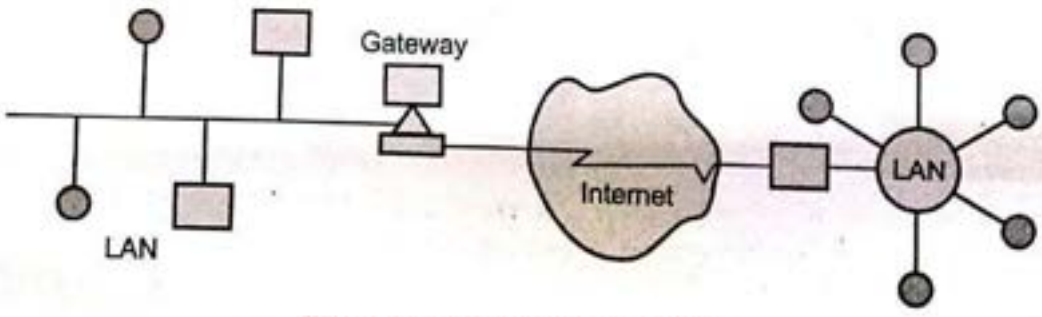


Fig. 1.6.1 Distributed system

- **WAN** provides long distance transmission of data and voice. Computers connected to a wide-area network are often connected through public networks, such as the telephone system. They can also be connected through leased lines or satellites.
- A **MAN** typically covers an area of between 5 and 50 km diameter. Many MANs cover an area the size of a city, although in some cases MANs may be as small as a group of buildings.
- A MAN often acts as a high speed network to allow sharing of regional resources. MAN provides the transfer rates from 34 to 150 Mbps.

Advantages of distributed OS :

1. **Resource sharing :** Sharing of software resources such as software libraries, database and hardware resources such as hard disks, printers and CDROM can also be done in a very effective way among all the computers and the users.
2. **Higher reliability :** Reliability refers to the degree of tolerance against errors and component failures. Availability is one of the important aspects of reliability. Availability refers to the fraction of time for which a system is available for use.
3. **Better price performance ratio :** Reduction in the price of microprocessor and increasing computing power gives good price-performance ratio.
4. **Shorter responses times and higher throughput.**
5. **Incremental growth :** To extend power and functionality of a system by simply adding additional resources to the system

Difficulties in distributed OS are

1. There are no current commercially successful examples.
2. Protocol overhead can dominate computation costs.
3. Hard to build well.
4. Probably impossible to build at the scale of the internet.

1.6.1 Client-Server Computing

- The system is structured as a set of processes, called servers that offer services to the users, called clients.
- Server systems are of two types : compute servers and file servers
- **Compute-server system :** It provides an interface to which a client can send a request to perform an action. In response, the server performs some operation and sends the results to the client. Server contains database.

- **File-server system :** Client can performs various operation like create, read, update and delete file on file server. Web server is best example of this type.
- Fig. 1.6.2 shows the client server model.

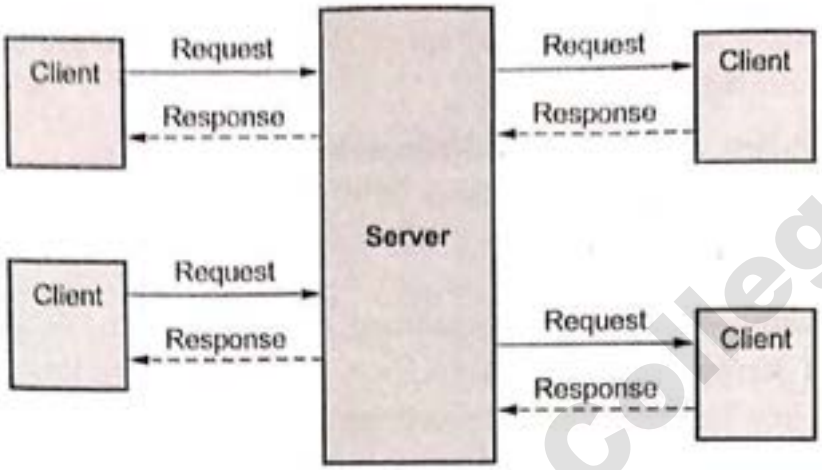


Fig. 1.6.2 Client server model

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using Remote Procedure Calls (RPC) or Remote Method Invocation (RMI) :
  1. The client sends a request (invocation) message to the server asking for some service;
  2. The server does the work and returns a result (e.g. the data requested) or an error code if the work could not be performed.
- Server is a process; Client is a process. Clients invoke servers, servers send results to clients. Servers can be clients of other servers.
- HTTP (Web) server is a server process to its client processes (web browsers). HTTP server may be a client of a database server. Service may be provided by multiple servers, as is most often the case within a large enterprise.
- Cache is a repository of recently accessed objects (files, graphics) that is physically closer to the client than the server from which it originated. Proxy server sits in between clients and servers and can play many mitigation roles.

Advantages :

1. Simple to implement.
2. Provides good security
3. All files are stored in a central location.

Disadvantages :

1. Single point of failure
2. A specialist network operating system is needed.

1.6.2 Peer-to-Peer System

- All processes (objects) play similar role. Do not require a server process. Processes (objects) interact without particular distinction between clients and servers.
- The pattern of communication depends on the particular application. Fig. 1.6.3 shows the peer-to-peer model.

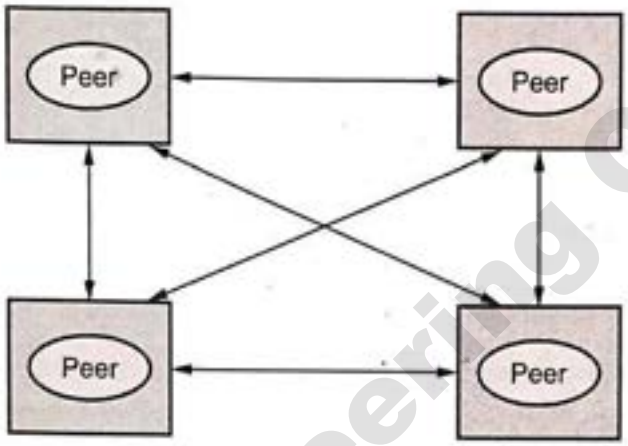


Fig. 1.6.3 Peer to peer model

- A large number of data objects are shared; any individual computer holds only a small part of the application database. Processing and communication loads for access to objects are distributed across many computers and access links. This is the most general and flexible model.
- A group of computers connected together to combine their computing and processing abilities to search the Internet or solve very complex problems
- Problems with peer-to-peer : High complexity due to
  1. Cleverly place individual objects
  2. Retrieve the objects
  3. Maintain potentially large number of replicas.

1.6.3 Distinguish between Client - Server and Peer-to-Peer Model

Client–server model	Peer-to-peer model
The client-server model firmly distinguishes the roles of the client and server. Under this model, the client requests services that are provided by the server	The peer-to-peer model doesn't have such strict roles. In fact, all nodes in the system are considered peers and thus may act as either clients or servers or both

Single point of failure	No single point of failure
Need for dedicated application and database servers	No need for dedicated application and database servers
Resources may be removed at any time	Resources are not removed from the network until they are no longer being requested.
Storage and bandwidth must be provided by the host.	Storage and bandwidth are distributed and provided by the entire network.
Provides good security	Provides Poor security

University Question

1. *Distinguish between the client-server and 'peer-to-peer' models of distributed systems.*

AU : May-16, Marks 8

1.7 Network Operating System

- Network operating system is designed by using existing operating system.
- In network operating system, user can login to the remote resources and then access that resource. There are multiple machines are connected by communication link. User can transfer data from their own machine to remote machine and vice versa.
- Telnet and remote file transfer are the example of network operating system.
- NOS supports multiple user accounts at the same time and enables concurrent access to shared resources by multiple clients.
- A function of network operating system is distributed over the network computers. It adds new functions whenever required. Server machine support multitasking. Operating system must be capable of executing multiple tasks at the same time.
- User knows the location of the resources before using that resource. Each machine has its own network operating system.

1.8 Real Time Systems

- Time constraints is the key parameter is real time systems. It controls autonomous system such as robots, satellites, air traffic control and hydroelectric dams.
- When user gives an input to the system, it must process within the time limit and result is sent back. Real time system fails if it does not give result within the time limits.

- Real time systems are of two types : **Hard real time and soft real time.**
- Critical task is completed within the time limit in hard real time system. All the delay in the system is fixed and time bounded. Existing general purpose operating system does not support the hard real time system functions. Real time task cannot keep waiting for longer time without allocating kernel.
- Soft real time system is less restrictive type. Soft real time cannot guarantee that it will be able to meet deadline under all condition. Example of soft real time system is digital telephone and digital audio.
- Real time operating system uses priority scheduling algorithm to meet the response requirement of a real time application. General real time applications with their example are listed below :
  1. Transportation : Air traffic control and traffic light system
  2. Communication : Digital telephone
  3. Process control : Petroleum and paper mill
  4. Detection : Burglar system and radar system
  5. Flight simulation : Auto pilot shuttle mission simulator.
- Memory management in real time system is less demanding than in other type of multiprogramming operating systems. Time critical device management is one of the main characteristics of the real time systems.

1.9 Handheld Systems

- Mobile phone is the example of handheld system. Physical size is the main constraints for handheld systems. Development of handheld system is major problem.
- Handheld system contains slow processor, small display and less memory. Maximum memory supported by handheld system is 8 MB. Operating system and applications must manage memory more efficiently. Memory manager take care of allocating and de-allocating of memory. Makes memory immediately free when task is completed.
- Processor speed is limited because of the power supply. Handheld system operates on the battery and battery power is limited. To increase the processor speed, battery capacity must be increased or recharged more frequently.
- Operating system used now a day for handheld device is Blackberry, Android, windows etc.
- Designing the display screen for such type of device is critical job. User requires the bigger screen but providing bigger screen means consuming space of other

components. Keyboard is provided with screen which is called touch screen keyboard. When keyboard is not used that time it is used for display screen. User can watch video, TV and play online games.

- Handheld device must support wireless technology. Bluetooth, wi-fi connectivity are the major features of handheld device. Using these facility, user can connect to the internet and check the email, browse the web.
- Bluetooth facility is used to connect with other handheld system or other devices. User can transfer files, audio, video using Bluetooth facility.

1.10 Characteristics of Modern Operating Systems

1. Microkernel architecture assigns only a few major functions to the kernel. It includes basic CPU scheduling, IPC, allocation of address space etc.
2. Process : Collection of threads.
3. Thread : Thread is light weight process. A thread is a flow of execution through the process code. It contains program counter, stack pointer and stack for data storage.
4. Multithreading : Process creates number of multiple threads. These all thread can run simultaneously.
5. Distributed operating systems : It provides transparency to the end user.

1.11 Operating System Services

- Operating system provides different types of services to different users. It provides services to program like load of data into memory, allocating disk for storage, file or directory for open or read etc.
- Services will change according to the operating system. May be the two different operating system provides same type of services with different names. OS makes programmer job easy by providing different services.
- Following are the list of services provided by operating systems :
  1. Program execution
  2. Input-output operation
  3. Error detection
  4. File and directory operation
  5. Communication
  6. Graphical User Interface

1. Program execution : Before executing the program, it is loaded into the memory by operating system. Once program loads into memory, its start execution. Program finishes its execution with error or without error. It is up to the user for next operation.

2. **Input - output operation :** Program is combination of input and output statement. While executing the program, it requires I/O device. OS provides the I/O devices to the program.
3. **Error detection :** Error is related to the memory, CPU, I/O device and in the user program. Memory is full, stack overflow, file not found, directory not exist, printer is not ready, attempt to access illegal memory are the example of error detection.
4. **File and directory operation :** User wants to read and writes the file and directory. User wants to search the file/directory, rename file, and modify the file etc. user also create the file or directory. All these operation is performed by user by using help of operating system.
5. **Communication :** Communication may be inter-process communication and any other type of communication. Data transfer is one type of communication. Communication is in between two process of same machine or two process of different machine. Pipe, shared memory, socket and message passing are the different methods of communication.
6. **Graphical user interface :** User interacts with operating system by using user interface. All operating system provides user interface. Command line interface and batch interface are two types of user interface used in the operating system. In command line interface, user enters the text command for performing operation. Batch interface uses files. A file contains the command for various operations. When file executes, command output is displayed on the screen.

- All above services provided by operating system is only for single user. If suppose there are multiple user in the system, then operating system provides some additional services. These services are resource allocation, accounting, protection of data and security.
- Different types of resources are managed by the operating system. Multiple users require different resource for their execution. One type of resource may be required by two different users. So resource allocation is necessary.
- Accounting means keeping information of resources and users. Which types of resources are allocated to which user and whether particular user has permission for using this type of resources.

1.12 Operating System Structure

AU : Dec.-15, May-17, 18

- Modern operating system is complex in nature. Here we study different structure of the operating systems.

1.12.1 Simple Structure

- Microsoft-Disk Operating System (MS-DOS) is example of simple structure operating system. Most of the commercial systems do not have well defined structure. Initially DOS is small and simple in size. When new versions are introduced, size goes on increasing.
- There is no CPU Execution Mode (user and kernel), and so errors in applications can cause the whole system to crash.
- MS-DOS is layered structure operating system. It consists of following layers :
  1. Application program layer
  2. System program layer for resident program
  3. Device driver layer
  4. ROM BIOS device driver layer.
- In DOS, application program directly interact with BIOS device driver. If user makes any changes in the BIOS device driver, it creates the problem and affects the whole system. Memory size is also limited so after use memory must be made free for other users.
- Another example of this type is UNIX operating system. Initially it provided limited hardware functionality. UNIX is divided into two parts : **Kernel and system programs.**
- Kernel provides system calls for CPU scheduling, I/O management, file management and memory management. System call uses application program interface in UNIX. API defines user interface by using set of system programs. Kernel supports an API and user interface.
- To support more advanced version of hardware, new UNIX operating system was developed. It is consists of different modules for controlling the hardware. Module will communicate with each other for proper operation. Designers are free to design operating systems which support all new versions of hardware and user requirements.

1.12.2 Layered Approach

- Second type of operating system design is layered approach. Operating system is divided into number of layers. Each layer boundary is properly defined. Bottom layer is called layer 0 and top most layer is called layer N. Layer N provides user interface.

- A function of layer is also fixed. Fig. 1.12.1 shows the operating system layer. Each layer consists of data structure and a set of routines. Layer provides services to upper and lower layers.

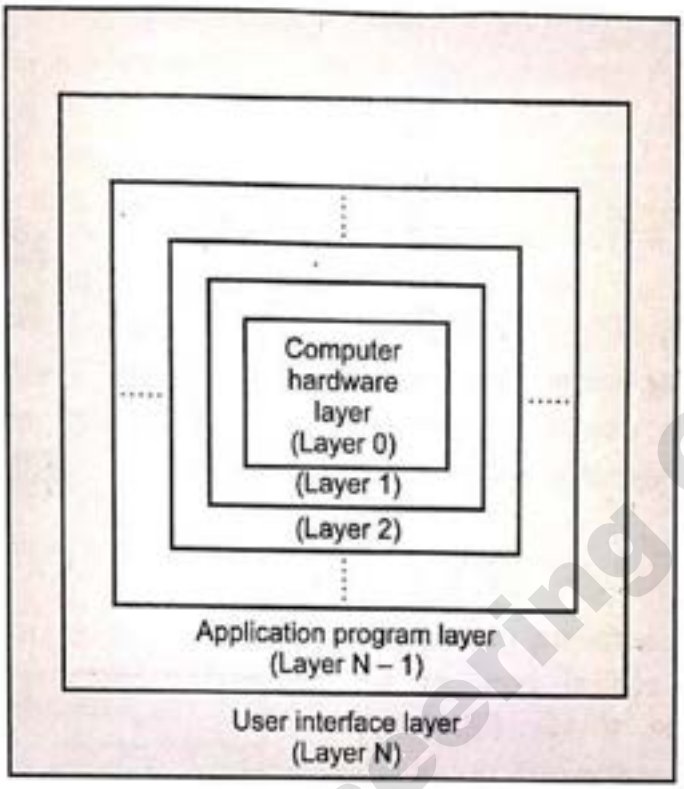


Fig. 1.12.1 Operating system layers

- Modularity is the advantage of the layered system. Number of layers are selected properly and each layer uses the services and functions of lower level layer (Below layer of the current layer). This type of design simplifies the debugging and system verification.
- First layer contains only basic hardware to implement function. So only first layer is debugged for checking the whole system. If error is not found then the system will works properly. If error is encounter while debugging second layer, then error is related to the second layer only. In this operating system is designed and implemented. It simplifies the design task of the operating system.
- Care should be taken while designing the layers. Which functions are added to which layer must be designing properly? Layer boundaries must be defined properly. Some of the function includes only in the lower layer. Device driver, memory management and input/output operation function must be included in the lower layer.
- Secondary storage is required for all operation. When CPU changes the process as per scheduling method, currently executing process is stored on the secondary

storage. Disk space is required for this purpose. So CPU scheduling is includes i the above layer of the secondary storage layer.

- Each layer uses system calls for performing their operation. Layer adds overhea to the system call.

University Questions

1. *Enumerate the different operating system structures and explain with neat sketch.* **AU : Dec.-15, Marks 8**
2. *Explain the various structures of an operating system.* **AU : May-17, Marks 8**
3. *State the operating system structure. Describe the operating system operations in detail. Justify the reason why the lack of a hardware-supported dual mode can cause serious shortcoming in an operating system. ?* **AU : May-18, Marks 13**

1.13 Kernel

AU : Dec.-15

- The Kernel is a software code that resides in the central core of a operating system. It has complete control over the system.
- A Kernel is different than the shell. The shell is the outermost part of an operating system and a program that interacts with user commands.
- Fig. 1.13.1 shows the position of the Kernel.

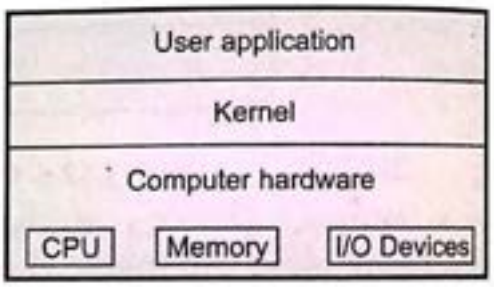


Fig. 1.13.1 Kernel

- The kernel does not interact directly with the user. But it interacts with the shell other programs and hardware.
- When operating system boots, kernel is the first part of the operating system t load into memory. Kernel remains in memory for the entire duration of th computer session. The kernel code is usually loaded into a protected area o memory.
- The kernel performs its tasks in kernel space. Executing processes and handlin interrupts are performed by kernel in kernel space. User performs its task in use area of memory. Memory is divided into system area and user area. This memor separation is made in order to prevent user data and kernel data from interferin with each other.

- When a computer crashes, it actually means the kernel has crashed. Single program crash is not a kernel crash. The kernel provides services for process management, memory management, file management and I/O management. System calls are used for providing this type of services.
- Kernel content will change according to the operating system but typically it includes :

  1. Scheduler 2. Supervisor 3. Interrupt handler 4. Memory manager
- Scheduler allocates the kernel's processing time to various processes. Supervisor grants the permission to use computer system resources to each process. Interrupt handler handles all requests from the various hardware devices which compete for the kernel's services. Memory manager allocates space in memory for all users of the kernel services.

Types of kernel :

1. Monolithic kernel 2. Microkernel 3. Hybrid kernel 4. Exo-kernel

1.13.1 Monolithic Kernel

- Traditional UNIX operating system uses monolithic kernel architecture. The entire operating system runs as a single program in kernel mode. Program contains operating system core function and device drivers.
- Most of the operation performed by kernel is via system call. Required system calls are made within programs and a checked copy of the request is passed through a system call. Fig. 1.13.2 shows monolithic kernel.

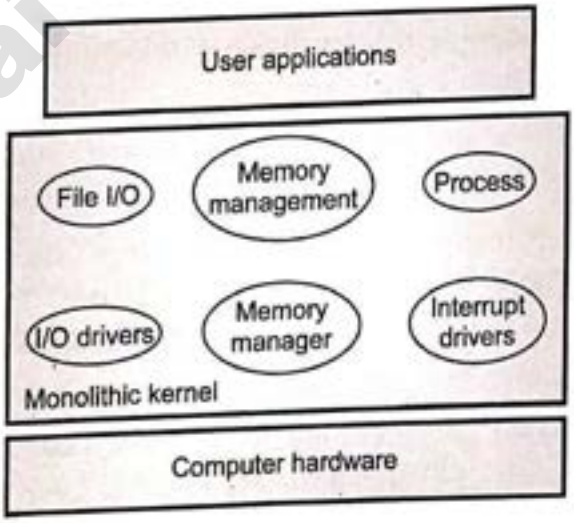


Fig. 1.13.2 Monolithic kernel

- LINUX operating system and FreeBSD uses modern monolithic kernel architecture. It loads the modules at run time. There is easy access of kernel function as required and minimize the code running in kernel space.
- Monolithic kernel used in Windows 95, Windows 98, Linux and FreeBSD etc.

Advantages :

1. Simple to design and implement.
2. Simplicity provides speed on simple hardware.
3. It can be expanded using module system.
4. Time tested and design well known.

Disadvantages :

1. Runtime loading and unloading is not possible because of module system.
2. If code base size increases, maintain is difficult.
3. Fault tolerance is low.

1.13.2 Microkernel

- Microkernel provides minimal services like defining memory address space, IPC and process management. It is small operating core. Hardware resource management is implemented whenever process is executing.
- The function of microkernel is to provide a communication facility between the client programs. It also provides facility to various services which are running in user space. Message passing method is for communication two processes.
- Microkernel runs in kernel mode and rest run in normal user processes. Microkernel also provides more security and reliability. Most of the services are running as user rather than kernel processes.
- By running device driver and file system as a separate user process, a error in one can crash only single component. Fig. 1.13.3 shows microkernel.
- Mach operating system uses microkernel architecture. Microkernel in Windows NT operating system provides portability and modularity. The kernel is surrounded by a number of compact subsystems so that the task of implementing Windows NT on a variety of platform is easy.
- Microkernel architecture assigns only a few essential functions to the kernel, including address space, IPC and basic scheduling. QNX is a real time operating system that is also based upon the microkernel design.
- Main disadvantage is poor performance due to increased system overhead from message passing.

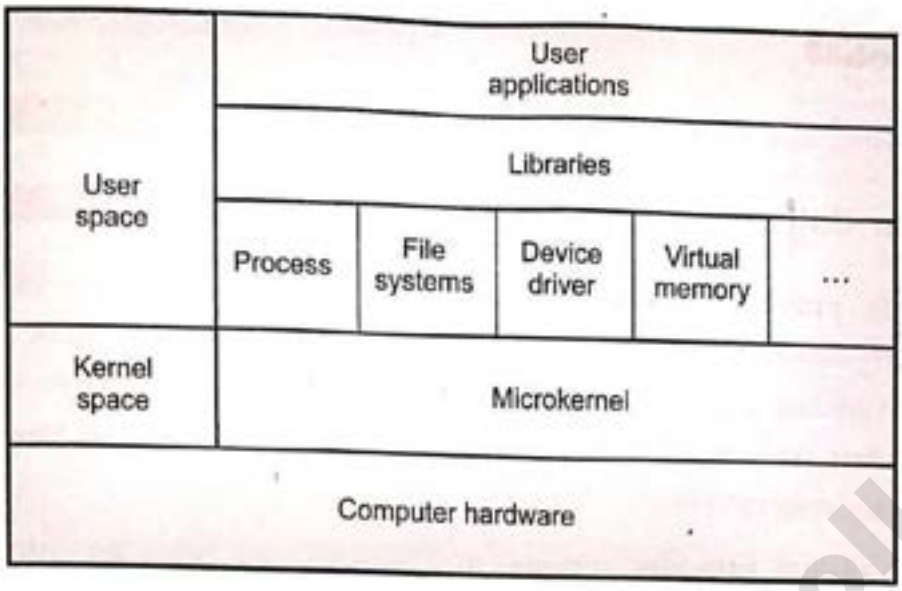


Fig. 1.13.3 Microkernel

Advantages of microkernel :

1. Microkernel allows the addition of new services.
2. Microkernel architecture design provides a uniform interface on requests made by a process.
3. Microkernel architecture supports object oriented operating system.
4. Modular design helps to enhance reliability.
5. Microkernel lends itself to distributed system support.
6. Microkernel architecture support flexibility. User can add or subtract services according to the requirement.

1.13.3 Comparison between Monolithic Kernel and Microkernel

Sr. No.	Monolithic Kernel	Microkernel
1.	Kernel size is large.	Kernel size is small.
2.	OS is complex to design.	OS is easy to design, implement and install.
3.	Request may be serviced faster.	Request may be serviced slower than monolithic Kernel.
4.	All the operating system services are included in the Kernel.	Kernel provides only IPC and low level device management services.
5.	No message passing and no context switching are required while the Kernel is performing the job.	Microkernel requires message passing and context switches.

University Question

1. *Explain system calls system programs and OS generation.* **AU : Dec.-15, Marks 8**

1.14 System Call

AU : Dec.-15,16, May-15,16,17

- System calls provide the interface between a running program and the operating system. Any single CPU computer can execute only one instruction at a time. If a process is running a user program in user mode and needs a system service, such as reading a data from a file, it has to execute a trap instruction to transfer control to the operating system.
- Operating system provides services and system call provides interface to these services. System call is written in language C and C++ as routines. System calls are performed in a series of steps.
- System call is a technique by which a program executing in user mode can request the kernel's service.
- An application programmer interface is a function definition that specifies how to obtain a given service.
- Fig. 1.14.1 shows the working of system call. (See Fig. 1.14.1 on next page)
- When application program calls the stub, trap instruction is executed and CPU switches to supervisor mode. Each system call contains its identification number.
- OS maintains the table of system call number. Operating system executes the system call using that number.
- When the function completes, it switches the processor to user mode and then returns control to the user process.
- A system call is an explicit request to the kernel mode via a software interrupt. When user mode process invokes a system call, the CPU switches to kernel mode and starts the execution of the kernel function.
- Making a system call is like making a special kind of procedure call. Only system call enters the kernel and procedure call does not enter into the kernel.
- Kernel implements many different types of system calls. The user mode process must pass a parameter called the system call number to identify the required system call. All system calls return an integer value. In the kernel, positive or 0 values denote a successful termination of the system call and negative values denote an error condition.

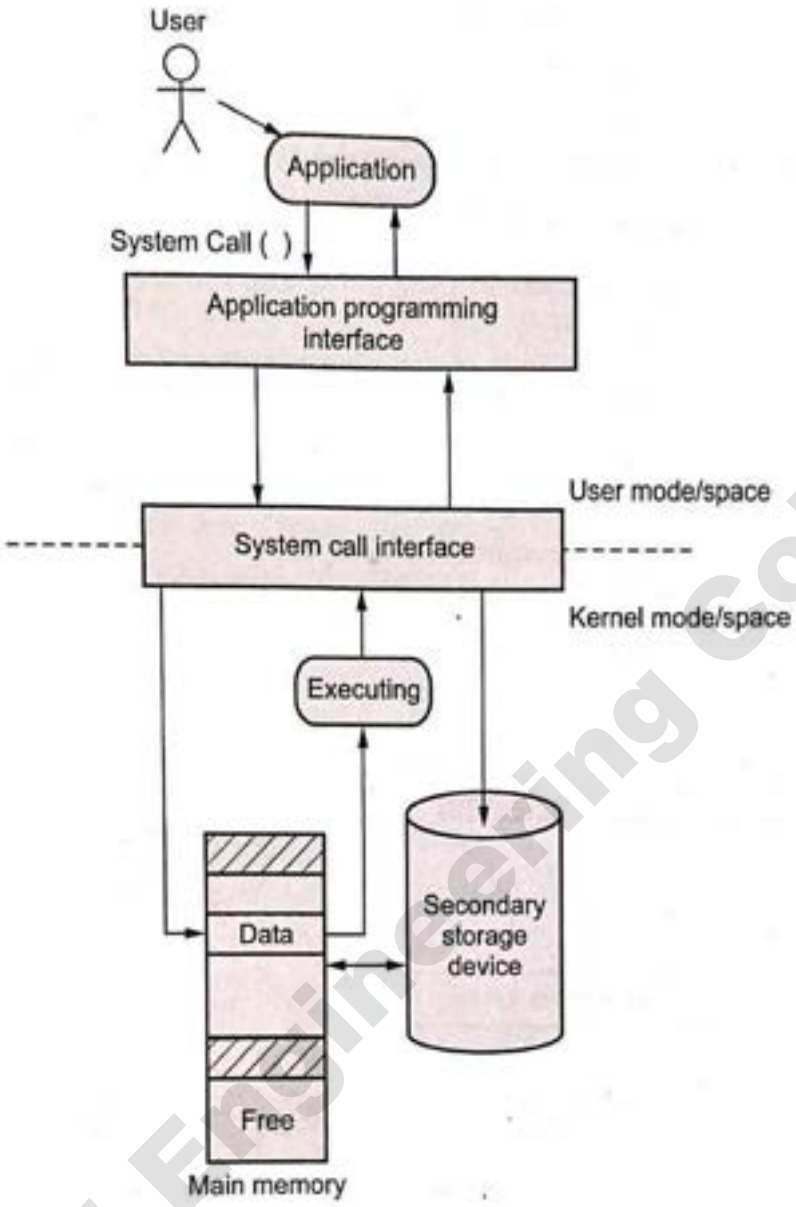


Fig. 1.14.1 Working of system call

- An API does not necessarily correspond to a specific system calls.
  1. API could offer its services directly in user mode.
  2. Single API function could make several system calls.
  3. Several API functions could make the same system call.
- API supports a set of functions for application programmer. It includes passing parameter to each function and return values. API used by application programmer are as follows :
  1. Windows system : win32 API
  2. POSIX system : POSIX API
  3. Virtual machine : Java API

- User function uses trap instruction for using kernel services. Application programmer uses ordinary procedure call. Operating system provides a library of user functions with name corresponding to each actual system call. Each of these stub function contains a trap to the operating system function.
- Trap is also called system call interface.
- Three general methods are used to pass parameters to the operating system.
  a. Pass parameters in registers
  b. Registers pass starting addresses of blocks of parameters
  c. Parameters can be placed, or pushed, onto the stack by the program, and popped off the stack by the OS
- Fig. 1.14.2 shows passing of parameters as a table

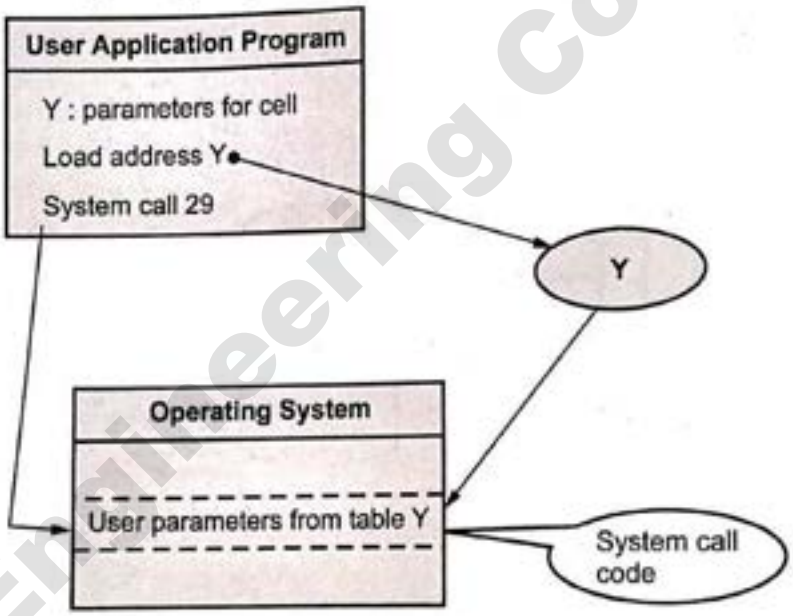


Fig. 1.14.2 Passing of parameters

1.14.1 Classification of System Call

- System call is divided into following types :
  1. File management
  2. Process management
  3. Interprocess communication
  4. I/O device management
  5. Information processing and maintenance

1. File management

- File management system calls are *create file, delete file, open file, close file, read file, write file, get and set file attribute.*

- User can create a file using create( ) system call. File name with attributes are required for creating and deleting a file through system call. After creating a file, user can performs various operations on the file.
- Read, write, reposition are the operation performed on the file. File is closed after finished using. Same type of operation is performed on directory.
- Every file has file attributes. File attributes includes name of file, type of file, accounting information etc. To perform any operation on the file, set file attribute and get file attribute executed to check the attributes.

2. Process management

- System calls for process management are create, terminate, load, execute, abort, set and get process attributes. Other system call for process management is wait for time, wait event, allocate and free memory.
- In some situation user want to terminate the currently running process abnormally then system call used. Other reasons for abnormal process termination are error message generated, memory dump, error trap.
- Operating system provides debugging facility to determine the problem of dump. Dump is written to secondary storage disk.
- Debugger is a one type of system program. It provides facility for finding and correcting bug.

3. Interprocess communication

- Pipe, socket, message passing and shared memory are used for interprocess communication. Send message, receive message, create and delete connection, attach remote device are the system calls used in interprocess communication.
- Shared memory : A process uses memory for communication. One process will create a memory portion which other processes can access. A shared segment can be attached multiple times by the same process. Shared memory is the fastest form of IPC because data does not need to be copied between processes.
- A *socket* is a bidirectional communication device that can be used to communicate with another process on the same machine or with a process running on other machine.
- Message passing : Two processes communicate with each other by passing messages. Message passing is direct and indirect communication. Indirect communication uses mailbox for sending receiving message from other process.

4. I/O device management

- System calls for device management are request ( ) device, release ( ) device, get and set device attributes, read, write etc.

- Process needs several resources in its life time. When process request for resources, request is granted if it is free otherwise request is rejected. Once request is granted, control is transfer to the process.

5. Information processing and maintenance

- System calls for this category is set time and date, get time and date, get system data, get system data, get and set process, file and device.
- Most of the operating system provides system call for set and get the time and date.
- This type of system call is used for transferring information from user to operating system and vice versa.

University Questions

1. *Explain the various types of system calls with an example for each.* **AU : May-15, Marks 10**
2. *Explain system calls system programs and OS generation.* **AU : Dec.-15, Marks 10**
3. *Describe three general methods for passing parameters to the operating system with example.* **AU : May-16, Marks 8**
4. *What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices ?* **AU : Dec.-16, Marks 8**
5. *Discribe system calls and system programs in detail with neat sketch.* **AU : May-17, Marks 5**

1.15 Essential Properties of the Operating System

1. **Batch system :** Jobs with similar needs are grouped together and feed to the computer system. Operator executes these jobs in batch on computer system. By keeping CPU and I/O device busy, it increases the performance. Buffering and spooling methods are also used for increasing performance of batch system. There is little or no memory management. This type of system is suitable for executing larger jobs that needs little interaction.
2. **Time sharing system :** It increases degree of multiprogramming. It uses CPU scheduling for better performance. CPU switches rapidly from one user to another. CPU is shared between a number of interactive users.
3. **Real time system :** Time constraints is the key parameter is real time systems. It controls autonomous system such as robots, satellites, air traffic control and hydroelectric dams. Real time system are usually dedicated, embedded system. The system must guarantee response to events within fixed periods of time to ensure correct performance.

4. **Distributed system :** Each processor has its own local memory. They communicate with each other through various communication lines. Processor does not share memory and clock.

1.16 Operating System Design and Implementation

Design :

- Before designing the system, first define goals and specification of the system. Goals and specification will give the idea about the system and it also understands the requirement of the user. At the higher level, system design is dominated by the choice of hardware and system type.
- System type may be batch system, multiprogramming, multitasking, real time and distributed system. Hardware must support all these type of operating system.
- Once the goal and specification is clearly understood, then a requirement is related to user and system. These requirements are called user goals and system goals.
- User goals are as follows :
  1. Simple to use and understand.
  2. Provides reliability.
  3. Provides security.
  4. Fast speed.
- System goals are as follows :
  1. Easy to design
  2. Implementation and maintenance is also simple and easy
  3. Free from error
  4. Efficient to use.
- Operating system requirements is not fixed. It will change according to the user requirements. Single user operating system is required and multiple user operating systems is also required.

Implementation

- At first, operating systems were written in assembly, but now a days C/C++ is the language commonly used.
- Small blocks of assembly code are still needed, especially related to some low level I/O functions in device drivers, turning interrupt on and off and the Test and Set instruction for synchronization facilities.
- Using higher level language allow code to be written faster, is easier to read, and can be debugged easier. It also makes the OS much easier to port to different hardware platforms.

1.17 OS Operations

- For single-user programmer operating systems, programmer has the complete control over the system. They operate the system from the console. When new operating systems developed with some additional features, the system control transfers from programmer to the operating system.
- Early operating systems were called resident monitors and starting with the resident monitor, the operating system began to perform many of the functions, like input-output operation.
- Before the operating system, programmer is responsible for the controls of input-output device operations. As the requirements of programmers from computer systems go on increasing and development in the field of communication helps to the operating system.
- Sharing of resource among different programmers is possible without increasing cost. It improves the system utilization but problems increase. If single system was used without share, an error occurs, that could cause problems for only the one program which was running on that machine.
- In sharing, other programs also affected by single program. For example, batch operating system faces the problem of infinite loop. This loop could prevent the correct operation of many jobs. In multiprogramming system, one erroreous program affects the other program or data of that program.
- For proper operation and error free result, protection of error is required. Without protection, only single process will execute one at a time otherwise the output of each program is separated. While designing the operating system, this type of care must be taken into consideration.
- Computer hardware detects the errors. Operating system handled this type of errors. Execution of illegal instruction or access of memory that is not in the user's address space, this type of operation found by the hardware and will trap to the operating system.
- The trap transfers control through the interrupt vector to the operating system. Operating system must abnormally terminate the program when program error occurs. To handle this type of situation, different types of hardware protection is used.

1.17.1 Dual Mode Operation

- For proper operation and correct output, operating system must be protected. The users program and data must be protected from any malfunctioning program. Shared resource also needs some kind of protection.

- In dual mode operation, two separate modes are used for working of operating system. These modes are **user mode** and **monitor mode.** The monitor mode also called system mode, supervisor mode or privileged mode.
- For indicating mode of the system, mode bit is used in the computer hardware. The mode bit is 0 for monitor and 1 for user. With the mode bit, we are able to distinguish between a task that is executed in user mode or monitor mode. This feature helps to the operating system in many ways.
- At the booting time, the hardware starts in the monitor mode, then operating system is loaded. The hardware switches from user mode to monitor mode when interrupts occur. When the operating system gains control of the system, it is in monitor mode.
- The dual mode operation provides the protection to the operating system from unauthorised users. The privileged instructions are executed only in the monitor mode. The computer hardware is not allowed for executing the privillege instructions in other mode, i.e. user mode. If anybody tries to execute the instructions in user mode, it is considered as illegal instruction and also traps it to the operating system.
- Software may trigger an interrupt by executing a special operation called a system call. System call is one type of request which is invoked by the user or system. Using privileged instructions, user will interact with the operating system. This type of request is invoked by user to execute the privileged instructions. As said earlier, this request is called system call or monitor call. When a system call is executed, it is treated by the hardware as a software interrupt.

1.18 System Programs

- Modern operating system consists of a collection of system programs. System program that provides an application programming environment on top of the hardware. Some of them are simply user interfaces to system calls. They can be divided into these categories :
  1. File management
  2. Status information
  3. File modification
  4. Programming language support
  5. Program loading and execution
  6. Communications.

- File management programs are used to create, delete, copy, rename, list, dump or files and directory. Status information system programs covers the date, time, disk space, available memory and users. All this information is formatted and displayed on output device or printed.
- Text editors are used for file modification. In this, new file is created and content of file is modified. Programming language support includes the compilers, assemblers, interpreters for common programming language like C, C++, Java, Visual Basic.
- For program loading and execution, it is loaded into memory then program is executed by processor. Operating system provides different types loaders and linkers to complete execution operation.
- Debugging facility is provided by the operating system with the help of application program. It is used for checking errors.
- Communication between two devices are performed by creating temporary connection. Communication is in between process, users and other I/O devices. File transfer, remote login, electronic mail, browsing web, downloading multimedia data are the example of communication.

1.19 Virtual Machines

AU : May-15

- In a pure virtual machine architecture the operating system gives each process the illusion that it is the only process on the machine. The user writes an application as if only its code were running on the system.
- Each user interacts with the computer by typing commands to the virtual machine on a virtual system console and receiving results back from the machine as soon as they are computed.
- Each user directs the virtual machine to perform different commands. These commands are then executed on the physical machine in a multiprogramming environments.
- Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility.
- It allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine. Fig. 1.19.1 shows virtual machine.
- Each virtual machine has its own set of virtual hardware (e.g., RAM, CPU, NIC etc.) upon which an operating system and applications are loaded. The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

- The main components of virtual machine are the control program, conversational monitor system, remote spooling communication and interactive problem control system.
- The control program creates the environments in which virtual machines can executes. It also manages the real machines underlying the virtual machine environment.
- The java virtual machine is one of the most widely used virtual machines.
- Virtual machines tend to be less efficient than real machines because they access the hardware indirectly.

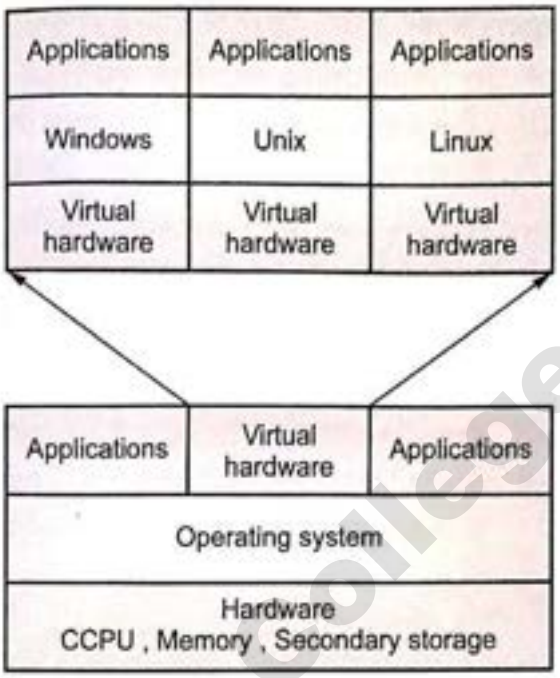


Fig. 1.19.1 Virtual machine

Benefits

1. There is no overlap amongst memory as each Virtual Memory has its own memory space.
2. Virtual machines are completely isolated from the host machine and other virtual machines.
3. Data does not leak across virtual machines.

Virtualization

- Virtualization means running multiple machines on a single hardware. The "Real" hardware invisible to operating system. OS only sees an abstracted out picture. Only Virtual Machine Monitor (VMM) talks to hardware.
- It is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. This includes making a single physical resource appear to function as multiple logical resources; or it can include making multiple physical resources appear as a single logical resource."
- It is divided into two main categories :
  1. Platform virtualization involves the simulation of virtual machines.
  2. Resource virtualization involves the simulation of combined, fragmented, or simplified resources.

Hypervisor

- In computing, a hypervisor is a virtualization platform that allows multiple operating systems to run on a host computer at the same time. The term usually refers to an implementation using full virtualization.
- Hypervisors are currently classified in two types :

1. Type 1 hypervisor is software that runs directly on a given hardware platform. A "guest" operating system thus runs at the second level above the hardware.
2. Type 2 hypervisor is software that runs within an operating system environment. A "guest" operating system thus runs at the third level above the hardware.

- Bochs and QEMU are PC emulators that allow operating systems such as Windows or Linux to be run in the user-space of a Linux operating system.
- VMware is a popular commercial full-virtualization solution that can virtualizes unmodified operating systems.
- Xen is an open source para-virtualization solution that requires modifications to the guest operating systems but achieves near native performance by collaborating with the hypervisor.
- Microsoft Virtual PC is a para-virtualization virtual machine approach.
- User-mode Linux (UML) is another para-virtualization solution that is open source. Each guest operating system executes as a process of the host operating system.
- Cooperative Linux, is a virtualization solution that allows two operating systems to cooperatively share the underlying hardware.
- Linux-Vserver is an operating system-level virtualization solution for GNU/Linux systems with secure isolation of independent guest servers.
- The Linux KVM is virtualization technology that has been integrated into the mainline Linux kernel . Runs as a single kernel loadable module, a Linux kernel running on virtualization-capable hardware is able to act as a hypervisor and support unmodified Linux and Windows guest operating systems.
- Fig. 1.19.2 shows para-virtualization architecture. In Para-virtualization, the virtual machine does not necessarily simulate hardware, but instead offers a special API that can only be used by modifying the "guest" OS. This system call to the hypervisor is called a "hypercall" in Xen.

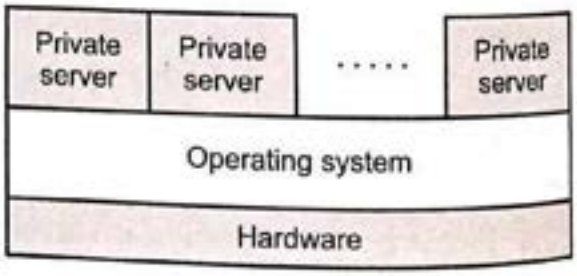


Fig. 1.19.2 Para-virtualization architecture

- Para-virtualization shares the process with the guest operating system

Problems with para-virtualization

1. Para-virtualized systems won't run on native hardware
2. There are many different para-virtualization systems that use different commands, etc.

Platform virtualization

- The creation of a virtual machine using a combination of hardware and software is referred to as platform virtualization
- Platform virtualization is performed on a given hardware platform by "host" software , which creates a simulated computer environment for its "guest" software.
- The "guest" software, which is often itself a complete operating system, runs just as if it were installed on a stand-alone hardware platform.
- Typically, many such virtual machines are simulated on a given physical machine.
- For the "guest" system to function, the simulation must be robust enough to support all the guest system's external interfaces, which may include hardware drivers.

Resource virtualization

- The basic concept of platform virtualization, was later extended to the virtualization of specific system resources, such as storage volumes, name spaces, and network resources.
- Resource aggregation, spanning, or concatenation combines individual components into larger resources or resource pools. For example : RAID and volume managers combine many disks into one large logical disk.
- Virtual Private Network (VPN), Network Address Translation (NAT), and similar networking technologies create a virtualized network namespace within or across network subnets. Multiprocessor and multi-core computer systems often present what appears as a single, fast processor.

University Question

1. *Discuss about the evolution of virtual machines. Also explain how virtualization could be implemented in operating systems.* **AU : May-15, Marks 10**

1.20 System Boot

AU : May-15,16

- Booting is the process of loading operating system into main memory. For a computer to start running, it needs to have an initial program to load and execute the boot program, which in turn loads the operating system.
- During bootstrapping, the kernel is loaded into memory and begins to execute. A variety of initialization tasks are performed and the system is then made available to users.
- The primitive loader program that can load and execute the boot program is called **Bootstrap program.** Bootstrap program is typically stored in ROM. On-start up, the computer automatically reads the bootstrap program. This primitive loader is then executed and loads the boot program in predetermined memory locations.
- The boot program is generally stored on disk with predetermined address, called **boot sector.** The boot program then loads the operating system into memory. This arrangement is known as **bootstrapping.**
- Boot time is a period of special vulnerability. Errors in configuration files, missing or unreliable equipment, and damaged file systems can all prevent a computer from coming up. Boot configuration is often one of the first tasks an administrator must perform on a new system. Unfortunately, it is also one of the most difficult, and it requires some familiarity with many other aspects of UNIX.

Recovery boot to a shell

- In normal operation, systems boot themselves independently and are then accessed remotely by administrators and users. Administrator need a recovery tool they can use if a disk crashes or configuration problem prevents the system from completing the normal boot process.
- Instead of shooting for full system operation, UNIX systems can boot just enough to run a shell on the system console. This option is traditionally known as booting to single user mode, recovery mode, or maintenance mode.
- Single user mode can be useful for checking and repairing operating systems, particularly those that have been damaged and will not allow booting into the default GUI or console multi-user mode.
- In single-user mode, your computer boots to *runlevel 1*. Your local file systems are mounted, but your network is not activated. You have a usable system maintenance shell. Unlike rescue mode, single-user mode automatically tries to mount your file system. Do not use single-user mode if your file system cannot be mounted successfully. You cannot use single-user mode if the *runlevel 1* configuration on your system is corrupted.

- Run levels are numbered mode of operation which activated as preconfigured set of services. It is defined in */etc/inittab* file and default run level 5. A run level is a state of init and the whole system that defines what system services are operating. Run levels are identified by numbers. Some system administrator's uses run levels to define which subsystems are working, e.g., whether X is running, whether the network is operational, and so on.

Run level in Linux are as follows :

Run level	Name	Descriptions
0	Halt	Shut down all services when the system will not be restarted.
1	Single user mode	Used for system maintenance. Do not require logging in with a user name and password. It operates as root but do not provide networking capabilities.
2	Multi-user mode without networking enabled	Rarely used except for system maintenance or testing.
3	Regular multi-user networking mode	Standard multi-user text mode operation. Non-graphical systems use this mode for normal operation.
4	--	Not used
5	Graphical login	Identical to run level 3 except a graphical login is used.
6	Reboot	Shuts down all services so the system can be restarted.

- On most system, you request a boot to single user mode by passing an argument to the kernel at boot time. If the system is already up and running, you can bring it down to single user mode with the **shutdown** or **telinit** command.

1.20.1 Steps in Boot Process

- Normal UNIX boot process has these main phases :
  1. Basic hardware detection.
  2. Executing the firmware system initialization program.
  3. Locating and running the initial boot program, usually from the predefined location on the disk.
  4. Locating and stating the UNIX kernel.
  5. The kernel initializes itself and then performs final, high level hardware checks, loading device drivers and/or kernel modules are required.

6. The kernel starts the init process, which in start the system processes and initializes all active subsystems. When everything is ready, the system begins accepting user logins.

1.20.2 Kernel Initialization

- Once the bootstrap program has installed the kernel into memory the kernel will
  1. Initialize its internal data structures,
  2. Perform some further hardware checking,
  3. Verify the integrity of the root file system and then mount it, and
  4. Create the process 0 (swapper) and process 1 (init).
- The swapper process is actually part of the kernel and is not a "real" process. The init process is the ultimate parent of all processes that will execute on a UNIX system.
- The UNIX kernel is a program. The kernel usually resides in the root partition of the UNIX file system.
- Most Linux distributions call this partition either "/vmlinuz" or "/boot". Other UNIX flavored systems may call this partition "/unix", "/vmunix", or "/kernel".
- Almost every vendor of Linux or UNIX calls their kernel pathname something different. Most systems implement a two stage loading process.
  1. First stage : The system ROM loads a small boot program into memory from disk. This program called the boot loader, and then arranges for the kernel to be loaded. This procedure occurs outside the domain of UNIX.
  2. The kernel probes the system to learn how much RAM is available. Size of the internal data structure for some kernel is fixed and it set aside some memory for itself when it starts. This memory is reserved for the kernel and cannot be used by the user level processes. The kernel prints a message on the console that reports the total amount of physical memory and the amount of available to user processes.

Hardware configuration

- The kernel inspects the installed hardware on the system and attempts to mount the proper drivers for each device that you tell the kernel it should expect to find. This is visually noticeable during the cryptic messages that appear on the screen while the computer boots.
- Devices that have missing drivers or that did not respond during boot-up will be disabled and will not be accessible to UNIX until the system is rebooted.

- Hardware configuration is relatively transparent process for administrators, especially under Linux. Kernels distributed by vendors are extremely modular and will automatically detect most hardware.

Creation of Kernel processes (System processes)

- After basic initializations have been completed, processes which are *NOT* created by the UNIX **fork** mechanism execute. The number of these processes varies in the various flavors of UNIX and Linux, but **init** is the common process throughout them all.
- The purpose of these system processes is to handle tasks that the kernel would handle but have been created separately for scheduling or architectural reasons.
- At this point, the bootstrapping process is complete, but none of the processes that handle basic operations have been created. Those processes are taken cared of by **init.** Only **init** is really a full fledged user process.
- Following are some common kernel processes on Linux systems.

Thread	What it does ?
kjournald	Commits file system journal updates to disk
kswapd	Swaps processes when physical memory is low.
Ksoftirqd	Handles soft interrupts if they can not be dealt with at content switch time
khubd	Configures USB devices

- A UNIX system creates similar kernel processes but since these processes represents aspects of the kernel implementations, none of the names or functions are necessarily common among systems. Administrators never need to interact with these processes directly.

Operator intervention (Manual boot)

- A command-line flag is brought up to signal init to go into single-user mode. From here you can choose which scripts to run, but only the root partition is mounted so you must first mount other partitions before you can start processed off of those partitions.
- Daemons do not typically run in single-user mode.
- Many single-user environments are mounted as read-only by default, therefore if "/tmp" is apart of the root partition then programs that need to write to that location, will not work.

- Normally, the file system is checked automatically by *fsck*, but you must do this manually during single-user mode. After the single-user shell has completed, the system will resume automated boot-up.

Multi-user operation

- After Initialization scripts have ran, the system is fully operational, but additional processes have to run in order to allow multiple users to log on to the system.
- The **getty** process listen for logins which is spawned from **init.**
- Graphical login systems include **xdm, gdm,** and **dtlogin.**

1.20.3 BIOS Initialization

- On a desktop PC, the boot loader resides on the Master Boot Record (MBR) of the hard drive and is executed after the PC's Basic Input Output System (BIOS) performs system initialization tasks.
- BIOS is Basic Input Output Software, it is a complex of system configuration software routines that is stored on flash memory to facilitate upgrade after that. BIOS run immediately after power on PC and used to initialize hardware specially the system and load the operating system.
- When machine boots, it begins by executing code stored in ROMs. The exact location and nature of this code varies, depending on the type of machine you have. For UNIX type machine, the code is typically firmware that knows how to use devices connected to the machine, how to talk to the network on a basic level, and how to understand disk based file systems.
- BIOS refer to the software code run by a computer when first powered on. The primary function of BIOS is code program embedded on a chip that recognizes and controls various devices that make up the computer.

1.20.4 Master Boot Record (MBR)

- OS is booted from a hard disk, where the MBR contains the primary boot loader. The MBR is a 512-byte sector, located in the first sector on the disk (sector 1 of cylinder 0, head 0). After the MBR is loaded into RAM, the BIOS yields control to it. In general, a boot loader gets installed in the MBR, removing its previous content. Fig. 1.20.1 shows MBR.
- The first 446 bytes are the primary boot loader, which contains both executable code and error message text. The next sixty-four bytes are the partition table, which contains a record for each of four partitions.
- The MBR ends with two bytes that are defined as the magic number (0xAA55). The magic number serves as a validation check of the MBR.

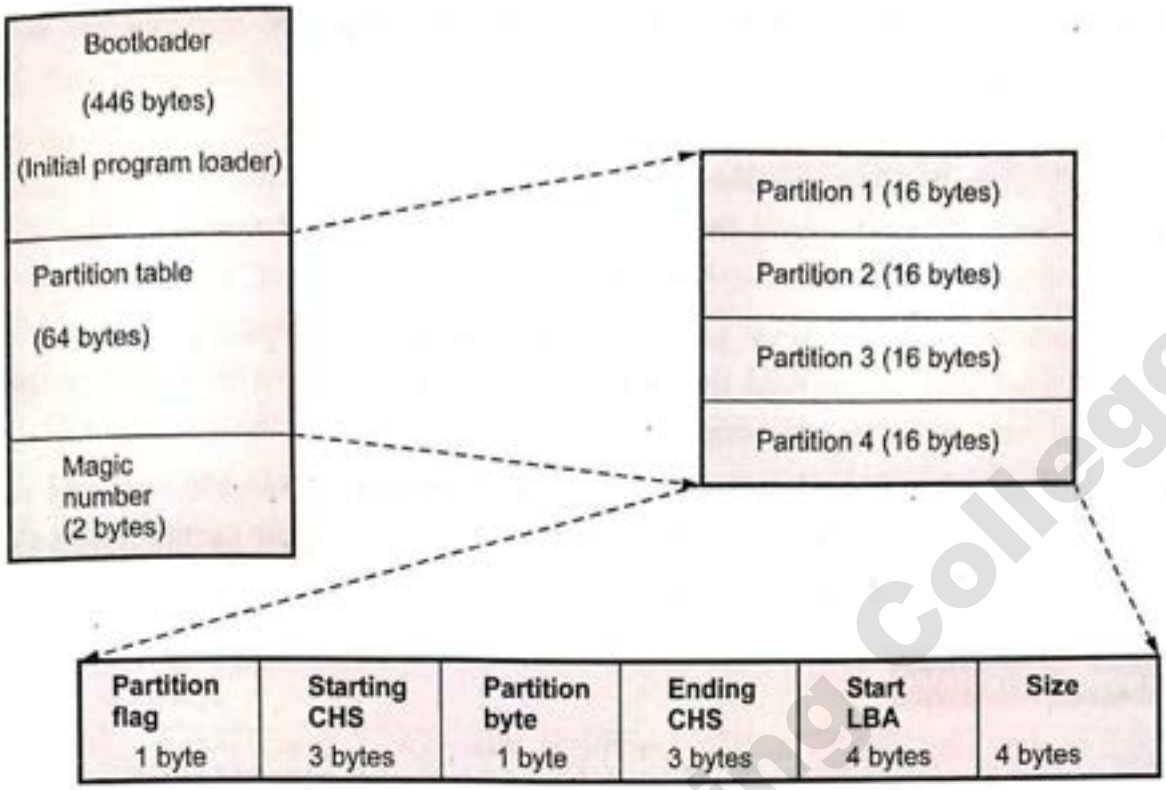


Fig. 1.20.1 Master boot record

- Partition table occupy the area of 64 bytes of the partition table. In this area, it stores the data structure which provides basic information of the operating system and the division of the HDD into primary partitions.
- Primary partitions are the first four partitions on a HDD. The final two bytes of this sector are a signature to indicate the end of the boot sector.
- This leaves only 446 bytes for the code used by the MBR to perform its tasks of locating the partition that is marked active (which is referred to as the active partition), loading the first sector from that partition into the appropriate memory address, and then transferring execution to that code.
- The starting and ending cylinder, head, and sector fields are collectively known as the CHS fields. These are additional elements of the partition table. These fields are essential for starting the computer. The master boot code uses these fields to find and load the boot sector of the active partition.
- Boot loader could be more rightly called the **kernel loader.** The task at this stage is to load the Linux kernel. GRUB and LILO are the most popular Linux boot loader.
- A popular Linux boot loader is LILO. LILO can be installed on either the MBR or the boot record of the Linux root partition. Installing LILO on the boot record allows you to use a different boot loader for another operating system like Windows XP.

- LILO can be configured through the "lilo" command. Its contents are stored at **"/etc/lilo.conf".**
- The default MBR contains a simple program that tells the computer to get its boot loader from the first partition on the disk. Once the MBR has chosen a partition to boot from, it tries to load the boot loader specific to that partition. This loader is then responsible for loading the kernel.
- An active partition is a partition that contains the operating system that a computer attempts to load into memory by default when it is started or restarted. Every hard disk has only one active partition.
- To boot the computer other than the active partition is possible by using LILO or GRUB. Logical partition can be used for booting. A logical partition is a partition that has been created inside of a primary partition.

University Questions

1. *Discuss about the functionality of system boot with respect to operating system.* **AU : May-15, Marks 6**
2. *How could a system be designed to allow a choice of operating systems from which to boot? What would the bootstrap program need to do?* **AU : May-16, Marks 8**

1.21 Process Management

- Process refers to a program in execution. The process abstraction is a fundamental operating system mechanism for management of concurrent program execution. The operating system responds by creating a process.
- A process needs certain resources, such as CPU time, memory, files and I/O devices. These resources are either given to the process when it is created or allocated to it while it is running.
- When the process terminates, the operating system will reclaim any reusable resources.
- The term process refers to an executing set of machine instructions. Program by itself is not a process. A program is a passive entity.
- The operating system is responsible for the following activities of the process management.
  1. Creating and destroying the user and system processes.
  2. Allocating hardware resources among the processes.
  3. Controlling the progress of processes.

4. Providing mechanisms for process communications.
5. Also provides mechanisms for deadlock handling.

1.22 Memory Management

AU : Dec.-16

- The memory management modules of an operating system are concerned with the management of the primary (main memory) memory. Memory management is concerned with following functions :
  1. Keeping track of the status of each location of main memory. i.e. each memory location is either free or allocated.
  2. Determining allocation policy for memory.
  3. Allocation technique i.e. the specific location must be selected and allocation information updated.
  4. Deallocation technique and policy. After deallocation, status information must be updated.
- Memory management is primarily concerned with allocation of physical memory of finite capacity to requesting processes. The overall resource utilization and other performance criteria of a computer system are affected by performance of the memory management module. Many memory management schemes are available and the effectiveness of the different algorithms depends on the particular situation.

University Question

1. *Describe a mechanism for enforcing memory protection in order to prevent a program from modifying the memory associated with other programs.* AU : Dec.-16, Marks 8

1.23 Storage Management

AU : Dec.-16, May-18

- OS provides uniform, logical view of information storage.
- It abstracts from the physical properties of its storage devices to define a logical storage unit, the **file**. It also maps files onto physical media and accesses these files via the storage devices.

1.23.1 File System Management

- Logically related data items on the secondary storage are usually organized into named collections called files. In short, file is a logical collection of information. Computer uses physical media for storing the different information.

- A file may contain a report, an executable program or a set of commands to the operating system. A file consists of a sequence of bits, bytes, lines or records whose meanings are defined by their creators. For storing the files, physical media (secondary storage device) is used.
- Physical media are of different types. These are magnetic disk, magnetic tape and optical disk. All the media has its own characteristics and physical organization. Each medium is controlled by a device.
- The operating system is responsible for the following in connection with file management.
  1. Creating and deleting of files.
  2. Mapping files onto secondary storage.
  3. Creating and deleting directories.
  4. Backing up files on stable storage media.
  5. Supporting primitives for manipulating files and directories.
  6. Transmission of file elements between main and secondary storage.
- The file management subsystem can be implemented as one or more layers of the operating system.

1.23.2 Secondary Storage Management

- A storage device is a mechanism by which the computer may store information in such a way that this information may be retrieved at a later time. Secondary storage device is used for storing all the data and programs. These programs and data access by computer system must be kept in main memory. Size of main memory is small to accommodate all data and programs. It also lost the data when power is lost. For this reason secondary storage device is used. Therefore the proper management of disk storage is of central importance to a computer system.
- The operating system is responsible for the following activities in connection with the disk management.
  1. Free space management 2. Storage allocation 3. Disk scheduling
- The entire speed and performance of a computer may hinge on the speed of the disk subsystem.

1.23.3 I/O System Management

- The module that keeps track of the status of devices is called the I/O traffic controller. Each I/O device has a device handler that resides in a separate process associated with that device.

- The I/O subsystem consists of
  1. A memory management component that includes buffering, caching and spooling.
  2. A general device driver interface.
  3. Drivers for specific hardware devices.

1.23.4 Caching

- Information is normally kept in some storage system i.e. main memory. While using, it is copied into a faster storage system-the cache-on a temporary basis.
- When user required a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache; if it is not, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.
- There are also caches that are implemented totally in hardware. For instance, most systems have an instruction cache to hold the next instructions expected to be executed. Without this cache, the CPU would have to wait several cycles while an instruction was fetched from main memory.
- Because caches have limited size, **cache management** is an important design problem. Careful selection of the cache size and of a replacement policy can result in greatly increased performance.
- The movement of information between levels of a storage hierarchy may be either explicit or implicit, depending on the hardware design and the controlling operating-system software. For instance, data transfer from cache to CPU and registers is usually a hardware function, with no operating-system intervention. In contrast, transfer of data from disk to memory is usually controlled by the OS.

University Questions

1. *State and explain the major activities of an operating system with regard to file management ?* **AU : Dec.-16, Marks 8**
2. *Describe the major activities of operating system with regards to file management.* **AU : May-18, Marks 5**

Two Marks Questions with Answers

Q.1 Define the degree of multiprogramming.

Ans. : Degree of multiprogramming is the number of processes in the memory.

AU : May-18

Q.2 Mention the purpose of system calls.

Ans. : System calls allow user-level processes to request services of the operating system.

Q.3 What is spooling ?

Ans. : The use of secondary memory as buffer storage to reduce processing delays when transferring data between peripheral equipment and the processors of a computer.

Q.4 What is meant by "hard real systems and soft real systems"?

Ans. : Hard real systems guarantee that critical tasks complete on time. In Soft real system a critical task get priority over other tasks and remains that priority until it completes.

Q.5 Explain what is batch processing.

Ans. : Here jobs with similar requirements are batched together and run through the computer as a group. Thus a batch operating system reads a stream of separate jobs, each with its own control cards that predefine what the job does, feed the batches one after another and send the output of each job to the appropriate destination.

Q.6 What are the main differences between operating systems for mainframe computers and personal computers ?

Ans. : Generally, operating systems for batch systems have simpler requirements than for personal computers. Batch systems do not have to be concerned with interacting with a user as much as a personal computer. As a result, an operating system for a PC must be concerned with response time for an interactive user. Batch systems do not have such requirements. A pure batch system also may not have to handle time sharing, whereas an operating system must switch rapidly between different jobs.

Q.7 What is difference between networked O.S. and distributed O.S. ?

Ans. : Difference between networked O.S. and distributed O.S. is an follows :

Sr. No.	Disributed O.S.	Networked O.S.
1.	Degree of transparency is high.	Degree of transparency is low.
2.	Same O.S. on all the nodes.	May be different or same O.S. on all the nodes.
3.	Shared memory is used for communication.	Files are used for communication.
4.	Scalability is less.	Scalability is high.
5.	Resource management is done globally or central.	Resource management is done by per node.

Q.8 State the function of Printer Daemon.

Ans. : The Printer Daemon provides printer services for local and remote users. It manages the printer spool area and the print queues. Printer Daemon is started at boot time from a startup script. It is generally included in the startup of Linux and BSD systems by default so you might not need to add it to your startup script.

Q.9 Distinguish between tightly coupled system and loosely coupled system.

Ans. :

Sr. No.	Tightly coupled system	Loosely coupled system
1.	Also known as parallel system.	Also known as distributed system.
2.	Shares computer bus, clock and sometimes memory.	Do not share memory or a clock.
3.	More than one processor in close communication.	Single processor system.

Q.10 What is the main advantage of multiprogramming ?

Ans. : Multiprogramming makes efficient use of the CPU by overlapping the demands for the CPU and its I/O devices from various users. It attempts to increase CPU utilization by always having something for the CPU to execute.

Q.11 What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment ?

Ans. : The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system it is running. Therefore when writing an operating system for a real-time system, the writer must be sure that his scheduling schemes don't allow response time to exceed the time constraint.

Q.12 What are the three main purposes of an operating system ?

Ans. :

- To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.
- To allocate the separate resources of the computer as needed to solve the problem given. The allocation process should be as fair and efficient as possible.
- As a control program it serves two major functions : 1) Supervision of the execution of user programs to prevent errors and improper use of the computer and 2) Management of the operation and control of I/O devices.

Q.13 What are the main advantages of the microkernel approach to system design ?

Ans. : Advantages are as follows,

1. Adding a new service does not require modifying the Kernel.
2. It is more secure as more operations are done in user mode than in Kernel mode.
3. A simpler Kernel design and functionality typically results in a more reliable operating system.

Q.14 Define real time system.

Ans. : Real time system is one that must react to inputs and responds to them quickly. A real time system has well defined, fixed time constraints.

Q.15 Describe the differences between symmetric and asymmetric multiprocessing.

Ans. : Symmetric multiprocessing treats all processors as equals and I/O can be processed on any CPU. Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes tasks among the slaves and I/O is usually done by the master only.

Q.16 What is the purpose of system programs ? AU : May-16

Ans. : System program that provides an application programming environment on top of the hardware.

Q.17 Mention the advantages in the design of distributed operating systems.

Ans. : Advantages are as follows :

1. Resource sharing
2. Higher reliability
3. Better price performance ratio
4. Shorter response times and higher throughput.

Q.18 List down the functions of operating systems. AU : CSE/IT : Dec.-10

Ans. : Functions of operating systems are convenience, efficiency and ability to evolve.

Q.19 What do you mean by multiprogramming ? AU : CSE/IT : Dec.-10

Ans. : Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

Q.20 What are the five major categories of system calls ? AU : CSE/IT : May-11

Ans. : Five main categories of system calls are : File management, IPC, process management, I/O devices management, information management.

Q.21 What is the function of system programs ? Write the name of the categories in which the system programs can be divided. AU : CSE/IT : May-11

Ans. : System programs provide a convenient environment for program development and execution. System programs are divided into these categories : File management, status management, file modification, programming language support, program loading and execution, communications.

Q.22 Define operating systems. AU : CSE/IT : Dec.-11

Ans. : An operating system is a program that manages the computer hardware.

Q.23 What is meant by a system call ? AU : CSE/IT : Dec.-11

Ans. : System calls provide the interface between a process and the OS.

Q.24 What does the CPU do when there are no user programs to run ? AU : CSE/IT : Dec.-11

Ans. : The CPU will always do processing. Even though there are no application programs running, the OS is still running and the CPU will still have to process many system processes during the operation of computer.

Q.25 What are the goals of OS ?

Ans. : Operating system goals :

a. Execute user programs and make solving user problems easier.

b. Make the computer system convenient to use.

Q.26 What is virtual machine ?

Ans. : A virtual machine is software that creates a virtualized environment between the computer platform and the end user in which the end user can operate software.

Q.27 List the disadvantages of multiprogrammed batched systems.

Ans. : Disadvantages

1. Users cannot interact with their jobs, while executing.
2. A programmer cannot modify a program as it executes to study its behavior

Q.28 What is tightly coupled system ?

Ans. : Processors share memory and a clock; communication usually takes place through the shared memory.

Q.29 What is command interpreter system ?

Ans. : Command-Interpreter system is a system program, which is the interface between the user and the operating system. Command-Interpreter system is known as the shell.

Q.30 List general methods used to pass parameters in system call.

Ans. : Three general methods are used to pass parameters between a running program and the operating system.

a. Pass parameters in registers.

b. Store the parameters in a table in memory, and the table address is passed as a parameter in a register.

c. Push (store) the parameters onto the stack by the program, and pop off the stack by operating system.

Q.31 What are the main advantages of layered approach ?

Ans. :

a. Each layer is implemented using those operations provided by lower-level layers.

b. A layer does not need to know how the low-level operations are implemented; it needs to know what these operations do.

c. Each layer hides the existence of data structures, operations and hardware from higher-level layer.

Q.32 What is graceful degradation ?

Ans. : In multiprocessor systems, failure of one processor will not halt the system, but only slow it down. If there are 15 processors and if one fails the remaining 14 processors pick up the work of the failed processor. This ability to continue providing service is proportional to the surviving hardware is called graceful degradation.

Q.33 What is a bootstrap program ?

Ans. : A bootstrap is a small initialization program to start up the computer.

Q.34 Do time-sharing differs from multiprogramming ? If so, How ?

AU : CSE/IT : May-15

Ans. : Time sharing is the sharing of resources among several processes at the same time. Multiprogramming is the allocation of more than one process on a computer system and its resources. Time sharing minimizes the response time and multiprogramming maximizes the processor use. Time sharing systems use the concept of multiprogramming to share the CPU time between multiple users at the same time.

Q.35 Why APIs need to be used rather than system calls ? **AU : CSE/IT : May-15**

Ans. : An application programmer designing a program using an API can expect his program to compile and run on any system that supports the same API. Actual system calls can often be more detailed and difficult to work with the API available to an application programmer.

Q.36 Compare and contrast DMA and Cache memory. **AU : Dec.-15**

Ans. :

- DMA technology provides special channels for CPU and I/O devices to exchange I/O data, and the memory is used for buffering the I/O data. When the CPU wants to handle I/O data, it triggers the DMA write operations that transfer the I/O data from I/O devices to the memory.

- Caches are "automatically" managed in that the hardware, when the requested memory contents are not in the cache, fetches that data from main memory.

Q.37 Write the difference between batch system and time sharing systems.

AU : Dec.-15

Ans. :

- A batch system executes jobs, whereas a time-shared system has user programs, or tasks.
- Batch systems are inconvenient for users because users cannot ir teract with their jobs to fix problems. User interacts with system in time sharing system.

Q.38 What are the advantages of peer-to-peer systems over client server systems?

AU : May-16

Ans. : Peer-to-peer system is more reliable as central dependency is eliminated. All the resources and contents are shared by all the peers, unlike server-client architecture where Server shares all the contents and resources.

Q.39 How does an interrupt differ from trap ? AU : [EIE] : Dec.-16, May-18

Ans. : Trap is a software-generated interrupt caused either by an error or by a specific request from a user program that an operating-system service be performed. A trap usually results in a switch to kernel mode. Interrupt signals can cause a program to suspend itself temporarily to service the interrupt. An interrupt is a hardware-generated signal that changes the flow within the system.

Q.40 What are the disadvantages of multiprocessor systems ? AU : [EIE] : Dec.-16

Ans. : Multiprocessor systems are more complex in both hardware and software. Additional CPU cycles are required to manage the cooperation, so per-CPU efficiency goes down.

Q.41 What are the objective of operating systems ? AU : [CSE] : May-17, Dec.-17

Ans. : The objective of operating systems are efficient use, user convenience, ability to evolve.

Q.42 What is SYSGEN and system boot ? AU : Dec.-17

Ans. : The system must then be configured or generated for each specific computer site, a process sometimes known as system generation SYSGEN. The procedure of starting a computer by loading the kernel is known as booting the system.

Q.43 List the advantages and disadvantages of writing an operating system in high level languages such as C. AU : May-18

Ans. : Advantages :

- The code can be written faster and more compact.
- Easier to understand and debug
- Recompilation is simple

Disadvantages :

- There could be a performance overhead introduced by the compiler and runtime system used for the high-level language.

Certain operations and instructions that are available at the machine-level might not be accessible from the language level, thereby limiting some of the functionality available to the programmer.

❑❑❑

UNIT - II

2 Process Management

Syllabus

Processes - Process Concept, Process Scheduling, Operations on Processes, Inter-process Communication; CPU Scheduling - Scheduling criteria, Scheduling algorithms, Multiple-processor scheduling, Real time scheduling; Threads- Overview, Multithreading models, Threading issues; Process Synchronization - The critical-section problem, Synchronization hardware, Mutex locks, Semaphores, Classic problems of synchronization, Critical regions, Monitors;

Contents

2.1 Process Concept

AU : Dec.-17

- Process term is used in MULTICS system in the 1960. Process is an asynchronou activity.
- Process is an active entity that requires a set of resources, including a processo program counter, registers to perform its function. Multiple processes may b associated with one program.
- Process means a program in execution. Process execution must progress i sequential order. It also called task. Task is a single instance of an executabl program. Fig. 2.1.1 shows memory layout for a process.
- Each process has its own address space. Address space is divided into two regions : 1. Text region 2. Data region 3. Stack region
- **Text region :** It stores the code that that the processor executes.
- **Data region :** It stores variables and dynamically allocated memory that th process uses during execution.
- **Stack region :** It stores instructions and local variables for active procedure calls.
- Process is a dynamic entity that executes a program on a particular set of data using resources allocated by the operating system.
- A process can run to completion only when all requested hardware and software resources have been allocated to the process.
- Program is a passive entity. Program becomes process when executable file loaded into memory. A process may be independent of other processes in the system.

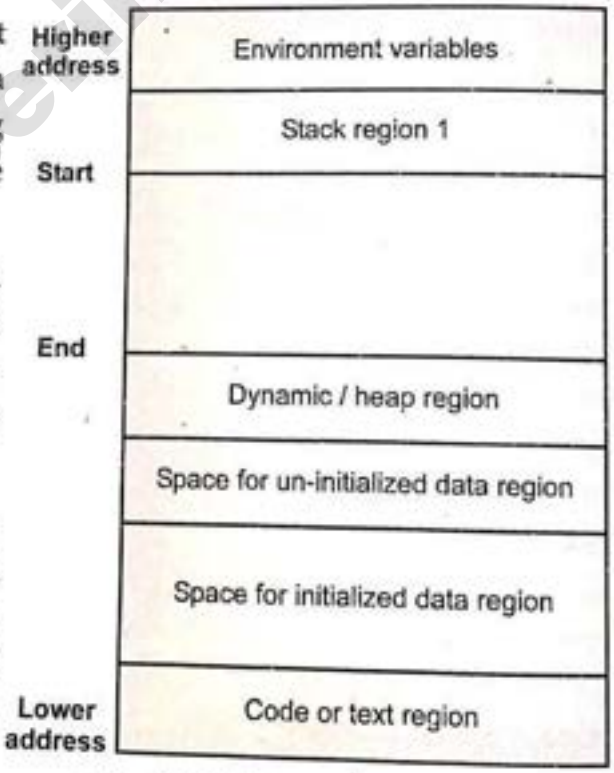


Fig. 2.1.1 Memory layout for a process

- After booting operating system, it creates foreground processes and background processes.
- Foreground processes interact with user and background processes is used b system. Background processes are related e-mail, web pages, news and printing.

- A process is used as a fundamental unit for resource allocation in operating system. A process normally has its own private memory area in which it runs.
- Text region contains executable instructions. It is be placed below the heap or stack.
- An initialized data region contains the static and global variables that are initialized by the user.
- Un-initialized data region contains all static and global variables that are initialized by kernel to zero.
- Heap is used for dynamic memory allocation and stack contains program counter.

2.1.1 Difference between Process and Program

Sr. No.	Process	Program
1.	Process is active entity.	Program is passive entity.
2.	Process is a sequence of instruction executions.	Program contains the instructions.
3.	Process exists in a limited span of time.	A program exists at single place and continues to exist.
4.	Process is a dynamic entity.	Program is a static entity.

2.1.2 Process Control Block

- Operating system keeps an internal data structure to describe each process it manages.
- When OS creates process, it creates this process descriptor. In some operating system, it calls Process Control Block (PCB).
- Fig. 2.1.2 shows process control block.
- Process control block will change according to the operating system. PCB is also called **task control block.**
- The PCB is identified by an integer Process ID (PID). When a process is running, its hardware state is inside the CPU.

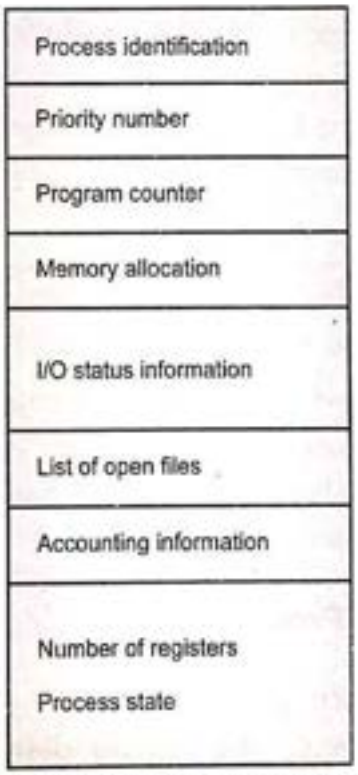


Fig. 2.1.2 Process control block

- When the OS stops running a process, it saves the register's values in the PCB.
- When a process is created by operating system, it allocates a PCB for it. OS initializes PCB and puts PCB on the correct queue. Following information is stored in process control block.

1. **Process identification** : Each process is uniquely identified by the user's identification and a pointer connecting it to its descriptor.
2. **Priority number** : Operating system allocates the priority number to each process. According to the priority number it allocates the resources.
3. **Program counter** : The PC indicates the address of the next instruction to be executed for this current process.
4. **Memory allocation** : It contains the value of the base registers, limit registers and the page tables depending on the memory system used by the operating system.
5. **I/O status information** : It maintains information about the open files, list of I/O devices allocated to the process etc.
6. **List of open files** : Process uses number of files for operation. Operating system keeps track of all opened file by this process.
7. **Process state** : Process may be in any one of the state : new, ready, running, and waiting, terminate.

- When process changes the state, the operating system must update information in the process's process control block. Process control block maintain other information which is not included in PCB block diagram. This information includes CPU scheduling, file management, input-output management information.
- Fig. 2.1.3 shows process table with PCB.
- When process is created, hardware registers and flags values are set by loader or linker.
- Operating system maintains pointers to each process's PCB in a per user process table or system wide process table. This information is used to access PCB quickly.

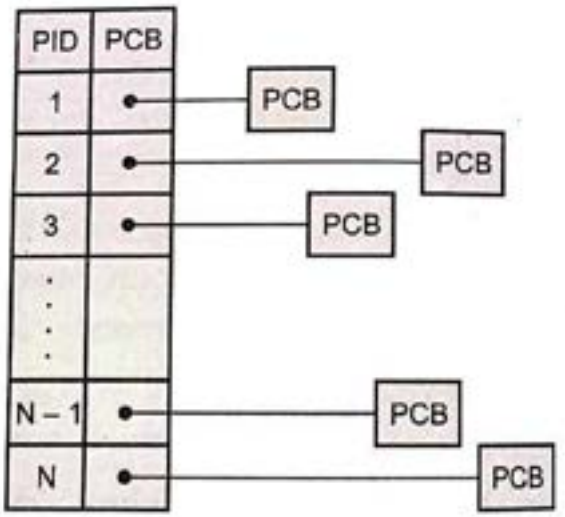


Fig. 2.1.3 Process table with PCB

2.1.3 Process States

- Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process.

- Fig. 2.1.4 shows a process state diagram. A state diagram is composed of a set of states and transitions between states.

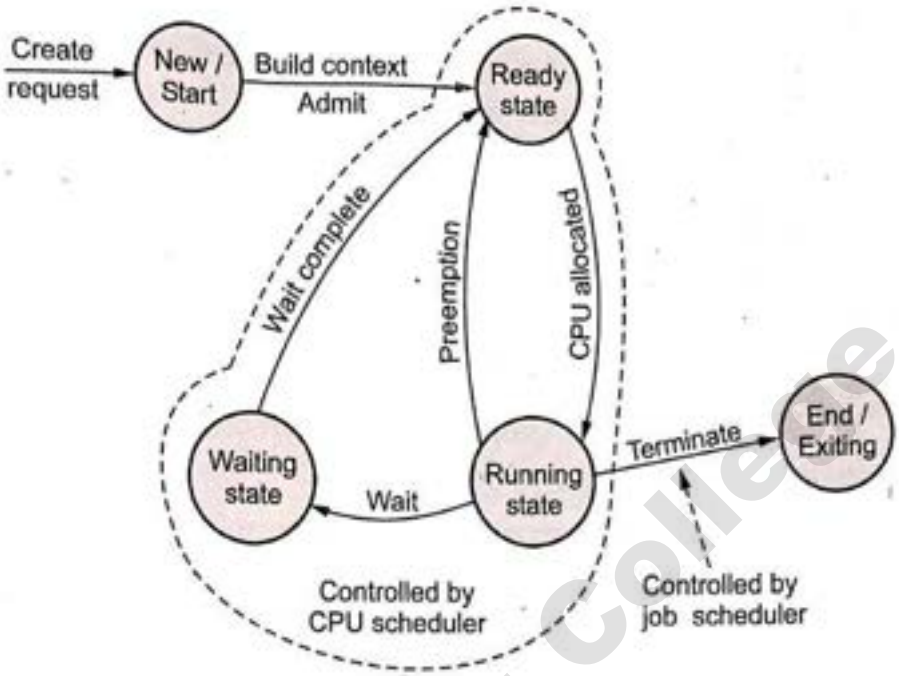


Fig. 2.1.4 Process state diagram

- State diagram is used by process manager to determine the type of service to provide to the process.
- The process states are as follows :
  1. New 2. Ready 3. Running 4. Blocked/Waiting 5. Exit/End
- **New :** Operating system creates new process by using fork( ) system call. These process are newly created process and resources are not allocated.
- **Ready :** The process is competing for the CPU. Process reaches to the head of the list (queue).
- **Running :** The process that is currently being executed. Operating system allocates all the hardware and software resources to the process for execution.
- **Blocked/Waiting :** A process is waiting until some event occurs such as the completion of an input-output operation.
- **Exit/End :** A process completes its operations and releases all resources.
- Operating system maintains a ready list of ready process and a blocked list of blocked processes. The ready list is maintained in priority order and blocked list is typically unordered.
- The act of assigning a processor to the first process on the ready list is called **dispatching** and is performed by a system entity called the **dispatcher.**
- When process is in new state, the program remains in the secondary storage. Here process itself is not in the main memory and space is not allocated.
- The process exists a system because of two reasons :
  1. Process is terminated when it completes its operation.
  2. Process aborts due to an unrecoverable error.

- To prevent any one process from continuously using the system, the operating system sets a hardware interrupting clock to allow a process to run for a specific time interval or time quantum.
- The process may request a resource when it is in the running state. In most of the operating system, if a running process requests an immediately available resources the process is allowed to continue in the running state.
- From the blocked state, the process can move to the ready state only by being allocated the requested resource.
- User only initiate process state transition is **blocked** and remaining all other state transitions is initiated by the operating system.

Sr. No.	State transition	Remarks
1.	Ready to running	Process is dispatched.
2.	Running to ready	Process time slice expires.
3.	Running to blocked	When a process blocks.
4.	Blocked to ready	When the event for which it has been waiting occurs.
5.	Ready to exit	This is possible when parent process may terminate child process at any time.
6.	Running to exit	When currently running process completes its operation then OS terminates the process.

2.1.4 Suspended Processes

- Each process to be executed must be loaded into main memory. In uniprocessor system, processor remains idle because of execution speed mismatch of processor and I/O devices. An I/O device is slower than processor.
- Main memory accommodates multiple processes and the processor select next process when one process is blocked. So even with multiprogramming operating system, a processor could be idle most of the time. Main memory size is increased for accommodating more processes but cost will increase.
- **Suspended state :** when all of the processes in main memory are in the blocked state, the operating system can suspend one process by putting it in the suspend state and transferring it to disk. The space that is freed in main memory can then be used to bring in another process.
- Characteristics of a suspended process :
  1. The process cannot execute immediately.

2. The process may or may not be waiting for an event.
3. The process was placed in a suspended state by a parent process, or itself or by operating system.
4. The process may not be removed from suspended state until its turn comes.

University Question

1. *What is a process ? Discuss components of process and various states of a process with the help of a process state transition diagram.* **AU : Dec.-17, Marks 8**

2.2 Process Scheduling

AU : May-16, 18, Dec.-16

- CPU utilization is maximizing by using multiprogramming concept. Processor is not idle, it is executing a process. Processor scheduler selects one process for execution from the ready queue.

Scheduling Queue

- Scheduling queue is queue of processes or input-output devices. When the user process enters into the system, they put into the job queue. Job queue consists of all processes of the system.
- Operating system maintains different types of queues for different purposes. The queue are ready queue, device queue etc. Fig. 2.2.1 shows ready queue and device queue.

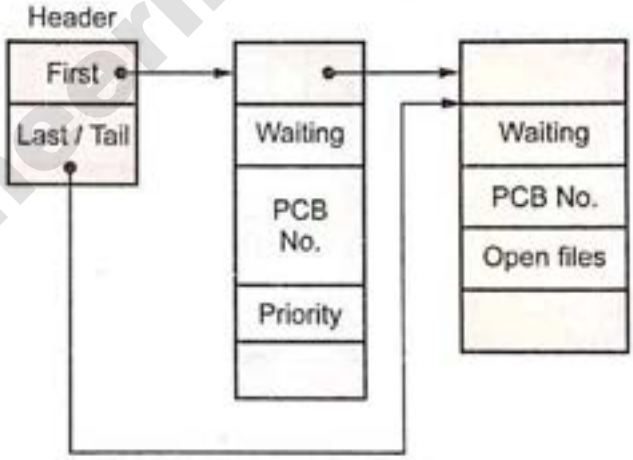


Fig. 2.2.1 Ready and device queue

- **Ready queue :** The processes which are ready and waiting to execute are kept in the ready queue. Ready queue is stored in main memory. Linked list is used for representing ready queue. Pointer field of PCB is used for this.
- **Device queue :** Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has maintained its own device queue.
- Process scheduling is represented by queueing diagram. Newly created process is kept into the ready queue and waits for processor. When CPU select the process for executing, following things happens :

1. Process may require I/O device to perform operation. So it is put into the I/O queue.
2. Process may create child process and wait for child process termination.
3. Because of interrupt, process preempted from CPU and put into the ready queue.

2.2.1 Schedulers

- Schedulers are used to handles process scheduling. It is one type of system software and selects the jobs from the system and decide which process to run. Schedulers are of three types -
  1. Long term scheduler 2. Short term scheduler 3. Medium term scheduler
- Fig. 2.2.2 shows queueing diagram for process scheduling.

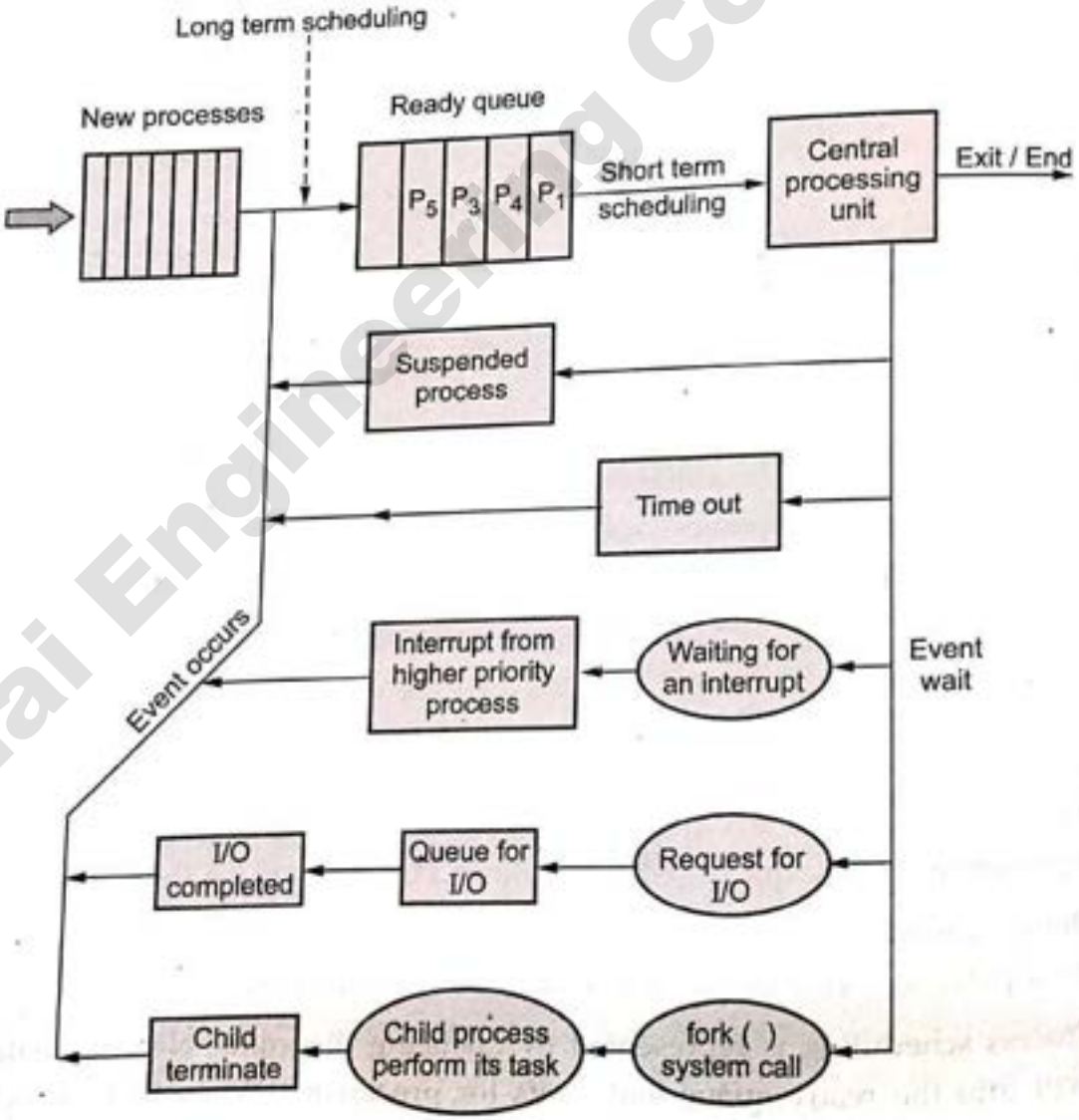


Fig. 2.2.2 Process scheduling queueing diagram

Long term scheduler

- Long term scheduler is also called job scheduler. It determines which process are admitted to the system for processing. Processes are selected from the queue and loads into the main memory for execution.
- Long term scheduling controls the degree of multiprogramming in multitasking systems. It provides a balanced mix of jobs, such as I/O bound and CPU bound.
- Long term scheduling is used in real time operating system. Time sharing operating system has no long term scheduler.

Medium term scheduler

- Medium term scheduler is part of swapping function. Sometimes it removes the process from memory. It also reduces the degree of multiprogramming.
- If process makes an I/O request and it is in memory then operating system takes this process into suspended states. Once the process becomes suspended, it cannot make any progress towards completion.
- In this situation, the process is removed from memory and makes free space for other process.
- The suspended process is stored in the secondary storage device i.e. hard disk. This process is called swapping.

Short term scheduler

- Short term scheduler is also called CPU scheduler. It selects the process from queue which are ready to execute and allocate the CPU for execution.
- Short term scheduler is faster that long term scheduler. This scheduler makes scheduling decisions much more frequently than the long-term or mid-term schedulers.
- A scheduling decision will at a minimum have to be made after every time slice, and these are very short.
- It is also known as dispatcher.

2.2.2 Difference between Long Term, Short Term and Medium Term Scheduler

Sr. No.	Long term	Short term	Medium term
1.	It is job scheduler.	It is CPU scheduler.	It is swapping.
2.	Speed is less than short term scheduler.	Speed is very fast.	Speed is in between both.
3.	It controls the degree of multiprogramming.	Less control over degree of multiprogramming.	Reduce the degree of multiprogramming.

4.	Absent or minimal in time sharing system.	Minimal in time sharing system.	Time sharing system use medium term scheduler.
5.	It select processes from pool and load them into memory for execution.	It select from among the processes that are ready to execute.	Process can be reintroduced into memory and its execution can be continued.
6.	Process state is (New to Ready).	Process state is (Ready to Running)	–
7.	Select a good process, mix of I/O bound and CPU bound.	Select a new process for a CPU quite frequently.	–

2.2.3 Context Switch

- A context switch is the switching of the CPU from one process or thread to another. A context is the contents of a CPU's registers and program counter at any point in time.
- Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a **context switch**.
- A context switch can mean a register context switch, a task context switch, a thread context switch or a process context switch.
- A register is a small amount of very fast memory inside of a CPU that is used to speed the execution of computer programs by providing quick access to commonly used values.
- A program counter is a specialized register that indicates the position of the CPU in its instruction sequence and which holds either the address of the instruction being executed or the address of the next instruction to be executed, depending on the specific system.
- Context switching can be described in more detail as the Kernel performing the following activities with regard to processes (including threads) on the CPU :
  1. Suspending the progression of one process and storing the CPU's state (i.e. the context) for that process somewhere in memory.
  2. Retrieving the context of the next process from memory and restoring it in the CPU's registers and
  3. Returning to the location indicated by the program counter in order to resume the process.
- Context switches can occur only in Kernel mode (system mode). Kernel mode is a privileged mode of the CPU in which only the Kernel runs and which provides access to all memory locations and all other system resources.

- Other programs, including applications, initially operate in user mode, but they can run portions of the Kernel code via system calls. Software context switching can be used on all CPUs and can be used to save and reload only the state that needs to be changed.
- To use the hardware context switch you need to tell the CPU where to save the existing CPU state and where to load the new CPU state from. The CPU state is always stored in a special data structure called a TSS (Task State Segment).
- Context switch times are highly dependent on hardware support. Context switching represents a substantial cost to the system in terms of CPU time and it can be the most costly operation on an operating system.
- There are three situations where a context switch needs to occur. They are multitasking, interrupt handling, user and Kernel mode switching.

University Questions

1. *Describe the actions taken by a kernel to context-switch between processes*
   AU : May-16, Marks 8
2. *Describe the differences among short-term, medium-term, and long term scheduling*
   AU : Dec.-16, Marks 8
3. *Describe the difference among short-term, medium-term and long-term scheduling with suitable example.*
   AU : May-18, Marks 13

2.3 Operations on Processes

- Following operations are performed on the process :
  1. Process creation 2. Process termination
- A mechanism for process creation and termination via system calls. Programmers usually access these system calls via application interface (API) / library.

2.3.1 Process Creation

- Operating system creates the process in following situations :
  1. Starting of new batch job. 2. User request for creating new process.
  3. To provide new services by OS. 4. System call from currently running process.
- Operating system creates a new process with the specified or default attributes and identifier. A process may create several new sub-process.
- Parent process is creating process and the new processes are called the children of the process. When operating system creates process, it builds the data structure for managing process and allocates address space in primary main memory.

- Operating system creates foreground and background process. Process is identified by unique process identifier (PID) in UNIX and windows operating system. PID value is an integer number.
- All processes in UNIX are created using the fork( ) system call. The forking process is called the parent process. The new process is called the child process.
- Both the parent and child process have their own and private memory. Open files are shared between parent and child.
- If the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork( ) calls will not be affected even though they have identical variable names.
- When a process is created, OS assigns some attributes. These are priority, privilege level, requirement of memory, access right, memory protection, PID etc. To perform operation, process needs software and hardware resources. It includes CPU time, files, memory, I/O device.
- Relation between parent process and child process is as follows :
  1. Parent process continues to execute concurrently with its child process.
  2. Parent process waits until some or all of its children have terminated.

```
void main( )
    {
        printf("Operating System\n");
        fork( );
            printf("Technical Publications\n");
      return 0;
}
```

- In above program Operating System is printed only once and Technical Publications is printed two times.

2.3.2 Process Termination

- When process finishes its normal execution then that process is terminate. Operating system delete that process using exit ( ) system call. After deleting process, memory space becomes free.
- OS passes the child's exit status to the parent process and then discards the process. At the same time, it de-allocate all the resources hold by this process.
- Following are the various reasons for process termination :
  1. Normal completion of operation
  2. Memory is not available
  3. Time slice expired
  4. Parent termination
  5. Failure of I/O
  6. Request from parent process
  7. Misuse of access rights

2.4 Interprocess Communication

AU : Dec.-16

- Exchange of data between two or more separate, independent processes/threads is possible using IPC. Operating systems provide facilities/resources for Inter-Process Communications (IPC), such as message queues, semaphores, and shared memory.
- A complex programming environment often uses multiple cooperating processes to perform related operations. These processes must communicate with each other and share resources and information. The Kernel must provide mechanisms that make this possible. These mechanisms are collectively referred to as **interprocess communication.**
- Distributed computing systems make use of these facilities/resources to provide Application Programming Interface (API) which allows IPC to be programmed at a higher level of abstraction. (e.g., send and receive).
- Five types of inter-process communication are as follows :
  1. Shared memory permits processes to communicate by simply reading and writing to a specified memory location.
  2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
  3. Pipes permit sequential communication from one process to a related process.
  4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.
  5. Sockets support communication between unrelated processes even on different computers.

Name	Description	Scope	Use
File	• Data is written to and read from a typical UNIX file. • Any number of processes can interoperate.	Local	Sharing large data sets.
Pipe	• Data is transferred between two processes using dedicated file descriptors. • Communication occurs only between a parent and child process.	Local	Simple data sharing, such as producer and consumer.
Named pipe	• Data is exchanged between processes via dedicated file descriptors. • Communication can occur between any two peer processes on the same host.	Local	Producer and consumer, or command-and-control, as demonstrated with MySQL server and its command-line query utility.

Signal	• An interrupt alerts the application to a specific condition.	Local	Cannot transfer data in a signal, so mostly useful for process management.
Shared memory	• Information is shared by reading and writing from a common segment of memory.	Local	Cooperative work of any kind, especially if security is required.
Socket	• After special setup, data is transferred using common input/output operations.	Local or remote	Network services such as FTP, ssh and the Apache web server.

Table 2.4.1 Interprocess communication in UNIX

- **Purposes of IPC**
  1. **Data transfer :** One process may wish to send data to another process.
  2. **Sharing data :** Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it.
  3. **Event modification :** A process may wish to notify another process or set of processes that some event has occurred.
  4. **Resource sharing :** The Kernel provides default semantics for resource allocation; they are not suitable for all application.
  5. **Process control :** A process such as a debugger may wish to assume complete control over the execution of another process.

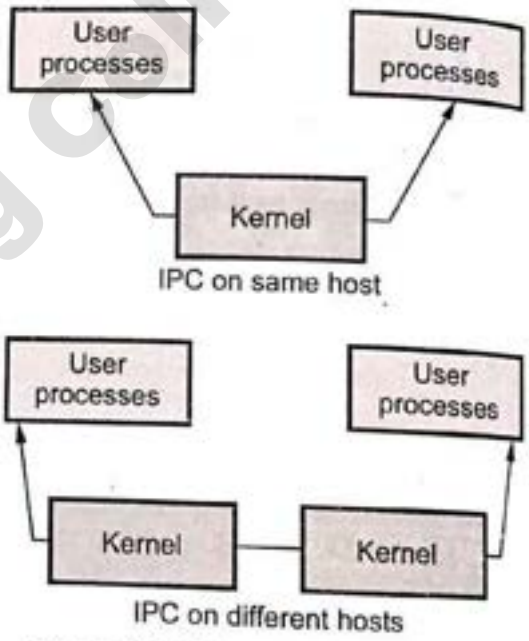


Fig. 2.4.1 IPC on different hosts

- IPC has two forms : IPC on same host and IPC on different hosts

IPC is used for 2 functions :

1. **Synchronization :** Used to coordinate access to resources among processes and also to coordinate the execution of these processes. They are record locking, semaphores, mutexes and condition variables.
2. **Message passing :** Used when processes wish to exchange information. Message passing takes several forms such as : Pipes, FIFOs, message queues and shared memory.

2.4.1 Pipes

- A pipe is a unidirectional, first-in first-out, unstructured data stream of fixed maximum size. Writers add data to the end of the pipe; readers retrieve data from the front of the pipe.
- Once read, the data is removed from the pipe and is unavailable to other readers. A pipe provides a simple flow control mechanism.
- A process attempting to read from empty pipe blocks until more data is written to the pipe. A process trying to write to a full pipe lock until another process reads data from pipe.
- The pipe system call creates a pipe and returns two file descriptors: one for reading and one for writing. These descriptors are inherited by child processes, which thus share access to the file.
- Each pipe can have several readers and writers. A given process may be a reader or writer or both. Fig. 2.4.2 shows data flow through a pipe.
- Pipes can be used only between processes that have a common ancestor. Normally, a pipe is created by a process, that process calls fork and the pipe is used between the parent and the child.
- Example to show how to create and use a pipe :

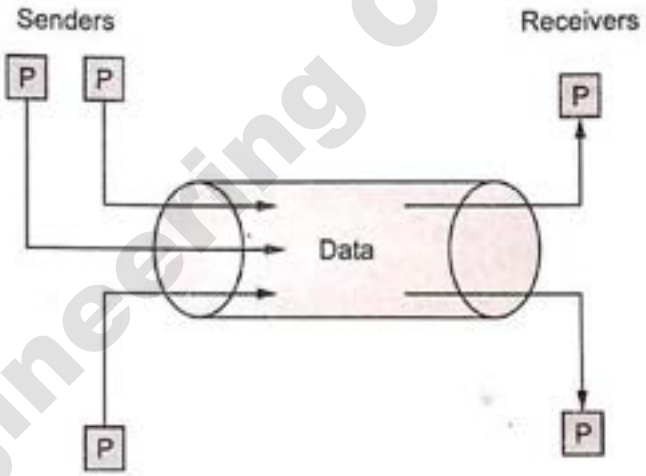


Fig. 2.4.2 Data flow through a pipe

```
main( )
{
   int pipefd[2], n;
   char buff[100];

   if (pipe(pipefd) < 0 )
  err_sys("pipe error");
  printf("read fd = %d, write fd = %d\n", pipefd[0], pipefd[1]);

 if (write(pipefd[1], "hello world\n", 12) != 12)
  err_sys("write error");
if ( (n=read(pipefd[0], buff, sizeof(buff))) < =0)
 err_sys("read error");
 write (1, buff, n); /*fd=1=stdout*/
 }
```

Properties of pipe :

1. Pipes do not have a name. For this reason, the processes must share a parent process. This is the main drawback to pipes. However, pipes are treated as file descriptors, so the pipes remain open even after fork and exec.
2. Pipes do not distinguish between messages; they just read a fixed number of bytes.
3. Pipes can also be used to get the output of a command or to provide input to a command

- The most common use of pipes is to let the output of one program become the input for another. Users typically join two programs by a pipe using the shell's pipe operator (|).

Limitations of pipes :

1. Reading data removes it from the pipe, a pipe cannot be used to broadcast data to multiple receivers.
2. Data in a pipe is treated as a byte stream and has no knowledge of message boundaries.
3. If there are multiple readers on a pipe, a writer cannot direct data to a specific reader.

- **Program for sending data from parent process to child process over a pipe**

```
#include <stdio.h>
int main(void)
{
int n;
int fd[2];
pid_t pid;
char line[MAXLINE];

if (pipe(fd) < 0)
err_sys("pipe error");

if ((pid = fork( )) < 0)
{
  err_sys("fork error");

} else if (pid > 0)
 { /* parent */
  close(fd[0]);
 write(fd[1], "hello world\n", 12);

 } else { /* child */
  close(fd[1]);
```

```
        n = read(fd[0], line, MAXLINE);
        write(STDOUT_FILENO, line, n);

        }
        exit(0);
        }
```

- Pipe is created by calling the pipe function.

```
#include < unistd.h>
int pipe ( int filedes[2]);
```

- Two file descriptor are returned through the *filedes* argument : *filedes[0]* is open for reading and *filedes[1]* is open for writing. The output of *filedes[1]* is the input for *filedes[0]*.
- Fig. 2.4.3 shows two ways to view the UNIX pipe.
- The *fstat* function returns a file type of FIFO for the file descriptor of either end of a pipe. A pipe in a single process is not useful.
- Normally, the process that calls pipe then calls fork, creating an IPC channel from the parent to the child or vice versa.

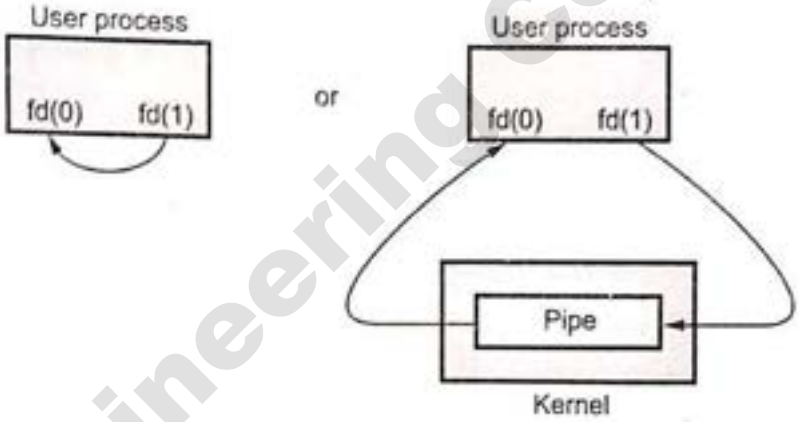


Fig. 2.4.3 Two ways to view the UNIX pipe

- Fig. 2.4.4 shows half-duplex pipe after a fork.

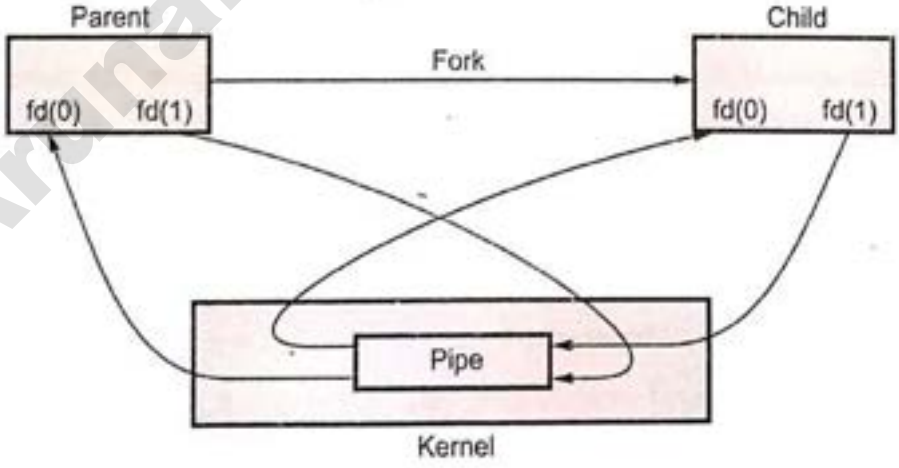


Fig. 2.4.4 Half-duplex pipe after a fork

- Fig. 2.4.5 shows pipe from parent to child. For a pipe from the parent to the child, the parent closes the read end of the pipe (fd[0]), and the child closes the write end (fd[1]).

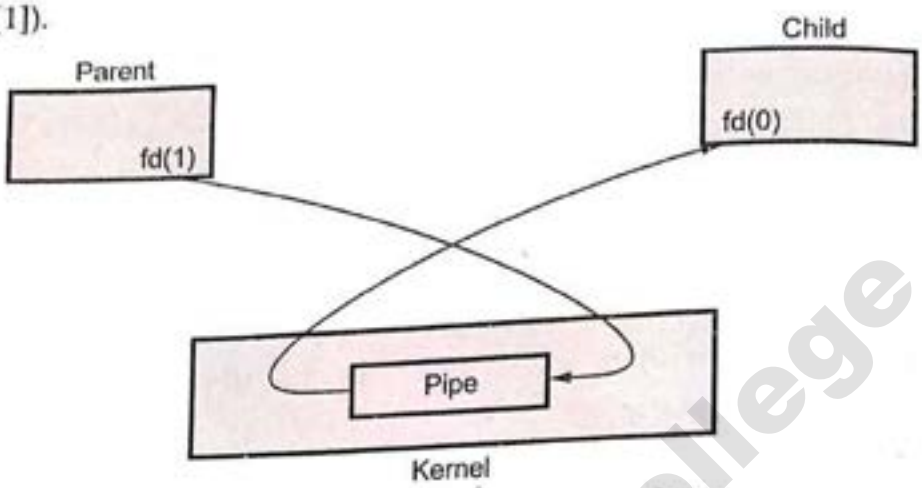


Fig. 2.4.5 Pipe from parent to child

- For a pipe from the child to the parent, the parent closes fd[1], and the child closes fd[0]. When one end of a pipe is closed, the following two rules apply.
  1. If we read from a pipe whose write end has been closed, read returns 0 to indicate an end of file after all the data has been read.
  2. If we write to a pipe whose read end has been closed, the signal SIGPIPE is generated. If we either ignore the signal or catch it and return from the signal handler, write returns 1 with errno set to EPIPE.

2.4.2 Features of Message Passing

1. **Simplicity :** Message passing system should be simple and easy to use. It should be possible to communicate with old and new applications.
2. **Uniform semantics :** Message passing is used for two types of IPC.
   a. **Local communication :** Communicating processes are on the same node.
   b. **Remote communication :** Communicating processes are on the different nodes.
3. **Efficiency :** IPC become so expansive if message passing system is not effective. Users try avoiding to IPC for their applications. Message passing system will become more efficient if we try to avoid more message exchanges during communication process. For examples :
   a. Avoiding the costs of establishing and terminating connection.
   b. Minimizing the costs of maintaining the connections.
   c. Piggybacking of acknowledgement.

4. **Reliability :** Distributed systems are prone to different catastrophic events such as node crashes or physical link failures. Loss of message because of communication link fails. To handle the loss messages, we required acknowledgement and retransmission policy. Duplicate message is one of the major problems. This happens because of timeouts or events of failures.
5. **Correctness :** Correctness is a feature related to IPC protocols for group communication. Issues related to correctness are as follows :
   i. **Atomicity :** Every message sent to a group of receivers will be delivered to either all of them or none of them.
   ii. **Ordered delivery :** Messages arrive to all receivers in an order acceptable to the application.
   iii. **Survivability :** Messages will be correctly delivered despite partial failures of processes, machines, or communication links.
6. **Security :** Message passing system must provide a secure end to end communication.
7. **Portability :** Message passing system should itself be portable.

2.4.3 IPC Message Format

- Message passing system requires the synchronization and communication between the two processes. Message passing used as a method of communication in microkernels. Message passing systems come in many forms. Messages sent by a process can be either fixed or variable size. The actual function of message passing is normally provided in the form of a pair of primitives.
  a) Send (destination_name, message)
  b) Receive (source_name, message).
- Send primitive is used for sending a message to destination. Process sends information in the form of a message to another process designated by a destination. A process receives information by executing the receive primitive, which indicates the source of the sending process and the message.
- Design characteristics of message system for IPC.
  1. Synchronization between the process
  2. Addressing
  3. Format of the message
  4. Queueing discipline.

Issues in IPC by Message Passing

- Message is a block of information.
- A message is a meaningful formatted block of information sent by the sender process to the receiver process.

- The message block consists of a fixed length header followed by a variable size collection of typed data objects.
- The header block of a message may have the following elements :
  1. **Address :** A set of characters that uniquely identify both the sender and receiver.
  2. **Sequence number :** It is the message identifier to identify duplicate and lost messages in case of system failures.
  3. **Structural information :** It has two parts. The type part that specifies whether the data to be sent to the receiver is included within the message or the message only contains a pointer to the data. The second part specifies length of the variable-size message.
- Fig. 2.4.6 shows the typical message format.

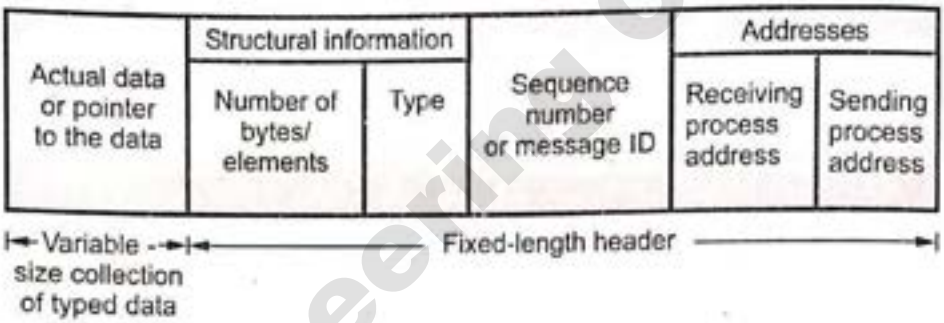


Fig. 2.4.6 Message structure

- Some important issues to be considered for the design of an IPC protocol based message passing system :
  i. The sender's identity
  ii. The receiver's identity
  iii. Number of receivers
  iv. Guaranteed acceptance of sent messages by the receiver
  v. Acknowledgment by the sender
  vi. Handling system crashes or link failures
  vii. Handling of buffers
  viii. Order of delivery of messages

2.4.4 IPC Synchronization

- Send operation can be synchronous or asynchronous. Receive operation can be blocking or nonblocking.
- Sender and receiver process can be blocking mode or nonblocking mode. Different possibility of sender and receivers are as follows:
  1. Blocking send, blocking receive
  2. Nonblocking send, blocking receive
  3. Nonblocking send, Nonblocking receive

Blocking send, blocking receive

- Blocking send must wait for the receiver to receive the message. Synchronous communication is an example of blocking send. Both processes (sender and receiver) are blocked until the message is delivered.
- **Rendezvous** : sending a message to another process leaving the sender process suspended until the message is received and processed.

Nonblocking send, blocking receive

- Sender is free to send the messages but receiver is blocked until the requested message arrives.
- Asynchronous communication is an example of nonblocking send. Asynchronous communication with nonblocking sends increases throughput by reducing the time that processes spend waiting.
- Natural concurrent programming task uses the nonblocking send. Nonblocking send is the overhead on the programmer for determine that a message has been received or not.

2.4.5 Shared Memory

- A region of memory that is shared by co-operating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer. Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of Kernel intervention.
- In contrast, in shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the Kernel is required.

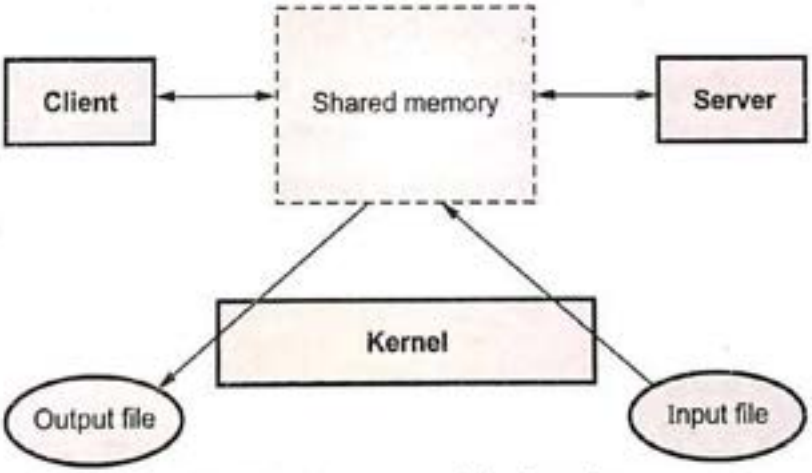


Fig. 2.4.7 Client / server with shared memory

- Fig. 2.4.7 shows client/server with shared memory.

Advantages

1. Good for sharing large amount of data. 2. Very fast.

Limitations

1. No synchronization provided - applications must create their own.
2. Alternative to shared memory is *mmap* system call, which maps file into the address space of the caller.

University Question

1. *Give an example of a situation in which ordinary pipes are more suitable than named pipes and an example of a situation in which named pipes are more suitable than ordinary pipes.* **AU : Dec.-16, Marks 8**

2.5 CPU Scheduling

- In a multiprogramming environment, usually more programs to be executed than could possibly be run time at one time. In CPU scheduling it switches from one process to another process. CPU resource management is commonly known as scheduling.
- Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundamental operating system functions.
- User program contains combination of CPU burst cycle and I/O burst cycles. Process starts with CPU burst cycle then I/O burst cycle again CPU burst cycle again I/O burst cycle. In this way the process completes its executions. Process will terminates after final CPU burst cycle completions.

  Program execution sequence : *CPU burst cycle → I/O burst cycle → CPU burst cycle → I/O burst cycle → CPU burst cycle → Program terminates.*
- A CPU bound process tends to use the processor time that the system allocates for it. An I/O bound process tends to use the processor only briefly before generating an I/O request. The CPU bound processes spend most of their time using the processor.

CPU Scheduler

- CPU scheduler is also called as short scheduler. System programmers use parameters like program size, resources required for program and any other special devices are needed to determine scheduling policies.

- Scheduling mechanism is the part of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- Scheduler is responsible for multiplexing processes on the CPU.

2.5.1 Preemptive and Non-preemptive Scheduling

- The scheduling policy determines when it is time for process to be removed from the CPU and which ready process should be allocated the CPU next. The scheduling mechanism is composed of several different parts, depending on exactly how it is implemented in any particular operating system.
- CPU scheduling is divided into two types : Preemptive scheduling and non-preemptive scheduling.
- **Preemptive scheduling :** A scheduling method that interrupts the processing of a process and transfers the CPU to another process is called a preemptive CPU scheduling. The process switches from running state to the ready state and waiting state to the ready state.
- **Non-preemptive scheduling :** Non-preemptive operation usually proceeds towards completion uninterrupted. Once the system has assigned a processor to a process, the system cannot remove that processor from the process. The process switches from running state to the waiting state and termination of process.
- Preemptive scheduling increases the cost and it has higher overheads. This scheduling method is useful only in the high priority processes require rapid response.
- Non-preemptive scheduling is simple to implement. It requires some hardware platform. This method is attractive because of its simplicity. Windows 3.1 and Apple Macintosh operating system uses this scheduling method.

2.5.2 Difference between Preemptive and Non-preemptive Scheduling

Sr. No.	Preemptive scheduling	Non-preemptive scheduling
1.	Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process.	Non-preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.
2.	Preemptive scheduling incurs a cost associated with access shared data.	Non-preemptive scheduling does not increase the cost.
3.	It also affects the design of the operating system Kernel.	It does not affects the design of OS Kernel.

4.	Preemptive scheduling is more complex.	Simple, but very inefficient.
5.	Example : Round robin method.	Example : First come first serve method.

2.5.3 CPU Scheduling Criteria

- Depending on the system, CPU scheduling criteria will change.
  1. **Throughput :** CPU scheduling should attempt to service the maximum number of processes per unit time. The higher is the number, the more work is done by the system.
  2. **Waiting time :** The average period of time a process spends waiting. Process is normally in the ready queue in waiting time.
  3. **Turnaround time :** Turnaround time start from process submission to completion of process.

     Turnaround time = Burst time + Waiting time
  4. **Response time :** It is the time from the submission of a request until the first response is produced.
  5. **CPU utilization :** It is average function of time during which the processor is busy.
  6. **Fairness :** Avoid the process from the starvation. All the processes must be given equal opportunity to execute.
  7. **Priority :** If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher priority processes.
  8. **Predictability :** A given process always should run in about the same amount of time under similar system loads.
- Depending upon the nature of operations the scheduling policy may differ. The CPU utilization and throughput are the system centered parameters. Fairness is affect by user and system.

2.5.4 Dispatcher

- Using dispatcher, CPU selects the process from the short term scheduler. Functions of the dispatcher is as follows:

  1. Switching context 2. Switching to user mode

  3. Jump to proper location in the user program for restarting that program.
- Dispatcher is a small program that switches the processor from one process to another.

- **Dispatch Latency :** It is time taken by dispatcher to stop running one process and start other process running is called as dispatch latency.

2.6 Scheduling Algorithm

AU : Dec.-15, 17, May-15,17, 18

- CPU scheduling algorithms are as follows :
  1. First In First Out (FIFO) scheduling
  2. Shortest job first scheduling
  3. Priority scheduling
  4. Round robin scheduling

2.6.1 First Come First Serve Scheduling

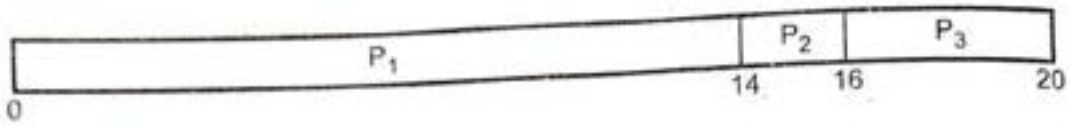
- This method of scheduling means that the first process to request the processor gets it until it finished execution. With this algorithm, processes are assigned the CPU in the order they request it.
- Normally there is a single queue of ready processes. When the first process enters the system from the outside, it is started immediately and allowed to run as long as it wants it. At the same time, other process enters into the system; they are put onto the end of the queue.
- New process enters the tail of the queue and the schedule selects the process from the head of the queue.
- When new process enters into the system, its process control block is linked to the end of the ready queue and it is removed from the front of the queue.
- When a process waits or blocks, it is removed from the queue and it queues up again in FCFS queue when it gets ready.
- FCFS is non-preemptive CPU scheduling algorithm. It is also called First In First Out method (FIFO).
- FCFS is simple to implement because it uses a FIFO queue. This algorithm is fine for most of the batch operating systems.
- FCFS is not useful in scheduling interactive processes because it cannot guarantee short response time. This algorithm performs much better for long processes than short ones.
- Turnaround time is unpredictable with the FCFS algorithm. Average waiting time of the FCFS is often quite long.
- Real life analogy is **buying tickets**.

Example 2.6.1 *Consider the following set of process that arrives at time 0, with the length of the CPU burst given in milliseconds*
Calculate the average waiting time.

Process	*Burst time*
P_1	14
P_2	2
P_3	4

Solution :

Gantt chart :



Such a diagram is called "Gantt charts", showing when each CPU burst uses the CPU. Here, each CPU burst comes from a different thread.

Waiting Time :

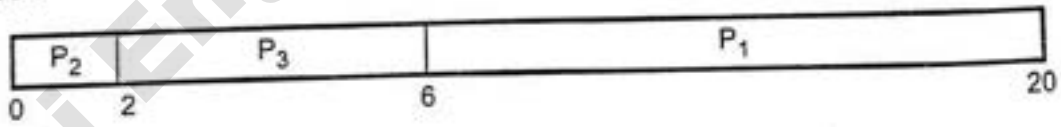
Process	Waiting time
P_1	0
P_2	14
P_3	16

$$\text{Average waiting time} = \frac{\text{Sum of } P_1, P_2 \text{ and } P_3 \text{ waiting time}}{3}$$

$$= \frac{0+14+16}{3} = 10$$

- If the processes change their order of arrive i.e. P_2, P_3, P_1, then the results will be different than first one and it is shown below:

Gantt chart :



Waiting Time :

Process	Waiting time
P_1	6
P_2	0
P_3	2

$$\text{Average waiting time} = \frac{\text{Sum of } P_1, P_2 \text{ and } P_3 \text{ waiting time}}{3}$$

$$= \frac{6+0+2}{3} = 2.66$$

Convey effect

- To reduce I/O device utilization, all I/O bound processes will be waiting excessively long for processor bound ones. This is called as **convey effect.**
- If one process monopolizes the system, the extent of its overall effect on system performance depends on the scheduling policy and whether the process is processor bound or I/O bound.

Advantages

1. Simple to implement
2. Fair

Disadvantages

1. Waiting time depends on arrival order
2. Convoy effect : short process stuck waiting for long process
3. Also called head of the line blocking

Example 2.6.2 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time when the processes arrive in the following order :*

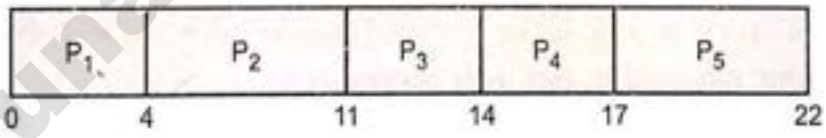
a. P_1, P_2, P_3, P_4, P_5

Provide the Gantt chart for the same.

Process	*Burst time*
P_1	4
P_2	7
P_3	3
P_4	3
P_5	5

Solution :

Gantt chart :



Such diagram is called Gantt charts, showing when each process burst uses the CPU.

Waiting time :

Process	Waiting time
P_1	0 – 0 = 0
P_2	4 – 0 = 4

P_3	11 – 0 = 11
P_4	14 – 0 = 14
P_5	17 – 0 = 17

$$\text{Average waiting time} = \frac{0+4+11+14+17}{5} = \frac{46}{5} = 9.2$$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	0 + 4 = 4
P_2ws	4 + 7 = 11
P_3	11 + 3 = 14
P_4	14 + 3 = 17
P_5	17 + 5 = 22

$$\text{Average turnaround time} = \frac{4+11+14+17+22}{5} = \frac{68}{5} = 13.60$$

2.6.2 Shortest Job First Scheduling

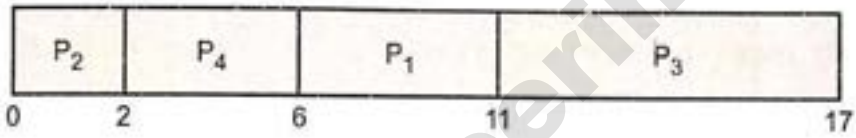
- Shortest job first scheduling algorithm is also known as Shortest Job Next (SJ scheduling algorithm. It handles the process based on length of their CPU cyc time. It reduces average waiting time over FIFO algorithm.
- SJF is a non-preemptive CPU scheduling algorithm.
- It does not work in interactive system because users do not estimate in advan the CPU time required to run their processes.
- SJF scheduling algorithm is used frequently in long term scheduling.
- When a process request a CPU, it must inform the system for how long it want to use the CPU.
- When CPU become available, the system allocates into processes with the lea expected execution time.
- To break ties, it follows the FCFS algorithm.
- Preemptive version of SJF is called as Shortest Remaining Time Next (SRTN)

- Preemptive SJF algorithm will preempt the currently exceutig process, whereas a non preemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- SIF selects processes for service in a manner ensuring the next one will complete and leave the system as soon as possible.

Example 2.8.3 *Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same.*

Process	*Burst time*
P_1	5
P_2	2
P_3	6
P_4	4

Solution : Gantt chart :



Waiting time :

Process	Waiting time
P_1	6 – 0 = 6
P_2	0 – 0 = 0
P_3	11 – 0 = 11
P_4	2 – 0 = 2

$$\text{Average waiting time} = \frac{6+0+11+2}{4} = \frac{19}{4} = 4.75$$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	6 + 5 = 11
P_2	0 + 2 = 2
P_3	11 + 6 = 17
P_4	2 + 4 = 6

$$\text{Average turnaround time} = \frac{11+2+17+6}{4} = \frac{36}{4} = 9$$

- The SJF scheduling algorithm is optimal only when all of the processes are available at the same time and the CPU estimates are available.
- SJF scheduling algorithm may be **preemptive or non-preemptive.**

2.6.3 Priority Scheduling

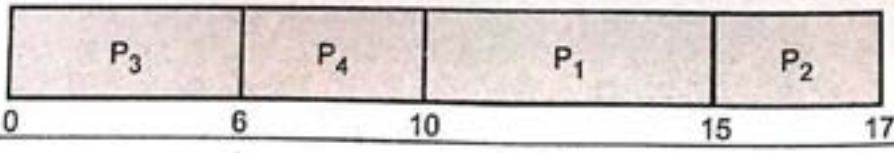
- Priority CPU scheduling algorithm is preemptive and non-preemptive algorithm. It is one of the most common scheduling algorithm in batch system. Priority can be assigned by a system admin using characteristics of the process.
- In this scheduling algorithm, CPU select higher priority process first. If the priority of two process is same then FCFS scheduling algorithm is applied for solving the problem.
- The priority of a process determines how quickly its request for a CPU will be granted if other processes make competing requests.
- Each process is assigned a priority number for the purpose of CPU scheduling.
- The priority number is normally a non negative integer number.
- Non preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.
- Preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

Example 2.6.4 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same. (Time slice = 2)*

Process	*Burst time*	*Priority*
P_1	5	3
P_2	2	4
P_3	6	1
P_4	4	2

Solution : For non-preemptive priority scheduling

Gantt chart :



Waiting time :

Process	Waiting time
P_1	10 – 0 = 10
P_2	15 – 0 = 15
P_3	0 – 0 = 0
P_4	6 – 0 = 6

$$\text{Average waiting time} = \frac{10+15+0+6}{4} = \frac{31}{4} = 7.75$$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	10 + 5 = 15
P_2	15 + 2 = 17
P_3	0 + 6 = 6
P_4	6 + 4 = 10

$$\text{Average turnaround time} = \frac{15+17+6+10}{4} = \frac{48}{4} = 12$$

Problem with Priority Scheduling

- Waiting time is more for lower priority process even if there required CPU burst time is less. Priority scheduling algorithm faces the starvation problem. Starvation problem can be solved by using **aging techniques.**
- In aging, the priority of the process will increase which is waiting for long time in the ready queue.

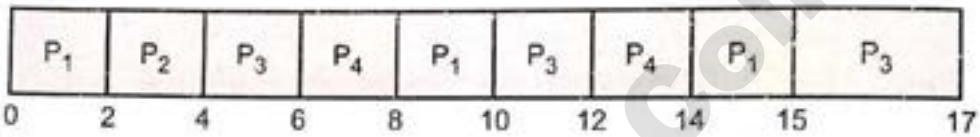
2.6.4 Round Robin Scheduling

- Round robin is a preemptive scheduling algorithm. It is used in interactive system.
- Here process are given a limited amount of time of processor time called a time slice or time quantum. If a process does not complete before its quantum expires , the system preempts it and gives the processor to the next waiting process. The system then places the preempted process at the back of the ready queue.
- Processes are placed in the ready queue using a FIFO scheme. With the RR algorithm, the principle design issue is the length of the time quantum to be used. For short time slice, processes will move through the system relatively quickly. It increases the processing overheads.

Example 2.6.5 *Consider the following set of process that arrive at time 0, with the length of CPU burst given in milliseconds. Calculate the average waiting time and average turnaround time. Provide the Gantt chart for the same. (Time slice = 2)*

Process	*Burst time*
P_1	5
P_2	2
P_3	6
P_4	4

Solution : Gantt chart :



Waiting time :

Process	Waiting time
P_1	0 – 0 + 8 – 2 + 14 – 10 = 10
P_2	2 – 0 = 2
P_3	4 – 0 + 10 – 6 + 15 – 12 = 11
P_4	6 – 0 + 12 – 8 = 10

$$\text{Average waiting time} = \frac{10+2+11+10}{4} = \frac{33}{4} = 8.25$$

Turnaround time : Turnaround time = Waiting time + Burst time

Process	Turnaround time
P_1	10 + 5 = 15
P_2	2 + 2 = 4
P_3	11 + 6 = 17
P_4	10 + 4 = 14

$$\text{Average turnaround time} = \frac{15+4+17+14}{4} = \frac{50}{4} = 12.5$$

2.6.5 Comparison between FCFS and RR

Sr. No.	FCFS	Round robin
1.	FCFS decision made is non-preemptive.	RR decision made is preemptive.
2.	It has minimum overhead.	It has low overhead.
3.	Response time may be high.	Provides good response time for short processes.
4.	It is troublesome for time sharing system.	It is mainly designed for time sharing system.
5.	The workload is simply processed in the order of arrival.	It is similar like FCFS but uses time quantum.
6.	No starvation in FCFS.	No starvation in RR.

2.6.6 Comparison of CPU Scheduling Algorithm

Algorithm	Policy type	Used in	Advantages	Disadvantages
FCFS	Non-preemptive	Batch	1. Easy to implement. 2. Minimum overhead.	1. Unpredictable turn around time. 2. Average waiting is more.
RR	Preemptive	Interactive	1. Provides fair CPU allocation. 2. Provides reasonable response times to interactive users.	1. Requires selection of good time slice.
Priority	Non-preemptive	Batch	1. Ensures fast completion of important jobs.	1. Indefinite postponement of some jobs. 2. Faces starvation problem.

SJF	Non-preemptive	Batch	1. Minimizes average waiting time. 2. SJF algorithm is optimal.	1. Indefinite postponement of some jobs. 2. Cannot be implemented at the level of short term scheduling. 3. Difficulty is knowing the length of the next CPU request.
Multilevel queues	Preemptive or Non-preemptive	Batch/Interactive	1. Flexible. 2. Gives fair treatment to CPUbound jobs.	1. Overhead incurred by monitoring of queues.

Example 2.6.6 *Consider the following process, with the CPU burst time given in milliseconds*

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

Processes are arrived in P_1, P_2, P_3, P_4, P_5 order at time 0.

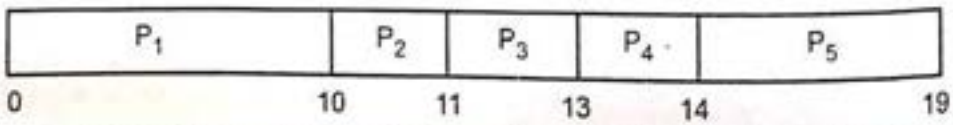
(i) Draw four Gantt charts to show execution using FIFO, SJF, non-preemptive priority and Round Robin (Quantum = 1) scheduling.

(ii) Also calculate waiting time and turnaround time for each scheduling algorithm.

AU : Dec.-15, Marks 16, May-18, Marks 15

Solution :

1. FIFO : Gantt chart :



Process	Waiting time for FIFO	Turnaroud time for FIFO
P_1	0	0 + 10 = 10
P_2	10	10 + 1 = 11

P_3	11	11 + 2 = 13
P_4	13	13 + 1 = 14
P_5	14	14 + 5 = 19

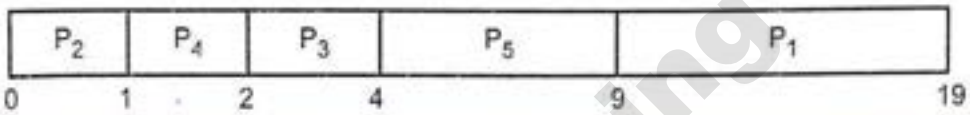
$$\text{Average waiting time} = \frac{\text{Sum of all processes waiting time}}{5}$$

$$= \frac{0 + 10 + 11 + 13 + 14}{5} = 9.6$$

$$\text{Average turnaround time} = \frac{\text{Sum of all processes turnaround time}}{5}$$

$$= \frac{10 + 11 + 13 + 14 + 19}{5} = 13.4$$

2) SJF



Process	Waiting time for SJF	Turnaroud time for SJF
P_1	9	9 + 10 = 19
P_2	0	0 + 1 = 1
P_3	2	2 + 2 = 4
P_4	1	1 + 1 = 2
P_5	4	4 + 5 = 9

$$\text{Average waiting time} = \frac{\text{Sum of all processes waiting time}}{5}$$

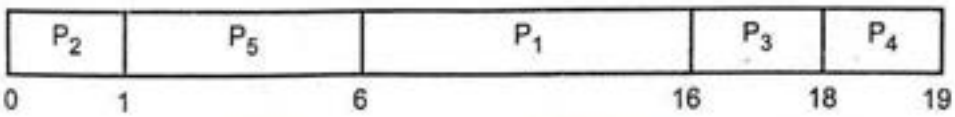
$$= \frac{9 + 0 + 2 + 1 + 4}{5} = 3.2$$

$$\text{Average turnaround time} = \frac{\text{Sum of all processes turnaround time}}{5}$$

$$= \frac{19 + 1 + 4 + 2 + 9}{5} = 7$$

3) Non-Preemptive Priority

Gantt chart : (1 is highest priority and 4 is lowest priority)



Process	Waiting time for Non-Preemptive Priority	Turnaroud time for Non-Preemptive Priority
P_1	6	6 + 10 = 16
P_2	0	0 + 1 = 1
P_3	16	16 + 2 = 18
P_4	18	18 + 1 = 19
P_5	1	1 + 5 = 6

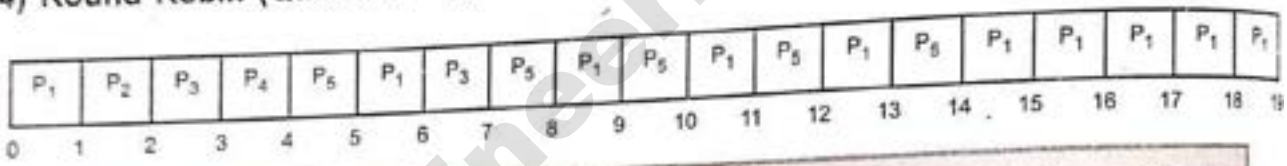
$$\text{Average waiting time} = \frac{\text{Sum of all processes waiting time}}{5}$$

$$= \frac{6 + 0 + 16 + 18 + 1}{5} = 8.2$$

$$\text{Average turnaround time} = \frac{\text{Sum of all processes turnaround time}}{5}$$

$$= \frac{16 + 1 + 18 + 19 + 6}{5} = 13.2$$

4) Round Robin (Quantum = 1)



Process	Waiting time for Round Robin	Turnaroud time for Round Robin
P_1	9	9 + 10 = 19
P_2	1	1 + 1 = 2
P_3	3	3 + 2 = 5
P_4	3	3 + 1 = 4
P_5	9	9 + 5 = 14

$$\text{Average waiting time} = \frac{\text{Sum of all processes waiting time}}{5}$$

$$= \frac{9 + 1 + 3 + 3 + 9}{5} = 5$$

$$\text{Average turnaround time} = \frac{\text{Sum of all processes turnaround time}}{5}$$

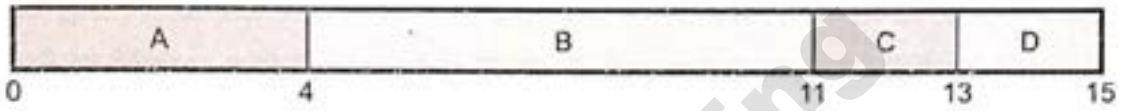
$$= \frac{19 + 2 + 5 + 4 + 14}{5} = 8.8$$

Example 2.6.7 *Explain the various CPU scheduling techniques with Gantt charts clearly as indicated by (process name, arrival time, process time) for the following (A, 0, 4), (B, 2, 7) (C, 3, 2) and (D, 2, 2) for FCFS, SJF, SRT, RR with quantum 1 and 2.*

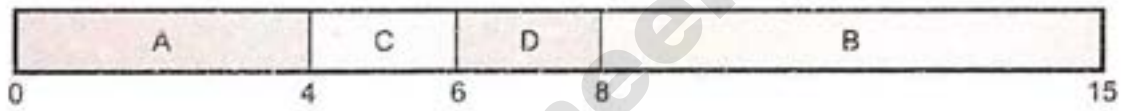
Solution :

Process Name	Arrival time	Process time
A	0	4
B	2	7
C	3	2
D	3	2

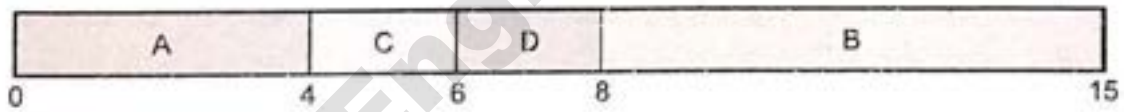
i) FCFS



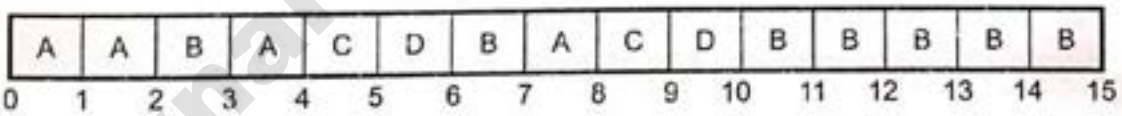
ii) SJF (Non-preemptive)



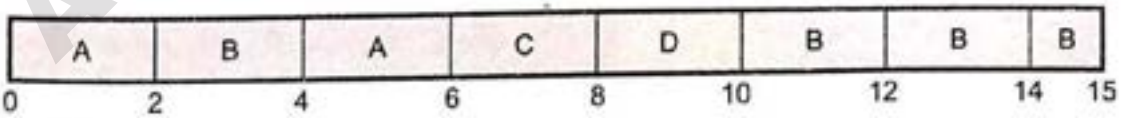
iii) SRT



iv) RR (quantum = 1)



v) RR (quantum = 2)



Example 2.6.8 *Consider the following set of processes, with the length of the CPU burst time given in milliseconds.*

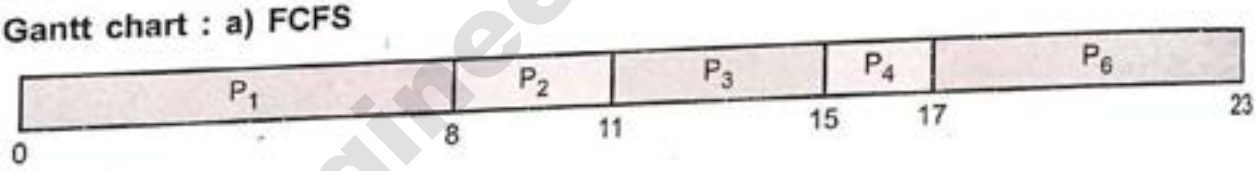
Process	Burst time	Priority
P_1	8	3
P_2	3	1
P_3	4	4
P_4	2	2
P_5	6	5

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4 and P_5 all at time '0'.

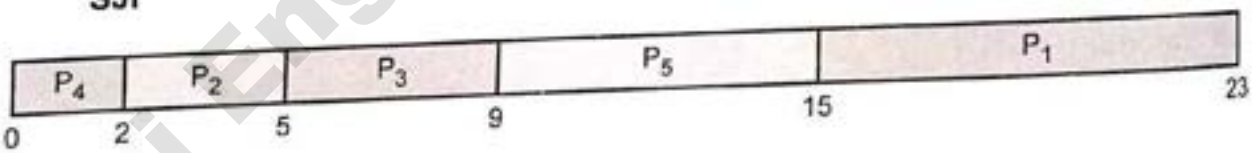
1) *Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF a non-preemptive priority (a smaller priority number implies a higher priority and RR (quantum = 2) scheduling.*
2) *What is the turn around time of each process for each of the scheduling algorithms in part (1).*
3) *What is the waiting time of each process for each of the scheduling algorithms in part (1).*
4) *Which of the schedules in part (1) results in the minimal average waiting time (over all processes).*

Solution :

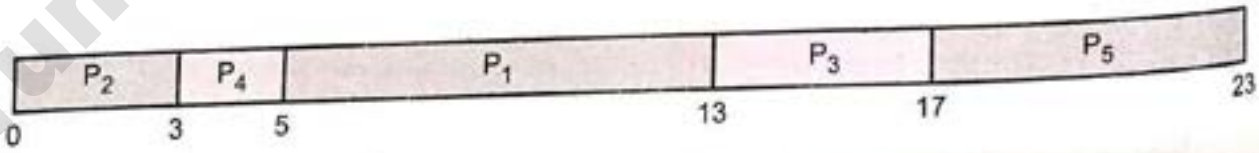
1. Gantt chart : a) FCFS



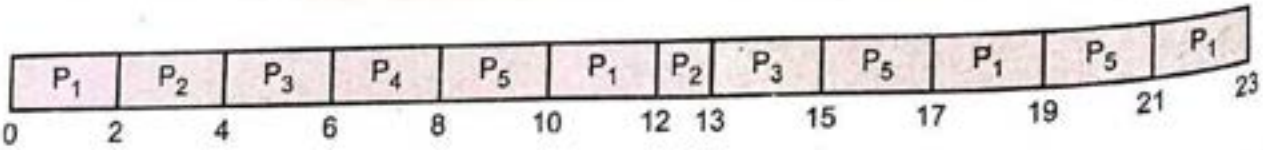
b) SJF



c) Non-preemptive priority



d) RR



2) Turn around time

Process	FCFS	SJF	Non-preemptive Priority	Round Robin
P_1	08	23	13	23
P_2	11	05	03	15
P_3	15	09	17	15
P_4	17	05	05	08
P_5	13	15	23	21

3) Waiting time

Process	FCFS	SJF	Non-preemptive Priority	Round Robin
P_1	0	15	5	15
P_2	8	0	0	10
P_3	11	13	13	11
P_4	15	3	3	6
P_5	17	9	17	15

4) Minimal average waiting time is SJF scheduler

Average waiting time for FCFS = 10.2

Average waiting time for SJF = 6.8

Average waiting time for non-preemptive priority = 7.6

Average waiting time for round robin = 11.4

Example 2.6.9 *Assume the following workload in a system. All jobs arrive at time 0 in the order given*

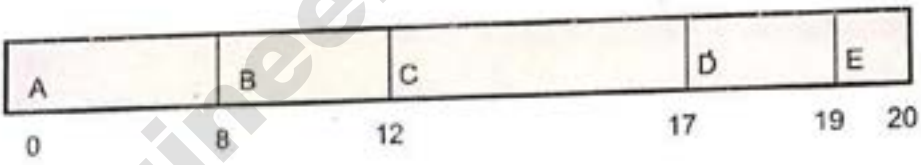
Job	Burst time (ms)	Priority
A	8	2
B	4	1
C	5	4
D	2	2
E	1	3

i) Draw a Gantt chart illustrating the execution of these job using FCFS, RR (quantum = 4), non-preemptive priority (a smaller priority number implies a higher priority) and SJF CPU scheduling.

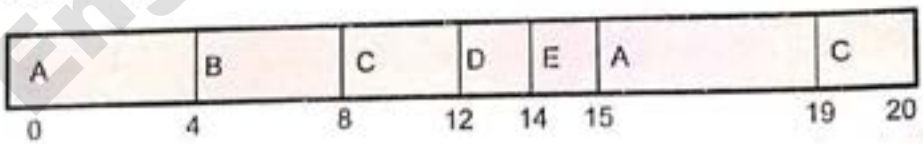
ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm.

Solution : i) Gantt chart

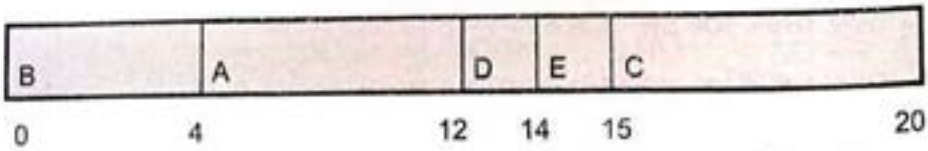
1) FCFS



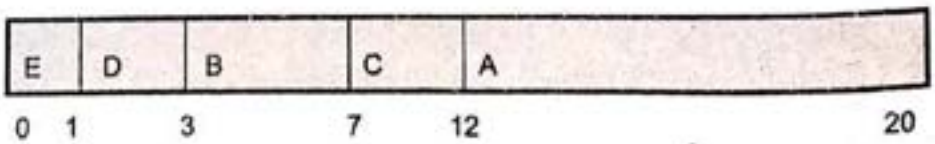
2) RR (quantum = 4)



3) Non - preemptive priority



4) SJF



ii) Waiting time and turnaround time

Job	Waiting time			Turnaround time		
	FCFS	RR	SJF	FCFS	RR	SJF
A	0	11	12	8	19	20
B	8	4	3	12	8	7
C	12	15	7	17	20	12
D	17	12	1	14	14	3
E	19	14	0	20	15	1

Non preemptive priority

Job	Waiting time	Turnaround time
A	4	12
B	0	4
C	15	20
D	12	14
E	14	15

Average waiting time :

i) FCFS = 11.20

ii) RR = 11.20

iii) SJF = 4.6

iv) Non preemptive priority = 9

Average turnaround time

i) FCFS = 14.20

ii) RR = 15.20

iii) SJF = 8.6

iv) Non preemptive priority = 13

Example 2.6.10 *Assume the following processes arrive for execution at the time indicated and also mention with the length of the CPU-burst time given in milliseconds.*

Job	Burst time (ms)	Priority	Arrival time (ms)
A	10	5	0
B	6	2	0
C	7	4	1
D	4	1	1
E	5	3	2

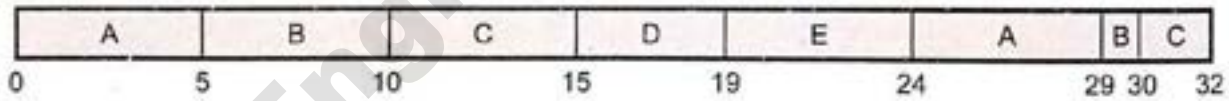
i) Give a Gantt chart illustrating the execution of these processes using FCFS, Round Robin (quantum = 5), and Priority (Preemptive and Non Preemptive).

ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm.

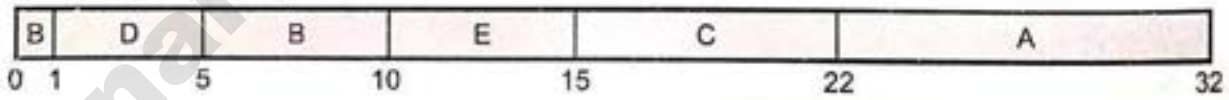
Solution : i) Gantt chart

a) FCFS

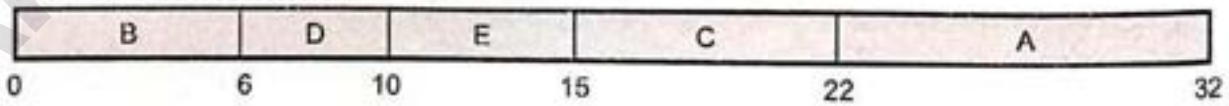
b) RR (Quantum = 5)



c) Preemptive priority



d) Non preemptive priority



ii) **Average waiting time and turnaround time**

Waiting time and turnaround time :

Job	Waiting time				Turnaround time			
	FCFS	RR	Preemptive Priority	Nonpreemptive priority	FCFS	RR	Preemptive priority	Nonpreemptive priority
A	00	19	22	22	10	29	32	32
B	10	24	04	00	16	30	10	06
C	15	24	14	14	22	31	21	21
D	22	14	00	05	26	18	04	09
E	25	17	08	08	30	22	13	13

Average waiting time and average turnaround time

Method	Average waiting time	Average turnaround time
FCFS	$= \frac{00+10+15+22+25}{5} = \frac{72}{5} = 14.4$	$\frac{10+16+22+26+30}{5} = \frac{104}{5} = 20.8$
RR	$= \frac{19+24+24+14+17}{5} = \frac{98}{5} = 19.6$	$\frac{29+30+31+18+22}{5} = \frac{130}{5} = 26$
Preemptive priority	$= \frac{22+4+14+0+8}{5} = \frac{48}{5} = 9.6$	$\frac{32+10+21+4+13}{5} = \frac{80}{5} = 16$
Non-preemptive priority	$= \frac{22+0+14+5+8}{5} = \frac{49}{5} = 9.8$	$\frac{32+06+21+9+13}{5} = \frac{81}{5} = 16.2$

Example 2.6.11 *Assume the following processes arrive for execution at the time indicated and also mention with the length of the CPU-burst time given in milliseconds.*

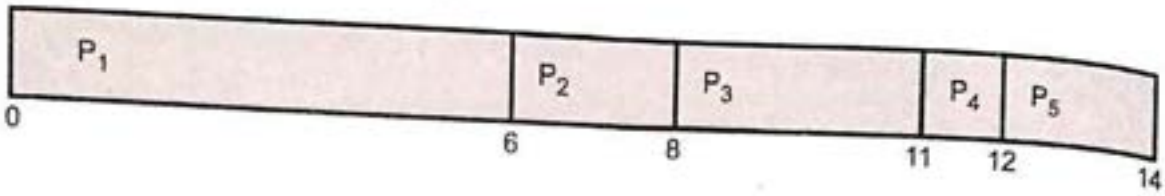
Job	*Burst time (ms)*	*Priority*	*Arrival time (ms)*
P_1	6	2	0
P_2	2	2	1
P_3	3	4	1
P_4	1	1	2
P_5	2	3	2

i) Give a Gantt chart illustrating the execution of these processes using FCFS, Round Robin (quantum = 1), and Priority (Preemptive and Non-preemptive).

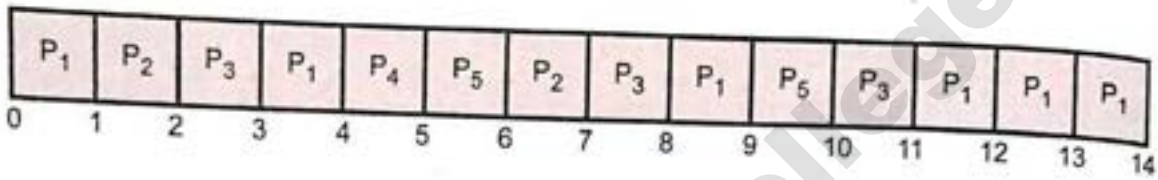
ii) Calculate the average waiting time and average turnaround time for each of the above scheduling algorithm.

Solution : i) Gantt chart

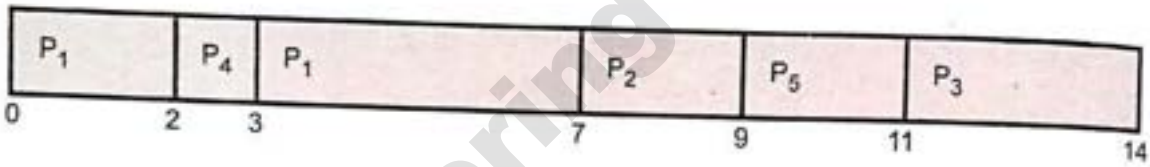
a) FCFS



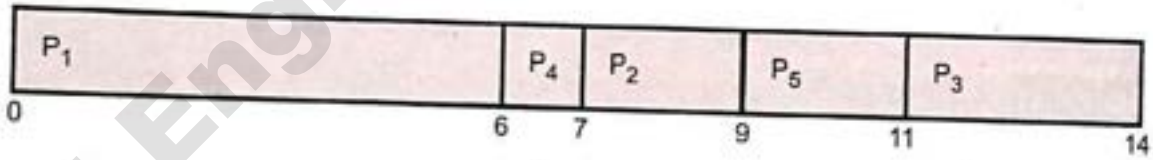
b) Round Robin (quantum = 1)



c) Preemptive priority



d) Non-preemptive priority



ii) Waiting time and turnaround time

Process	Waiting time			
	FCFS	RR	Preemptive priority	Non-preemptive priority
P_1	0	8	1	0
P_2	5	4	6	6
P_3	7	7	10	10
P_4	9	2	0	4

P_5	10	6	7	7
Average waiting time	6.2	5.4	4.8	5.2

Turnaround time

Process	Turnaround time			
	FCFS	RR	Preemptive priority	Non-preemptive priority
P_1	6	14	7	6
P_2	7	6	8	8
P_3	10	10	13	13
P_4	10	3	1	5
P_5	12	8	9	9
Average waiting time	9	8.2	7.6	8.2

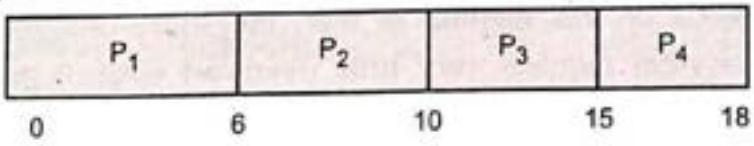
Example 2.6.12 *Consider the following set of processors with the length of CPU burst time given in milliseconds.*

Process	*Arrival time*	*Burst time*
P_1	0	6
P_2	1	4
P_3	3	5
P_4	5	3

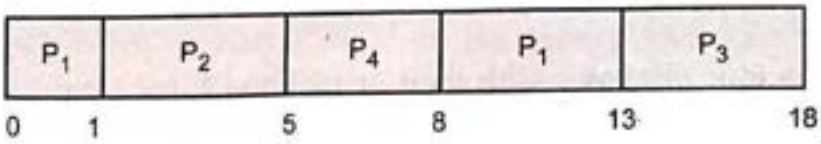
Draw the Gantt charts illustrating the execution of these processes using SJF (preemptive and non-preemptive) and FCFS. Calculate average turnaround time, average waiting time in each case.

Solution : Gantt chart

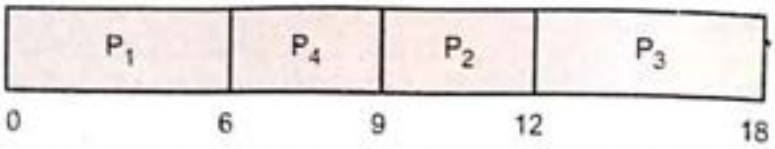
i) FCFS



ii) SJF (preemptive)



iii) SJF (Non-preemptive)



	FCFS		SJF Preemptive		SJF Non-preemptive	
Process	Waiting time	Turnaround time	WT	TT	WT	TT
P1	0	6	7	13	0	6
P2	5	9	0	4	8	12
P3	7	12	10	15	10	15
P4	10	13	0	3	1	4
Total	22	40	17	35	19	37

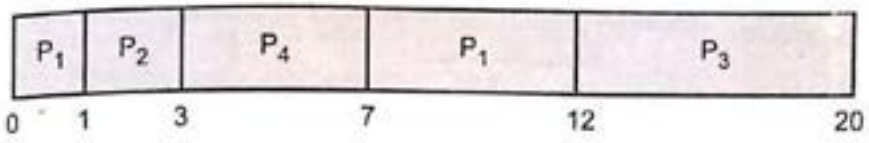
Average waiting time	Average turnaround time
FCFS = $\frac{22}{4} = 5.5$	FCFS = $\frac{40}{4} = 10$
Preemptive SJF = $\frac{17}{4} = 4.25$	Preemptive SJF = $\frac{35}{4} = 8.75$
Non-preemptive SJF = $\frac{19}{4} = 4.75$	Non-preemptive SJF = $\frac{37}{4} = 9.25$

2.6.7 Shortest Remaining Time Next

- Preemptive SJF is called *shortest remaining time first*. In this algorithm, the scheduler selects the process with the smallest estimated run time to completion.
- This scheduler gives minimum wait times in theory but in certain situation due to preemption overhead, shortest process first might performance better.
- An advantageous of this method is that, the short processes are handled very quickly. The system requires very little overhead since it only makes a decision when a process completes or a new process is added. When a new process is admitted the SRTN algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.
- Consider the four processes with their arrival and burst time.

Process	Burst Time	Arrival time
P1	6	0
P2	2	1
P3	8	2
P4	4	3

- The execution of the process is shown by the Gantt chart:



- Now we calculate waiting time and turnaround time for all processes.

Process	Waiting Time	Turnaround time
P1	(0 - 0) + (7 –1) = 6	6 + 6 = 12
P2	1 – 1 = 0	2 + 0 = 2
P3	12 – 2 = 10	8 + 10 = 18
P4	3 – 3 = 0	4 + 0 = 4

Average waiting time = $\frac{6+0+10+0}{4} = \frac{16}{4} = 4$

Average waiting time = $\frac{12+2+18+4}{4} = \frac{36}{4} = 9$

Advantages :

1. Very short processes get very good service.
2. The penalty ratios are small; this algorithm works extremely well in most cases
3. This algorithm provably gives the highest throughput of all scheduling algorithms if the estimates are exactly correct.

2.6.8 Multilevel Queue Scheduling

- Multilevel queue is not separate scheduling algorithm but it works in conjunction with several CPU scheduling algorithms. The processes can be grouped according to common characteristics.
- One type of multilevel queue is based on the priority based system with different queues for each priority level.

- System consists of two types of processes or jobs : Interactive and batch. Interactive jobs are shorter and batch jobs are longer. Fig. 2.6.1 shows multilevel queue scheduling.

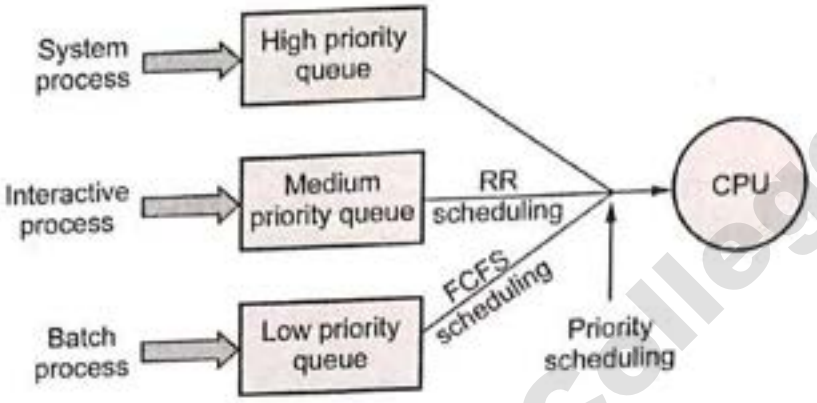


Fig. 2.6.1 Multilevel queue scheduling

- To keep minimum response time for user, interactive jobs are scheduled before the batch jobs. So ready queue is partitioned into two separate queue: interactive (foreground) and batch (background). Each queue uses their own scheduling algorithm.
- Once the jobs assigns to the queue, it will remain in that queue. Jobs never change the queue. Here we assume that FCFS is applied to one queue and round robin method is applied to another queue.
- The batch jobs are put in one queue called the background queue while the interactive jobs are put in a foreground queue. Batch jobs are scheduled by first come first serve method and interactive jobs by round robin method.
- The scheduler now has to decide which queue to run. The methods are as follows :

1. Higher priority queues can be processed until they are empty before the lower priority queues are executed.
2. Each queue can be given a certain amount of the CPU. Maybe, the interactive queue could be assigned 75 % of the CPU, with the batch queue being given 25%.

- It should also be noted that there can be many other queues. Many systems will have a *system queue*. This queue will contain processes that are important to keep the system running. For this reason these processes normally have the highest priority.
- It should be also need to specify which queue a process will be put to when it arrives to the system and/or when it starts a new CPU burst.

Advantages :

1. It can be preemptive or non-preemptive.
2. Short CPU bound processes are executed first.
3. Simple to implement.

Disadvantage :

1. Queues require monitoring, which is a costly activity.

2.6.9 Multilevel Feedback Queue Scheduling

- Multilevel Feedback Queue (MLFQ) allows moving the processes between the queues. MLFQ has a number of distinct queues; each assigned a different priority level. At any given time, a process that is ready to run is on a single queue. MLFQ uses priorities to decide which process should run at a given time. A process with higher priority is selected for execution.
- Consider processes with different CPU burst characteristics. If a process uses too much of the CPU it will be moved to a lower priority queue. This will leave I/O bound and interactive processes in the higher priority queue.
- Fig. 2.6.2 shows multilevel feedback queue scheduling.

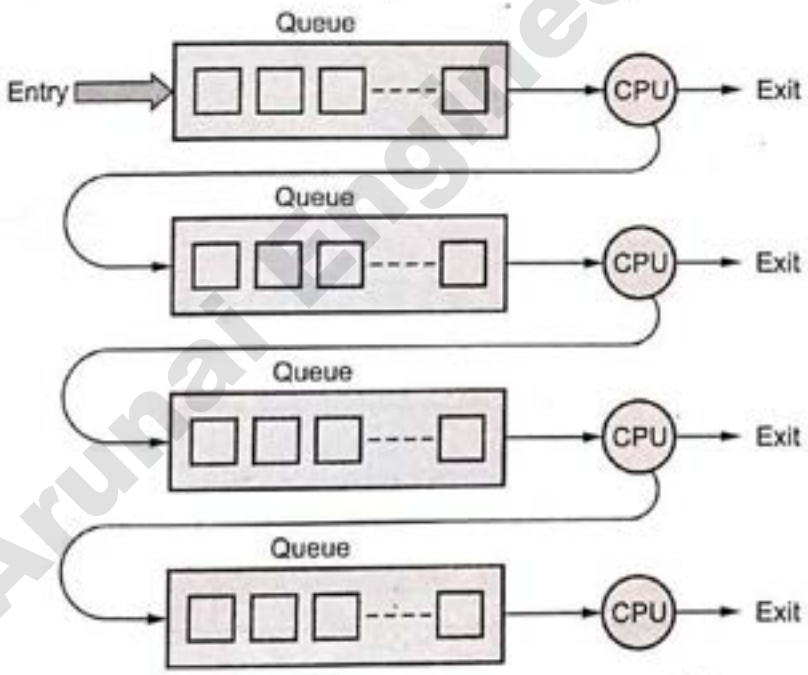


Fig. 2.6.2 Multilevel feedback queue scheduling

- MLFQ also uses concept of aging to prevent starvation. The processes with lower priority will move towards the higher priority queue in aging concept. The key to MLFQ scheduling lies in how the scheduler sets priorities. Rather than giving a

fixed priority to each process, MLFQ varies the priority of a process based on its observed behavior.

- Assume we have three queues (Q0, Q1 and Q2). Q0 is the lowest priority queue and Q2 is the highest priority queue. The process enters at the highest priority (Q2). After a single time-slice of 10 ms, the scheduler reduces the process's priority by one, and thus the job is on Q1. After running at Q1 for a time slice, the job is finally lowered to the lowest priority in the system (Q0), where it remains.
- Following parameters are defined by scheduler for implementing MLFQ :

1. Number of queue required.
2. Scheduling algorithm for each queue.
3. Method is required for finding to increase and/or decrease priority of processes.

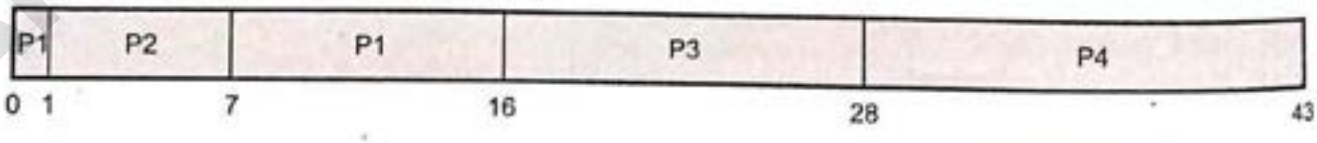
Example 2.8.13 *Explain the FCFS, preemptive and non preemptive versions of shortest-job-first and roudn robin (time slice = 2) schedulign algorithms with Gantt chart for the four processes given. Compare their average turn around and waiting time.*

Process	Arrival time	Burst time
P1	0	10
P2	1	6
P3	2	12
P4	3	15

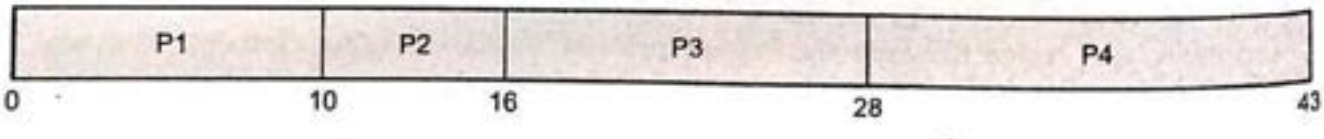
AU : May-15, Marks 12

Ans. : **a) Gantt chart for FCFS :**

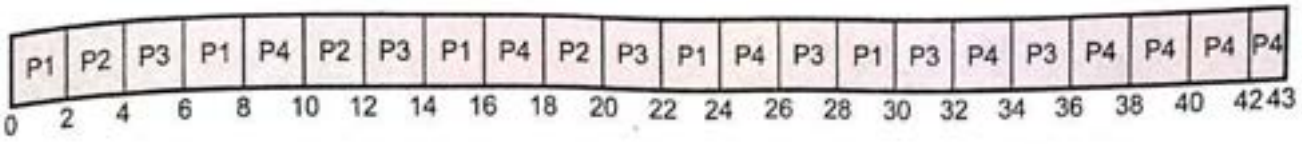
b) Gantt chart for preemptive SJF :



c) Gantt chart for non-preemptive SJF :



d) Gantt chart for round robin :



Waiting time

Process	FCFS	Preemptive SJF	Nonpreemptive SJF	RR
P1	0	6	0	20
P2	9	0	9	13
P3	14	14	14	22
P4	25	25	25	25

Average waiting time

$$\text{FCFS} = \frac{0+9+14+25}{4} = \frac{48}{4} = 12$$

$$\text{Preemptive SJF} = \frac{6+0+14+15}{4} = \frac{35}{4} = 8.75$$

$$\text{Nonpreemptive SJF} = \frac{0+9+14+15}{4} = \frac{48}{4} = 12$$

$$\text{RR} = \frac{20+13+22+25}{4} = \frac{80}{4} = 20$$

Turnaround time

Process	FCFS	Preemptive SJF	Nonpreemptive SJF	RR
P1	10	16	10	30
P2	15	6	15	19
P3	26	26	26	34
P4	40	40	40	40

Average turnaround time

$$\text{FCFS} = \frac{10+15+26+40}{4} = \frac{91}{4} = 22.75$$

$$\text{Preemptive SJF} = \frac{16+6+26+40}{4} = \frac{88}{4} = 22$$

$$\text{Nonpreemptive SJF} = \frac{10+15+26+40}{4} = \frac{91}{4} = 22.75$$

$$\text{RR} = \frac{30+19+34+40}{4} = \frac{123}{4} = 30.75$$

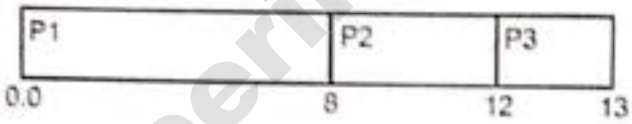
Example 2.6.14 *What is the average turnaround time for the following process using a) FCFS b) SJF non-prmptive c) Preemptive SJF.*

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

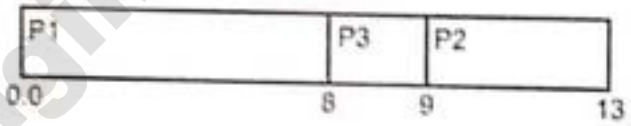
AU : Dec.-17, Marks 9

Ans. : Gantt chart :

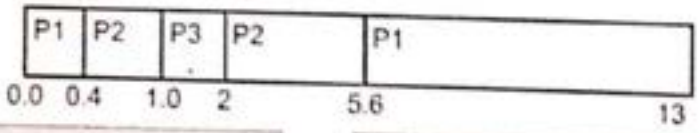
a) FCFS



b) SJF non-preemptive



c) Preemptive SJF



Process	Waiting Time		
	FCFS	SJF Non-preemptive	Preemptive SJF
P1	0	0	5.2
P2	7.6	8.6	1
P3	11	7	0

Process	Turnaround Time		
	FCFS	SJF Non-preemptive	Preemptive SJF
P1	8	8	13.2
P2	11.6	12.6	5
P3	12	8	1
Average turnaround time	**7.2**	**9.53**	**6.4**

University Questions

1. *Consider the following set of processes, with the length of the CPU-burst time in given ms :*

Process	Burst Time	Arrival Time
P1	8	0.00
P2	4	1.001
P3	9	2.001
P4	5	3.001
P5	3	4.001

Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, priority and RR (quantum = 2) scheduling. Also calculate waiting time and turnaround time for each scheduling algorithms.

AU : May-17, Marks 13

2. *Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short process. :*
   i) RR ii) Multilevel feedback queues.

AU : May-18, Marks 13

3. *Consider the following set of processes, with the length of the CPU burst given in milliseconds :*

Process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The process are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5 all at time 0.

i) Draw Gantt chart that illustrate the execution of these processes using the scheduling algorithms FCFS (smaller priority number implies higher priority) and RR (quantum = 1). **[10]**

ii) What is the waiting time of each process for each of the scheduling algorithms ? **[5]**

AU : May-18, Marks 15

2.7 Algorithm Evolution

- Performance evaluation of the CPU scheduling is required for selection of th algorithm. The maximizing CPU utilization and maximizing throughput are tw parameters used for evaluation of the scheduling algorithm.
- Following are the different evaluation methods used for this :
  1. Deterministic Modeling
  2. Queueing Models
  3. Simulations
  4. Implementation

2.7.1 Deterministic Modeling

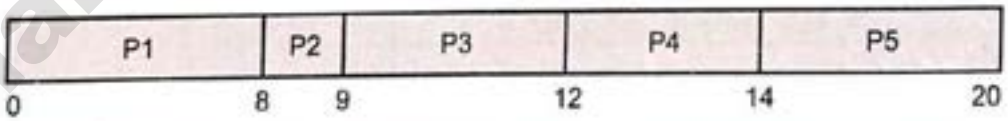
- Deterministic modeling uses an analytic method. It takes a particular predetermined workload and defines the performance of each algorithm for that workload. It gives real calculation of each case.

Example 2.7.1 *Consider the following set of processes with the length of CPU burst time given in the milliseconds.*

Process	Arrival time	Burst time	Priority
P1	0	8	3
P2	1	1	1
P3	2	3	2
P4	3	2	3
P5	4	6	4

Calculate average turnaround time and average waiting time for first come first served, Shortest job first and priority algorithm.

Solution : For the FCFS the processes are executed as follows :

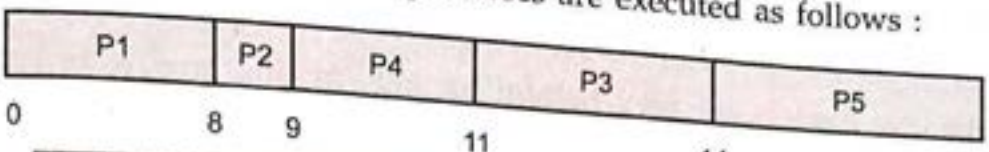


Process	Waiting time	Turnaround time
P1	0	8
P2	7	8
P3	7	10
P4	9	11
P5	10	16

$$\text{Average waiting time} = \frac{0+7+7+9+10}{5} = 6.6$$

$$\text{Average turnaround time} = \frac{8+8+10+11+16}{5} = 10.6$$

- For the non-preemptive SJF the processes are executed as follows :



Process	Waiting time	Turnaround time
P1	0	8
P2	7	8
P3	9	12
P4	6	8
P5	10	16

$$\text{Average waiting time} = \frac{0+7+9+6+10}{5} = 6.4$$

$$\text{Average turnaround time} = \frac{8+8+12+8+16}{5} = 10.4$$

- For the non - preemptive priority scheduling the processes are executed same as FCFS algorithm. So average waiting is same as FCFS waiting time.
- The average waiting time of SJF is small as compared to FCFS and priority scheduling algorithm.
- Deterministic modeling is simple to implement and accurate also.

2.7.2 Queueing Models

- Each of us has spent a great deal of time waiting in lines. In this chapter, we develop mathematical models for waiting lines, or queues. Queuing theory is the mathematics of waiting lines. It is extremely useful in predicting and evaluating system performance.
- Queuing theory has been used for operations research. Traditional queuing theory problems refer to customers visiting a store, analogous to requests arriving at a device.
- The theory of queuing systems was developed to provide models for forecasting behaviors of systems subject to random demand. The first problems addressed concerned congestion of telephone traffic. Erlang observed that a telephone system

can be modeled by Poisson customer arrivals and exponentially distributed service times.

- A queuing system can be described as follows :

"Customers arrive for a given service, wait if the service cannot start immediately and leave after being served".

- The term "customer" can be men, products, machines, ...
- The Queuing models are very helpful for determining how to operate a queuing system in the most effective way if too much service capacity to operate the system involves excessive costs. The models enable finding an appropriate balance between the cost of service and the amount of waiting.
- Following Fig. 2.7.1 shows the queueing system.

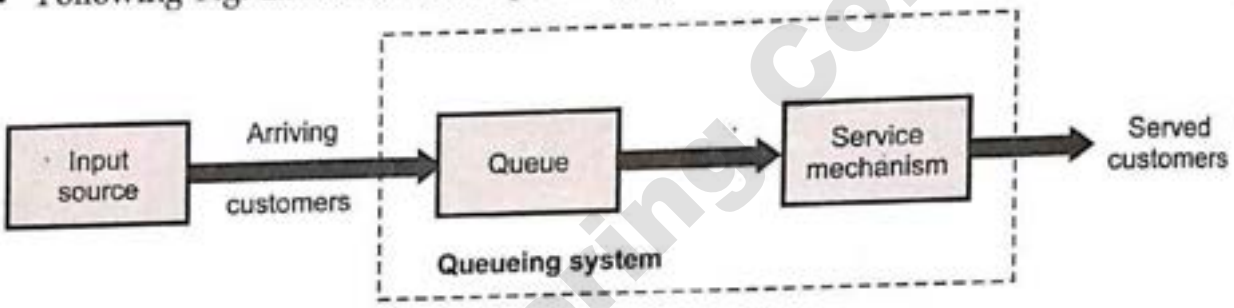


Fig. 2.7.1 Queueing system

- The basic queueing process is as follows :
  1. Customers are generated over time by an input source.
  2. The customers enter a queueing system.
  3. A required service is performed in the service mechanism.
- **Distribution of arrivals :** Generally , the arrival of customers into the system is a random event. Frequently the arrival pattern is modeled as a Poisson process.
- **Distribution of service times :** Service time is also usually a random variable. A distribution commonly used to describe service time is the exponential distribution.
- **Queue discipline :** Most common queue discipline is first served (FCFS). Other disciplines assign priorities to the waiting units and then serve the unit with the highest priority first.

2.7.3 Simulations

- Simulations create a model of the system, then process statistical or real data. A component of the system is the major software data structures.
- Clock represents the variable in the system. Variable values are increases for reflecting system state.

- Simulator executes the algorithm and algorithm performances are gathered and printout is taken.
- Required data is generated in different ways : Random number generator one of the method for generating input for the system. Random number generator generates the data for processes, CPU burst times, arrival times, departures timing, priority etc.
- The distribution can be defined **mathematically or empirically.**
- **Empirically :** Measurements of the actual system under study are taken. In real system, results are defining the distribution of events. Simulation uses this result for calculating output.
- Simulation methods are expensive and time consuming.

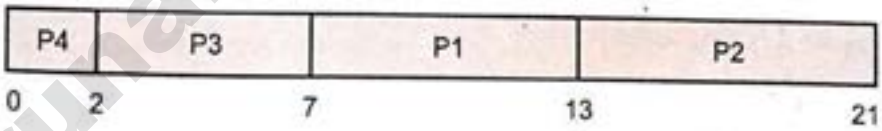
Example 2.7.2 *Find average waiting time for shortest job first scheduling and round robin scheduling algorithm.*

Process	*CPU burst time*
P1	*6*
P2	*8*
P3	*5*
P4	*2*

CPU burst time is given in millisecond and time quantum is 4.

Solution : 1) Shortest job first scheduling (Non-preemptive)

Gantt chart :



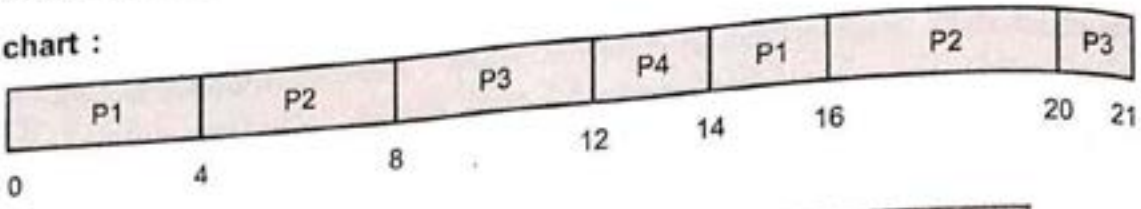
Waiting time :

Process	Waiting time
P1	7
P2	13
P3	2
P4	0

$$\text{Average waiting time} = \frac{7+13+2+0}{4} = \frac{22}{4} = 5.5$$

2. Round Robin (time quantum = 4)

Gantt chart :



Waiting time :

Process	Waiting time
P1	0 + (14 – 4) = 10
P2	4 + (16 – 8) = 12
P3	8 + (20 – 12) = 16
P4	12

Average waiting time : $\frac{10+12+16+12}{4} = \frac{50}{4} = 12.5$

2.8 Multiprocessor Scheduling

- Designs of processor scheduling parameters are changed when more than one processor is used.
- Classification of multiprocessor systems are as follows :

1. **Loosely coupled :** It is also called distributed multiprocessor or cluster. It is a collection of autonomous systems with own main memory and I/O channels.
2. **Functionally specialized processors :** It consists of a master, general-purpose processor. Master processors are controlled the system and provides services. An example is an I/O processor
3. **Tightly coupled multiprocessor :** It consists of a set of processors that share a common main memory. The popular multi-core architecture falls into this category.

2.8.1 Issue Relating to The Scheduling

1. **Granularity :** It is one of the characteristics of the multiprocessor system. It is used with frequency of synchronization between processes in a system. We can distinguish five categories of parallelism that differ in the degree of granularity.

a. **Independent parallelism :** Each process in the system represents a separate independent application. There is no synchronization among processes. Used in time sharing machine.

b. **Coarse and very coarse-grained parallelism :** Minimum synchronizations among the processes. Set of concurrent processes running on a multi-programmed uni-processor system uses this type of parallelism.

c. **Medium-grained parallelism :** High degree of coordination is needed among those threads, which leads to a medium-grained parallelism.

d. **Fine-grained parallelism** represents a much more complex use of parallelism, and remains a very difficult area.

2.8.2 Various Design Issues

- Scheduling on a multiprocessor involves three issues:

a. The assignment of processes to processors

b. The use of multiprogramming on individual processors

c. The actual dispatching of a process

1. Processor assignment

- Regarding the actual assignment, at least two approaches can be followed :

a. With a master/slave approach, key kernel functions, including the scheduler of the operating system, always run on a "master" processor, while the rest of the processors are used to run user processes. When a slave process needs some service, it simply sends a request to the master processor, and waits for its response. This approach is very simple and does not need a conflicting resolution mechanism. But, the master can become a bottleneck, and it can even bring down the whole system when it fails.

- **In a peer structure,** the OS can execute on any processor, and each of them does its own scheduling among the available processes.

2. Use of multiprogramming on individual processors

- For coarse-grained, or independent processes, each process contains a large amount of instructions, thus, it is much more likely that a processor can be idle when processes are blocked for various service. It is necessity to processor to switch among processes to have a better performance. But, for a medium-grained applications running in an environment with many processors available, it is no longer that important to keep all processors busy all the time.

- A key design issue for a multiprocessor scheduling is the actual selection of a process for execution. In a multi-programmed environment, more sophisticated scheduling, based on such factors as priority and past usage, may lead to a better performance, compared with simpler scheduling algorithm such as FCFS.

- But, a sophisticated algorithm often leads to overhead, thus unnecessary or even counter-productive for the overall performance.

- In most traditional multiprocessor systems, processes are not associated with dedicated processors. But, a single queue for all the processors is used to serve all the competing processes.
- If priority is an important issue, then several queues will be used, each of which takes care of a class of processes with the same priority.
- Simple FCFS strategy or the use of FCFS coupled with a static priority scheme may suffice for a multiprocessor system.

2.8.3 Thread Scheduling

- An application can be implemented as a set of threads, which cooperate and execute concurrently in the same address space. Four general approach for multiprocessor thread scheduling are as follows :

1. Load sharing
2. Gang scheduling
3. Dedicated processor scheduling
4. Dynamic scheduling

1. Load sharing

- Threads are not assigned to a particular processor. When a processor is idle, it selects a thread from a global queue serving all processors.
- In this method, load is evenly distributed among processors. No centralized scheduler is required.
- This policy is implemented in different ways: FCFS and priority method is used. These are depends upon queue structure.
- Three different versions of load sharing :

a. First-come-first-served (FCFS) : On arrival of the thread, it is placed at the end of the shared queue. Next ready thread is selected when processor is idle. Processor executes the thread until its completion or blocking.

b. Smallest number of threads first : Shared ready queue is organized according to the priority. Processor selects highest priority thread with the smallest number of unscheduled threads. If the priority of the two thread is same, then FCFS method is applied.

c. Preemptive smallest number of threads first : Highest priority is given to jobs with the smallest number of unscheduled threads.

Advantages of Load sharing

1. No centralized scheduler is required.

2. Load distribution is equal.
3. Global queue is maintained by the system.

Disadvantages of Load sharing

1. It enforces mutual exclusion because of memory.
2. High degree of coordination is required between the threads of a program.
3. Caching becomes less efficient.

2. Gang scheduling

- With gang scheduling, a set of related threads is scheduled to run on a set of processors at the same time, on a one-to-one basis.
- Synchronization blocking may be reduced or increased.
- If closely related processes executes in parallel, synchronization blocking may be reduced.
- Set of related threads is scheduled to run on a set of processors.
- Gang scheduling has three parts.

1. Group of related threads are scheduled as a unit, a gang.
2. All members of a gang run simultaneously on different timeshared CPUs.
3. All gang members start and end their time slices together.

- The trick that makes gang scheduling work is that all CPU are scheduled synchronously. This means that time is divided into discrete quanta.
- An example of how gang scheduling works is given in the Fig. 2.8.1. Here we have a multiprocessor with six CPU being used by five processes, A through E, with a total of 24 ready threads.

	CPU					
Time slot	0	1	2	3	4	5
0	A_0	A_1	A_2	A_3	A_4	A_5
1	B_0	B_1	B_2	C_0	C_1	C_2
2	D_0	D_1	D_2	D_3	D_4	E_0
3	E_1	E_2	E_3	E_4	E_5	E_6
4	A_0	A_1	A_2	A_3	A_4	A_5
5	B_0	B_1	B_2	C_0	C_1	C_2
6	D_0	D_1	D_2	D_3	D_4	E_0
7	E_1	E_2	E_3	E_4	E_5	E_6

Fig. 2.8.1 Gang scheduling

- During time slot 0, threads A_0 through A_5 are scheduled and run.
- During time slot 1, threads B_0, B_1, B_2, C_0, C_1, C_2 are scheduled and run.
- During time slot 2, D's five threads and E_0 get to run.
- The remaining six threads belonging to process E run in the time slot 3. Then cycle repeats, with slot 4 being the same as slot 0 and so on.
- Gang scheduling is useful for applications where performance severly degra when any part of the application is not running.

3. Dedicated processor assignment

- When application is scheduled, its threads are assigned to a processor.
- Some processor may be idle and no multiprogramming of processors.
- Provides implicit scheduling defined by assignment of threads to processors. F the duration of program executation, each program is allocated a set of process equal in number to the number of threads in the program. Processors are chos from the available pool.

4. Dynamic scheduling

- Number of threads in a process are altered dynamically by the application.
- Operating system and the application are involved in making scheduling decisio The OS is responsible for partitioning the processors among the jobs.
- Operating system adjusts load to improve the use

1. Assign idle processors.
2. New arrivals may be assigned to a processor that is used by a job curren using more than one processor.
3. Hold request until processor is available.
4. New arrivals will be given a processor before executing running applications.

2.9 Real Time Scheduling

- Many real time systems are built with operating systems providing multitaskin facilities.
- **A hard real-time system** guarantees that real-time tasks be completed within the required deadlines. Failure to meet a single deadline may lead to a critic catastrophic system failure such as physical damage or loss of life.
- **A firm real-time system** tolerates a low occurrence of missing a deadline. A fe missed deadlines will not lead to total failure, but missing more than a few may lead to complete and catastrophic system failure.

- A **soft real-time system** provides priority of real-time tasks over non real-time tasks. Performance degradation is tolerated by failure to meet several deadline time constraints with decreased service quality but no critical consequences.
- Task is the schedulable unit of the system. Task is characterized by timing constraints and resource requirements. It can be preemptive or non-preemptive.
- Scheduling algorithm guarantees a newly arrived task meets its deadline if the algorithm can find a feasible schedule.

2.9.1 Characteristics of Real-Time Operating Systems

- RTOS are characterized by these main features

1. Determinism
2. Responsiveness
3. User control
4. Reliability
5. Fail-soft operation

1. **Determinism :** Operations are performed at fixed, predetermined times or within predetermined time intervals.
2. **Responsiveness :** How long, after acknowledgment, the operating system takes to service the interrupt. It includes the time to begin execution of the interrupt service routine (ISR). If a context switch is necessary, the delay is longer than an ISR executed within the context of the current process.
3. **User Control :** User should be able to
   a) Specify paging or process swapping
   b) Decide which processes must reside in main memory
   c) Establish the rights of processes
   d) Fine-grained control over task priorities
   e) Select algorithms for disks scheduling
4. **Reliability :** Real time system must be reliable. Reliability means that system should not fail. The mean time between the failures should be very high.
5. **Fail-soft operation :** It is ability of a system to fail in such a way as to preserve as much capability and data as possible.

- **Periodic tasks :** All jobs of a periodic task T_i have a regular inter arrival time T_i, we call T_i the period of the periodic task T_i. If a job for a periodic task T_i arrives at time t, then the next job of task T_i must arrive at $t + T_i$.

- **Sporadic tasks :** The jobs of a sporadic task T_i arrive irregularly, but they have minimum inter-arrival time T_i, we call T_i the period of the sporadic task T_i. If job of a sporadic task T_i arrives at t, then the next job of task T_i can arrive at an time at or after $t + T_i$.

2.9.2 Class of Algorithms

- Real time scheduling algorithm depends upon following factors :

1. Schedulability analysis is performed or not.
2. Which method is used? Static or dynamic method
3. Schedule is produced by result of analysis

- Based on the above considerations, the following classes of algorithms ar identified :

1. Static table driven
2. Static priority driven preemtive
3. Dynamic planning
4. Dynamic best effort

1. Static table-driven approaches :

- It is applicable mainly to periodic tasks. This performs a static analysis of feasibl schedules.
- Input required for analysis : Periodic arrival times, execution time, periodic endin deadline and relative priority of tasks.
- The scheduler tries to develop schedule to meet all needs.
- Earliest deadline first is an example of this type of scheduling algorithm. A set o tasks is schedulable if the sum of task loading is less than 100%. If the load i above 100%, then it is not stable.

2. Static priority-driven preemptive approaches :

- A static analysis is done, and the result is used to assign priorities to tasks. A traditional priority driven preemptive scheduler can then be used.
- At run-time, tasks are executed highest-priority-first, with preemptive-resum policy. When resources are used, need to compute worst-case blocking times.
- Usually, in a real-time system, the priority is related to the time constraints on th tasks.
- Rate Monotonic scheduling assigns priorities based on their periods.

3. Dynamic planning-based approaches :

- Feasibility is determined at run-time rather than an offline analysis prior to the start of execution.
- An arriving task is accepted only if it is feasible to meet its time constraints.
- Requires constant reworking of the schedule to accommodate new tasks and existing ones

4. Dynamic Best Effort :

- There is no feasibility analysis.
- System tries to meet all deadlines and aborts any started process whose deadline is missed.

Preemption

- A computation or task is preemptable if it can be interrupted when another more critical task needs to be executed. Fig. 2.9.1 shows the effect of preemption.

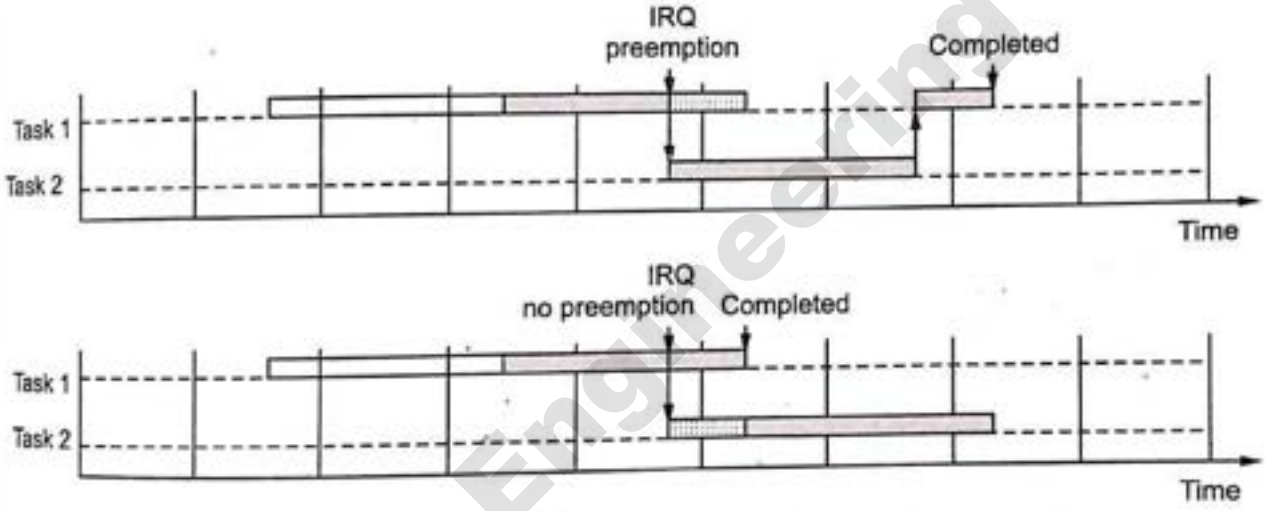


Fig. 2.9.1 Effect of preemption

1. Round-robin preemptive scheduler

- Real-time task would be added to the ready queue to await its next time slice.
- Any preemption means some delay in executing the task instance, a delay which the RTOS has to take care of as it has to guarantee respecting the deadline.
- Fig. 2.9.2 shows round robin preemptive scheduler.

Fig. 2.9.2

- All real-time scheduling algorithms strictly rely on priorities. A real time process may divide its instructions into separate tasks, each of which must complete by a deadline.

2.9.3 Rate Monotonic Priority Assignment

- Rate Monotonic Priority Assignment (RM) is a so called static priority round robin scheduling algorithm.
- In this algorithm, priority is increases with the rate at which a process must be scheduled. The process of lowest period will get the highest priority.
- In RM algorithms, the assigned priority is never modified during runtime of the system. RM assigns priorities simply in accordance with its periods, i.e. the priority is as higher as shorter is the period which means as higher is the activation rate. So RM is a scheduling algorithm for periodic task sets.

Advantages :

1. Simple to understand.
2. Easy to implement.
3. Stable algorithm.

Disadvantages :

1. Lower CPU utilization.
2. Only deal with independent tasks.
3. Non-precise schedulability analysis

Rate monitoring example

Tasks (T)	Period (P)	CPU burst (C)
T_1	2	1
T_2	3	1.01

$$\text{CPU Utilization} = \frac{C_1}{P_1} + \frac{C_2}{P_2} = \frac{1}{2} + \frac{1.01}{3}$$

$$= 0.5 + 0.3366 = 0.8366$$

$$U = 83.66\ \%$$

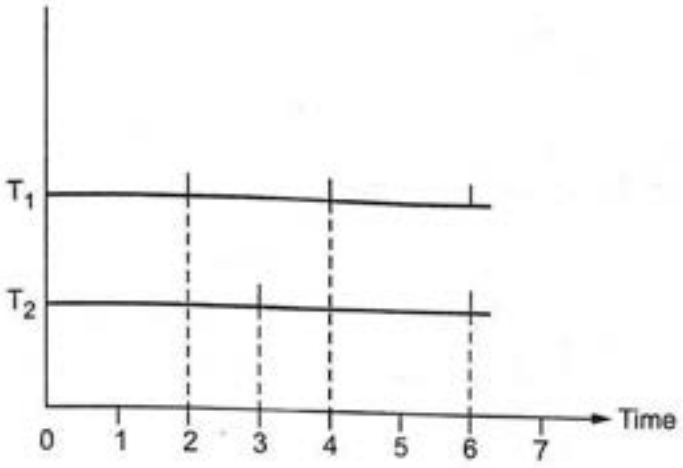


Fig. 2.9.3 (a)

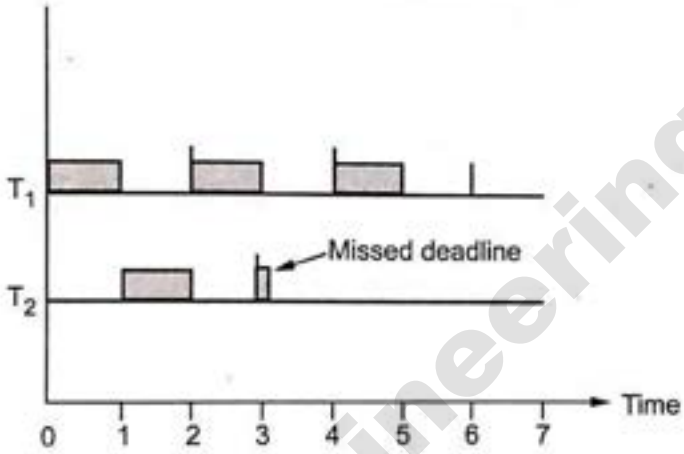


Fig. 2.9.3 (b)

2.9.4 Earliest Deadline First Scheduling Algorithm

- Earliest Deadline First (EDF) is one of the best known algorithms for real time processing. It is an optimal dynamic algorithm. In dynamic priority algorithms, the priority of a task can change during its execution It produces a valid schedule whenever one exists.
- EDF is a preemptive scheduling algorithm that dispatches the process with the earliest deadline. If an arriving process has an earlier deadline than the running process, the system preempts the running process and dispatches the arriving process.
- A task with a shorter deadline has a higher priority. It executes a job with the earliest deadline. Tasks cannot be scheduled by rate monotonic algorithm.
- EDF is optimal among all scheduling algorithms not keeping the processor idle at certain times. Upper bound of process utilization is 100%.

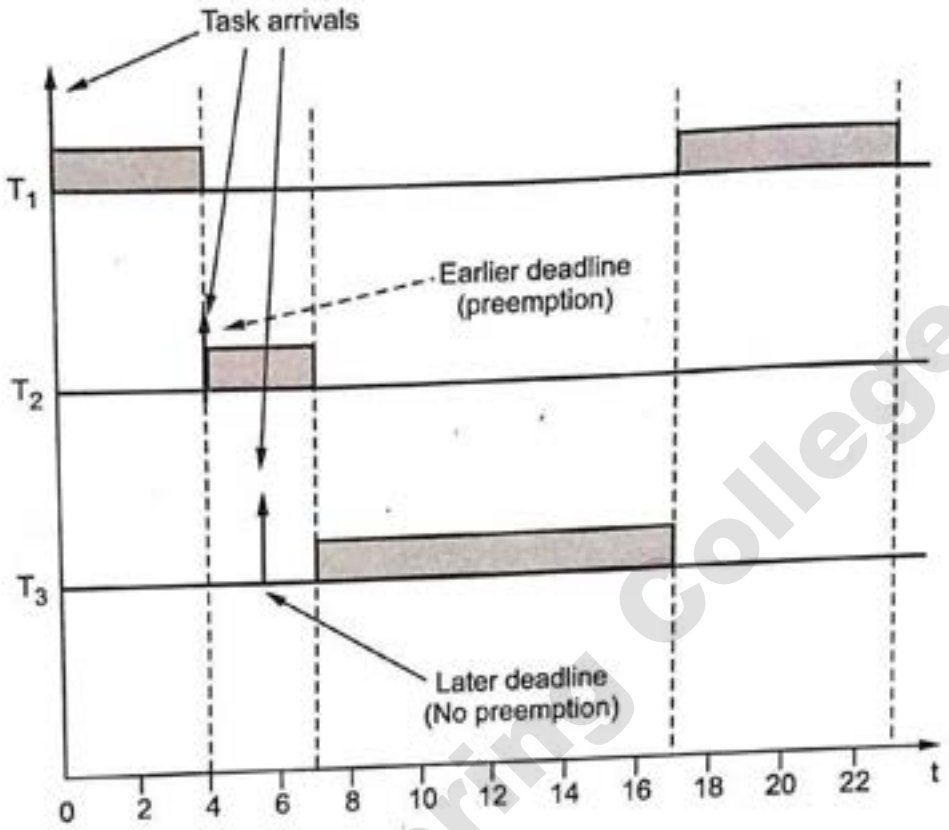


Fig. 2.9.4

- Whenever a new task arrive, sort the ready queue so that the task closest to the end of its period assigned the highest priority. System preempt the running task if it is not placed in the first of the queue in the last sorting.
- If two tasks have the same absolute deadlines, chose one of the two at random (ties can be broken arbitrarily). The priority is dynamic since it changes for different jobs of the same task.
- EDF can also be applied to aperiodic task sets. Its optimality guarantees that the maximal lateness is minimized when EDF is applied.
- Many real time systems do not provide hardware preemption, so other algorithm must be employed.
- In scheduling theory, a real-time system comprises a set of real-time tasks; each task consists of an infinite or finite stream of jobs. The task set can be scheduled by a number of policies including fixed priority or dynamic priority algorithms.
- The success of a real-time system depends on whether all the jobs of all the tasks can be guaranteed to complete their executions before their deadlines. If, they can, then we say the task set is **schedulable.**

- The schedulability condition is that the total utilization of the task set must be less than or equal to 1.

Advantages

1. It is optimal algorithm.
2. Periodic, aperiodic and sporadic tasks are scheduled using EDF algorithm.
3. Gives best CPU utilization

Disadvantages

1. Needs priority queue for storing deadlines
2. Needs dynamic priorities
3. Typically no OS support
4. Behaves badly under overload
5. Difficult to implement.

2.9.5 Comparison between RMS and EDF

Parameters	RMS	EDF
Priorities	Static	Dynamic
Works with OS with fixed priorities	Yes	No
Uses full computational power of processor	No	Yes
Possible to exploit full computational power of processor without provisioning for slack	No	Yes

Example 2.9.1 *Schedulability test for EDF.*

Task	Period	CPU Burst
T_1	9	6
T_2	15	5
T_3	5	1

Test if the given task set is schedulable with EDF.

Solution :

$$\text{CPU utilization (U)} = \frac{C_1}{P_1} + \frac{C_2}{P_2} + \frac{C_3}{P_3}$$

$$= \frac{6}{9} + \frac{5}{15} + \frac{1}{5}$$

$$= 0.6666 + 0.3333 + 0.2$$

$$U = 1.1999$$

Utilization (1.1999) is greater than 1. So test fails. EDF schedule does not meet all deadlines.

2.10 Thread

AU : Dec.-16, 17

- Thread is a dispatchable unit of work. It consists of thread ID, program counter, stack and register set. Thread is also called a Light Weight Process (LWP). Because they take fewer resources then a process. A thread is easy to create and destroy.
- It shares many attributes of a process. Threads are scheduled on a processor. Each thread can execute a set of instructions independent of other threads and processes. Fig. 2.10.1 shows a thread.

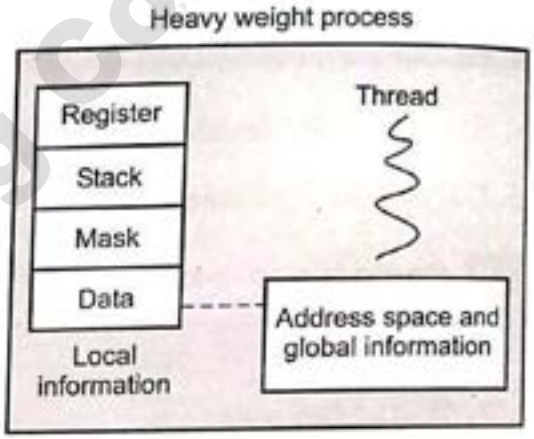


Fig. 2.10.1 Thread

- Every program has at least one thread. Programs without multithreading executes sequentially. That is, after executing one instruction the next instruction in sequence is executed.
- Thread is a basic processing unit to which an operating system allocates processor time and more than one thread can be executing code inside a process.
- When user double click on an icon by using mouse, the operating system creates a process and that process has one thread that runs the icon's code.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
- Normally, an operating system will have a separate thread for each different activity. The OS will have a separate thread for each process and that thread will perform OS activities on behalf of the process. In this condition, each user process is backed by a kernel thread. When process issues a system call to read a file, the process's thread will take over, find out which disk accesses to generate, and issue

the low level instructions required to start the transfer. It then suspends until the disk finishes reading in the data.

- When process starts up a remote TCP connection, its thread handles the low-level details of sending out network packets.
- In Remote Procedure Call (RPC), servers are multithreaded. Server receives the messages using a separate thread.
- Different operating system uses threads in different ways.
  1. Free memory is managed by kernel thread in LINUX OS.
  2. Solaries uses a kernel thread for interrupt handling.

2.10.1 Thread Advantages

1. Context switching time is minimized.
2. Thread support for efficient communication.
3. Resource sharing is possible using threads.
4. A thread provides concurrency within a process.
5. It is more economical to create and context switch threads.

2.10.2 Difference between Thread and Process

Sr. No.	Thread	Process
1.	Thread is also called lightweight process.	Process is also called heavyweight process.
2.	Operating system is not required for thread switching.	Operating system interface is required for process switching.
3.	One thread can read, write or even completely clean another threads stack.	Each process operates independently of the other process.
4.	All threads can share same set of open files and child processes.	In multiple processing, each process executes the same code but has its own memory and file resources.
5.	If one thread is blocked and waiting then second thread in the same task can run.	If one server process is blocked then other server process cannot execute until the first process is unblocked.
6.	Uses fewer resources.	Uses more resources.

2.10.3 Thread Lifecycle

- Fig. 2.10.2 shows a thread lifecycle. A thread has one of the following states.

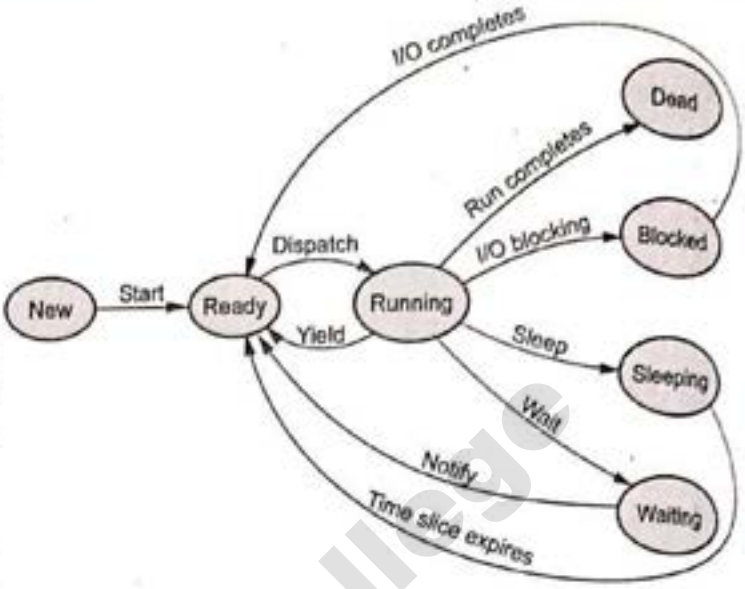


Fig. 2.10.2 Thread lifecycle

1. **New :** Thread is just created.
2. **Ready :** Thread's start( ) method invoked and now it is executing. OS put thread into Ready queue.
3. **Running :** Highest priority ready thread enters the running state. Thread is assigned a processor and now is running.
4. **Blocked :** This is the state when a thread is waiting for a lock to access an object.
5. **Waiting :** Here thread is waiting indefinitely for another thread to perform an action.
6. **Sleeping :** Thread sleep for a specified time of period. When sleeping time expires, it enters to ready state. CPU is not used by sleeping thread.
7. **Dead :** When thread completes task or operation.

2.10.4 Thread Programming and Libraries

- The subroutines which comprise the Pthreads API can be informally grouped into four major groups :

1. **Thread management :** Routines that work directly on threads - creating, detaching, joining, etc. They also include functions to set/query thread attributes (joinable, scheduling etc.)
2. **Mutexes :** Routines that deal with synchronization, called a "mutex", which is an abbreviation for "mutual exclusion". Mutex functions provide for creating, destroying, locking and unlocking mutexes.
3. **Condition variables :** Routines that address communications between threads that share a mutex. Based upon programmer specified conditions. This group includes functions to create, destroy, wait and signal based upon specified variable values. Functions to set/query condition variable attributes are also included.
4. **Synchronization :** Routines that manage read/write locks and barriers.

Basic thread functions include creation and termination of threads. There are five such thread function.

2.10.4.1 pthread_create Function

- A program is started by exec, a single thread is created known as initial thread or main thread. Additional threads are created by pthread_create

```
#include<pthread.h>
int pthread_create (pthread_t *tid, const pthread_attr_t *attr,
                    void *(*func) (void*), void* arg);
```

- Each thread is identified by a thread ID, of which data type is pthread_t. Each thread has numerous attributes, its priority, its initial stack size. The thread starts by calling this function and then terminates explicitely or implicitly. The address of function is specified as **func** argument which is called with a single pointer argument, arg.

2.10.4.2 pthread_join Function

- A thread can be terminated by calling **pthread_join.** In Unix process pthread_create is similar to **fork** and pthread_join is similar to **wait pid.**

```
#include<pthread.h>
int pthread_ join (pthread_t tid, void **status);
```

- The tid is specified as the wait. But there is no way to wait for any of the threads.

2.10.4.3 pthread_self Function

- Each thread has an ID which identifies it within a given process. This thread ID is returned by **pthread_create** and is used by **pthread_join.**
- A thread fetches this value for itself by using ptheread_self.

```
#include<pthread.h>
pthread_t pthread_self (void);
```

2.10.4.4 pthread_detach Function

- A thread is joinable or detachable. When joinable thread terminates, its thread ID and exit status are retained until another thread calls pthread_join.
- A detached thread is like a daemon process i.e. when it is terminated all its resources are released. The pthread_detach function changes the specified thread so that it is detached.

```
#include<pthread.h>
int pthread_detach (pthread_t tid);
```

2.10.4.5 pthread_exit Function

- A way for a thread to terminate is to call **pthread_exit.**

```
#include<pthread.h>
    void pthread_exit (void *status);
```

- But if thread is not detached, its thread ID and exit status are retained for a later pthread_join by any other thread in calling processes.

2.10.4.6 pthread_sigmask, pthread_kill

The prototype of the *pthread_sigmask* and *pthread_kill* functions are :

```
#include <signal.h>
#include <pthread.h>
int pthread_sigmask (int mode, sigset_t *sigsetp, sigset_t *oldstp);
int pthread_kill (pthread_t tid, int signum);
```

- The *pthread_sigmask* function sets the signal mask of a calling thread. The *sigsetp* argument contains one or more of the signal numbers to be applied to the calling thread.
- The mode argument specifies how the signal specified in the *sigsetp* argument is to be used. Possible value of mode argument is declared in <signal.h> header and their meanings are :

Mode value	Meaning
SIG_BLOCK	Adds signals contained in the *sigsetp* argument to the thread signal mask.
SIG_UNBLOCK	Removes signals contained in the *sigsetp* argument from the thread signal mask.
SIG_SETMASK	Replace the thread signal mask with the signal specified in the *sigsetp* argument.

- The *pthread_kill* function sends a signal, as specified in the *signum* argument, to a thread whose ID is given by the *tid* argument. The sending and receiving threads must be in the same process.
- If the default action for a signal is to terminate the process, then sending the signal to a thread will still kill the entire process.

2.10.4.7 sched_yield

- The function prototype of the *sched_yield* API is :

```
#include <pthread.h>
int sched_yield (void);
```

- The *sched_yield* function is called by a thread to yield its execution to other threads with the same priority. This function returns 0 on success and –1 when fails.

University Question

1. *What are the different thread libraries used ? Explain any one with example.* **AU : Dec.-16, Marks 8**
2. *Write the difference between user thread and kernel thread.* **AU : Dec.-17, Marks 5**

2.11 Thread Models

- Threads are of two types :
  1. User level thread 2. Kernel level thread
- All modern operating system support threading model. Implementation of thread will change according to the operating system.

2.11.1 User Level Thread

- User level thread uses user space for thread scheduling. These threads are **transparent** to the operating system. User level threads are created by runtime libraries that cannot execute privileged instructions.
- User-level threads have low overhead but it can achieve high performance in computation. User-level threads are managed entirely by the run-time system.
- User-level threads are small and faster. A thread is simply represented by a PC, registers, stack and small thread control block. Fig. 2.11.1 shows user level thread.
- The code for creation and destroying thread, message passing and data transfer, thread scheduling is included into thread library. Kernel is unaware of user level thread.
- User level threads do not invoke the Kernel for scheduling decision.

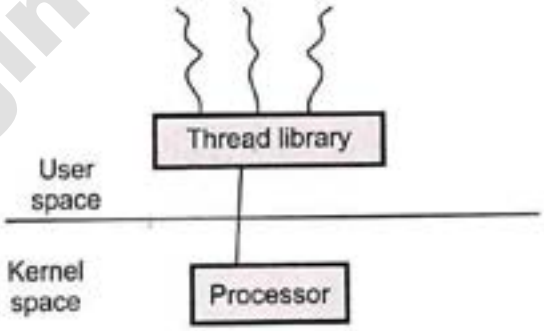


Fig. 2.11.1 User level thread

- User level thread are also called **many to one mapping** thread because the operating system maps all threads in a multithreaded process to a single execution context. The operating system considers as each multithreaded processes as a single execution unit.
- Example : POSIX Pthreads and Mach C-threads.

Advantages :

1. Kernel mode privilege does not require for thread switching.
2. These threads are fast to create and manage.
3. User level thread works even if the OS does not support threads.
4. User level threads are more portable.
5. Threading library controls flow of thread.

Disadvantages :

1. If thread blocks, the Kernel may block the all threads.
2. Not suitable for multiprocessor system.
3. User level threads also do not support system wide scheduling priority.

2.11.2 Kernel Level Thread

- In Kernel level thread, thread management is done by Kernel. Operating system support the Kernel level thread.
- Since Kernel managing threads, Kernel can schedule another thread if a given thread blocks rather than blocking the entire processes. Fig. 2.11.2 shows Kernel level thread.
- Kernel level thread support one to one thread mapping. This mapping requires each user thread with kernel thread. Operating system performs this mapping.

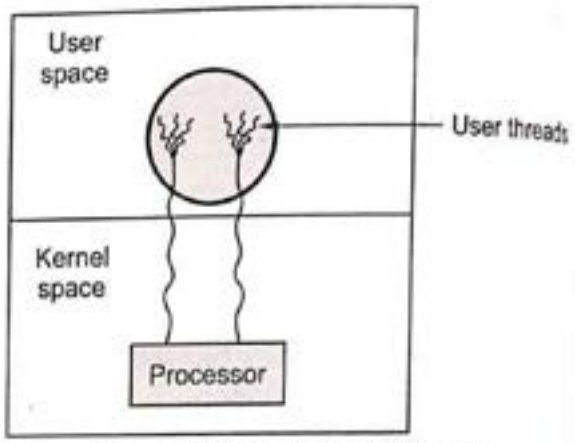


Fig. 2.11.2 Kernel level thread

- Threads are constructed and controlled by system calls. The system knows the state of each thread.
- Thread management code is not included in the application code. It is only API to the Kernel thread. Windows operating system uses this facility.
- Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- Kernel performs scheduling on a thread basis. The Kernel support for scheduling and management, thread creation only in Kernel space.
- Kernel level threads are slower than user level threads.
- Example : Windows 95/98/NT, Sun Solaris and Digital UNIX.

Advantages :

1. Each thread can be treated separately.
2. A thread blocking in the Kernel does not block all other threads in the same process.
3. Kernel routines itself as multithreaded.

Disadvantages :

1. Slower than the user level thread.
2. There will be overhead and increased in Kernel complexity.

2.11.3 Difference between User Level and Kernel Level Thread

Sr. No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads are slower to create and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to Kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.
4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called Kernel level threads.
5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.
6.	Example : POSIX Pthreads and Mach C-threads	Example : Windows 95/98/NT, Sun Solaris and Digital UNIX
7.	User level threads are also called many to one mapping thread.	Kernel level thread support one to one thread mapping.

2.12 Multithreading Models

AU : May-15,16

- Operating system uses user level thread and kernel level thread. User level threads are managed without kernel support. Operating system support and manage the kernel level threads.
- Modern operating system support kernel level threads. Kernel performs multiple simultaneous tasks in operating system. In most of the application, user threads are mapped with kernel level threads.
- Different methods of mapping is used in operating system are as follows :
  1. One to one 2. Many to one 3. Many to many

1. One to one

- In this model, one user level thread maps with one kernel level thread. If there are three user threads then system creates three separate kernel thread. This model provides more concurrency because of separate thread. Fig. 2.12.1 shows one to one mapping.
- In this method, the operating system allocates data structure that represents kernel threads. Here multiple threads are run in parallel on multiprocessor system.
- Number of threads in the system increases, the amount of memory required is also increases.

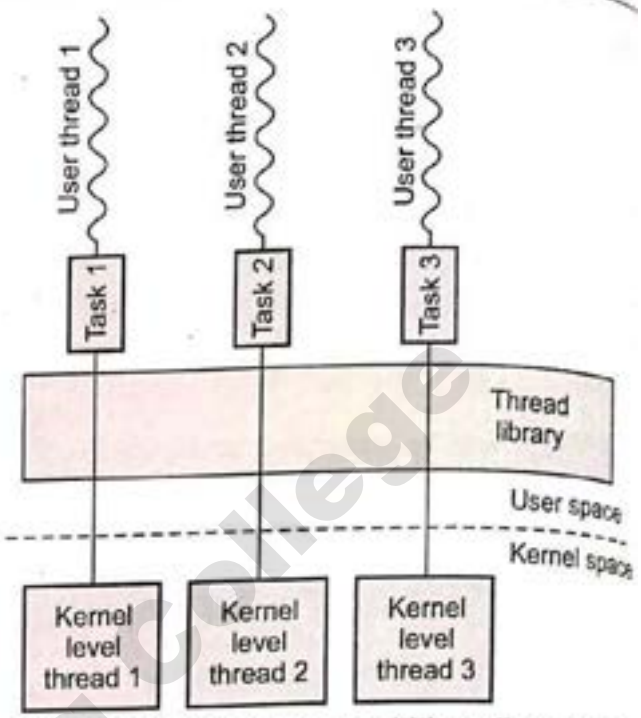


Fig. 2.12.1 One to one multithreading model

- Windows 95/XP and Linux operating system uses this one to one thread mapping.
- Only overhead in this method is creation of kernel level thread for user level thread. Because of this overhead, system performance is slowdown.

2. Many to one

- Many to one mapping means many user thread maps with one kernel thread. Fig. 2.12.2 shows many to one mapping.
- Operating system blocks the entire multi-threaded process when a single thread blocks because the entire multi-threaded process is a single thread of control. So when operating system receive any a blocking I/O request, it blocks the entire process.
- Thread library handle thread management in user space. The many-to-one model does not allow individual processes to be split across multiple CPUs.

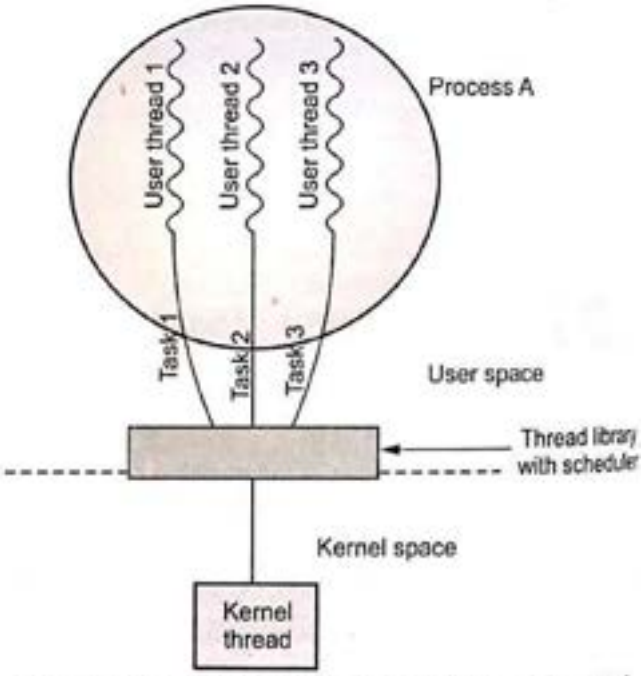


Fig. 2.12.2 Many to one multithreading model

- This type of relationship provides an effective context-switching environment, easily implementable even on simple kernels with no thread support.
- Green threads for Solaris and GNU portable threads implement the many-to-one model in the past, but few systems continue to do so today.
- **Advantage** : System performance is improved by customizing the thread library scheduling algorithm.

3. Many to many

- In the many to many mapping, many user-level threads maps with equal number of kernel threads. Fig. 2.12.3 shows many to many thread mapping.

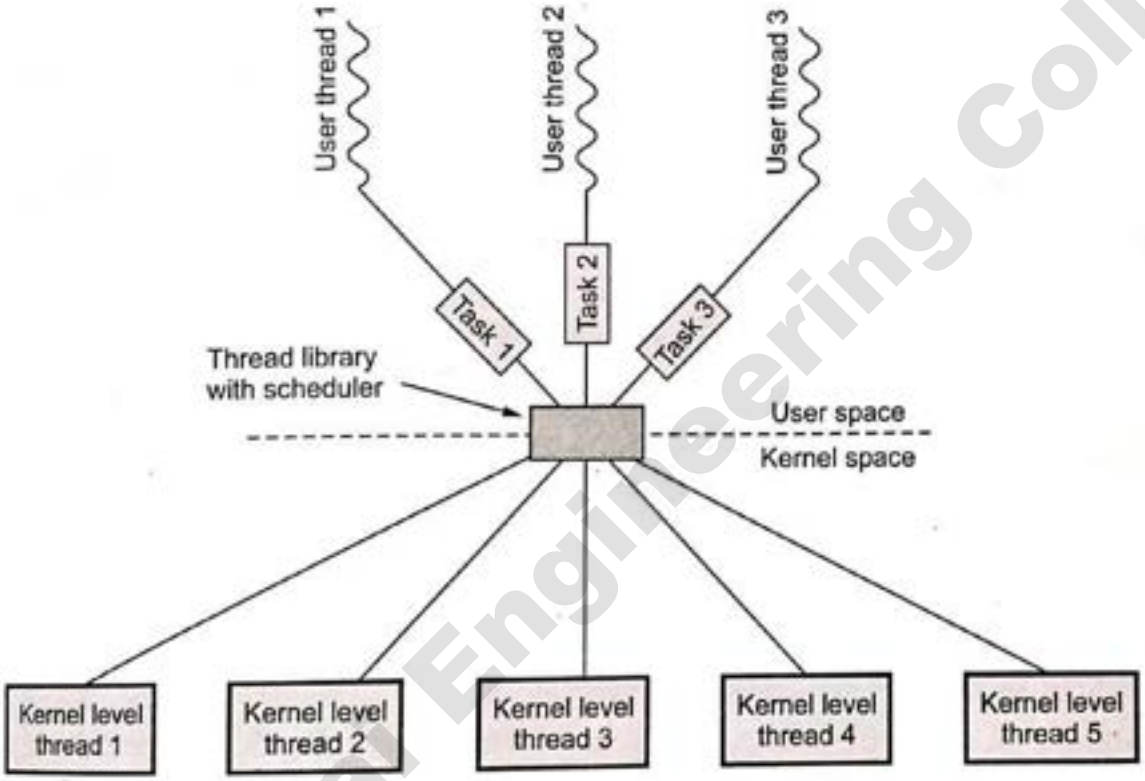


Fig. 2.12.3 Many to many multithreading model

- Many to many mapping is also called M-to-N thread mapping. Thread pooling is used to implement this method.
- Users can create required number of thread and there is no limitation for user. Again here blocking Kernel system calls do not block the entire process.
- This model support for splitting processes across multiple processors.
- **Limitations** : Operating system design becomes complicated.

Example 2.12.1 *Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.* **AU : May-16, Marks 8**

Solution :

a. A Web server that services each request in a separate thread.

b. A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel.

c. An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

University Questions

1. *Discuss about the issues to be considered with multithreaded programs.* **AU : May-15, Marks 8**

2. *Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.* **AU : May-16, Marks 8**

2.13 Threading Issues

- Some other issue related to the thread is discussed in this section. The issue include fork and exec system call, thread cancellation, signal handling, thread pool etc.

2.13.1 The fork ( ) and exec ( ) system call

- In a multithreaded programming environment, fork and exec system calls are changed. UNIX operating system uses two versions of fork system calls.
  1. Fork call duplicates all threads.
  2. In second option, only thread created by fork duplicates.
- Based on the application, system uses fork system calls. Sometimes, duplicating all the threads are unnecessary. If we call exec immediately after fork then there is no use of duplicating threads.
- Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell.
- This is where the exec system call comes into play. An exec will replace the contents of the currently running process with the information from a program binary. Thus the process the shell follows when launching a new program is to firstly fork, creating a new process, and then exec the program binary it is supposed to run.

2.13.2 Thread Cancellation

- Cancellation allows one thread to terminate another. One reason to cancel a thread is to save system resources such as CPU time. When your program determines that the thread's activity is no longer necessary then thread is terminated.
- Thread cancellation is a task of terminating a thread before it has completed.
- The thread cancellation mechanism allows a thread to terminate the execution of any other thread in the process in a controlled manner. Each thread maintains its own cancelability state. Cancellation may only occur at cancellation points or when the thread is asynchronously cancelable.
- The target thread can keep cancellation requests pending and can perform application-specific cleanup when it acts upon the cancellation notice.
- A thread's initial cancelability state is enabled. Cancelability state determines whether a thread can receive a cancellation request. If the cancelability state is disabled, the thread does not receive any cancellation requests.
- Target thread cancellation occurs in two different situations:
  1. Asynchronous cancellation
  2. Deferred cancellation
- **Asynchronous cancellation :** Target thread is immediately terminated. With the help of another thread, target thread is cancelled. When a thread holds no locks and has no resources allocated, asynchronous cancellation is a valid option.
- **Deferred cancellation :** When a thread has enabled cancellation and is using deferred cancellation, time can elapse between the time it's asked to cancel itself and the time it's actually terminated.

2.13.3 Signal Handling

- Signal is used to notify a process that a particular event has occurred. Signal may be synchronous or asynchronous. All types of signals are based on the following pattern :
  1. At a specific event, signal is generated.
  2. Generated signal is sent to a process/thread.
  3. Signal handler is used for handling the delivered signal.

- **Synchronous signals :** An illegal memory access, division by zero is the example of synchronous signals. These signals are delivered to the same process which performed the operation for generating the signal.
- **Asynchronous signals :** Terminating a process with certain keystrokes are signals that are generated by an event external to the running process.
- Synchronous signal is sent to same process whether as asynchronous signal is sent to another process.
- Signals may be handled in different ways :
  1. Some signals may be ignored. For example changing the windows size.
  2. Other signals may be handled by terminating the program. For example illegal access of memory.
- Delivery of signals in multithreaded programs is more complex than single thread.
- Following are the condition where/ how should the signals be delivered to threads/process :
  a. Thread applies the signal are received the signal.
  b. Every thread in the process received the signal.
  c. Deliver the signal to limited threads in the process
  d. All the signals are received to particular thread for the process.
- The method for delivering a signal depends on the type of signals generated.
  1. Synchronous signals is sent to the thread which causes the signal.
  2. All the thread received asynchronous signals.

2.13.4 Thread Pool

- A thread pool offers a solution to both the problem of thread life-cycle overhead and the problem of resource thrashing. By reusing threads for multiple tasks, the thread-creation overhead is spread over many tasks.
- Thread pools group CPU resources, and contain threads used to execute tasks associated with that thread pool. Threads host engines that execute user tasks, run specific jobs such as signal handling and process requests from a work queue.
- Multithreaded server has potential problems and these problems are solved by using thread pool. Problems with multithreaded server :
  1. Time for creating thread

2. Time for discarding thread

3. Excess use of system resources.

Advantages of thread pools :

1. Servicing a request with an existing thread is usually faster than waiting to create a thread.

2. Thread pool size is fixed.

2.14 Multi-core Programming

AU : May-17

- The era of programming with single processors has ended.
- A **multi-core processor** is a processing system composed of two or more independent cores or CPUs. The cores are typically integrated onto a single integrated circuit die or they may be integrated onto multiple dies in a single chip package.
- Two independent microprocessors are called dual-core processor.
- Multiprocessor is any computer with several processors. Multi-core processor is a special kind of a multiprocessor where all processors are on the same chip. Multi-core processors are MIMD.
- In single physical package, multi-core implements a multiprocessing system. CPU or cores are may be loosely coupled or tightly coupled. Cores in the multi-core processor share cache memory in some cases and sometimes cache memory is not shared. Fig. 2.14.1 shows shared memory multi-core.

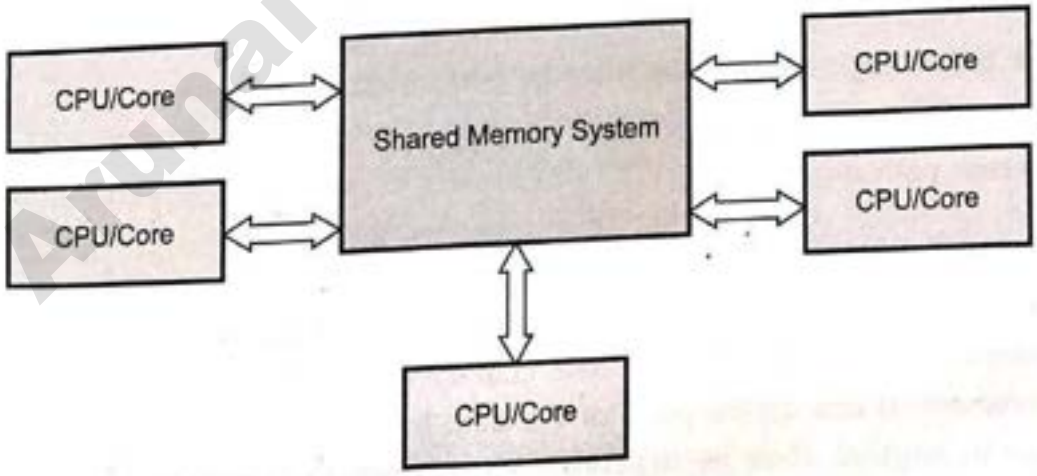


Fig. 2.14.1 Shared memory with core

- IPC is implemented by using message passing method or shared memory method. All cores are identical in symmetric multi-core systems and they are not identical in asymmetric multi-core systems.

Why multi-core ?

1. Difficult to make single-core clock frequencies even higher.
2. Deeply pipelined circuits face heat problems and speed of light problems.
3. Many new applications are multithreaded.
4. General trend in computer architecture shift towards more parallelism.

Programming for multi-core

1. Process or thread is used by programmer.
2. Use parallel algorithm.
3. Mapping of thread to cores or process to core done by operating system.
4. Spread the workload across multiple cores.

Programming challenges

- Following parameters are considered while doing programming on multi-core.

1. **Task identification :** One task is not depends upon other task. Both the task runs independently and in parallel. For this purposes, operating system finds application for running task in parallel. Here the given application is divided into number of task.
2. **Splitting of data :** Individual tasks are accessed data and operation on data is performed on separate core.
3. **Dependency of data :** It is necessary to check data dependency between two tasks. If there is any dependency exit, operating system must synchronize its operations.
4. **Task balancing :** All the task must be performed equal work.
5. **Testing and debugging :** These methods are difficult to implement because of different path used by task.

Types of parallelism

1. **Data parallelism :** The same types of task run on different data in parallel. Example: to convert all lower case characters in an array to upper-case. For this conversion, it can divide parts of the data between different tasks and perform the tasks in parallel. Here no dependencies between the tasks those cause their results to be ordered.

2. **Task parallelism :** Different tasks running on the same data. Task parallelism is also known as functional parallelism. Example : To perform several functions on the same data, i.e. average, minimum, binary etc. Here again no dependencies between the tasks, so all task can run in parallel.

University Question

1. *Explain the concept of multiprocessor and Multicore organization.* **AU : May-17, Marks 7**

2.15 Process Synchronization

AU : Dec.-12,16, May-13,16,17

- Logical control flows are concurrent if they overlap in time. This general phenomenon is known as **concurrency**. Concurrency refers to any form of interaction among processes or threads. The familiar examples are hardware exception handlers, processes and UNIX signal handlers.
- Modern operating systems can handle more than one process at a time. System scheduler manages processes and their competition for the CPU. The role of memory manager is to manage sharing of main memory between active processes in the system. Look at how processes coexist and communicate with each other on modern computer.
- Concurrency is good for users because working on the same problem, simultaneous execution of programs, background execution.
- Concurrency creates problems also because of access to shared data structures, deadlock due to resource contention and enabling process interaction.
- Applications that use application-level concurrency are known as concurrent programs. Modern operating systems provide three basic approaches for building concurrent programs : *Processes, I/O multiplexing and threads.*
- **Concurrency** means that two or more calculations happen within the same time frame and there is usually some sort of dependency between them. **Parallelism** means that two or more calculations happen simultaneously.
- But concurrency describes a problem, while parallelism describes a solution. Parallelism is one way to implement concurrency, but it is not the only one. Another popular solution is interleaved processing.

2.15.1 Principle of Concurrency

- Concurrent access to shared data may result in data inconsistency. Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

- Concurrency arises in the same way at different levels of execution streams. Following are the example of concurrency in different types of operating systems :
  1. **Concurrency in multiprogramming :** An interaction between multiple processes running on one CPU.
  2. **Concurrency in multithreading :** An interaction between multiple threads running in one process
  3. **Concurrency in multiprocessors :** An interaction between multiple CPUs running multiple processes or threads.
  4. **Concurrency in multi-computers :** An interaction between multiple computers running distributed processes or threads.
- Java is a concurrent programming language.
- Process synchronization is required in uni-processor system, multiprocessor system and network. If more than one thread exists in a system at the same time, then the threads are said to be concurrent. Synchronization problems can occur whenever two or more concurrent processes use any shared resource.
- Cooperating process share the logical address space.

Is concurrency is possible without parallelism ?

- **Yes, concurrency is possible without parallelism.**
- Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

2.15.2 Race Condition

- **Race condition** occurs when two or more operations occur in an undefined manner. When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called **race condition.**
- There is a "race condition" if the outcome depends on the order of the execution. The outcome depends on the CPU scheduling or "interleaving" of the threads.
- **Race condition** should be avoided because they can cause fine errors in applications and are difficult to debug.
- In banking system **balance** is a shared variable. After depositing the amount, it update by bank employee **balance = balance + amount**. At the same time, some amount is transfer then it becomes **balance = balance – amount.** These two tasks are in a race to write variable **balance**. In this the process that updates last determines the final value of balance.

- The concurrent execution of two processes is not guaranteed to be determinate, since different executions of the same program on the same data may not produce the same result.

Operating system concerns

- Design and management issue for concurrency are as follows :
  1. Track of various processes is kept by operating system.
  2. Operating systems allocates and deallocate software and hardware resources to active process.
  3. Operating system must protect user data and physical resources from un-authorized process.
  4. Process execution speed do not depends upon other process execution speed.

2.15.3 Critical Section Problem

- A critical section is a block of code that only one process at a time can execute; so, when one process is in its critical section, no other process may be in its critical section. The critical section problem is to ensure that **only one process at a time is allowed** to be operating in its critical section.
- Critical section means, process may change some common variable, writing files, updating memory location, updating a process table etc. When process is accessing shared modifiable data, it is said to be in a critical section.
- Each process takes permission from operating system to enter into the critical section. Structure of process is as follows :
  1. Entry section
  2. Remainder section
  3. Exit section
- **Entry section :** It is block of code executed in preparation for entering critical section.
- **Exit section :** The code executed upon leaving the critical section.
- **Remainder section :** Rest of the code is remainder section.
- Each process cycles through remainder, entry, critical, exit sections in this order.
- Consider the following example :

```
Int count = 5;

Process 1                Process 2
count = count +1;        count = count–1;
Output : count = ?       ( 6 or 5)
```

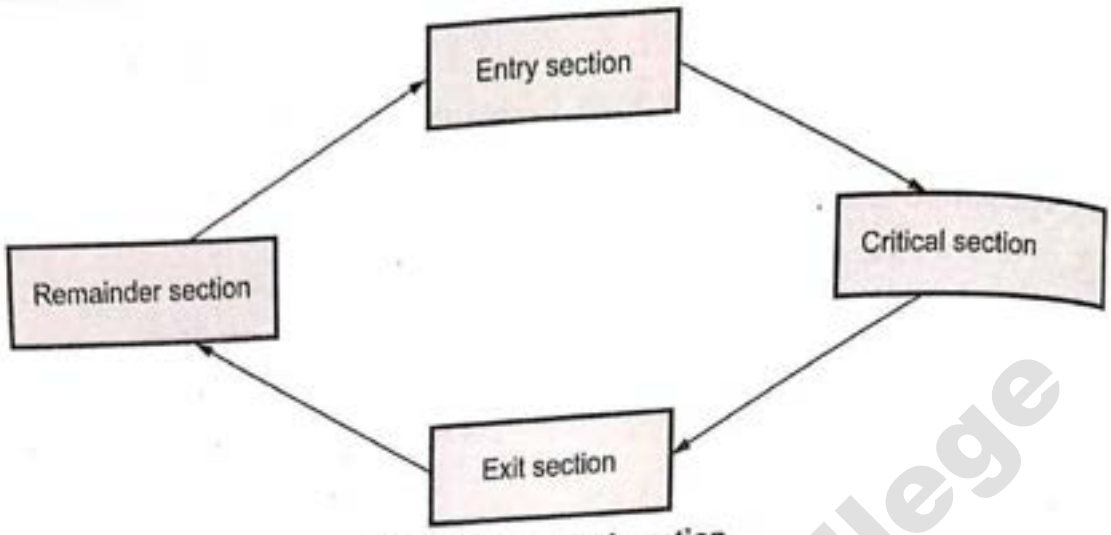


Fig. 2.15.1 Critical section

- **Problem :** Design a protocol to solve this problem.
- **General framework :** Every solution will have the following layout :
- To prevent critical section problem, the system should ensure that only one process at a time can execute the instruction in its critical section for a particular resource. If one process attempts to enter its critical section while another is in its own critical section, the first process should wait until the executing process exits the critical section.

Entry section
Critical section
Exit section
Remainder section

Fig. 2.15.2

- If a process in a critical section terminates by any reasons, then the operating system must release mutual exclusion so that other process may enter their critical sections.

Solution of critical section

- Solution to critical section problem must satisfy the mutual exclusion, progress and bounded waiting parameters.
- Software and hardware solution to critical section problem is available.
- Process using Kernel is active in the system at any given time. Race condition is with implementing a Kernel code. Kernel data structure maintains a list of all open files in the system. System always modifies this list depending upon the process open files and closed files.
- Operating system handles critical section problem by using Kernel. Kernel is classified as **preemptive Kernel and non-preemptive Kernel.**
- **Preemptive Kernel :** If process running in kernel mode then also it is preempted.
- **Non-preemptive Kernel :** It does not allow process to preempt.

2.15.4 Critical Region

- Writing set of data into a section of memory or reading it out, generally take some time and is performed through execution of a series of primitives operation.
- Critical region is an area of a process which is sensitive to inter-process complications. To guarantee mutual exclusion only one process is allowed to enter its critical region at a time.
- OS need a system to ensure process cannot enter critical region if another process is in it's.
- A process must be allowed to finish work on a critical part of the program before other processes can have access to it. It is called a critical region because it is a critical section and its execution must be handled as a unit. Other processes must wait before accessing critical region resources.
- Each conditional critical region is a "resource" that has a name and a set of variables that can only be accessed in that region. All variables can only be accessed within a **conditional critical region.**

2.15.5 Mutual Exclusion

- The need for mutual exclusion comes with concurrency. There are several kinds of concurrent execution :
  1. Interrupt handlers
  2. Interleaved preemptively scheduled processes/threads
  3. Multiprocessor clusters, with shared memory
  4. Distributed systems
- Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections
- Requirement of mutual exclusion is that, when process P1 is accessing a shared resource R1 then on other process should be able to access resource R1 until process P1 has finished its operation with resource R1.
- Examples of such resources include files, I/O devices such as printers and shared data structures.
- Approaches to implementing mutual exclusion :
  1. **Software method :** Leave the responsibility with the processes themselves. These methods are usually highly error-prone and carry high overheads.
  2. **Hardware method :** Special-purpose machine instructions are used for accessing shared resources. This method is faster but cannot provide complete

solution. Hardware solution cannot give guarantee the absence of deadlock and starvation

3. **Programming language method :** Provide support through the operating system or through the programming language.

Requirements of mutual exclusion

1. At any time, only one process is allowed to enter in its critical section.
2. Solution is implemented purely in software on a machine.
3. A process remains inside its critical section for a bounded time only.
4. No assumption can be made about relative speeds of asynchronous concurrent processes.
5. A process cannot prevent any other process for entering into critical section.
6. A process must not be indefinitely postponed from entering its critical section.

2.15.6 Lock

- Lock is a shared variable. Software mutual exclusion required that a process read a variable to determine that no other process is executing a critical section, then set a variable known as lock. It indicates that the process is executing its critical section.
- Initially lock variable is initialized with zero. The process set it to 1 and enters its critical section.

  Lock = 0 no process in the critical section

  Lock = 1 process in the critical section
- Using simple lock variable, process synchronization problem is not solved. To avoid this, spinlock is used. A lock that uses busy waiting is called a spin lock.

2.15.7 Mutex

- Mutex is used to ensure only one thread at a time can access the resource protected by the mutex. The process that locks the mutex must be the one that unlocks it. Mutex is good only for managing mutual exclusion to some shared resource.
- Mutex is easy and efficient for implement.
- Mutex can be in one of two states: locked or unlocked
- Mutex is represented by one bit. Zero (0) means unlock and any other value represents locked. It uses two procedures.

- When a process needs access to a critical section, it checks the condition of mutex_locks. If the mutex is currently unlocked then calling process enters into critical section.
- If the mutex is locked, the calling process is enters into blocked state and wait until the process in the critical section finishes its execution.
- Mutex variables have only two states so they are simple to implement. Their use is limited to guarding entries to critical regions.
- A segment of code in which a thread may be accessing some shared variable is called a **critical region.**
- Mutex variable is like a binary semaphore. But both are not same.

University Questions

1. *It is possible to have concurrency but not parallelism? Explain.* **AU : May-16, Marks 8**
2. *Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.* **AU : Dec.-16, Marks 8**
3. *What is a race condition ? Explain how a critical section avoids this condition. What are the properties which a data item should possess to implement a critical section ? Describe a solution to the Dining philosopher problem so that no races arise.* **AU : May-17, Marks 13**

2.16 Semaphores

AU : May-15, 17, Dec.-15, 17

- Semaphore is described by Dijkstra. Semaphore is a nonnegative integer variable that is used as a flag. Semaphore is an operating system abstract data type. It takes only integer value. Its used to solve critical section problem.
- Dijkstra introduced two operations (P and V) to operate on semaphore to solve process synchronization problem. A process calls the P operation when it wants to enter its critical section and calls V operation when it wants to exit its critical section. The P operation is called as wait operation and V operation is called as signal operation.
- A wait operation on a semaphore decease its value by one.

```
waits : while S < 0
            do loops;
        S:= S - 1;
```

- A signal operation increments its value:

```
signal:
    S:= S + 1;
```

- A proper semaphore implementation requires that P and V be indivisible operations. A semaphore operation is atomic. This may be possible by taking hardware support. The operation P and V are executed by the operating system in response to calls issued by any one process naming a semaphore as parameter.
- There is no guarantee that no two processes can execute wait and signal operations on the same semaphore at the same time.

2.16.1 Binary Semaphore

- Binary semaphore is also known as mutex locks. It deals with the critical section for multiple processes.
- Binary semaphore value is only between 0 and 1.

Counting semaphore

- Used with that resource which has a finite number of instances.
- It works over unrestricted domain.
- Nonbinary semaphore often referred to as either a counting semaphore or a general semaphore.
- Process waiting for semaphore is stored in the queue for binary and counting semaphore. Queue uses first in first out policy.
- A semaphore that does not specify the order in which processes are removed from the queue is called as **weak semaphore.** The process that has been blocked the longest is released from the queue first is called a **strong semaphore.**
- Semaphore can be implemented in user applications and in the kernel. Both operations can be implemented in the kernel by blocking waiting process to avoid busy waiting.

2.16.2 Busy Waiting

- Busy waiting is a situation in which a process is blocked on a resource but does not yield the processor. A busy wait keeps the CPU busy in executing a process even as the process does nothing.
- Busy waiting is also called as spin waiting.
- Busy waiting waste CPU cycles that some other process might be able to use productively.
- To avoid busy waiting, a process waiting for critical section, they put into the blocked state. When process allowed entering into critical section, then process is kept in ready state.
- To overcome the need for bust waiting :

In wait( ) operation :

a. When process reads semaphore value and it is not positive value then process blocked itself for some time.

b. Semaphore keeps all blocked process in the waiting queue so process is kept in this queue.

c. CPU scheduler selects another process for execution.

In signal ( ) operation :

a. The wakup ( ) operation is used for restating the blocked processes. When any other process executes signal ( ) operation, then blocked process state changed from blocked state to ready state.

b. This process is kept in the ready queue.

Properties of semaphore :

1. Semaphores are machine independent.
2. Semaphores are simple to implement.
3. Correctness is easy to determine.
4. Semaphore acquire many resources simultaneously.
5. Can have many different critical sections with different semaphores.

2.16.3 Drawbacks of Semaphore

1. They are essentially shared global variables.
2. Access to semaphores can come from anywhere in a program.
3. There is no control or guarantee of proper usage.
4. There is no linguistic connection between the semaphore and the data to which the semaphore controls access.
5. They serve two purposes, mutual exclusion and scheduling constraints.

2.16.4 Semaphore Programming

1. Semget ( )

- To create a semaphore or gain access to one that exists, the *semget* system call is used.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget (key_t key ,intnsems, int semflg);
```

- The semget system call takes three arguments :
  a. The first argument, *key*, is used by the system to generate a unique semaphore identifier.
  b. The second argument, *nsems*, is the number of semaphores in the set.
  c. The third argument, *semflg*, is used to specify access permission and/or special creation conditions.
- If the semget system call fails, it returns a – 1 and sets the value stored in errno.
- When a semaphore is first created, the kernel assigns to it an associated :
  a. Semaphore Control Block (SCB),
  b. A unique ID,
  c. A value (binary or a count), and
  d. A task - waiting list
- A Kernel can support many different types of semaphores, including binary, counting, and mutual-exclusion (mutex) semaphores.

2. semctl ( ) function

- The *semctl* system call allows the user to perform a variety of generalized control operations on the system semaphore structure, on the semaphores as a set, and on individual semaphores.
- The function prototype is as follows :

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, union semun arg);
```

- The *semctl* system call takes four arguments :
  1. The first argument, *semid* is a valid semaphore identifier that was returned by a previous *semget* system call.
  2. The second argument, *semnum*, is the number of semaphores in the semaphore set (array), 0 means this number (index) is not relevant.
  3. The third argument to *semctl*, *cmd*, is an integer command value. The *cmd* value directs *semctl* to take one of several control actions. Each action requires specific access permissions to the semaphore control structure.
  4. The fourth argument to *semctl*, *arg*, is a union of type *semun*. Given the action specified by the preceding *cmd* argument, the data in *arg* can be one of any of the following four values :
     a. An integer already was set in the val member of sem_union that used with SETVAL to indicate a change of specific value for a particular semaphore within the semaphore set.

b. A reference to a semid_ds structure where information is returned when IPC_STAT or IPC_SET is specified.

c. A reference to an array of type unsigned short integers; the array is used either to initialize the semaphore set or as a return location when specifying GETALL.

d. A reference to a seminfo structure when IPC_INFO is requested.

- If semctl fails, it returns a value of – 1 and sets errno to indicate the specific error.

3. semop ( ) function

- Additional operations on individual semaphores are accomplished by using the *semop* system call. Its syntax is :

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
```

- The semop system call takes three arguments :

a. The first argument, *semid*, is a valid semaphore identifier that was returned by a previous successful *semget* system call.

b. The second argument, sops, is a reference to the base address of an array of semaphore operations that will be performed on the semaphore set associated with by the *semid* value.

c. The third argument, nsops, is the number of elements in the array of semaphore operations.

- If *semop* fails, it returns a value of – 1 and sets errno to indicate the specific error.
- If a *semop* call must block after completing some of its component operations, the kernel *rewinds* the operation to the beginning to ensure atomicity of the entire call.
- When the *sem_op* value is negative, the process specifying the operation is attempting to decrement the semaphore. The decrement of the semaphore is used to record the acquisition of the resource affiliated with the semaphore.
- When a semaphore value is to be modified, the accessing process must have altered permission for the semaphore set.
- When the *sem_op* value is positive, the process is adding to the semaphore value. The addition is used to record the return (release) of the resource affiliated with the semaphore.
- Again, when a semaphore value is to be modified, the accessing process must have alter permission for the semaphore set.
- When the *sem_op* value is zero, the process is testing the semaphore to determine if it is at 0.

- When a semaphore is at 0, the testing process can assume that all the resources affiliated with the semaphore are currently allocated.

4. sem_post ( )

Syntax :

```
#include <semaphore.h>
int sem_post(sem_t *sem);
```

- The sem_post( ) function unlocks the specified semaphore by performing a semaphore unlock operation on that semaphore. When this operation results in a positive semaphore value, no threads were blocked waiting for the semaphore to be unlocked; the semaphore value is simply incremented.
- When this operation results in a semaphore value of zero, one of the threads waiting for the semaphore is allowed to return successfully from its invocation of the sem_wait( ) function. The sem_post( ) interface is reentrant with respect to signals and may be invoked from a signal-catching function.

5. sem_init ( )

Syntax :

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- The sem_init( ) function is used to initialize the semaphore's value. The pshared argument must be 0 for semaphores local to a process. The value argument specifies the initial value for the semaphore.
- The pshared argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.
- If pshared has the value 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads.
- If pshared is nonzero, then the semaphore is shared between processes, and should be located in a region of shared memory.

University Questions

1. *Show how wait ( ) and signal ( ) semaphore operations could be implemented in multiprocessor environments, using the test and set ( ) instruction. The solution should exhibit minimal busy waiting. Develop pseudocode for implementing the operations.*

AU : May-15, Marks 10

2. *Illustrate semaphores with neat example.*

AU : Dec.-15, Marks 8

3. What do you mean by term synchronization ? What is Semaphore ? Explain how semaphore can used as synchronization tool. Consider a coke machine that has 10 slots. The producer is the delivery person and the consumer is the student using the machine. It uses the following three semaphores :

semaphore mutex

semaphore full Buffer/ Number of filled slots */*

semaphore empty Buffer/ Number of empty slots */*

i) Write pseudo code for delivery person ( ) and student ( )

ii) What will be the initial values of the semaphores ?

iii) Write a solution that guarantees the mutual exclusion and has no deadlocks.

AU : May-17, Marks 15

Ans. :

```
* initialise */
  # define slots = 10
    semaphore mutex = 1
    semaphore fullBuffer = x
    semaphore emptyBuffer = y   //x + y must equal to 10//
       delivery_Person ()
       {
         wait (emptyBuffer);
         wait (mutex);
         - - - - - - -
         - - - - - - -
         put_one_coke_in_machine ();
            signal (mutex);
            signal (fullBuffer);
       }
       student ()
       {
       - - - - - - -
       - - - - - - -
       wait (fullBuffer);
       wait (mutex);
          take_one_coke_in_machine ();
            signal (mutex);
            signal (emptyBuffer);
       }
```

4. *Consider the atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x. Consider the following implementation of P and V functions on a binary semaphore*

```
void P (binary_semaphore*s) {
unsigned y;
unsigned*x = & (s->value)
do {
   fetch-and-set x, y;
} while (y);
}
void V (binary_semaphore*s) {
   S-> value = 0;
}
```

Write whether the implementation may or may not work if context switching is disabled in P.

AU : Dec.-17, Marks 4

Ans. : Implementation may not work if context switching is disabled in P. Let us talk about the operation P(). It stores the value of s in x, then it fetches the old value of x, stores it in y and sets x as 1. The while loop of a process will continue forever if some other process doesn't execute V() and sets the value of s as 0. If context switching is disabled in P, the while loop will run forever as no other process will be able to execute V().

2.17 Classical Problem of Synchronization

AU : Dec.-17

2.17.1 Dinning Philosopher Problem

- Dinning philosophers problem is one of classical process synchronization problem. Here five philosophers are seated around a circular table. They spend their lives in thinking and eating. Five plates with five forks are kept on the circular table.
- While philosophers think, they ignore the eating and do not require fork. When a philosopher decides to eat, then he/she must obtain a two fork, one from left side fork and another from right side fork. Philosophers can pick up only a single fork at one time.

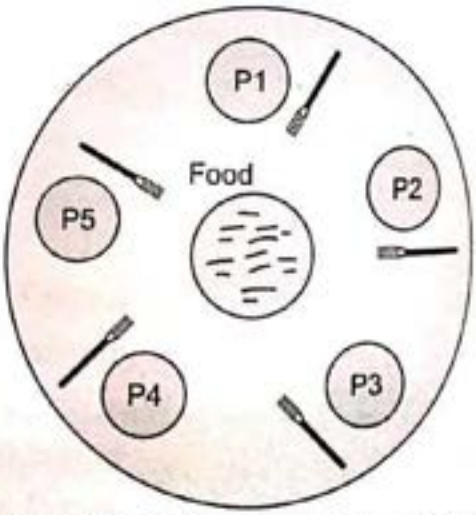


Fig. 2.17.1 Dinning philosophers

- After consuming a food, the philosophers replace the fork and resumes thinking. A philosophers to the left or right of a dinning philosopher cannot eat while the dinning philosophers is eating, since forks are a shared resource.
- Fig. 2.17.1 shows seating arrangement of five philosophers.
- Consider the following code :

```
void dinning_Philosopher ( )
    {
        while (true)
        {
          thinking ( );
          eating ( );
        }

    }
void eating ( )
{
    takeleftfork ( );
          takerightfork ( );
          eatfood(delay);
          putrighttfork ( );
    putleftfork ( );
}
```

- Here philosophers operates asynchronously and concurrently, it is possible for each philosophers to enter into eating mode. The procedure takeleftfork ( ) waits until the specified fork is available and then seizes it. This is not possible because of the following reasons.
- Each philosopher will hold exactly one fork, and no forks will remain available on the table. The philosophers will all deadlock.
- To solve this problem, write some extra code in takerightfork ( ) procedure. User can specify the philosophers to put down the left fork if that philosophers cannot obtain the right fork.

Problem analysis :

- **Shared resource :** Here each fork is shared by two philosophers so we called it is a shared resource.
- **Race condition :** we do not want a philosopher to pick up a fork that has already been picked up by his neighbor. This is a race condition.
- **Solution :** we consider each fork as a shared item and protected by a mutex lock. Before eating, each philosopher first lock left fork and then right fork. If philosopher acquires both locks successful, then philosophers have two locks (two

fork) so he can eat a food. After finishes easting, this philosopher releases both fork, and thinks.

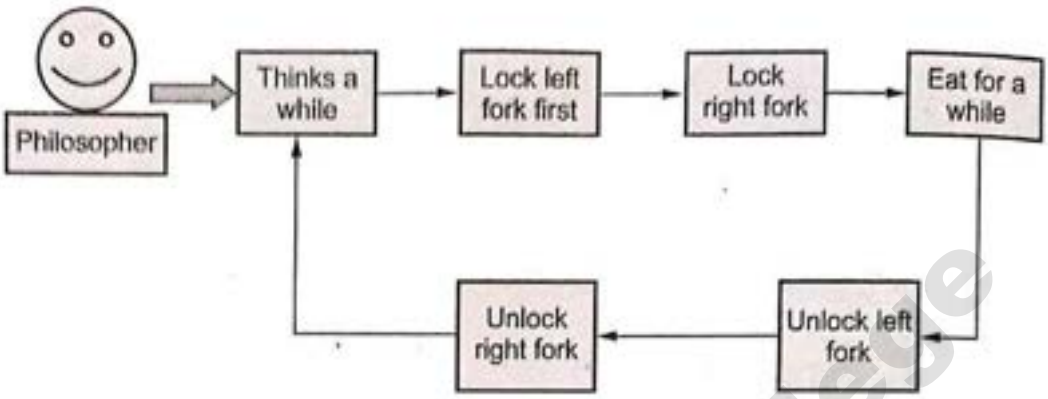


Fig. 2.17.2 Philosopher states

Some other alternatives :

1. Pick up the left fork, if the right fork is not available for a given time, put the left fork down, wait and try again. Even if each philosopher waits a different random time, an unlucky philosopher may starve.
2. Require all philosophers to acquire a binary semaphore before picking up any forks. This guarantees that no philosopher starves but limits parallelism dramatically.

- Because we need to lock and unlock a chopstick, each chopstick is associated with a mutex lock. Since we have five philosophers who think and eat simultaneously, we need to create five threads, one for each philosopher. Since each philosopher must have access to the two mutex locks that are associated with its left and right chopsticks, these mutex locks are global variables.

2.17.2 Reader-Writer Problem

- When two types of processes need to access a shared resource such as a file or database. They called these processes reader and writers.
- For example in railway reservation system, readers are those who want train schedule information. They are called reader because readers do not change the content of the database. Many readers may access the database at once. There is no need to enforce mutual exclusion among readers.
- The writers are those who making reservations on a particular train. Writer can modify the database, so it must have exclusive access. When writer is active, no other readers or writers may be active. Therefore, it must enforce mutual exclusion if there are groups of readers and a writer.
- Reader writer problem is similar to one which a file is to be shared among a set of processes. If a process want only to read the file, then it may share the file with any other process that also wants to read the file. If the writer wants to append

the file, then no other process should have access to the file when the writer has access to it.

- If reader having higher priority than the writer, then there will be starvation with writers. For writer having higher priority than reader then starvation with readers.

2.17.3 The Producer Consumer Problem

- Producer - consumer problem is example classic problems of synchronization. Producer process produce data item that consumer process consumes later.
- Buffer is used between producer and consumer. Buffer size may be fixed or variable. The producer portion of the application generates data and stores it in a buffer, and the consumer reads data from the buffer. Fig. 2.17.3 shows producer-consumer problem.

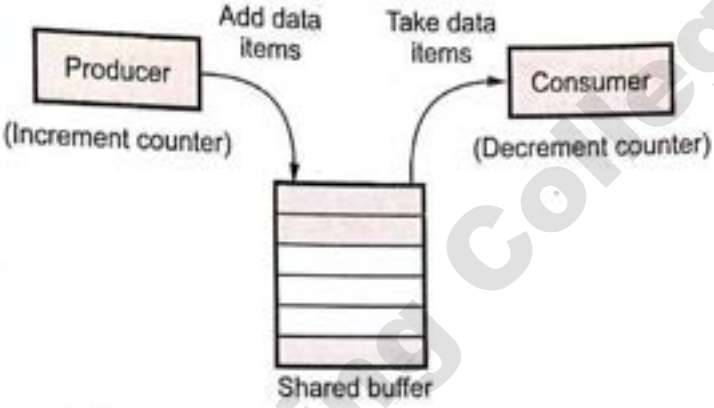


Fig. 2.17.3 Producer - consumer problem

- Producer-consumer is also called bounded buffer problem. There exist several producers and consumers.
- The producer-consumer problem shows the need for synchronization in systems where many processes share a resource. In the problem, two processes share a fixed-size buffer. One process produces data items and deposits it in the buffer, while the other process consumes data items from the buffer.
- The producer cannot deposit its data if the buffer is full. Similarly, a consumer cannot retrieve any data if the buffer is empty. If the buffer is not full, a producer can deposit its data. The consumer should be allowed to retrieve a data item if buffer contains.
- The consumer retrieves a data item, so the data from buffer start decreasing. When buffer becomes empty, the producer should be allowed to deposit its data.
- Problem arises when the buffer is full and producer wants to add a new data item. Solution to this problem is that producer goes to sleep until consumer removes data item from the buffer. Similar condition exists with the consumer. Consumer wants to consume data item from the buffer, but buffer is empty then consumer go to sleep. It wakeup when producer add some data items into the buffer.

- In order to synchronize these processes, both procedure and consumer are blocked on some condition. The producer is blocked when the buffer is full, and the consumer is blocked when the buffer is empty.
- Example of procedure consumer problem :
  1. Printing word file : When user gives the print command for printing file, a word processor spools data to a buffer and that data are subsequently consumed by the printer as it prints the document. Printer stops printing when buffer is empty.
  2. A compiler may produce assembly code, which is consumed by an assembler.
- Speed of the procedure and consumer is different. The producer process is faster than the consumer process. Suppose the CPU is producer and line printer is consumer, then CPU can generate data much faster than a line printer.
- To solve this above problem, buffer is used in between procedure and consumer. Buffer capacity is fixed (bounded) or variable (unbounded). Fig. 2.17.4 shows the buffer states between producer-consumer problem. Buffer is in any of the following states :
  1. Full buffer
  2. Empty buffer
  3. Partially empty buffer
- A solution to the producer-consumer problem satisfy the following conditions :
  1. When buffer is full, producer must wait. Producer do not produce data item.
  2. When buffer is empty, consumer must wait.
  3. Mutual exclusion is required for accessing the buffer.
- Producer-consumer problem is solved by using semaphore, mutex and monitor. Mutual exclusion must be enforced on the buffer itself.

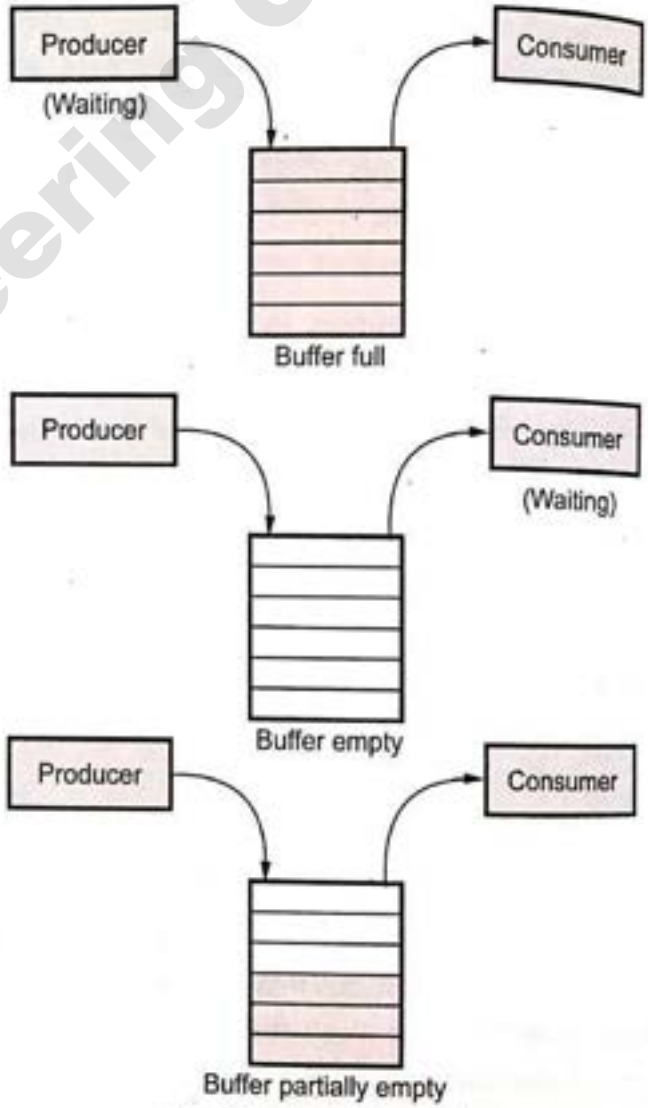


Fig. 2.17.4 Buffer states

- Each data item produce by producer to the shared buffer must be consumed exactly once by the consumer. Data item must be consumed in the same order (First in First Out Order) in which it is out into the buffer.
- If there is no synchronization between producer and consumer then data can be lost if the producer add new data item into the buffer before the consumer consumes the previous data item.
- We use variable counter to keep track of the number of data items in the buffer.
- Solution to single producer and single consumer is given below :

1. Code for producer

```
producer (void)
  {
        int item;
            while (TRUE)
              {
                produce item(&item);
                produce_if (counter  == N)
                 sleep( );
                 enter_item(item);
                  counter = counter + 1;
                      if (counter == 1)
                        wakeup(consumer);
              }
    }
```

2. Code for consumer process

```
consumer(void)
{
    int item;
        while (TRUE)
            {
                if (count == 0)
                sleep( );
                 remove_item(&item);
                 counter = counter - 1;
    if (count == N-1)
      wakeup(producer);
       consume_item(item);
            }
}
```

- Initially we assume that the buffer is empty. Consumer read the counter and context switch occurs. The producer process produce data item and increments the

counter by one. After incrementing the counter, it sends a wakeup signal to the consumer. Here one more context switching occurs and consumer process runs.

- Solution to producer consumer problem by using binary semaphore :

```
void main ( )
{
    count = 0;
}
 int i;
binarysemaphore s = 1;
int producer ( )
{
   While (true)
{
         produce_data_item ( );
        semwaitB(s);
        append ( );
        count = count + 1;
        if ( count == 1)
        semSignalB(delay);
        semSignalB(s);
        }
}
 int consumer ( )
{
   int p;
semWaitB(delay);
while (true)
{
  semWaitB(s);
consume( );
count = count - 1;
p = count;
semSignalB(s);
consumedata( );
if ( p == 0)
semWaitB(delay);
}
}
```

University Questions

1. *Consider a situation where we have a file shared between many people.*

If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her. However if some person is reading the file, then others may read it at the same time.

a) What kind of situation is this ? [3]

b) Consider the following problem parameters to solve this situation. [8]

Problem parameters :

1) One set of data is shared among a number of processes.

2) Once a writer is ready, it performs it write. Only one writer may write at a time.

3) If a process is writing no other process can read it.

4) If at least one reader is reading, no other process can read it.

5) Readers may not write and only read.

AU : Dec.-17, Marks 11

Ans. : It is reader and writer problem. Here we consider reader having higher priority than writer. Reader should not wait if the share is currently opened for reading.

- To implement this problem three variables are used; readcount, mutex and wrt.
- Semaphore function : wait() and signal()

```
semophore mutex, wrt;
int readcount;
```

- For writer process :

1. Writer requests the entry to critical section.
2. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
3. It exits the critical section.

```
do {
// writer requests for critical section
wait(wrt);
   // performs the write and  leaves the critical section
signal(wrt);
} while(true);
```

- For reader process:

1. Reader requests the entry to critical section.
2. If allowed: It increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.

- It then, signals mutex as any other reader is allowed to enter while others are already reading.

- After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore "wrt" as now, writer can enter the critical section.

3. If not allowed, it keeps on waiting.

```
do {
  // Reader wants to enter the critical section
  wait(mutex);
  // The number of readers has now increased by 1
  readcount = readcount +1;
  // there is atleast one reader in the critical section, this ensure no writer
  can enter if there is even one reader
  // thus we give preference to readers here
  if (readcnt==1)
    wait(wrt);
  // other r  neaders can enter while this current reader is inside
  // the critical section
  signal(mutex);
  // current reader performs reading here
  wait(mutex);  // a reader wants to leave
  readcount = readcount –1 ;
  // that is, no reader is left in the critical section,
  if (readcount == 0)
     signal(wrt);       // writers can enter
signal(mutex); // reader leaves
} while(true);
```

- Semaphore 'wrt' is queued on both readers and writers in a manner such that preference is given to readers if writers are also there. Thus, no reader is waiting simply because a writer has requested to enter the critical section.

2.18 Peterson's Solution

- Peterson's solution is software based solution for critical section problem. This algorithm will not work correctly on the modern computer architecture. It is simpler algorithm for two process mutual exclusion with busy waiting.
- This algorithm is less complicated than Dekker's algorithm.
- Peterson's algorithm solves critical section problem of two processes only. It does not require any special hardware. Two processes alternates execution between their critical section and remainder sections.

- Deadlock and indefinite postponement are impossible in Peterson's algorithm as long as no process or thread terminates unexpectedly.
- For indefinite postponement to occur, one process would have to be able to continually complete and reenter its critical section while the other process busy waited.

2.19 Synchronization Hardware

2.19.1 Test and Set Operations

- Test and set is a single individual machine instruction. It is simply known as TS. It was introduced by IBM for its multiprocessing system.
- In a one machine cycle, it tests to see if the key is available and, if it is, sets it to unavailable. TS are a hardware solution for critical section problem.
- TS instruction eliminates the possibility of preemption occurring during the interval. The instruction :

```
testAndset ( p, q)
```

- The instruction reads the value of q, which may be either true or **false.** Then the value is copied into "p" and the instruction sets the value of "q" to **true.**
- A process P1 would test the condition code using TS instruction before entering a critical section. If no other process was in this critical section, then process P1 would be allowed to proceed and the condition code would be changed from zero to one.
- When process P1 exist the critical section, the condition code is reset to zero so another process can enter into critical section.
- If process P1 finds a busy condition code then it is placed in a waiting loop where it continues to test the condition code and waits until it is free.
- The "TestandSet" instruction is a tool that programmers use to simplify software solution to mutual exclusion but the instruction itself does not enforce mutual exclusion.

Advantages :

1. Simple to implement.
2. It works well for a small number of processes.
3. It can be used to support multiple critical sections.

Drawbacks :

1. It suffers from starvation.
2. There is possibility of busy waiting.
3. There may be deadlock.

2.20 Monitors

- Monitor is an object that contains both data and procedures needed to perfo allocation of a shared resource. It is a programming language construct th support both data access synchronization and control synchronization.
- Fig. 2.20.1 shows structure of monitor.
- Monitor is implemented in programming languages like Pascal, java and C++. Ja makes extensive use of monitors to implement mutual exclusion.
- Monitor is an abstract data type for which only one process may be executing procedure at any given time. Monitor is a collection of procedure, variables a data structure. Data inside the monitor may be either global to all routines with the monitor or local to a specific routine.
- Monitor data is accessible only within the monitor. A shared data structure can b protected by placing it in a monitor. If the data in a monitor represents som resource, then the monitor provides a mutual exclusion facility for accessing th resource. A procedure defined within a monitor can access only those variabl declared locally within the monitor and its formal parameters.
- When a process calls a monitor procedure, the first few instruction of th procedure will check to see if any other process is currently active within th

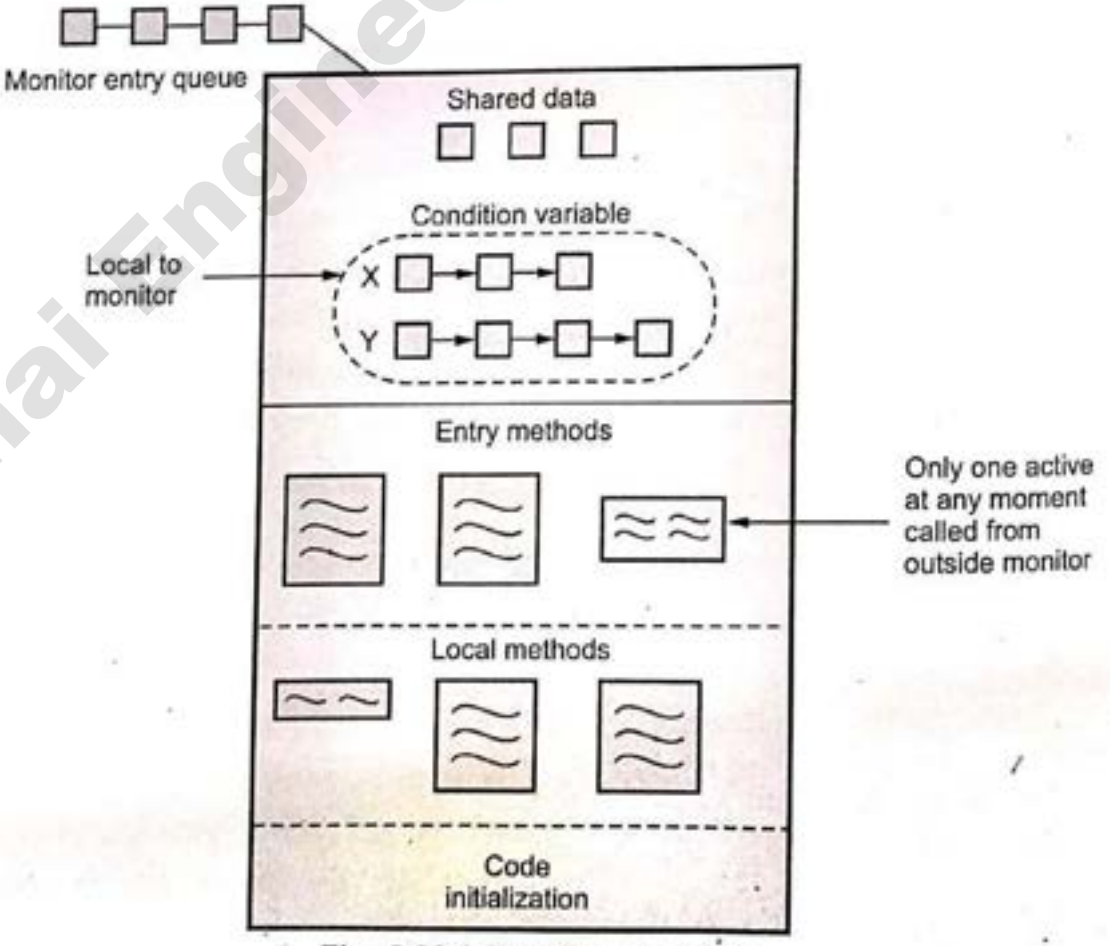


Fig. 2.20.1 Monitor structure

monitor. If process is active then calling process will be suspended until the other process has left the monitor. If no other process is using the monitor, the calling process may enter.

- Monitor supports synchronization by the use of condition variables that are contained within the monitor and accessible only within the monitor. Every conditional variable has an associated queue. Condition variables operates on two functions :

  cwait(condition variable)

  csignal (condition variable)

- **cwait** : It suspend execution of the calling process on condition.
- **csignal** : Resume execution of some process blocked after a cwait on the same condition.
- The cwait must come before the csignal.
- Fig. 2.20.2 shows monitor structure with condition variables.
- CPU is a resource that must be shared by all processes. The part of the kernel that apportions CPU time between processes is called the scheduler.

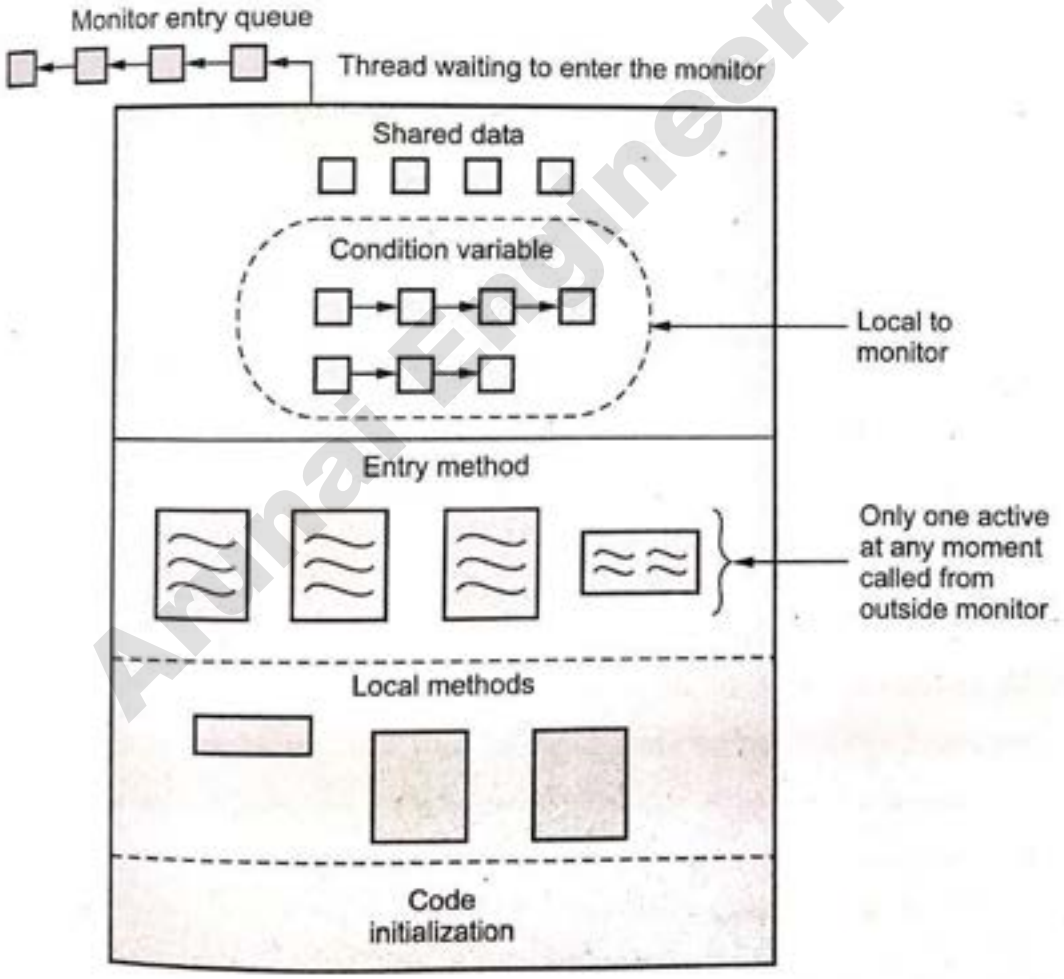


Fig. 2.20.2 Monitor structure with condition variable

- A condition variable is like a semaphore, with two differences :

1. A semaphore counts the number of excess up operations, but a signal operation on a condition variable has no effect unless some process is waiting. A wait on a condition varibale always blocks the calling process.
2. A wait on a condition variable automatically does an up on the monitor mutex and blocks the caller.

Example : Solve producer consumer problem by using monitors

Bounded Buffer problem using monitors

```
monitor Bounded Buffer
{
        private Buffer b = new Buffer (20);
        private int count = 0;
        private condition nonfull, nonempty;
        public void add (object item)
}
        if(count == 20)
        nonfull.wait ( );
        b.add (item);
        count ++;
        nonempty. signal ( );
}
public object remove ( )
{
        if (count == 0)
        nonempty.wait ( );
        item result = b.remove( );
        count = count-1;
        nonfull.signal( );
        return result ;
}
```

- Each condition variable is associated with some logical condition on the state of the monitor. Consider what happens when a consumer is blocked on the nonempty condition variable and producer calls add.

1. The producer adds the item to the buffer and calls nonempty signal ( ).
2. The producer is immediately blocked and the consumer is allowed to continue.
3. The consumer removes the item from the buffer and leaves the monitor.
4. The producer wakes up and since the signal operation wait the last statement in add, leaves the monitor.

- Monitors are a higher level concept than P and V. They are easier and safer to use but less flexible. Many languages are not supported by the monitor. Java is making monitors much more popular and well known.

Example 2.20.1 *Solve the reader-writer problem using monitor with readers priority.*

Solution :

```
readers-writers : monitor;
begin
        integer readercount;
        condition okread, okwrite;
        boolean busy;
procedure startread;
        begin
            if busy then ok read.wait;
            readercount := readercount+1;
            okread.signal;
        end startread;
        procedure endread;
        begin
            reader count:= readercount-1;
        if  reader count = 0 then okwrite.signal;
            end endread;
procedure startwrite;
        begin if busy OR readercount ≠ 0 then okwrite.wait;
        busy := true;
        end startwrite;
procedure endwrite;
        begin
        busy := false;
        if okread.queue then okread.signal
        else okwrite.signal
        end endwrite;
begin (*Initialisation*)
readercount :=0;
busy:=false;
end;
end readers-writers;
```

Example 2.20.2 *Solve procedure-consumer problem with monitors.*

Solution :

```
monitor procedure-consumer
condition full empty;
integer count;
```

```
Procedure insert (item : integer);
      begin
            if count = N then wait (full);
            insert-item(item);
            count := count+1;
            if count = 1 then signal (empty)
      end;
function remove:integer;
      begin
            if count = 0 then wait (empty);
            remove = remove_item;
            count := count–1;
            if count = N-1 then signal (full);
            end;
            count := 0;
      end monitor;
      procedure producer;
            begin
            while true do
        begin
            item = produce_item;
                  producer_consumer.insert(item)
            end
end;
            procedure consumer;
                  begin
                  while true do
                        begin
                        item = procedure_consumer.remove;
                        consume_item(item)
                  end
            end;
```

- By making the mutual exclusion of critical regions automatic, monitors make parallel programming much less error prone than with semaphores.

2.20.1 Drawbacks of Monitors

1. Major weakness of monitors is the absence of concurrency if a monitor encapsulates the resource, since only one process can be active within a monitor at a time.
2. There is the possibility of deadlocks in the case of nested monitors calls.
3. Monitor concept is its lack of implementation most commonly used programming languages.
4. Monitors cannot easily be added if they are not natively supported by the language.

2.20.2 Difference between Monitors and Semaphore

- A semaphore is an operating system abstract data type whereas monitors are based on abstract data types.
- Semaphore-level synchronization primitives are difficult to use for complex synchronization situations. Monitors were derived to simplify the complexity of synchronization problems by abstructing away details.
- Semaphores provide a general-purpose mechanism for controlling access to critical sections. Their use does not guarantee that access will be mutually exclusive or deadlock will be avoided. A monitor is a programming language construct that guarantees appropriate access to critical sections.
- Monitor uses condition variables. A condition variable is like a semaphore, with two differences

1) A semaphore counts the number of excess up operations, but a signal operation on a condition variable has no effect unless some process is waiting. A wait on a condition variable always blocks the calling process.
2) A wait on a condition variable automatically does an up on the monitor mutex and blocks the caller.

Two Marks Questions with Answers

Q.1 Provide two programming examples in which multithreading provides better performance than a single-threaded solution.

Ans. :

1. A Web server that services each request in a separate thread.
2. A parallelized application such as matrix multiplication where different parts of the matrix may be worked on in parallel.
3. An interactive GUI program such as a debugger where a thread is used to monitor user input, another thread represents the running application, and a third thread monitors performance.

Q.2 Define context switching.

Ans. : Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as context switch.

Q.3 State what does a thread share with peer threads.

Ans. : Thread share the memory and resource of the process.

Q.4 Define process. What is the information maintained in a PCB ?

Ans. : A process is simply a program in execution. i.e. an instance of a progra execution. PCB maintains pointer, state, process number, CPU register, PC, memor allocation etc.

Q.5 Can a multithreaded solution using multiple user-level threads achieve bett performance on a multiprocessor system than on a single processor system ?

Ans. : A multithreaded system comprising of multiple user-level threads cannot mak use of the different processors in a multiprocessor system simultaneously. Th operating system sees only a single process and will not schedule the different thread of the process on separate processors. Consequently, there is no performance benef associated with executing multiple user-level threads on a multiprocessor system.

Q.6 Define a thread. State the major advantage of threads.

Ans. : A thread is a flow of execution through the process's code with its own program counter, system registers and stack.

Advantages : 1. Minimize context switching time. 2. Efficient communication.

Q.7 What is co-operating processes ? Give on example.

Ans. : Co-operating process is a process that can affect or be affected by the other processes while executing.

Q.8 Define task control block.

Ans. : TCB is also called PCB.

Q.9 What are the advantages of co-operating processes ? AU : CSE/IT : Dec-13

Ans. : Advantages :

1. Sharing of information.
2. Increases computation speed.
3. Modularity.
4. Convenience.

Q.10 What are the differences between user-level threads and kernel-level threads ? (Refer section 2.6.3) AU : CSE/IT : May-12, Dec-13

Q.11 What is PCB ? Specify the information maintained in it. AU : CSE/IT : Dec-12

Ans. : Each process is represented in the operating system by a process control block. PCB contains information like process state, program counter, CPU register, accounting information etc.

Q.12 Differentiate a thread from a process. AU : CSE/IT : Dec-12

Ans. : Thread is called light weight process and process is heavy weight process. Thread switching does not need to call OS and cause an interrupt to the Kernel. Process switching needs interface with OS.

Q.13 What are the benefits of multithreads ? AU : CSE/IT : May-13

Ans. : Benefits of multithreading is responsiveness, resource sharing, economy and utilization of multiprocessor architecture.

Q.14 What are the properties of communication link ?

Ans. : Properties of communication link

1. Links are established automatically.
2. A link is associated with exactly one pair of communicating processes.
3. Between each pair there exists exactly one link.
4. The link may be unidirectional, but is usually bidirectional.

Q.15 Why a thread is called as light weight process ?

Ans. : Thread is light weight taking lesser resources than a process. It is called light weight process to emphasize the fact that a thread is like a process but is more efficient and uses fewer resources and they also share the address space.

Q.16 What are the reasons for terminating execution of child process ?

Ans. : Parent may terminate execution of children processes via abort system call for a variety of reasons, such as :

1. Child has exceeded allocated resources.
2. Task assigned to child is no longer required.
3. Parent is exiting and the operating system does not allow a child to continue if its parent terminates.

Q.17 What is independent process ?

Ans. : Independent process cannot affect or be affected by the execution of another process.

Q.18 Name one situation where threaded programming is normally used ?

Ans. : Threaded programming would be used when a program should satisfy multiple tasks at the same time. A good example for this would be a program running with GUI.

Q.19 Describe the actions taken by a thread library to context switch between user-level threads.

Ans. : Context switching between user threads is quite similar to switching between Kernel threads, although it is dependent on the threads library and how it maps user threads to kernel threads. In general, context switching between user threads involves taking a user thread of its LWP and replacing it with another thread. This act typically involves saving and restoring the state of the registers.

Q.20 What is socket ?

Ans. : A socket is defined as an endpoint for communication.

Q.21 What is a thread pool ?

Ans. : A thread pool is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application. The thread pool is primarily used to reduce the number of application threads and provide management of the worker threads.

Q.22 What is deferred cancellation in thread ?

Ans. : The target thread periodically checks whether it should terminate, allowing it an opportunity to terminate itself in an orderly fashion. With deferred cancellation, one thread indicates that a target thread is to be cancelled, but cancellation occurs only after the target thread has checked a flag to determine if it should be cancelled or not. This allows a thread to check whether it should be cancelled at a point when it can be cancelled safely.

Q.23 What is Pthread ?

Ans. : Pthreads refers to the POSIX standard defining an API for thread creation and synchronization. This is a specification for thread behavior, not an implementation. Operating system designers may implement the specification in any way they wish.

Q.24 What is thread cancellation ?

Ans.: Under normal circumstances, a thread terminates when it exits normally, either by returning from its thread function or by calling *pthread_exit*. However, it is possible for a thread to request that another thread terminate. This is called canceling a thread.

Q.25 What is ready queue ?

Ans. : The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue.

Q.26 List the benefit of multithreading.

Ans. : Benefit of multithreading :

1. It takes less time to create thread than a new process.
2. It takes less time to terminate thread than process.

Q.27 Name any two file system objects that are neither files nor directories and what the advantage of doing so is.

Ans. : Semaphore and monitors the system objects. Advantage is to avoid critical section problem.

Q.28 Define mutual exclusion. AU : CSE/IT : Dec.-11, May-13

Ans. : If a collection of processes share a resource or collection of resources, then often mutual exclusion is required to prevent interference and ensure consistency when accessing the resources.

Q.29 What is bounded buffer problem ?

Ans. : The bounded buffer producers and consumers assume that there are fixed buffer sizes i.e. a finite numbers of slots are available. To suspend the producers when the buffer is full, to suspend the consumers when the buffer is empty, and to make sure that only one process at a time manipulates a buffer so there are no race conditions or lost updates.

Q.30 How the lock variable can be used to introduce mutual exclusion ?

Ans. : We consider a single, shared, (lock) variable, initially 0. When a process wants to enter in its critical section, it first tests the lock. If lock is 0, the process first sets it to 1 and then enters the critical section. If the lock is already 1, the process just waits until (lock) variable becomes 0. Thus, a 0 means that no process in its critical section, and 1 means hold your horses - some process is in its critical section.

Q.31 What is binary semaphore ?

Ans. : Binary semaphore is a semaphore with an integer value that can range only between 0 and 1.

Q.32 What is the hardware feature provided in order to perform mutual exclusion operation indivisibly ?

Ans. : Hardware features can make any programming task easier and improve system efficiency. It provide special hardware instructions that allow user to test and modify the content of a word.

Q.33 What is semaphore ? Mention its importance in operating systems.

AU : CSE/IT : Dec.-11, 12

Ans. : Semaphore is an integer variable. It is a synchronization tool used to solve critical section problem. The various hardware based solutions to the critical section problem are complicated for application programmers to use.

Q.34 Discuss why implementing synchronization primitives by disabling interrupts is not appropriate in a single processor system if the synchronization primitives are to be used in user level programs.

Ans. : If a user level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes execute.

Q.35 State the assumption behind the bounded buffer producer consumer problem.

Ans. : Assumption : It is assume that the pool consists of 'n' buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.

Q.36 State the condition to avoid the race conditions.

Ans. : Where the several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called a race conditions. To guard against the race condition, we need to ensure that only one process at a time can be manipulating the variable counter.

Q.37 Explain the use of monitors.

Ans. : Use of monitors :

a) It provides a mutual exclusion facility. b) A monitor support synchronization by the use of condition variables. c) Shared data structure can be protected by placing it in a monitor.

Q.38 Define 'monitor'. What does it consist of ? AU : CSE/IT : Dec.-11

Ans. : Monitor is a highly structured programming language construct. It consists of private variables and private procedures that can only be used within a monitor.

Q.39 What is race condition ?

Ans. : A race condition is a situation where two or more processes access shared data concurrently and final value of shared data depends on timing.

Q.40 What is a critical section and what requirements must a solution to the critical section problem satisfy ? AU : CSE/IT : Dec.-13

Ans. : Consider a system consisting of several processes, each having a segment of code called a critical section, in which the process may be changing common variables, updating tables, etc. The important feature of the system is that when one process is executing its critical section, no other process is to be allowed to execute its critical section. Execution of the critical section is mutually exclusive in time.

A solution to the critical section problem must satisfy the following three requirements :

1. Mutual exclusion 2. Progress 3. Bounded waiting

Q.41 Define entry section and exit section.

Ans. : Each process must request permission to enter its critical section. The section of the code implementing this request is the entry section. The critical section is followed by an exit section. The remaining code is the remainder section.

Q.42 What is the meaning of the term busy waiting ? AU : May-16

Ans. : Busy waiting means a process waits by executing a tight loop to check the status/value of a variable.

Q.43 What is bounding waiting ?

Ans. : After a process made a request to enter its critical section and before it is granted the permission to enter, there exists a bound on the number of times that other processes are allowed to enter.

Q.44 Why can't you use a test and set instruction in place of a binary semaphore ?

Ans. : A binary semaphore requires either a busy wait or a blocking wait, semantics not provided directly in the Test and Set. The advantage of a binary semaphore is that it does not require an arbitrary length queue of processes waiting on the semaphore.

Q.45 Differentiate preemptive and non-preemptive scheduling.

Ans. :

No.	Preemptive scheduling	Non-preemptive scheduling
1.	It allows a process to be interrupted in the middle of its execution, taking the CPU away and allocating it to another process.	It ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.
2.	Complex to implement.	Simple to implement.
3.	Costly.	Cost is less.
4.	Higher overhead.	Less overhead.
5.	Process switches from running state to ready state and waiting state to ready state.	Process switches from running state to waiting state and process terminates.

Q.46 Define priority inversion problem.

Ans. : The higher priority process would be waiting for the low priority one to finish. This situation is known as priority inversion problem.

Q.47 What advantage is there in having different time-quantum sizes on different levels of a multilevel queuing system ?

Ans. : Processes that need more frequent servicing, for instance, interactive processes such as editors, can be in a queue with a small time quantum. Processes switch no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, and thus making more efficient use of the computer.

Q.48 What is non-preemptive scheduling ?

Ans. : Non-preemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

Q.49 What is context switching ?

Ans. : Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as context switch.

Q.50 Explain why two level scheduling is commonly used.

Ans. : It provides the hybrid solution to the problem of providing good system utilization and good user service simultaneously.

Q.51 What do you mean by short term scheduler ?

Ans. : Short term scheduler, also known as a dispatcher executes most frequently, and makes the finest-grained decision of which process should execute next. This scheduler is invoked whenever an event occurs.

Q.52 Which are the criteria used for CPU scheduling ? **AU : CSE/IT : May-14**

Ans. : Criteria used for CPU scheduling are CPU utilization, throughput, turnaround time, waiting time, response time.

Q.53 How does real-time scheduling differs from normal scheduling ?

AU : CSE/IT : Dec.-12

Ans. : Normal scheduling provides no guarantee on when a critical process will be scheduled; it guarantees only that the process will be given preference over non-critical processes. Real-time systems have stricter requirements. A task must be serviced by its deadline; service after the deadline has expired is the same as no service at all.

Q.54 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs ?

Ans. : I/O-bound programs have the property of performing only a small amount of computation before performing IO. Such programs typically do not use up their entire CPU quantum. CPU-bound programs, on the other hand, use their entire quantum without performing any blocking IO operations. Consequently, one could make better use of the computer's resources by giving higher priority to I/O-bound programs and allow them to execute ahead of the CPU-bound programs.

Q.55 What is Shortest-Remaining-Time-First (SRTF) ?

Ans. : If a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First.

Q.56 What is response time ?

Ans. : Response time is the amount of time it takes from when a request was submitted until the first response is produced, not output.

Q.57 Define waiting time.

Ans. : Amount of time a process has been waiting in the ready queue.

Q.58 What is round robin CPU scheduling ?

Ans. : Each process gets a small unit of CPU time (time quantum). After this time has elapsed, the process is preempted and added to the end of the ready queue.

Q.59 Define scheduling algorithm ?

Ans. : In multiprogramming systems, whenever two or more processes are simultaneously in the ready state, a choice has to be made which process to run next. The part of the OS that makes the choice is called the scheduler and the algorithm it uses is called the **scheduling algorithm.**

Q.60 Define the term dispatch latency.

AU : CSE/IT, April/May-2015

Ans. : Time it takes for the dispatcher to stop one process and start another running.

Q.61 What is meant by starvation in operating system ?

AU : CSE/IT : Dec.-13

Ans. : Starvation is a resource management problem where a process does not get the resources (CPU) it needs for a long time because the resources are being allocated to other processes.

Q.62 What is an aging ?

Ans. : Aging is a technique to avoid starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the requests priority as time passes and must ensure that a request will eventually be the highest priority request.

Q.63 How to solve starvation problem in priority CPU scheduling ?

Ans. : Aging - as time progresses increase the priority of the process, so eventually the process will become the highest priority and will gain the CPU. i.e., the more time is spending a process in ready queue waiting, its priority becomes higher and higher.

Q.64 What is preemptive priority method ?

Ans. : A preemptive priority will preempt the CPU if the newly arrived process is higher than the priority of the currently running process.

Q.65 What is medium term scheduling ?

Ans. : Medium-term scheduling used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.

Q.66 What is preemptive scheduling ?

Ans. : Preemptive scheduling can preempt a process which is utilizing the CPU in between its execution and give the CPU to another process.

Q.67 What is convoy effect ?

Ans. : A convoy effect happens when a set of processes need to use a resource for a short time, and one process holds the resource for a long time, blocking all of the other processes. Essentially it causes poor utilization of the other resources in the system.

Q.68 How can starvation / indefinite blocking of processes be avoided in priority scheduling ?

Ans. : A solution to the problem of indefinite blockage of processes is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

Q.69 What is the difference between long-term scheduling and short-term scheduling ?

Ans. : Long term scheduling adds jobs to the ready queue from the job queue. Short term scheduling dispatches jobs from the ready queue to the running state.

Q.70 What are the benefits of multithreads ? AU : CSE/IT : May-13

Ans. : Benefits of multithreading is responsiveness, resource sharing, economy and utilization of multiprocessor architecture.

Q.71 What are the reasons for providing process cooperation ?

Ans. : Shared memory and shared resources.

Q.72 List out any four scheduling criteria.

Ans. : Response time, throughput, waiting time and turn around time.

Q.73 List out the data fields associated with process control blocks. AU : May-15

Ans. : Data fields associated with process control block is CPU registers, PC, process state, memory management information, input-output status information etc.

Q.74 Define the term 'Dispatch latency'. AU : May-15

Ans. : Dispatch latency : Time it takes for the dispatcher to stop one process and start another running. It is the amount of time required for the scheduler to stop one process and start another.

Q.75 What is concept behind strong semaphore and spinlock ? AU : Dec.-15

Ans. :

- Semaphore can be implemented in user applications and in the kernel. The process that has been blocked the longest is released from the queue first is called a strong semaphore.
- Using simple lock variable, process synchronization problem is not solved. To avoid this, spinlock is used. A lock that uses busy waiting is called a spin lock.

Q.76 Under what circumstances is user level threads is better than the kernel level threads? AU : May-16

Ans. : User-level threads are much faster to switch between, as there is no context switch; further, a problem-domain-dependent algorithm can be used to schedule among them. CPU-bound tasks with interdependent computations, or a task that will switch among threads often, might best be handled by user-level threads

Q.77 Distinguish between CPU-bounded and I/O bounded processes. AU : EIE : Dec.-16

Ans. :

CPU-bounded	I/O bounded processes
CPU Bound processes are ones that are implementing algorithms with a large number of calculations	Processes that are mostly waiting for the completion of input or output (I/O) are I/O Bound
Given a lower priority by the scheduler	Given high priority by the scheduler
CPU bound does too much computation to keep I/O busy	I/O bound does too much I/O to keep CPU busy
Example : matrix multiplication	Example : netscape

Q.78 What resources are required to create threads ? AU : EIE : Dec.-16

Ans. : Thread is smaller than a process, so thread creation typically uses fewer resources than process creation.

Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

Q.79 "Priority inversion is a condition that occurs in real time systems where a low priority process is starved because higher priority processes have gained hold of the CPU" - Comment on this statement. AU : CSE : May-17

Ans. : A low priority thread always starts on a shadow version of the shared resource, the original resource remains unchanged. When a high-priority thread needs a resource engaged by a low -priority thread, the low priority thread is preempted, the original resource is restored and the high -priority thread is allowed to use the original resource.

Q.80 Differentiate single threaded and multi-threaded processes. AU : CSE : May-17

Ans. : Single-threading is the processing of one command at a time. When one thread is paused, the system waits until this thread is resumed. In Multithreaded processes, threads can be distributed over a series of processors to scale. When one thread is paused due to some reason, other threads run as normal.

Q.81 Name and draw five different process states with proper definition. AU : Dec.-17

Ans. : Process states are new, running, waiting, ready and terminated. Fig. 2.1 shows process state diagram.

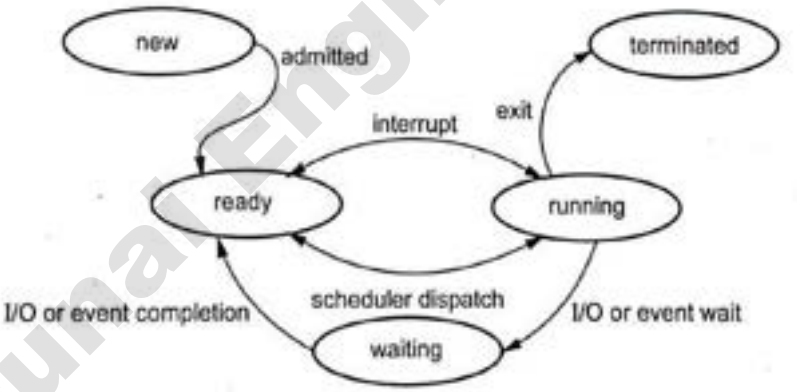


Fig. 2.1

Q.82 Elucidate mutex locks with its procedure. AU : Dec.-17

Ans. : Mutex lock is software tools to solve the critical-section problem. A mutex lock has a boolean variable available whose value indicates if the lock is available or not. If the lock is available, a call to acquire() succeeds, and the lock is then considered unavailable

Q.83 What are the benefits of synchronous and asynchronous communication ? AU : May-18

Ans. : Benefits of Synchronous Communication :

- Easy to program
- Outcome is known immediately
- Error recovery easier
- Better real-time response

Disadvantages of Synchronous Communication :

- Service must be up and ready.
- Usually requires connection-oriented protocol

Advantages of Asynchronous Communication :

- Requests need not be targeted to specific server.
- Service need not be available when request is made.
- No blocking, so resources could be freed.
- Could use connectionless protocol

Disadvantages of Asynchronous Communication :

- Response times are unpredictable.
- Error handling usually more complex.
- Usually requires connection-oriented protocol.

Q.84 Give an programming example in which multitreading does not provide better performance than single threaded solution. AU : May-18

Ans. : • Any kind of sequential program is not a good candidate to be threaded. An example of this is a program that calculates an individual tax return.

- Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

❑❑❑

UNIT - II

3 Deadlock

Syllabus

Deadlock - System model, Deadlock characterization, Methods for handling deadlocks, Deadlock prevention, Deadlock avoidance, Deadlock detection, Recovery from deadlock.

Contents

3.1 System Model

- A lack of process synchronization can result in two extreme conditions deadlock or starvation. Deadlock is the problem of multiprogrammed system.
- Deadlock can be defined as the permanent blocking of a set of processes th either complete for system resources. Deadlock is more serious than starvation.
- Deadlock can occur on sharable resources such as files, printers, database, disk tape drives, memory, CPU cycles etc.
- A process is in deadlock state if it was waiting for a particular event that will no occur. In a system deadlock, one or more processes are deadlocked.

Deadlock Example

- Computer system is collection of limited/finite number of resources. Thes resources are distributed among a number of competing processes.
  Resources are of two types :

1. Reusable resources
2. Consumable resources.

- **Reusable resource** is used only by one process at a time. Process can release resource after use. Processors, I/O channel, I/O device, files, database, primary and secondary memory, semaphore are example of the reusable resource.
- Consumable resource is one that can be created and destroyed. There is no limit on the number of consumable resources of a particular type.
- An interrupt, messages, signals and messages in I/O buffers are examples of consumable resources.
- Process utilize the resources in the following sequence :

1. Request for resource
2. Use of resource
3. Resource release.

- Process uses system call for requesting the resource. After allocating resource to the process, it use or operate the resource.
- The process releases the resource after use.
- If the resource is not available when it is requested, the requesting process is forced to wait.
- Fig. 3.1.1 shows the deadlock with 4 process.

Process 1	Process 2	Process 3	Process 4
Printf(" ")	int a;	Procedure ( )	sort ( )
- - - - -	- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -	- - - - -
request (resource 1); /* Holding res 1 */	request (resource 2); /* Holding res 2 */	request (resource 3); /* Holding res 3 */	request (resource 4) /* Holding res 4*/
- - - - -	- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -	- - - - -
request (resource 2);	request (resource 3);	request (resource 4);	request(resource 1);

Fig. 3.1.1 Deadlock with 4 process

- Process 1 is holding resource 1 and requesting resource 2; process 2 is holding resource 2 and requesting resource 3; process 3 is holding resource 3 and requesting resource 4. Process 1 is holding resource 4 also.
- Here deadlock occurs because none of the processes can proceed because all are waiting for a resource held by another blocked process.
- To break this situation, one of the process release the resource.
- Operating system must be handle the deadlock situation. OS detect the deadlock and solve the deadlock problem.

3.1.1 Resource Allocation Graphs

- Resource allocation graph is introduced by Holt. It is a directed graph that depicts a state of the system of resources and processes.
- Process and resource are represented by node in directed graph. Graph consists of a set of vertices (V) and set of edges (E).
- A process node is graphically represented by circle and shown in Fig. 3.1.2
- A resource node is graphically represented by a rectangle and represents one type of resources. Fig. 3.1.3 shows resources node.

Fig. 3.1.2 Process

- The number of bullet symbols in a resource node indicates how many units of that resource class exist in the system.

Fig. 3.1.3 Resource with 8 instances

- An arrow from the process to resource indicates the process is requesting the resource. An arrow from resource process shows an instance of the resource has been allocated to the process.
- Claim edge P → R indicated that process P may request resource R in the future represented by a dashed line. Claim edge converts to request edge when a proc requests a resource. Fig. 3.1.4 shows request and claim edge.
- Request edge converted to an assignment edge when the resource is allocated the process. When a resource is released by a process, assignment edge reconve to a claim edge.

(a) Request edge

(b) Claim edge

Fig. 3.1.4 Edges

- Fig. 3.1.5 shows a resource allocation graph. System consists of process a resources.

  Process : P_1, P_2

  Resource : R_1, R_2

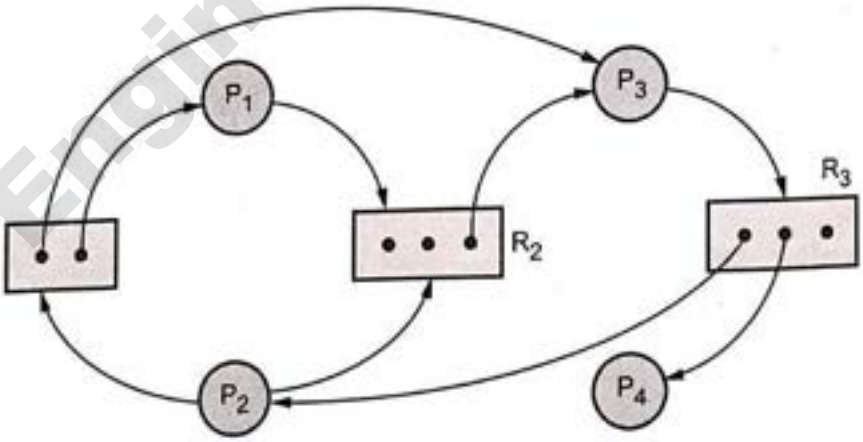


Fig. 3.1.5 Resource allocation graph

Resource	Number of instance
R_1	2
R_2	3
R_3	3

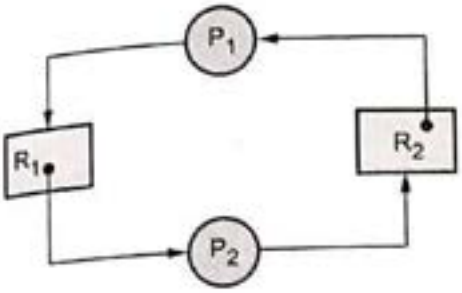


Fig. 3.1.6 Circular wait with deadlock

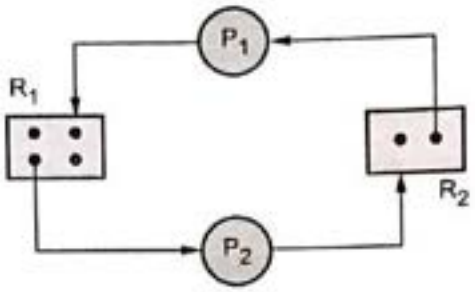


Fig. 3.1.7 Circular wait but no deadlock

Fig. 3.1.8 shows the resource allocation graph. System consists of four processes (P_1, P_2, P_3, P_4) and four resources (R_1, R_2, R_3, R_4).

Resource	Number of instance
R_1	2
R_2	2
R_3	3
R_4	6
R_5	3

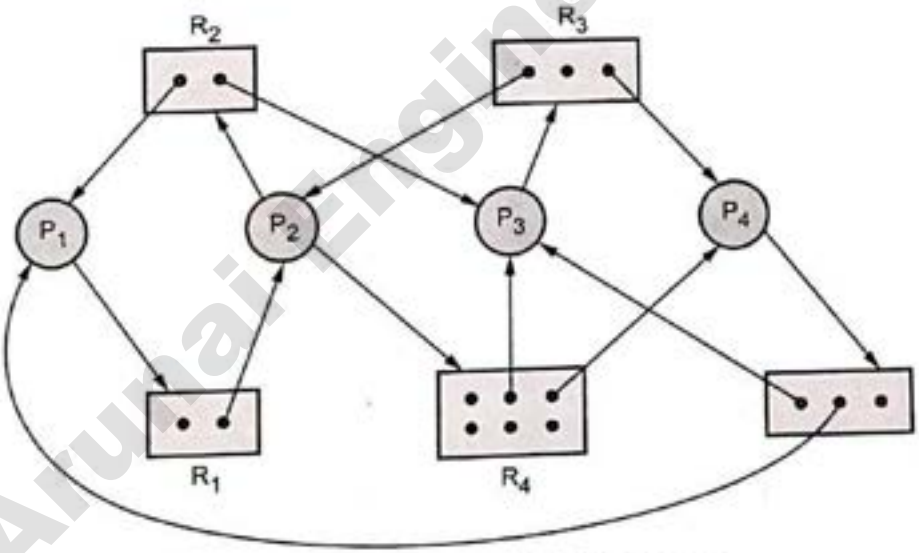


Fig. 3.1.8 Resource allocation graph

- Resource managers and other operating system processes can be involved in a deadlock. Deadlock is a global condition rather than a local one.
- An individual program cannot generally detect a deadlock. It is blocked and unable to use the processor to do any work. Deadlock detection must be handled by the operating system.

3.2 Deadlock Characteristics

3.2.1 Necessary Condition for Deadlock

- Following four conditions are necessary for deadlock to exist.

1. Mutual exclusion 2. Hold and wait
3. No preemption 4. Circular wait

1. **Mutual exclusion :** A resource may be acquired exclusively by only one proce at a time.
2. **Hold and wait :** Processes currently holding resources that were granted earli can request new resources.
3. **No preemption :** Once a process has obtained a resource, the system cann remove it from the process control until the process has finished using th resource.
4. **Circular wait :** A circular chain of hold and wait condition exists in the system.

- All four of these conditions must be present for a resource deadlock to occur. I one of them is absent, no resource deadlock is possible.

3.3 Deadlock Solution

- There are four approaches for deadlock solution.

1. Deadlock prevention 2. Deadlock avoidance
3. Deadlock detection 4. Deadlock recovery

- In deadlock **prevention,** aim is to condition a system to remove any possibility o deadlock occuring. Poor resource utilization may be possible.
- **Avoidance :** Deadlocks can be avoided by clearly identifying safe states and unsafe states.
- **Detection :** Deadlock detection methods are used in systems in which deadlock can occur.
- **Recovery :** Used to resolve the deadlock from a system.
- Deadlock prevention and detection algorithm is used for ignoring the deadlock.

3.4 Deadlock Prevention

AU : May-17, 18

- To prevent a deadlock, the OS must eliminate one of the four necessary conditions.

1. Mutual exclusion 2. Hold and wait 3. No preemption 4. Circular wait

1. **Mutual exclusion :** It is necessary in any computer system because some resources (memory, CPU) must be exclusively allocated to one user at a time. No other process can use a resource while it is allocated to a process.
2. **Hold and wait :** If a process holding certain resources is denied a further request, it must release its original resources and if required request them again.
3. **No preemption :** It could be bypassed by allowing the operating system to deallocate resources from process.
4. **Circular wait :** Circular wait can be bypassed if the operating system prevents the formation of a circle.

- A deadlock is possible only if all four of these conditions simultaneously hold in the system.
- Prevention strategies ensure that at least one of the conditions is always false.

Example 3.4.1 *Consider a system consisting of 'm' resources of the same type being shared by 'n' processes. Resource can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold :*

i) The maximum need of each process is between 1 and m resources.

ii) The sum of all maximum needs is less than m + n. **AU : May-18, Marks 15**

Solution : By contradiction, assume that the 4 conditions for deadlock exist in the system and thus there is a group of processes involved in a circular wait.

- Let these processes be $P_1, \ldots, P_k$, $k \leq n$, their current demands be $D_1,\ldots,D_k$ and the number of resources each of them holds be $H_1,\ldots,H_k$.
- The circular wait condition should look like : $P_1 -> P_2 -> \ldots -> P_k -> P_1$, but in fact it is simpler : Let $M_1, \ldots, M_n$ be total (maximum) demands of processes $P_1, \ldots, P_n$.
- Then a circular wait can occur only if all resources are in use and every process hasn't acquired all its resources : $H_1 + \ldots + H_k = m$ and $D_i >= 1$ for $1 \leq i \leq k$.
- Since $M_i = H_i + D_i$, the sum of maximum demands of the processes involved in a circular wait is : $M_1 + \ldots + M_k \geq m+k$.
- Note that the remaining processes' $P_{k+1}, \ldots, P_n$ maximum demands are at least 1 : $M_i \geq 1$, $k+1 \leq i \leq n$ and thus $M_{k+1} + \ldots + M_n \geq n-k$.
- The total sum of maximum demands is thus :
  $M_1 + \ldots + M_n = M_1 + \ldots + M_k + M_{k+1} + \ldots + M_n \geq m + k + (n-k) = m + n$.
- It is defined that sum of all maximal needs is strictly less than m+n, thus we have a contradiction.
- If we have a deadlock all resources are held by the various processes, otherwise some process can take a resource and "advance" and we are not in a deadlock.

- Therefore, every process needs at least 1 resource more, or else we don't have a deadlock. So the total sum of maximum demands is :

$$\sum_{i=1}^{n} M_i = \sum_{i=1}^{n} H_i + D_i = \sum_{i=1}^{n} H_i + \sum_{i=1}^{n} D_i \geq m + n$$

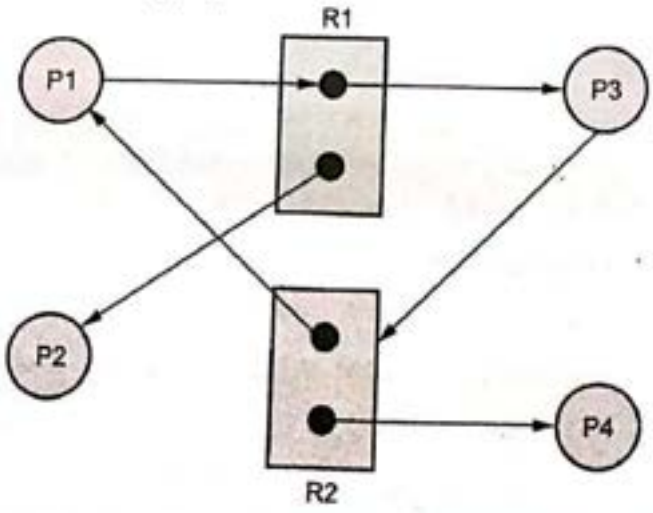
- This contradicts the assumption total sum of maximum demands is less than m+n.

University Question

1. *What is deadlock ? What are the necessary conditions for deadlock to occur ? Explain the deadlock prevention method of handling deadlock.*
   Consider the following information about resources in a system.
   i) There are two classes of allocatable resource labeled R1 and R2
   ii) There are two instances of each resource
   iii) There are four processes labeled p1 through p4
   iv) There are some resource instances already allocated to processes as follows :
   One instance of R1 held by p2, another held by p3
   One instance of R2 held by p1, another held by p4
   v) Some processes have requested additional resources, as follows :
   p1 wants one instance of R1
   p3 wants one instance of R2
   1) Draw the resource allocation graph for this system
   2) What is the state (runnable, waiting) of each process ? For each process that is waiting indicate what it is waiting for.
   3) In this system deadlocked ? If so, state which processes are involved. If not, give an execution sequence that eventually ends, showing resource acquisition and release at each step.

AU : May-17, Marks 15

Ans. : 1. Resource allocation graph :



2. Runnable state process = P4, P2

   Waiting state process = P1, P3

- Process P1 is waiting for one instance of R1 and process P3 is waiting for instance of R2.

3. System is not in deadlock state.

   Execution sequence is P4, P2, P3, P1 or P2, P4, P1, P3.

- P2 and P4 have all resources for completing so safe sequence is P2, P4, P1, P3.

3.5 Deadlock Avoidance

AU : Dec.-15, 17

- Deadlock avoidance depends on additional information about the long term resource needs of each process. The system must be able to decide whether granting a resource is safe or not and only make the allocation when it is safe.
- Fig. 3.5.1 shows safe and unsafe state.

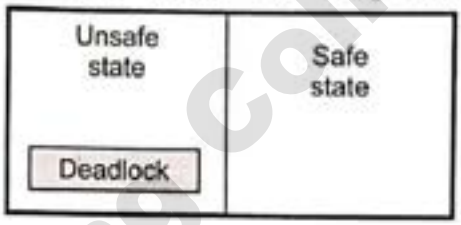


Fig. 3.5.1 Safe and unsafe state

- When a process is created, it must declare its maximum claim, i.e. the maximum number of unit resource. The resource manager can grant the request if the resources are available.
- For example, if process 1 requests a resource held by process 2 then make sure that process 2 is not waiting for resource head by first process 1.

3.5.1 Banker's Algorithm

- Banker's algorithm is the deadlock avoidance algorithm. Banker's is named because the algorithm is modeled after a banker who makes loads from a pool of capital and receives payments that are returned to that pool.
- Algorithm is check to see if granting the request leads to an unsafe state. If it does, the request is denied. If granting the request leads to a safe state, it is carried out.
- The Dijkstra proposed an algorithm to regulate resource allocation to avoid deadlocks. The banker's algorithm is the best known of the avoidance method.
- By using avoidance method, the system is always kept in a safe state.
- It is easy to check if a deadlock is created by granting a request, deadlock analysis method is used for this.
- Deadlock avoidance uses the worst-case analysis method to check for future deadlocks.

- **Safe state :** There is at least one sequence of resource allocations to processes that does not result in a deadlock.
- System is in a safe state only if there exist a safe sequence. A safe state is not a deadlocked state.
- Deadlocked state is an unsafe state. It does mean the system is in a deadlock.
- As long as the state is safe, the resource manager can be guaranteed to avoid a deadlock.
- Initially the system is in a safe state. When process requests a resource and the resource is available then the system must decide whether the resources can be allocated immediately or process must wait.
- If system remains in safe state after allocating resource then only OS allocate resources to process.
- Banker algorithm uses following data structures.

1. Allocation : Allocation is a table in which row represents process and column represents resources (R).

alloc [i, j] = Number of unit of resource R_j held by process P_i.

2. Max : Max be the maximum number of resources that process requires during its execution.

- **Need (claim) :** It is current claim of a process, where a process's claim is equal to its maximum need minus its current allocation.

  Need = Max – Allocation

- **Available :** There will be number of resources still available for allocation. This is equivalent to the total number of resources minus the sum of the allocation to all processes in the system.

  Available = Number of resources – Sum of the allocation to all process

$$= \text{Number of resources} - \sum_{i=1}^{n} \text{Allocation}(P_i)$$

- Each process cannot request more that the total number of resources in the system. Each process must also guarantee that once allocated a resource, the process will return that resource to the system within a finite time.

Weakness of Banker's algorithm

1. It requires that there be a fixed number of resources to allocate.
2. The algorithm requires that users state their maximum needs (request) in advance.
3. Number of users must remain fixed.

4. The algorithm requires that the bankers grant all requests within a finite time.
5. Algorithm requires that process returns all resource within a finite time.

Examples on Banker's algorithm

1. System consists of five processes (P_1, P_2, P_3, P_4, P_5) and three resources (R_1, R_2, R_3). Resource type R_1 has 10 instances, resource type R_2 has 5 instances and R_3 has 7 instances. The following snapshot of the system has been taken :

Process	Allocation			Max			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	0	7	5	3	3	3	2
P_2	2	0	0	3	2	2			
P_3	3	0	2	9	0	2			
P_4	2	1	1	2	2	2			
P_5	0	0	2	4	3	3			

Content of the matrix need is calculated as Need = Max – Allocation.

Process	Need		
	R_1	R_2	R_3
P_1	7	4	3
P_2	1	2	2
P_3	6	0	0
P_4	0	1	1
P_5	4	3	1

Currently the system is in safe state.

Safe sequence : Safe sequence is calculated as follows :

1) Need of each process is compared with available. If $need_i \leq available_i$, then the resources are allocated to that process and process will release the resource.
2) If need is greater than available, next process need is taken for comparison.
3) In the above example, need of process P_1 is (7, 4, 3) and available is (3, 3, 2).

   need ≥ available → False

   So system will move for next process.
4) Need for process P_2 is (1, 2, 2) and available (3, 3, 2), so

   need ≤ available (work)

   (1, 2, 2) ≤ (3, 3, 2) = True

   then Finish [i] = True

Request of P_2 is granted and processes P_2 is release the resource to the system.

Work := Work + Allocation

Work := (3, 3, 2) + (2, 0, 0)

:= (5, 3, 2)

This procedure is continued for all processes.

5) Next process P_3 need (6, 0, 0) is compared with new available (5, 3, 2).

Need > Available = False

(6 0 0) > (5 3 2)

6) Process P_4 need (0, 1, 1) is compared with available (5, 3, 2).

Need < Available

(0 1 1) < (5 3 2) = True

Available = Available + Allocation

= (5 3 2) + (2 1 1)

= (7 4 3) (New available)

7) Then process P_5 need (4 3 1) is compared with available (7 4 3)

Need < Available

(4 3 1) < (7 4 3) = True

Available = Available + Allocation

= (7 4 3) + (0 0 2) = (7 4 5) (New available)

8) One cycle is completed. Again system takes all remaining process in sequence. So process P_1 need (7 4 3) is compared with new available (7 4 5).

Need < Available = True

(7 4 3) < (7 4 5)

Available = Available + Allocation

= (7 4 5) + (0 1 0) = (7 5 5) (New available)

9) Process P_3 need is (6 0 0) is compared with new available (7 5 5).

∴ Need < Available = True

(6 0 0) < (7 5 5) = True

Available = Available + Allocation

= (7 5 5) + (3 0 2)

= (10 5 7) = (New available)

Safe sequence is < P_2 P_4 P_5 P_1 P_3 >

Example 3.5.1 *Consider the following snapshot of a system. Using the banker's algorithm.*

Process	Allocation				Request				Available			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

If a request from P_1 arrives for (0, 4, 2, 0), can the request be granted immediately ?

Solution :

i) **Content of need matrix is**

Process	R_1	R_2	R_3	R_4
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

Safe state

a) Need of P_0 = (0, 0, 0, 0)

Available = (1, 5, 2, 0)

Need < Available

(Required resources are allocated to P_0)

∴ New available = Allocation + Available

= (0, 0, 1, 2) + (1, 5, 2, 0)

= (1, 5, 3, 2)

b) For P_1 process

Need = (0, 7, 5, 0)

Available = (1, 5, 3, 2)

Need > Available

(Request is not granted)

New available = Available = (1, 5, 3, 2)

c) For P_2 process

Need = (1, 0, 0, 2)

Available = (1, 5, 3, 2)

Need < Available

(Request is granted)

New available = Allocation + Available = (1, 3, 5, 4) + (1, 5, 3, 2)

= (2, 8, 8, 6)

d) For process P_3

Need of P_3 = (0, 0, 2, 0)

Available = (2, 8, 8, 6)

Need < Available

New available = (2, 14, 11, 8)

e) Process P_4

Need of P_4 = (0, 6, 4, 2)

Available = (2, 14, 11, 8)

Need < Available

New available = (2, 14, 11, 8) + (0, 0, 1, 4)

= (2, 14, 12, 12)

f) Again for P_1 process

Need of P_1 = (0, 7, 5, 0), Available = (2, 14, 12, 12)

Need < Available

New available = (1, 0, 0, 0) + (2, 14, 12, 12)

= (3, 14, 12, 12)

So the safe sequence is P_0, P_2, P_3, P_4, P_1

Request from P_1 = (0, 4, 2, 0)

Available = (1, 5, 2, 0)

After granting the request of P_1, available resource is (1, 1, 0, 0)

a) Need of P_0 = (0, 0, 0, 0)

New available = (1, 1, 0, 0) + (0, 0, 1, 2)

= (1, 1, 1, 2)

b) P_1 need is greater than available.

c) P_2 need is (1, 0, 0, 2)

Need < Available

New available = (1, 1, 1, 2) + (1, 3, 5, 4)

= (2, 4, 6, 6)

d) P_3 Need is (0, 0, 2, 0)

Need < Available

New available = (2, 4, 6, 6) + (0, 6, 3, 2)

= (2, 10, 9, 8)

e) P_4 Need is (0, 6, 4, 2)

Available (2, 10, 9, 8)

Need < Available

∴ New available = (2, 10, 9, 8) + (0, 0, 1, 4)

= (2, 10, 10, 12)

f) Again P_1 need (0, 7, 5, 0)

Need < Available

If a request from P_1 arrives for (0, 4, 2, 0), then **the request is granted immediately.**

Example 3.5.2 *The operating system contains 3 resources, the number of instance of each resource type are 7, 7, 10. The current resource allocation state is as shown below.*

Process	Current allocation			Maximum need		
	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8
P_2	2	0	3	4	3	3
P_3	1	2	4	3	4	4

i) Is the current allocation in a safe state ?

ii) Can the request made by process P_1 (1 1 0) be granted ? **AU : Dec.-15, Marks 4**

Solution : **i)** First find the available resource in the system.

Available : = Number of instance – Sum of allocation

Process	Current allocation		
	R_1	R_2	R_3
P_1	2	2	3
P_2	2	0	3
P_3	1	2	4
	5	4	10 ⇐ Sum

Available = (7 7 10) – (5 4 10)

Available resources = (2 3 0)

Content of need matrix is

Process	Need		
	R_1	R_2	R_3
P_1	1	4	5
P_2	2	3	0
P_3	2	2	0

Safe sequence < P_3, P_2, P_1 >.

The system is in safe state.

ii) Process P_1 request (1 1 0), this request is less than need. Need for process P_1 is (1 4 5). Available resource is (2, 3, 0) and request is (1 1 0).

∴ Request < Available

(1 1 0) < (2 3 0)

After allocating (1 1 0) to process P_1 the need becomes as follows.

Process	Need		
	R_1	R_2	R_3
P_1	0	3	5
P_2	2	3	0
P_3	2	2	0

And available resource is (1 2 0).

Need of any process is never satisfied after granting the process P_1 request (1 1 0). So the system will be blocked. Therefore request of process P_1 (1 1 0) cannot be granted.

Example 3.5.3 *Consider following snapshot.*

Process	Allocation		Max		Available	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	1	2	4	2	1	1
P_2	0	1	1	2		
P_3	1	0	1	3		
P_4	2	0	3	2		

Whether the system is safe or unsafe ?

Solution : First calculate Need

Need = Max – Allocation

Process	Need	
	R_1	R_2
P_1	3	0
P_2	1	1
P_3	0	3
P_4	1	2

System is in safe state with safe sequence < P_2, P_4, P_1, P_3 >. System will complete its operation in this sequence.

Example 3.5.4 *Consider the following system snapshot using data structures in the Banker's algorithm, with resources A, B, C and D and process P_0 to P_4.*

	Max				Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	6	0	1	2	4	0	0	1					3	2	1	1
P_1	1	7	5	0	1	1	0	0								
P_2	2	3	5	6	1	2	5	4								
P_3	1	6	5	3	0	6	3	3								

Using Banker's algorithm, answer the following questions :

a) How many resources of type A, B, C and D are there ? [2]

b) What are the contents of the need matrix ? [3]

c) Is the system in a safe state ? Why ? [3]

d) If a request from process P_4 arrives for additional resources of (1, 2, 0, 0), can the Banker's algorithm grant the request immediately ? Show the new system state and other criteria. [7]

AU : Dec.-17, Marks 15

Solution : a) Resource types of A, B, C and D :

	Allocation			
	A	B	C	D
P_0	4	0	0	1
P_1	1	1	0	0
P_2	1	2	5	4
P_3	0	6	3	3
P_4	0	2	1	2
	6	11	9	10

	Available			
	A	B	C	D
	3	2	1	1
	6	11	9	10
Total	9	13	10	11

b) Need Matrix :

	A	B	C	D
P_0	2	0	1	1
P_1	0	6	5	0
P_2	1	1	0	2
P_3	1	0	2	0
P_4	1	4	4	4

c) Safe State :

Process	Need		Available	T/F	Available
P_0	(2, 0, 1, 1)	<	(3, 2, 1, 1)	T	(7, 2, 1, 2)
P_1	(0, 6, 5, 0)	<	(7, 2, 1, 2)	F	(7, 2, 1, 2)
P_2	(1, 1, 0, 2)	<	(7, 2, 1, 2)	T	(8, 4, 6, 6)
P_3	(1, 0, 2, 0)	<	(8, 4, 6, 6)	T	(8, 10, 9, 9)
P_4	1, 4, 4, 4)	<	(8, 10, 9, 9)	T	(8, 12, 10, 11)
P_1	(0, 6, 5, 0)	<	(8, 12, 10, 11)	T	(9, 13, 10, 11)

Safe sequence : P_0, P_2, P_3, P_4, P_1

d) Request from process P_4 (1, 2, 0, 0)

We assume that, requested is granted immediately. So calculate need matrix :

	A	B	C	D
P_0	2	0	1	1
P_1	0	6	5	0
P_2	1	1	0	2
P_3	1	0	2	0
P_4	0	2	4	4

Available = (2, 0, 1, 1)

Calculate safe sequence:

Process	Need		Available	T/F	Available
P_0	(2, 0, 1, 1)	<	(2, 0, 1, 1)	F	(2, 0, 1, 1)
P_1	(0, 6, 5, 0)	<	(2, 0, 1, 1)	F	(2, 0, 1, 1)
P_2	(1, 1, 0, 2)	<	(2, 0, 1, 1)	T	(3, 2, 6, 5)
P_3	(1, 0, 2, 0)	<	(3, 2, 6, 5)	T	(3, 8, 9, 8)
P_4	(0, 2, 4, 4)	<	(3, 8, 9, 8)	T	(4, 12, 10, 10)
P_0	(2, 0, 1, 1)	<	(4, 12, 10, 10)	T	(8, 12, 10, 11)
P_1	(0, 6, 5, 0)	<	(8, 12, 11, 11)	T	(9, 13, 10, 11)

Safe sequence : P_2, P_3, P_4, P_0, P_1

So requested is granted immediately.

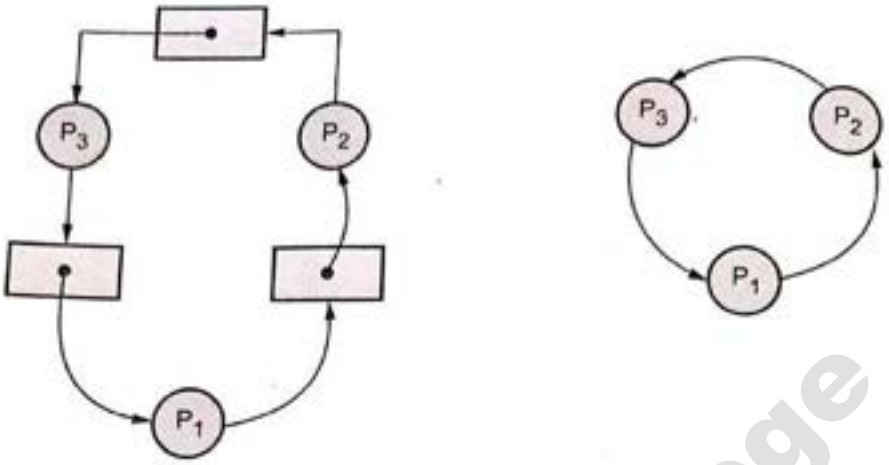
3.6 Deadlock Detection

AU : May-15

- Deadlock detection is the process of determining that a deadlock exists and identifing the processes and resources involved in the deadlock.
- If processes are blocked on resources for an inordinately long time, the detection algorithm is executed to determine whether the current state is a deadlock.
- Deadlock detection algorithms can incur significant runtime overhead.

3.6.1 Wait for Graph

- Any resource allocation graph with a single copy of resources can be transferred to a **wait for graph.**
- Fig. 3.6.1 shows resource allocation graph with corresponding wait for graph. The state of the system can be modeled by directed graph, called a wait for graph.



(a) Resource allocation graph

(b) Wait for graph

Fig. 3.6.1

- Wait for graph is a graph where each node represents a process. An edge $P_i \rightarrow P_j$ means that process P_i is blocked and waiting for process P_j to release a resource.
- The wait for graph of a system is always smaller than the resource allocation graph of that same system. There is a deadlock in a system if and only if there is a loop in the wait for graph of that system.
- Deadlock detection invloves two issuses :
  1. Maintenance of the wait for graph.
  2. Searching of the wait for graph for the presence of cycles.
- Deadlock detection requires overhead for run time cost of maintaining necessary information and executing the detection algorithm.
- For multiple instances of a resource type use an algorithm similar to banker's algorithm.

University Question

1. *Discuss how deadlocks could be detected in detail.* **AU : May-15, Marks 4**

3.7 Deadlock Recovery

- Once deadlock has been detected in the system, the deadlock must be broken by removing one or more of the four necessary conditions.
- Here one or more processes will have to be preempted, thus releasing their resources so that the other deadlocked processes can become unblocked.

1 Process termination

- Deadlock is removed by aborting a process. But aborting process is not easy. All deadlocked processes are aborted.
- Circular wait is eliminated by aborting one by one process. There will be lot of overhead. Deadlock detection algorithm must rerun after each process kill.
- Selection of process for aborting is difficult. Following parameters are considered :
  1. Priority of the process.
  2. What percentage the process finished its execution ?
  3. Resource used by process.
  4. Need of resources to complete process remaining operation.
  5. How many processes will need to be terminated ?
  6. Process type : Batch or interactive.

2 Resource preemption

- Some times, resource temporarily take away from its current process and allocate it to another process.
- For selecting victim, following factors are considered.
  1. Priority of the process. Higher priority process are usally not seleced.
  2. CPU time used by process. The process which is close to completion are usually not selected.
  3. The number of other process that would be affected if this process were selected as the victim.

3 Recovery through rollback

- When a process in a system terminates, the system performs a rollback by undoing every operation related to the terminated process.
- Checkpointing a process means that its state is written to a file so that it can be restarted later.
- Risk in this method is that the original deadlock may recover but the nondeterminancy of concurrent processing may ensure that this does not happen.

4 Starvation

- Starvation is one type situation in which a process waits for an event that might never occur in the system.
- Select the victim only for finite number of times. Use rollback method for selecting victim process.

3.8 Comparison between Detection, Prevention and Avoidance Methods of Handling Deadlock

AU : May-16

Parameters	Avoidance	Detection	Prevention
Resource allocation policy	Midway between that of detection and prevention.	Very liberal.	Conservative under commit resources.
Different schemes	Manipulate to find at least one safe path.	Invoke periodically to test for deadlock.	Preemption, resource ordering, requesting all resources at once.
Advantages	No preemption necessary.	Never delays process initiation.	No preemption necessary.
Disadvantages	Process can be blocked for long period.	Inherent preemption losses.	Delays process initiation.

Example 3.8.1 *Consider a system with five processes and three resource types and at time T_0 the following snapshot of the system has been taken :*

	Allocated			Maximum			Available		
Process Id	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	1	2	4	3	3	3	1	0
P2	2	1	2	3	2	2			
P3	4	0	1	9	0	2			
P4	0	2	0	7	5	3			
P5	1	1	2	11	2	3			

i) Determine the total amount of resources of each type.

ii) Compute the need matrix

iii) Determine if the state is safe or not using Banker's algorithm.

iv) Would the following request be granted in the current state?

a) P1 <3, 3, 1>

b) P2<2, 1,0>

Solution :

i) The total amount of resources of each type : Sum of allocated + Available

	Allocated			Available		
Process Id	R1	R2	R3	R1	R2	R3
P1	1	1	2	3	1	0

P2	2	1	2
P3	4	0	1
P4	0	2	0
P5	1	1	2
Total	8	5	7

	R1	R2	R3
The total amount of resources of each type =	8	5	0 (Allocated)
	3	1	0 (Available)
The total amount of resources of each type = R1 = 11, R2 = 6, R3 = 7	11	6	7

ii) **The need matrix**

Need matrix = Maximum – Allocated

	Maximum				Allocated				Need matrix		
Process Id	R1	R2	R3		R1	R2	R3		R1	R2	R3
P1	4	3	3		1	1	2		3	2	1
P2	3	2	2	–	2	1	2	=	1	1	0
P3	9	0	2		4	0	1		5	0	1
P4	7	5	3		0	2	0		7	3	3
P5	11	2	3		1	1	2		10	1	1

iii) State is safe or not using Banker's algorithm

Steps for safe state :

First Iteration :

1. Process P1 need is (3, 2, 1) and resource available with system is (3, 1, 0).

Here need is greater than available i.e.

Need (3, 2, 1) > Available (3, 1, 0)

Condition is false and algorithm selects next process for resource allocation.

2. Process P2 need is (1, 1, 0) and resource available with system is (3, 1, 0).

Here need is less than available i.e.

Need (1, 1, 0) < Available (3, 1, 0)

Condition is true and algorithm allocates required resources to that process (P2 Process). After completion of work, process P2 releases all the resources. So resources available with system are increased. So,

New available = Previous available + Allocated of particular process

= (3, 1, 0) + (2, 1, 2)

New available = (5, 2, 2)

3. Process P3 need is (5, 0, 1) and resource available with system is (5, 2, 2).

Here need is less than available i.e.

Need (5, 0, 1) < Available (5, 2, 2).

Condition is true and algorithm allocates required resources to that process (P3 Process). After completion of work, process P3 releases all the resources. So resources available with system are increased. So,

New available = Previous available + Allocated of particular process

= (5, 2, 2) + (4, 0, 1)

New available = (9, 2, 3)

4. Process P4 need is (7, 3, 3) and resource available with system is (9, 2, 3).

Here need is greater than available i.e.

Need (7, 3, 3) > Available (9, 2, 3)

Condition is false and algorithm selects next process for resource allocation.

5. Process P5 need is (10, 1, 1) and resource available with system is (9, 2, 3).

Here need is greater than available i.e.

Need (10, 1, 1) > Available (9, 2, 3)

Condition is false and algorithm selects next process for resource allocation.

One loop is completed; again we start for remaining three processes P1, P4 and P5

Second Iteration: for process P1, P4 and P5

1. Process P1 need is (3, 1, 2) and resource available with system is (9, 2, 3).

Here need is less than available i.e.

Need (3, 1, 2) < Available (9, 2, 3).

Condition is true and algorithm allocates required resources to that process (P1 Process). After completion of work, process P1 releases all the resources. So resources available with system are increased. So,

New available = Previous available + Allocated of particular process

= (9, 2, 3) + (1, 1, 2)

New available = (10, 3, 5)

2. Process P4 need is (7, 3, 3) and resource available with system is (10, 3, 5).

Here need is less than available i.e.

Need (7, 3, 3) < Available (10, 3, 5).

Condition is true and algorithm allocates required resources to that process (P4 Process). After completion of work, process P4 releases all the resources. So resources available with system are increased. So,

New available = Previous available + Allocated of particular process

= (10, 3, 5) + (0, 2, 0)

New available = (10, 5, 5)

3. Process P5 need is (10, 1, 1) and resource available with system is (10, 5, 5).

Here need is less than available i.e.

Need (10, 1, 1) < Available (10, 5, 5).

Condition is true and algorithm allocates required resources to that process (P5 Process). After completion of work, process P5 releases all the resources. So resources available with system are increased. So,

New available = Previous available + Allocated of particular process

= (10, 5, 5) + (1, 1, 2)

New available = (11, 6, 7)

Safe sequence = P2, P3, P1, P4, P5

iv) Would the following request be granted in the current state ?

a) P1 <3, 3, 1>

We assume that requested is granted for process P1. Then the need matrix will be

	Maximum				Allocated				Need matrix		
Process Id	R1	R2	R3		R1	R2	R3		R1	R2	R3
P1	4	3	3		4	4	3		0	0	0
P2	3	2	2		2	1	2		1	1	0
P3	9	0	2		4	0	1		5	0	1
P4	7	5	3	−	0	2	0	=	7	3	3
P5	11	2	3		1	1	2		10	1	1

But available with the system is less that the required one. So P1 requested is not granted.

b) P2 < 2, 1, 0 >

We assume that requested is granted for process P2. Then the need matrix will be

	Maximum				Allocated				Need matrix		
Process Id	R1	R2	R3		R1	R2	R3		R1	R2	R3
P1	4	3	3		1	1	2		3	2	1
P2	3	2	2		4	2	2		0	0	0
P3	9	0	2	–	4	0	1	=	5	0	1
P4	7	5	3		0	2	0		7	3	3
P5	11	2	3		1	1	2		10	1	1

New available = (3, 1, 0) – (2, 1, 0) = (1, 0, 0)

Compute the safe sequence :

1. Need of P1 (3, 2, 1) is greater than available (1, 0, 0). Condition is false. Select next process.
2. Need of P2 (0, 0, 0) is less than available (1, 0, 0). Condition is true. Then
   New available = (1, 0, 0) + (4, 2, 2) = (5, 2, 2).
3. Need of P3 (5, 0, 1) is greater than available (5, 2, 2). Condition is false. Select next process.
4. Need of P4 (7, 3, 3) is greater than available (5, 2, 2). Condition is false. Select next process.
5. Need of P5 (10, 1, 1) is greater than available (5, 2, 2). Condition is false. Select next process.

Again repeat for second iteration

1. Need of P1 (3, 2, 1) is greater than available (5, 2, 2). Condition is true. Then
   New available = (5, 2, 2) + (1, 1, 2) = (6, 3, 4)

There is no safe sequence because need of process P3, P4 and P5 is greater than the available. So request is not granted immediately.

Example 3.8.2 *Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show the system is deadlock-free.*

AU : May-16, Marks 8

Solution : Yes, this system is deadlock-free.

Maximum resources R1 = 4

Three processes : P1, P2, P3

Max matrix :

	P1	P2	P3
A	2	2	2

Allocation Matrix :

	P1	P2	P3
A	1	1	1

Available Resources = 1

Need Matrix :

	P1	P2	P3
A	1	1	1

- This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.
- Need of P1 process is one resource, so it get from available. After getting one resources, process P1 releases both the resources which it hold.
- So available resource will become 2 and it allocate one by one to all remaining process. In this way, system fulfill all the needs of process. System will not enter into deadlock state.

3.9 Live Lock

AU : Dec.-17

- Livelock : If two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work.
- These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.
- This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work.

This is similar to deadlock in that no progress is made but differs in that neith process is blocked or waiting for anything.

- Livelock is a risk with some algorithms that detect and recover from deadlock. more than one process takes action, the deadlock detection algorithm can repeatedly triggered.
- Fig. 3.9.1 shows process of livelock.

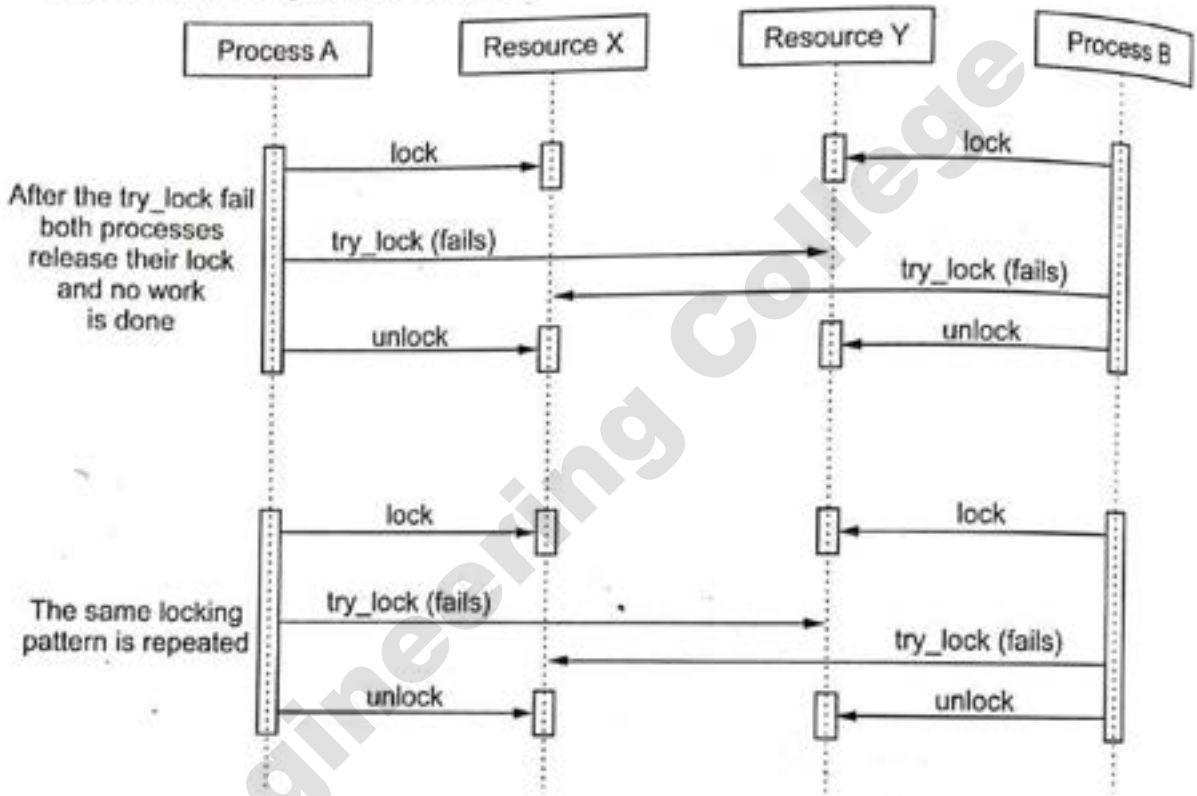


Fig. 3.9.1

University Question

1. *With example elucidate livelock.* **Dec.-17, Marks 4**

Two Marks Questions with Answers

Q.1 State the conditions for deadlock.

Ans. :

a. Mutual exclusion b. Hold and wait

c. No preemption d. Circular wait

1. Atleast one resource must be held in a nonsharable mode.
2. A process holding atleast one resource is waiting for more resources held by other processes.

3. Resources cannot be preempted.
4. There must be a circular waiting.

Q.2 What is deadlock ? What are the four necessary conditions for deadlock ?

Ans. : A deadlock occurs when two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes.

Four necessary conditions :

1. Mutual exclusion 2. Hold and wait 3. No preemption 4. Circular wait

Q.3 What is deadlock state ?

Ans. : If there is any process deadlock in a state, then that state is called deadlock state.

Q.4 Define the deadlock problem.

Ans. : A situation where every process is waiting for an event that can be triggered only by another process.

Q.5 What is a deadlock ? **AU : CSE/IT : Dec.-10**

Ans. : A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Usually the event is release of a currently held resource.

Q.6 Write the three ways to deal the deadlock program. **AU : CSE/IT : May-11**

Ans. : Three ways to deal with the deadlock problem :

1. Use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
2. Allow the system to enter a deadlock state, detect it and recover.
3. Ignore the problem altogether and pretend that deadlocks never occur in the system.

Q.7 Is it possible to have a deadlock involving only one process ? State your answer.

Ans. : No. This follows directly from the hold-and-wait condition.

Q.8 List two examples of deadlocks that are not related to a computer system environment.

Ans. : a. Two cars crossing a single-lane bridge from opposite directions.

b. A person going down a ladder while another person is climbing up the ladder.

Q.9 What is resource-allocation graph ?

Ans. : Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph.

Q.10 Define safe state.

Ans. : A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.

Q.11 When is a set of processes deadlocked ?

Ans. : Resource deadlock : Each process requests resources held by another process in the set and it must receive all the requested resources before it can become unblocked. Communication deadlock : Each process is waiting for communication from another process and will not communicate until it receives the communication for which it is waiting.

Q.12 Define request edge and assignment edge.

Ans. : There is a request edge from process P to resource R if and only if P is blocked waiting for an allocation of R. There is an assignment edge from resource R to process P if and only if P is holding an allocation of R.

Q.13 What is a knot ?

Ans. : A strongly connected sub-graph of a directed graph, such that starting from any node in the subset it is impossible to leave the knot by following the edges of the graph.

Q.14 What is hold and wait ?

Ans. : Hold and Wait : A process must be holding a resource and waiting for another.

Q.15 What is banker's algorithm ?

Ans. : Banker's algorithm is a deadlock avoidance algorithm that is applicable to a resource-allocation system with multiple instances of each resource type.

Q.16 Which are two options to break a deadlock ?

Ans. : There are two options for breaking a deadlock :

1. Process termination : To abort one or more processes to break the circular wait.
2. Resource preemption : To preempt some resources from one or more of deadlock processes.

❑❑❑

UNIT - III

4 Storage Management

Syllabus

Main Memory - Background, Swapping, Contiguous Memory Allocation, Paging, Segmentation, Segmentation with paging, 32 and 64 bit architecture Examples; Virtual Memory - Background, Demand Paging, Page Replacement, Allocation, Thrashing; Allocating Kernel Memory, OS Examples.

Contents

4.1 Main Memory

- Memory is used to store information. Secondary storage memory is long term persistent memory that is held in storage device such as disk drive.
- Primary memory is faster than secondary memory. Memory manager is responsible for allocating primary memory to processes.
- Memory management is performed by both software and special purpose hardware. The memory manager is an operating system component. Managing the sharing of primary memory and minimizing memory access time are the basic goals of the memory manager.

Primary memory requirements

1. **Access time :** It should be as small as possible. This need influences both software and hardware design.
2. **Size :** Size must be as large as possible. It can accomodate many programs into memory.
3. **Cost :** Cost of the memory is less than the total cost of the computer.

4.1.1 Memory Management Function

1. Allocate primary memory space to processes.
2. Minimize access time.
3. Determining allocation policy for memory.
4. Deallocation technique and policy.

4.1.2 Basic Hardware of Memory

- CPU can access content of main memory and register directly. If the data is not available into the memory, it load into memory from disk.
- Registers are built on the processor. Using one cycle of the CPU clock, processor access data from register.
- Accessing memory may take many CPU clock cycle. Mismatch of speed between CPU and memory is overcome by using cache memory.
- The use of base and bound (limit) registers are restrict a process memory references upto a certain limit. Hardware is used to protect user address space.
- Each process requires its own address space operating system define legal address for each process. Maximum and minimum limit is also decided so that process can access only these legal address.

- Fig. 4.1.1 shows the protection of process by using registers.
- An address space is the set of addresses that a process/program can use to address main memory. Each process has its own address space.
- User programs are loaded into consecutive memory locations by using base and limit register. When process is executing, the base register is loaded with the physical address where its program begins in memory and the limit register is loaded with the length of the program.

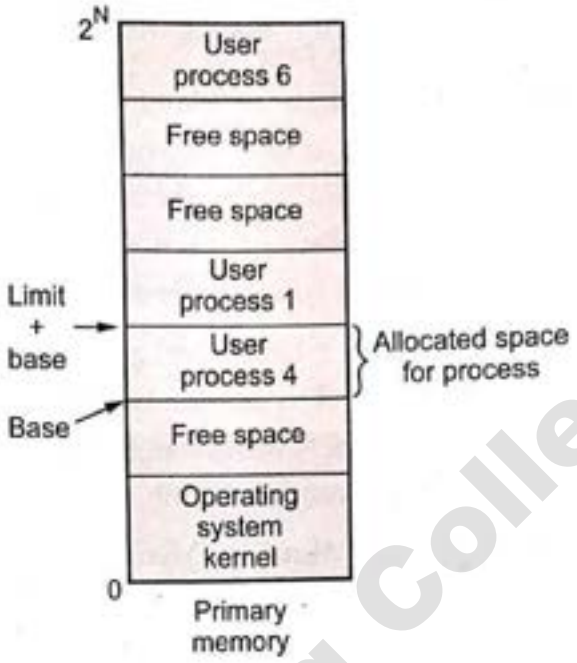


Fig. 4.1.1 Address space and memory

- Memory protection is used to avoid interference between programs existing in main memory. The memory protection hardware compares every memory address used by the program with the contents of two registers (base and limit) to ensure that it lies with the allocated memory area.
- Multiple hardware memories are used to provide a larger address space.
- The simplest method of memory protection is adding two registers to the CPU. This works good for all memory is allocated contiguously. Non-contiguous memory is harder to protect.
- When a process reads from or writes to address, the memory decoder adds on the value of the base register. The actual operation of read or write to address = base register + limit register.
- If the input address is higher than limit or lower than zero, then the memory hardware generates error. This is informed to the operating system by using interrupt. Processes can only access memory within these limits.
- Each process has its own pair of base register and limit register.

4.1.3 Address Space Mapping

- Secondary storage device stores program in binary executable format. Before executing, the program is loaded into the main memory.
- Most of the operating systems allow a user process to store in any section of the main memory. Source program uses symbolic addresses.
- Fig. 4.1.2 shows processing of user program.

- Binding of instruction and data to main memory address is following ways :
  1. Compile time 2. Load time 3. Execution time
- **Compile time :** Source program is translated at compile time to produce a relocatable object module. At compile time, the translator generates code to allocate storage for the variable. This storage address is used for code reference. Target address is unknown at compile time, it cannot be bound at compile time. Example of compile time binding is MS DOS.com programs.
- **Load time :** Compiler generates relocatable code if compile time binding is not performed. The loader modifies the addresses in the load module at load time to produce the executable image stored in main memory. Final binding is delayed until program load time.
- **Execution time :** Memory address of the program is changed at execution time, then execution time binding is used. Binding is delayed until the run time of the program. Normally all operating system uses execution time binding. Special hardware is used for execution time binding.

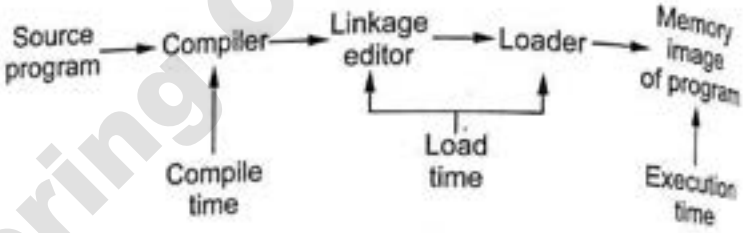


Fig. 4.1.2 Processing of user program

- Memory allocation and deallocation is done using run-time support of the programming language in which a program is coded. Allocation and deallocation requests are made by calling appropriate routines of the run time library.
- Kernel is not involved in this kind of memory management.

4.1.4 Concept of Memory Address

- **Logical address** is generated by the CPU. This address is also called virtual address.
- Main memory address uses **physical address**. This address also called real address.
- **Logical address space :** Set of all logical addresses generated by a program.
- Logical address and physical address is identical when load time and compile time address binding is performed. The execution time address binding generates different physical and logical address.
- Memory Management Unit (MMU) is responsible for run time address mapping from vitual to physical address.

Dynamic Relocation

- Base register is sometimes called as a relocation register. The value of the relocation register is added to every address generated by a user process at the time it is sent to main memory.
- User can load a process with only absolute addresses for instructions and data, only when those specific addresses are free in main memory. Program's instruction, data and any other data structure required by the process can be accesssed easily if the addresses are relative.
- Fig. 4.1.3 shows dynamic relocation. User programs never reads the main memory physical address.
- Dynamic relocation requires extra hardware. It is mapping of the virtual address space to the physical address space at run time.
- Dynamic relocation makes it possible to move a partially executed process from one area of main memory into another without affecting other process.

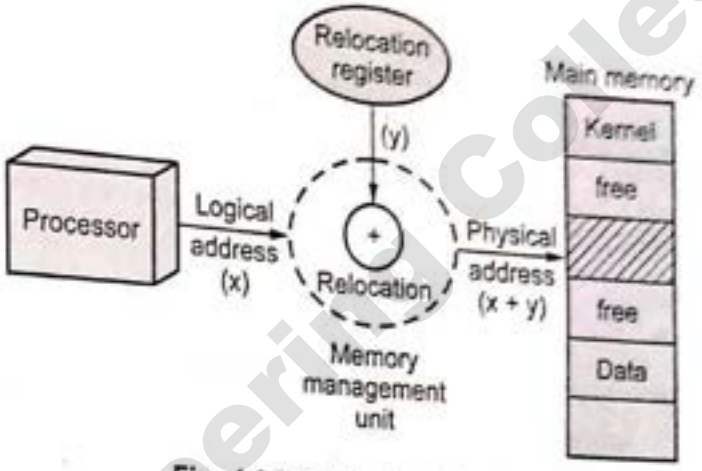


Fig. 4.1.3 Dynamic relocation

- Problem with relocation is that, it is necessary to perform an addition and a comparison on every memory reference.
- For good memory management, logical address space is bound with a separate physical address space.

4.1.5 Dynamic Loading

- Dynamic loading is used for better memory space utilization. User program size is large as compared to the memory size. Program or process are dynamically loaded into memory as per required.
- With dynamic loading, a routine is not loaded until it is called. All routines are kept on storage disk in a relocatable load format. The main program is loaded into memory and is executed.

Advantages

1. Unused routine is never loaded.
2. Special support is not required from OS.
3. Used with error routines.

4.2 Swapping

- Keeping all processes in memory all the time requires a large amount of main memory. For execution of the process, it must be swapped.
- **Swapping** is method of temporarily removing inactive process/programs from the physical memory.
- Inactive program means, which is neither executing on the CPU, nor performing an I/O operation.

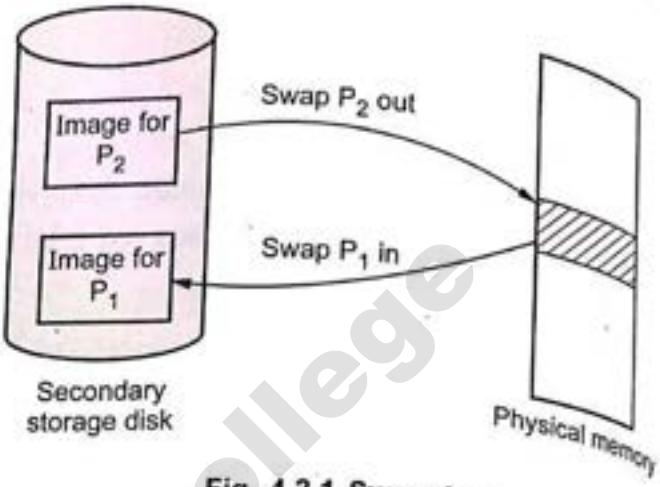


Fig. 4.2.1 Swapping

- Fig. 4.2.1 shows swapping.
- When process P_2 send requests for an I/O operation, OS blocked P_2 process and will not return to the ready state for long period of time. Process P_2 is in blocked state. Process manager place the process P_2 in blocked state and inform to the memory manager regarding this action.
- Process manager moves a blocked process P_1 into the ready state and inform to memory manager. Process P_1 is loaded into main memory by memory manager when there is space available.
- Swapped out process is stored on the secondary storage disk. It contains executable image with code, data and stack.
- Relocation hardware supports for swapping. Because of address binding problem, swapping is difficult without relocation hardware.
- Round robin and priority based CPU scheduling algorithm uses the swapping concept.
- Time sharing system uses swapping. In time sharing, kernel can estimate when a program is likely to be scheduled next.
- A swapped out process will occupy same memory location when process swap in by OS. But this totally depends on address binding. If address binding is load time or assembly time, then swapped out process uses same memory location when swap-in. But execution-time binding uses different memory location.
- Secondary storage is required to implement the swapping technique.

- Content switching time increases when swapping increases. Major part of the swap time is transfer time. Swapping increases the operating system overhead due to the disk I/O involved.
- When swapping creates multiple holes in memory, it is possible to combine them all into one by moving all the processes to one side of memory. This technique is known as **memory compaction.**
- Compaction requires a lot of CPU time, so most operating systems not used this technique.
- Compaction means movement of processes in the memory during their execution. It is sometimes called dynamic relocation of a program.

4.3 Contiguous Memory Allocation with Fixed Size Partitions

AU : Dec.-15, May-17

- In a fixed size partitioning of the main memory all partitions are of the same size. The operating system divides main memory into a number of fixed size partition. Each partition holds a single program.
- Fixed sized partitions are relatively simple to implement. Degree of multiprogramming depends on the number of partitions.
- Normally main memory is divided into two partitions :
  1. For resident program
  2. For user processes
- Batch operating system uses the fixed size partition scheme. The operating system keeps the record of memory allocation and deallocation in the form of table. Initially all the memory is available for user processes.
- When the process arrives in the system and needs memory, operating system search for large hole for this process. Hole is one large block of free available memory. If any free hole is found, process is allocated to the free hole of memory as is needed.
- After allocating number of holes for the processes, a set of various size holes is scattered throughout memory at any given time. When a process arrives and searches for (memory) set of holes. But the holes must be large enough to accomodate the process.
- Fixed sized partitions are simple to implement. Any process whose size is less than or equal to the partition size can be loaded into any available partition.
- Fig. 4.3.1 shows an example of fixed partitioning of a memory.

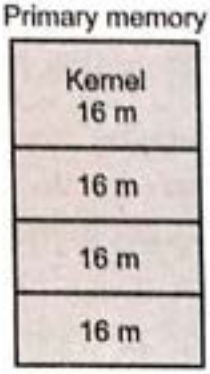


Fig. 4.3.1 Fixed size partition

- There are two difficulties with the use of equal size fixed partitions :
  1. A program may be too big to fit into a partition.
  2. Memory utilization is extremely inefficient.
- First difficulty is solved by using overlays. Overlays involves moving data a program segments in and out of memory.

Advantages and disadvantages of fixed partition size

1) Advantages

1. Simple to implement.
2. Does not require expertise to understand and use such a system.
3. Less overhead.

2) Disadvantages

1. Memory is not fully utilized.
2. Poor utilization of processors.
3. User's process being limited to the size of available main memory.
4. Requires contiguous loading of entire program.

4.3.1 Dynamic Memory Partitions

- Memory partitions are of variable length and number. Processes are loaded into the size nearest to its requirements.
- Fig. 4.3.2 shows the effect of dynamic memory partitioning. Initially in main memory, only OS is loaded and rest of the memory is free. Memory size is 64 Mbytes. How the process is loaded with dynamic partitions is shown below. After loading all the process some free space remains at one side of the memory.
- Four process/programs are loaded into memory. These program starts loading where the OS ends. After loading four programs, it leaves a hole at the end of memory that is too small for a next process.
- Operating system swap out program 2 and leaves sufficient room to load new program into memory. The size of newly loaded program is smaller than the program in memory. Again another small hole is created.

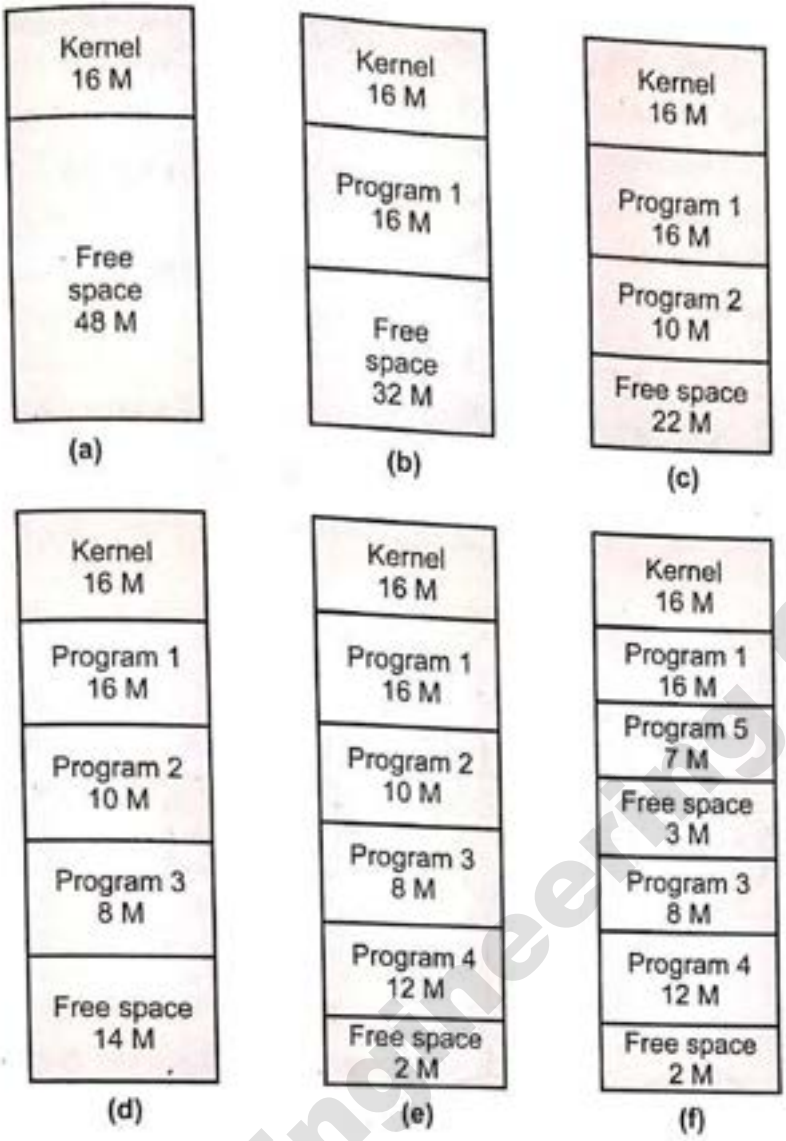


Fig. 4.3.2 Dynamic partitioning example

4.3.2 Memory Protection and Mapping

- Memory allocated to the process need to be protected against interference by other processes. Memory protection provides this facility.
- Relocation register and limit register are used for memory protection. Physical address is stored in the relocation register and range of virtual address is stored in the limit register.
- Fig. 4.3.3 shows logical address space.
- Use of registers method for memory protection requires proper arrangement. This method has to be used individually for each memory area allocated to a process.
- Every logical address generated by the CPU is checked against these registers. A protection violation interrupt is raised if any of these checks fail.

- Operating system load a base register and limit register. OS uses privileged instruction for this purpose. Operating system executes in monitor mode and privilege instruction also executes in privilege mode.
- Fig. 4.3.4 shows memory protection by using relocation and limit registers.

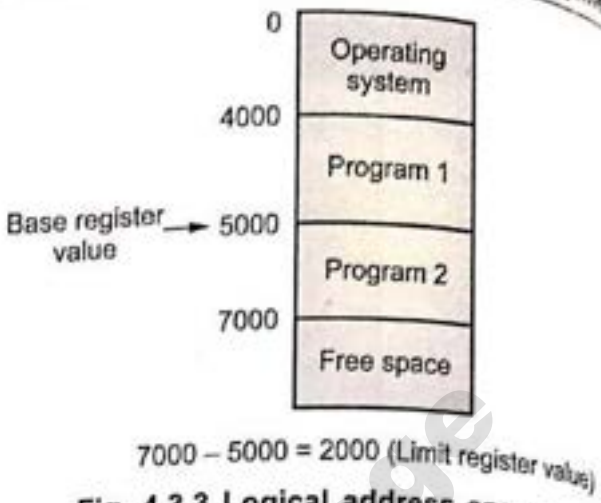


Fig. 4.3.3 Logical address space

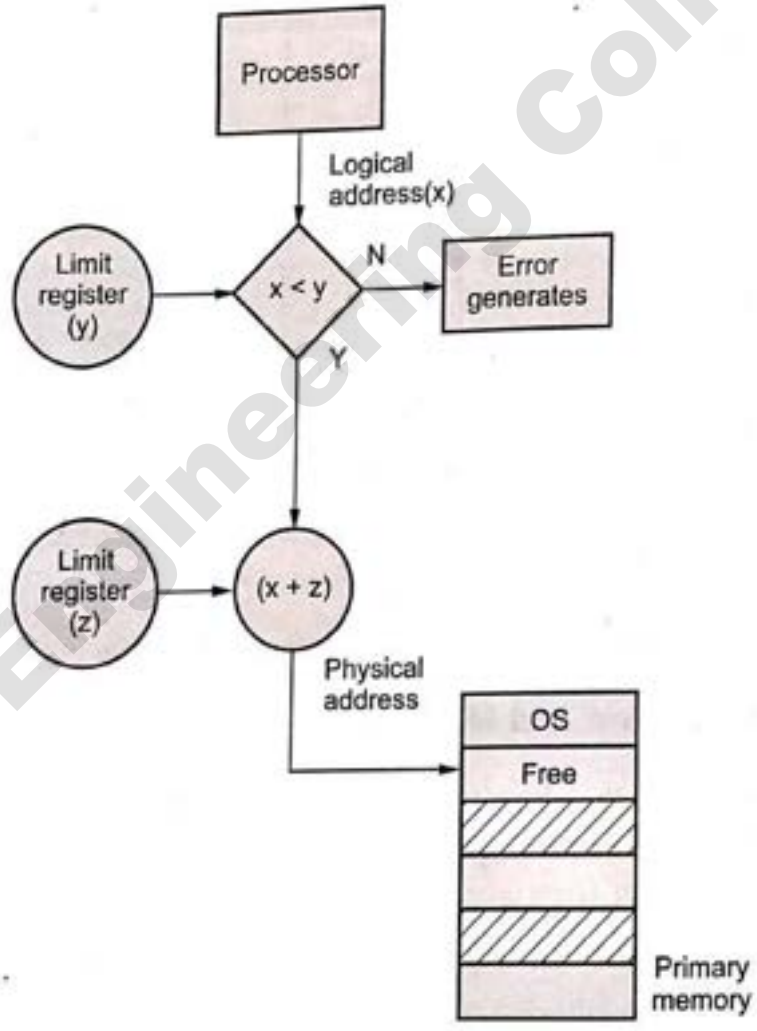


Fig. 4.3.4 Memory protection hardware

- Operating system prevents the user programs from changing the content of the registers. While executing in the monitor mode, operating system is given unrestricted access to both monitor and user's memory.

- In a multiprogramming environment, protection of main memory is essential. Paging or segmentation or both in combination provides an effective means of managing main memory.
- An example of the hardware support that can be provided for memory protection is that of the IBM system/370 family of machines, on which VMS runs. Microsoft Windows 3.1/95 offer memory protection. Microsoft WinNT also offers memory protection. In UNIX, almost impossible to corrupt another process memory.

University Question

1. *Discuss the given memory management technique with diagram : Partition Allocation Methods.* **AU : Dec.-15, May-17, Marks 8**

4.4 Contiguous Memory Allocation with Variable Size Partitions

AU : Dec.-12, 16, 17

- The use of unequal size partitions provides a degree of flexibility to fixed partitioning. In dynamic partitioning, the partitions are of variable length and number.
- In noncontiguous memory allocation, a program is divided into blocks that the system may place in nonadjacent slots in main memory.
- This allocation method do not suffer from internal fragmentation, because a process partition is exactly the size of the process.
- Fig. 4.4.1 shows noncontiguous memory allocation method. Operating system maintain the table which contains the memory areas allocated to process and free memory. Memory management unit use this information for allocating processes.
- Table contains information about memory starting address for process/program and their size.

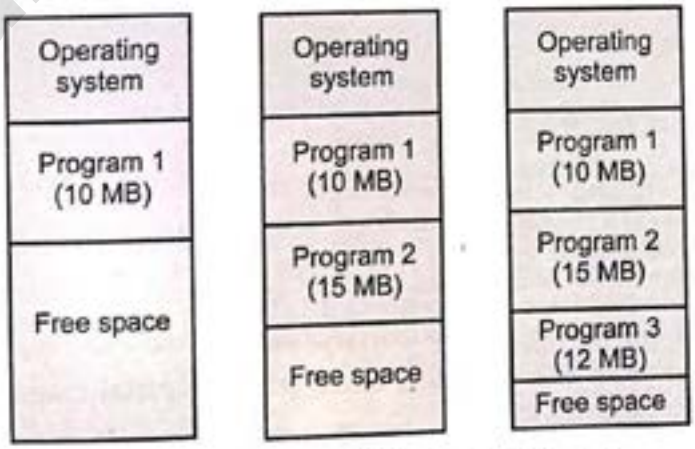


Fig. 4.4.1 (a) Variable size partition

- CPU sends the logical address of the process to the MMU and the MMU uses the memory allocation information stored in the table for calculating logical address. We called this address as effective memory address of the data/instruction.

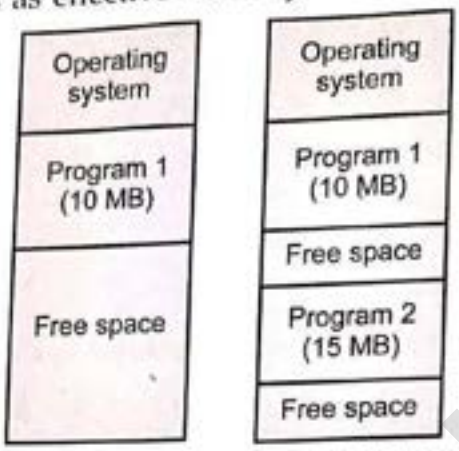


Fig. 4.4.1 (b) Noncontiguous memory allocation

4.4.1 Difference between Contiguous and Noncontiguous Memory Allocation

Sr. No.	Contiguous allocation	Noncontiguous allocation
1.	Program execution take place without overhead.	Address translation is overhead.
2.	Swapped-in processes are placed in the original area.	Swapped-in processes can be placed any where in memory.
3.	Suffer from internal fragmentation.	Only paging, suffers from internal fragmentation.
4.	Allocates single area of memory for process.	Allocates more than one block of memory for process.
5.	Wastage of memory.	No wastage of memory.

4.4.2 Fragmentation

- Fragmentation are of two types :
  1. Internal fragmentation.
  2. External fragmentation.
- In fragmentation, OS cannot use certain area of available memory.

Internal fragmentation

- There is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition is called as internal fragmentation.
- Fig. 4.4.2 shows an internal fragmentation.
- To keep track of this free hole is overhead for system.

External fragmentation

- It occurs when enough total main memory space exists to satisfy a request, but it is not contiguous, storage is fragmentated into a large number of small holes.
- Fig. 4.4.3 shows external fragmentation.
- Following are the solution for external fragmentation :
  1. Compaction
  2. Logical address space of a process/program

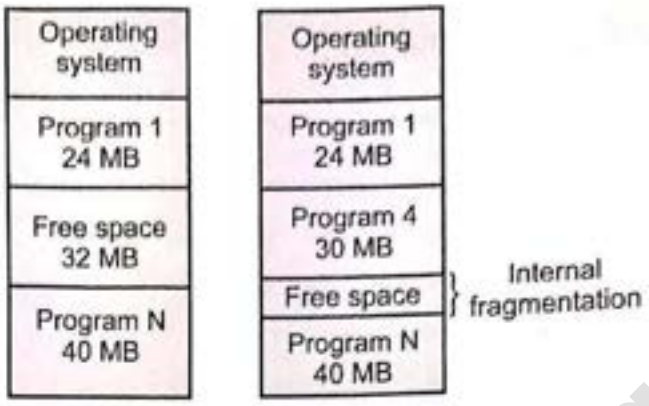


Fig. 4.4.2 Internal fragmentation

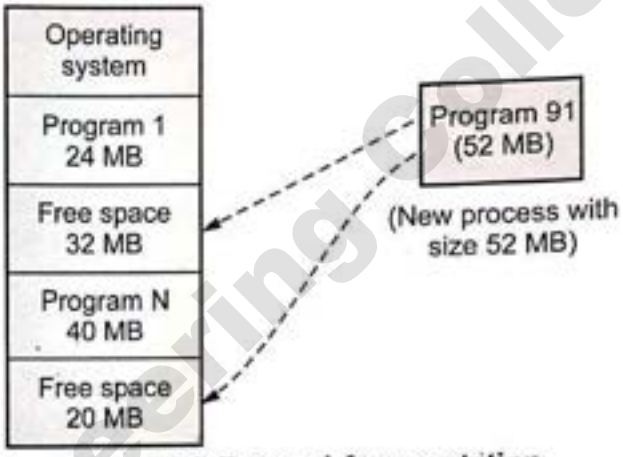


Fig. 4.4.3 External fragmentation

4.4.3 Difference between Internal and External Fragmentation

Internal Fragmentation	External Fragmentation
Internal fragmentation is the area occupied by a process but cannot be used by the process.	External fragmentation exists when total free memory is enough for the new process but it's not contiguous and can't satisfy the request.
First fit and best fit memory allocation does not suffer from internal fragmentation.	First fit and best fit memory allocation suffers from external fragmentation.
In fixed partitioning, inefficient use of memory due to internal fragmentation.	In dynamic partitioning, inefficient use of processor due to need for compaction to counter external fragmentation.
Paging's suffer from internal fragmentation.	Paging does not suffer from external fragmentation
Segmentation does not suffer from internal fragmentation.	Segmentation suffer from external fragmentation

4.4.4 Compaction

- Compaction solves problem of external fragmentation. Fig. 4.4.4 shows compaction.
- Operating system moves all the free holes to one side of main memory and creates large block of free size.
- It must be performed on each new allocation of process to memory or completion of process for memory. System must also maintain relocation information.
- All free blocks are brought together as one large block of free space. Compaction requires dynamic relocation.

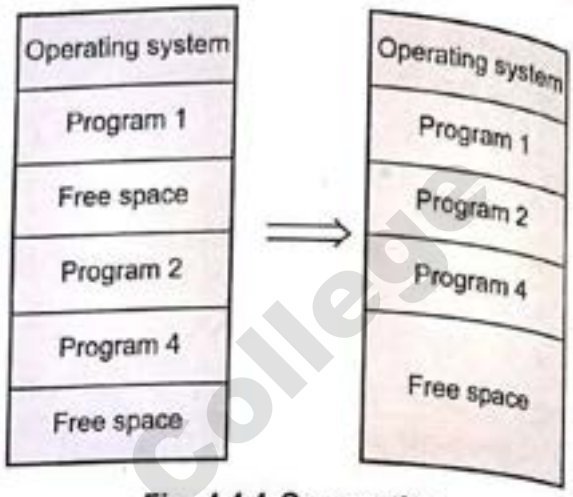


Fig. 4.4.4 Compaction

- Compaction has a cost and selection of an optimal compaction strategy is difficult. One method for compaction is swapping out those processes that are to be moved within the memory and swapping them into different memory locations.
- Compaction is not always possible. Compaction is time consuming method and wastage of the CPU time.

Garbage collection

- Some programs use dynamic data structures. These programs dynamically use and discard memory space. Technically, the deleted data items release memory locations.
- But, in practice the OS does not collect such free space immediately for allocation. This is because that affects performance. Such areas, therefore are called **garbage.**
- When such garbage exceeds a certain threshold the OS would not have enough memory available for any further allocation.

4.4.5 Placement Algorithms

- In an environment that supports dynamic memory allocation, the memory manager must keep a record of the usage of each allocatable block of memory. This record could be kept by using almost any data structure that implements linked lists.

- An obvious implementation is to define a free list of block descriptors, with each descriptor containing a pointer to the next descriptor, a pointer to the block and the length of the block.
- The memory manager keeps a free list pointer and inserts entries into the list in some order conductive to its allocation strategy. A number of strategies are used to allocate space to the processes that are competing for memory.
- Fig. 4.4.5 shows the placement algorithm.

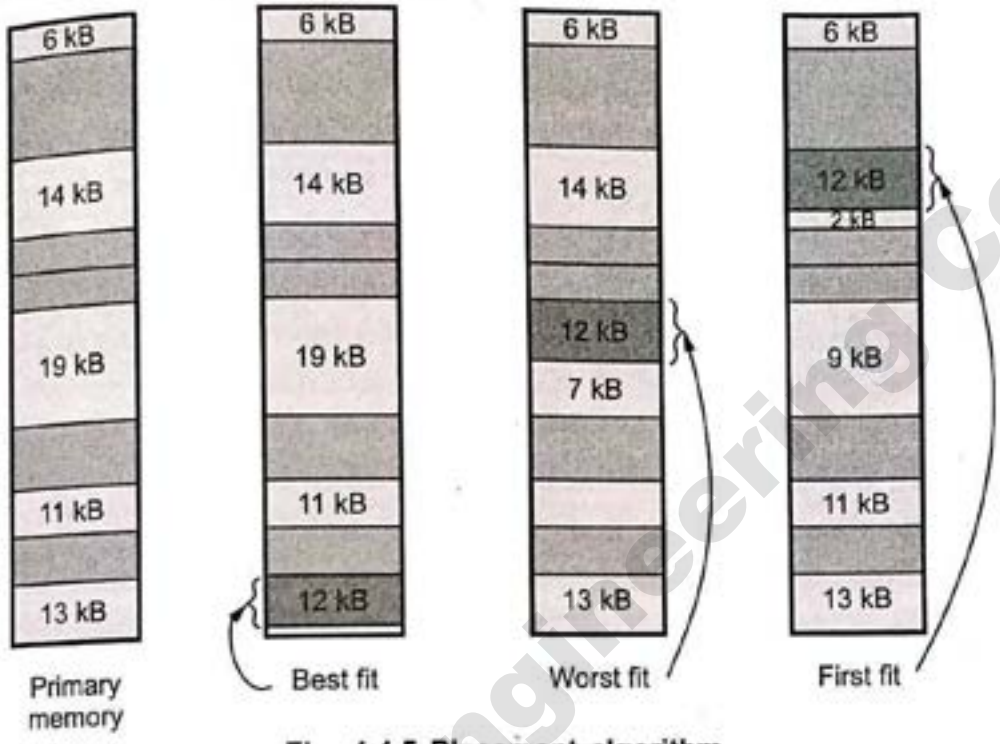


Fig. 4.4.5 Placement algorithm

1. **Best fit :** The allocator places a process in the smallest block of unallocated memory in which it will fit. For example, suppose a process requests 12 kB of memory and the memory manager currently has a list of unallocated blocks of 6 kB, 14 kB, 19 kB, 11 kB and 13 kB blocks. The best-fit strategy will allocate 12 kB of the 13 kB block to the process.
   - Best fit chooses the block that is closest in the size to the request.

Problems of best fit

- It requires an expensive search of the entire free list to find the best hole.
- More importantly, it leads to the creation of lots of little holes that are not big enough to satisfy any requests. This situation is called *fragmentation* and is a problem for all memory-management strategies, although it is particularly bad for best-fit.

Solution : One way to avoid making little holes is to give the client a bigger block than it asked for. For example, we might round all requests up to the next larger

multiple of 64 bytes. That doesn't make the fragmentation go away, it just hides it. Unusable space in the form of holes is called *external fragmentation.*

2. **Worst fit :** The memory manager places a process in the largest block of unallocated memory available. The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that compared to best fit ; another process can use the remaining space. Using the same example as above, worst fit will allocate 12 kB of the 19 kB block to the process, leaving a 7 kB block for future use.
3. **First fit :** There may be many holes in the memory, so the operating system, to reduce the amount of time it spends analyzing the available spaces, begins at the start of primary memory and allocates memory from the first hole it encounters large enough to satisfy the request. Using the same example as above, first fit will allocate 12 kB of the 14 kB block to the process.
4. **Next fit :** The first fit approach tends to fragment the blocks near the beginning of the list without considering blocks further down the list. Next fit is a variant of the first-fit strategy. The problem of small holes accumulating is solved with next fit algorithm, which starts each search where the last one left off, wrapping around to the beginning when the end of the list is reached (a form of one-way elevator).

- Notice in the diagram above that the best fit and first fit strategies both leave a tiny segment of memory unallocated just beyond the new process.

Example 4.4.1 *Given memory partitions of 100 K, 500 K, 200 K, 300 K and 600 K (in order) how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 K, 417 K, 112 K and 426 K (in order) ? Which algorithm makes the most efficient use of memory ?* **AU : Dec.-12**

Solution : First-fit :

212 K is put in 500 K partition

417 K is put in 600 K partition

112 K is put in 288 K partition (new partition 288 K = 500 K – 212 K)

426 K must wait.

Best-fit :

212 K is put in 300 K partition

417 K is put in 500 K partition

112 K is put in 200 K partition

426 K is put in 600 K partition

Worst-fit :

212 K is put in 600 K partition

417 K is put in 500 K partition

112 K is put in 388 K partition (600 K – 212 K)

426 K must wait

In this example, Best-fit turns out to be the best.

University Questions

1. *Explain the difference between internal and external fragmentation.* **AU : Dec.-16, Marks 8**
2. *In a variable partition scheme, the operating system has to keep track of allocated and free space. Suggest a means of achieving this. Describe the effects of new allocations and process terminations in your suggested scheme.* **AU : Dec.-17, Marks 5**

4.5 Paging

AU : Dec.-15,16, May-15,17

- In paging, operating system divides each incoming programs into pages of equal size. The sections of a disk are called block or sectors. The sections of main memory are called page frames. One sector will hold one page of job instructions and fit into one page frame of memory.
  - In paging, logical address space of a program can be noncontiguous. It solves external fragmentation problem.
  - The relation between virtual addresses and physical memory addresses given by page table.
- Fixed sized blocks are called frames and breaking of logical memory into blocks of same size called pages.
- Memory manager prepares following things before executing a program :
  1. Find out the number of pages in the program.
  2. Free space in the main memory.
  3. Loading of all the programs pages into memory.
- Pages are not loaded continuosly in main memory. Each page can be stored in any available page frame anywhere in main memory. Memory manager keeps the track of pages of program. Paging avoids external fragmentation and need for compaction.
- Memory manager uses three tables to keep track of process and memory.
  1. **Job table :** It stores the size of the active job, and memory location where its page table is stored.
  2. **Page map table :** It contains vital information about each page. PMT contains apage number and corresponding page frame memory address. It includes only one entry per page. Page numbers are in sequential order.
  3. **Memory map table :** MMT is used to store page frame location and its status.

- Page size depends upon underlying hardware. Operating system maintains a page table for each process. The page table shows the frame location for each page of a process. Fig. 4.5.1 shows the virtual address format for paging system.

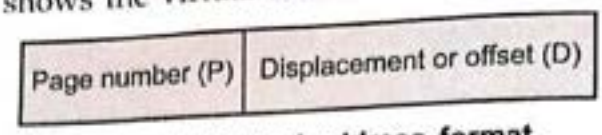


Fig. 4.5.1 Virtual address format

- Processor hardware performs the logical to physical address translation. Logical address contains page number and offset. The physical address contains frame number and offset. Offset is a relative factor. It is used to locate that line within its page frame.
- Fig. 4.5.2 shows address translation.
- Suppose a system uses n-bit for representing both physical and virtual address. The page number is represented by the most significant bit (n - m bits) and the displacement is represented by m bit.
- The table, which holds virtual address to physical address translations is called page table. As displacement is constant, so only translation of virtual page number to physical page is required.

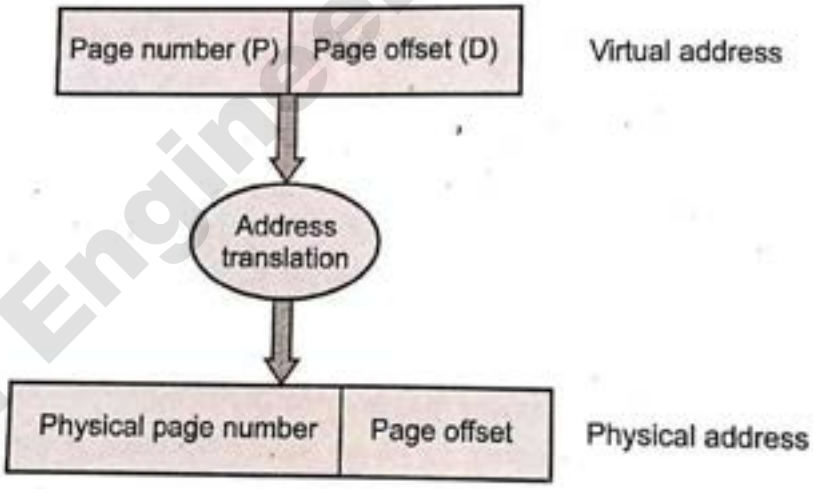


Fig. 4.5.2 Address translation

- Fig. 4.5.3 shows paging hardware. CPU generates logical addresses containing page number and an offset. This page number is used to retrieve the frame number from a page table which gives the base address so the physical address is calculated as base + offset.
- Paging implementation requires CPU and operating system support. Page table may itself be resident in main memory.
- Active process is loaded into memory and it occupies number of pages. This set of pages forms its resident set. Hardware defined the page size.

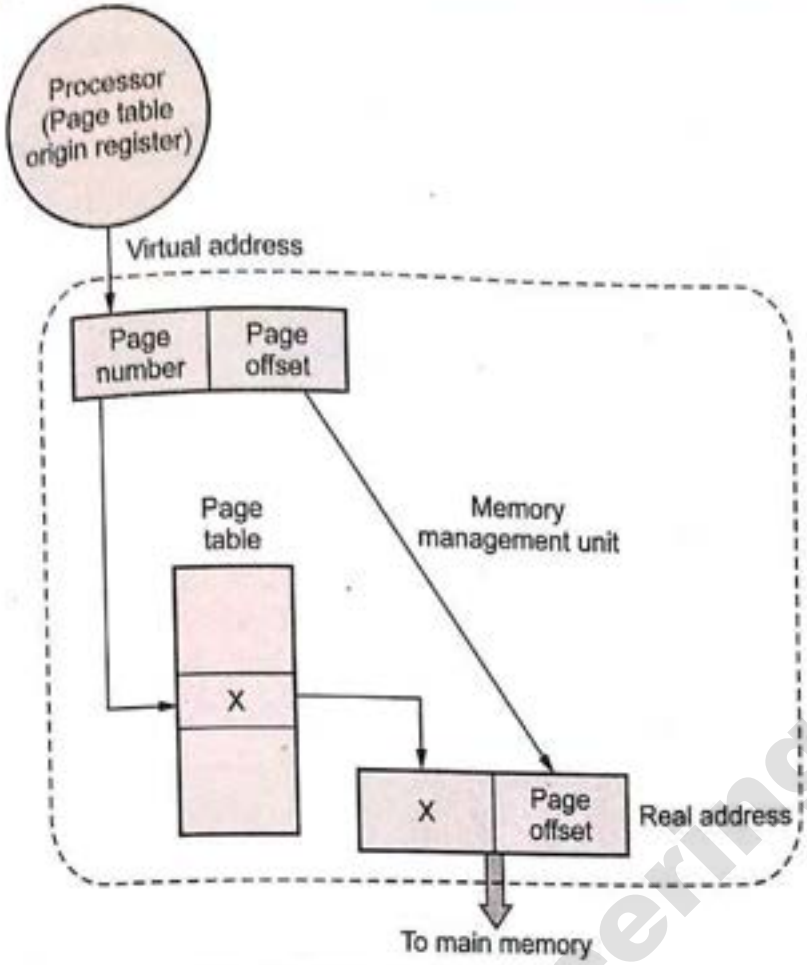


Fig. 4.5.3 Hardware for paging

- The processor generates virtual (logical) address and it consists of two parts :
  1. Page number 2. Page offset
- Page table contains page number and index frame number. Page number is used as an index into a page table.
- In case of simple partition, a logical address is the location of a word relative to the beginning of the program the processor translates that into a real address.
- But in the paging method the logical to real address translation is still done by processor hardware. The processor must know how to access the page table of the current process.
- The operating system partitions the memory into areas called **page frames.** Each page frame size is equal. Page size is normally power of 2.
- Fig. 4.5.4 shows a paging model. Operating system maintains the free frame list. During execution of process the memory management unit consults the page table to perform address translation.
- The physical address is the portion of the primary memory allocated to the process. The page frames allocated to the process need not be contiguous.

- The aim of the paging system is to identify the set of pages needed for the process current locality and then to load only those pages into page frames in primary memory.
- The logic of the address translation process in paged system is shown in the following Fig. 4.5.5.
- For example, the virtual address is 03200H. This virtual address is split by hardware into the page number and offset within that page. High order 12 bit is used as page number i.e. 003H and lower order 12 bit is used for the offset i.e. (200H).

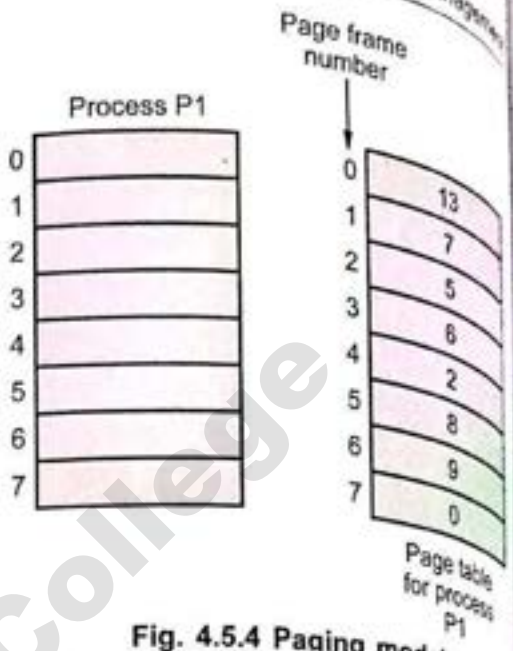


Fig. 4.5.4 Paging model

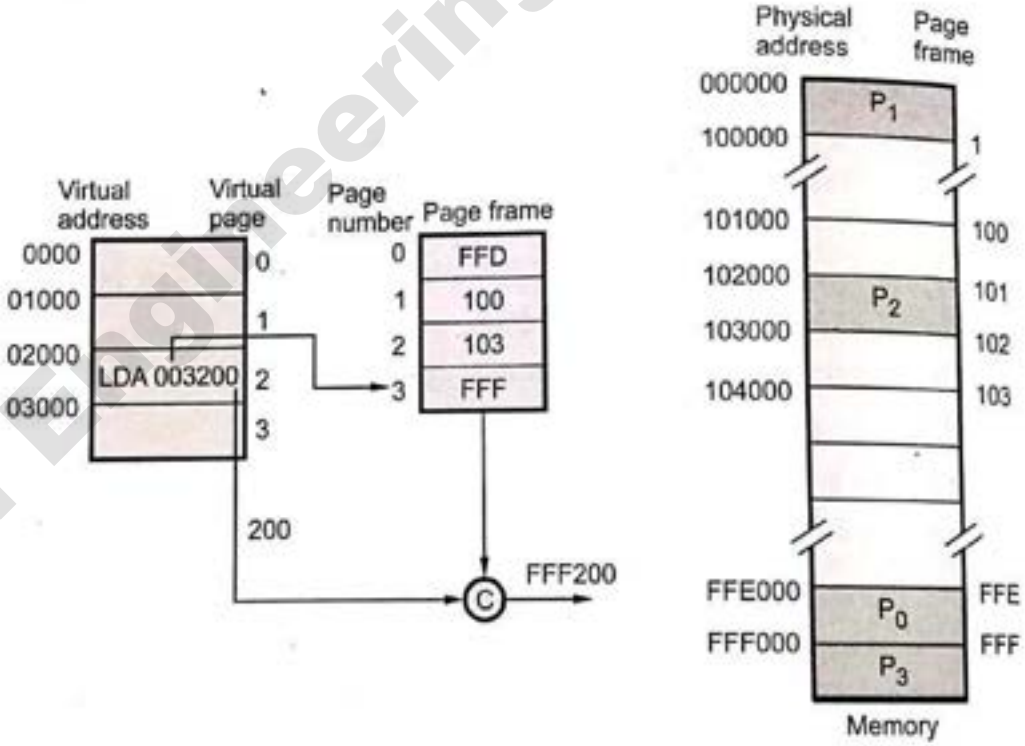


Fig. 4.5.5 (Virtual address to physical address) Paging

- Page number is used to index the page table and to obtain the correspond physical frame number (FFFH). This value is then concatenated with the offse produce the physical address (FFF200H) which is used to reference the target i in memory.

- The operating system keeps track of the status of each page frame by means of a physical memory map that may be structured as a static table. **The logical address space may be the same size as the physical address space or it may be smaller.**
- If logical address space is small, it prevents one process from monopolizing all of the memory. If the logical address space is larger than the physical address space, all pages could not be resident in physical memory. Simple paging has no capability for dealing with references to pages that are not in memory.

4.5.1 Protection and Sharing

- Protection bit is used for provide protection to the memory. Page table adds one more column for providing protection bit.
- One bit field define a page to be read only or read-write operation. This field is read at the time of calculating physical address of memory.
- Fig. 4.5.6 shows protection bit with page table.

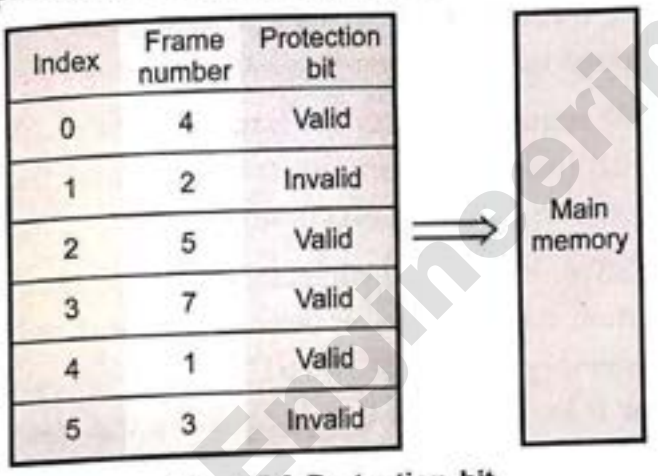


Fig. 4.5.6 Protection bit

- When valid bit is set, the required page is in the process logical address space and the page is valid. If invalid bit is set, the page is not in the process logical address space.

4.5.2 Paging with TLB

- Set of registers are used for implementing page table. Registers are suitable only for small page table. If page table size increases then special purpose hardware cache is used for page table.
- Special cache memory called a Translation Lookaside Buffer (TLB) is used with the address translation hardware. The full page table is kept in primary memory.
- TLB is very effective in improving the performance of page frame access. The cache buffer is implemented in a technology which is faster than the primary memory technology.

- Fig. 4.5.7 shows paging with TLB.
- The TLB contents may be controlled by the operating system or by hardware, depending on the architecture.
- When a page is first translated to a page frame, the map is read from primary memory into the TLB. The TLB entry contains the page number, page frame's physical address etc.

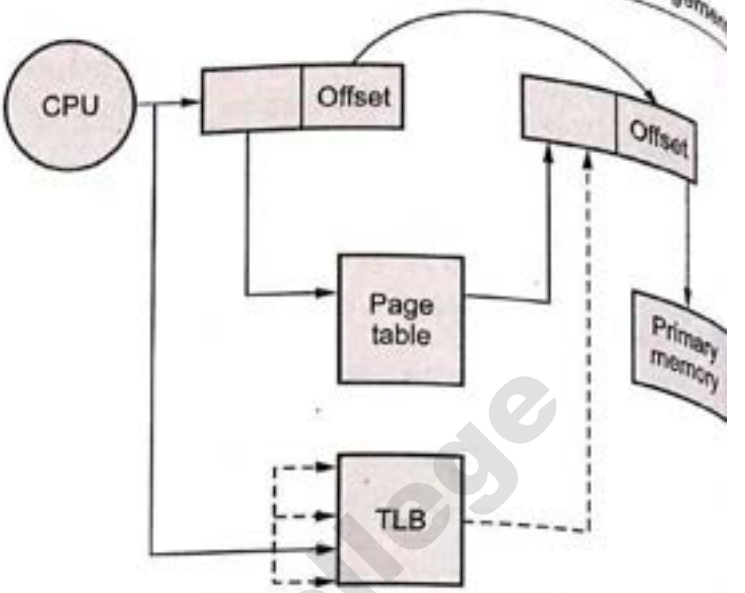


Fig. 4.5.7 Paging with TLB

- The page mapping mechanism first tries to find page number in the TLB. If the TLB contains the page number, then it is called as **page hit** or **TLB hit.**
- If the TLB does not contain an entry for page p, then it called as page miss or TLB miss. In case of TLB miss, the operating system locates the page table entry in the primary memory, which increases execution time.
- The TLB is associative, high-speed memory. A translation buffer is used to store a few of the translation table entries. It is very fast, but only for a small number of entries. On each memory reference.
  1. First ask TLB if it knows about the page. If so, the reference proceeds fast.
  2. If TLB has no information for page, must go through page and segment table to get information. Reference takes a long time, but gives the info for this page to TLB so it will know it for next reference.
- The greatest performance improvement is achieved when the associative memory is significantly faster than the normal page table lookup and the hit ratio is high. The hit ratio is the ratio between accesses that find a match in the associative memory and those that do not.
- **Disadvantage of TLB is that :** If two pages use the same entry of the memory only one of them can be remembered at once. If process is referencing both pages at same time, TLB does not work very well.
- Each entry in the TLB must include the page number as well as the complete page table entry. The processor is equipped with hardware that allows it to simultaneously interrogate a number of TLB entries to determine if there is a match on page number. This technique is referred to as *associative mapping*.

4.5.3 Page Table Structure

4.5.3.1 Multilevel or Hierarchical Page Table

- Page tables can consume a significant amount of memory.
- For example : A 32-bit virtual address space using 4 kB pages.

Page size $= 2^{12}$ bytes

Space for page numbers $= 2^{32} - 2^{12}$

$= 2^{20}$ bytes

- So page table may consists of upto 1 million entries, one for each page, for a total address capacity of about four billion bytes.
- Hierarchical page table is also called as forward mapped page table. This approach is so effective that many modern operating systems.employ it.
- In this method, each level containing a table that stores pointers to tables in the level below. Each table in the hierarchy is the size of one page. It enables the operating system to transfer page tables between main memory and secondary storage device easily.

For two levels of page table

- Virtual address contains page number and displacement into that page.

Virtual address (v) = (p, t, d)

where (p, t) = Page number

d = Displacement

- Here p is an index into the outer page table.
- Fig. 4.5.8 shows hierarchical page table.

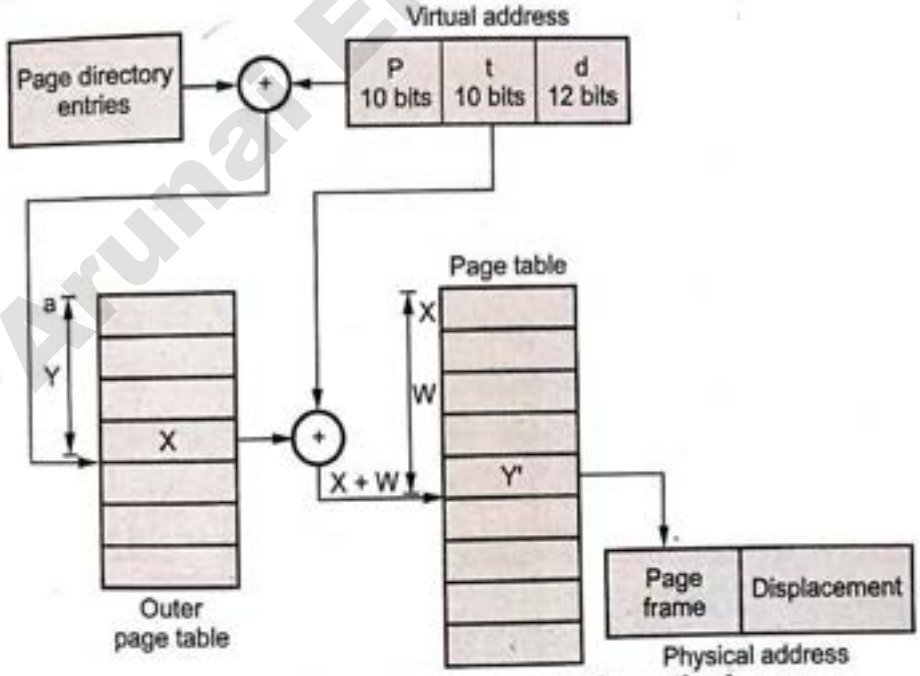


Fig. 4.5.8 Two level page table method

- Intel IA - 32 architecture support two levels of page tables.

Advantages

1. It is compact and support sparse address space..
2. It easily manage the memory.
3. It reduce table fragmentation.

Disadvantages

1. Overhead due to one more page table.
2. On TLB miss, two loads from memory will be required to get the right address translation information from the page table.

4.5.3.2 Hashed Page Tables

- Hashed page table support address space of 32 bits and more. Hash value is used as virtual page number.
- Hash table contains a link list of elements. Each elements consists of following fields :
  1. Virtual page number
  2. Mapped page frame value.
  3. Pointer to the next element.
- Hash table improves search speed.
- Fig. 4.5.9 shows block diagram of hashed page table.

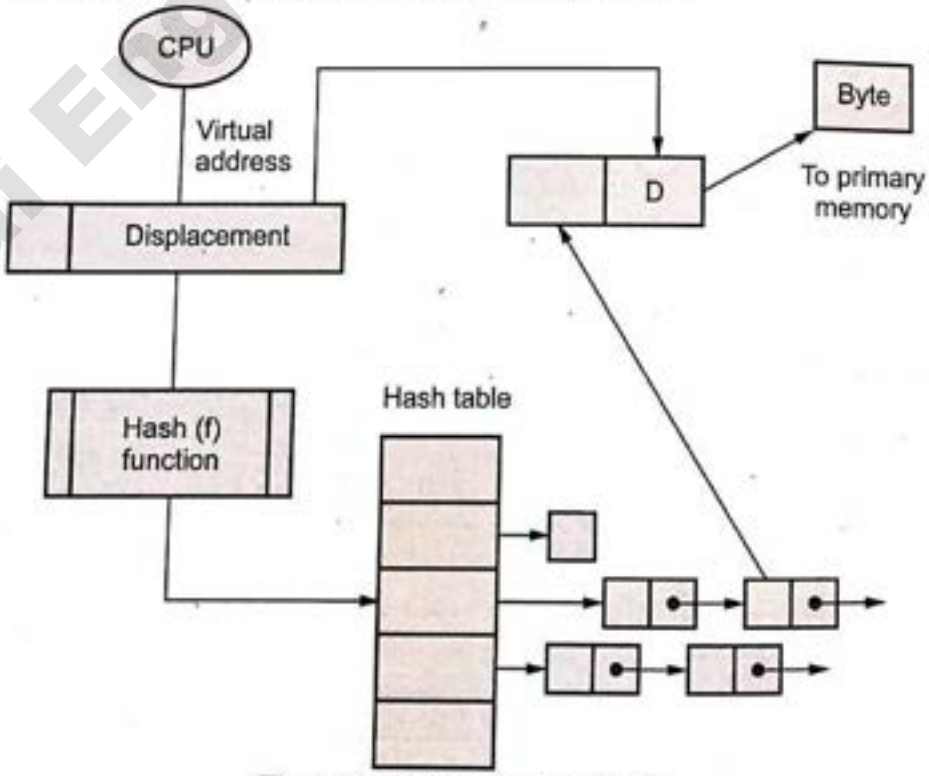


Fig. 4.5.9 Hashed page table

Working

1. Virtual page number is taken from virtual address.
2. Virtual page number is hashed into the hash table.
3. Virtual page number is compared with the first element of linked list.
4. Both the value is matched, that value is (i.e. page frame) used for calculating physical address.
5. If the both value is not matched, the entire linked list is searched for a matching.

- Clustered page table is same as hashed page table but only difference is that each entry in the hash table refers to several pages.

4.5.3.3 Inverted Page Table

- It uses a page table that contains an entry for each physical frame. This ensure that the page table occupies a fixed fraction of memory. The size is proportional to the physical memory.
- Page table overhead increases with address space size. Page table get too big to fit in memory.
- Inverted page table has one entry for each real page of memory. Lookup time is increased because it requires a search on the inverted table.
- Logical address is as follows

Process id	Page number	Offset

- Inverted page table is used by Itanium, Power PC and Ultra SPARC.

4.5.4 Advantages of Paging

1. Paging eliminates fragmentation.
2. Support higher degree of multiprogramming.
3. Paging increases memory and processor utilization.
4. Compaction overhead required for the relocatable partition scheme is also eliminated.

4.5.5 Disadvantages of Paging

1. Page address mapping hardware usually increases the cost of the computer.
2. Memory must be used to store the various tables like page table, memory map table etc.
3. Some memory will still be unused if the number of available block is not sufficient for the address spaces of the jobs to be run.

University Questions

1. *With a neat sketch, explain how logical address is translated into physical address using paging mechanism.* **AU : CSE/IT : May-15, Marks 10**
2. *Discuss the given memory management technique with diagram : Paging and Translation Look-aside Buffer.* **AU : Dec.-15, Marks 8**
3. *Consider a system that allocates pages of different sizes to its processes. What are the advantages of such a paging scheme ? What modifications to the virtual memory system provide this functionality ?* **AU : Dec.-16, Marks 8**
4. *Discuss the given Memory Management techniques with diagrams : Paging and Translation Look-aside Buffer.* **AU : May-17, Marks 6**

4.6 Segmentation

AU : Dec.-17, May-16, 18

- In segmentation, a program's data and instructions are divided into blocks called **segments.** A segment is a logical entity in a program.
- Fig. 4.6.1 shows programmer's view.
- **Logical view :** A process consists of a set of segments.
- **Physical view :** It consists of non-adjacent areas of memory allocated to segments.

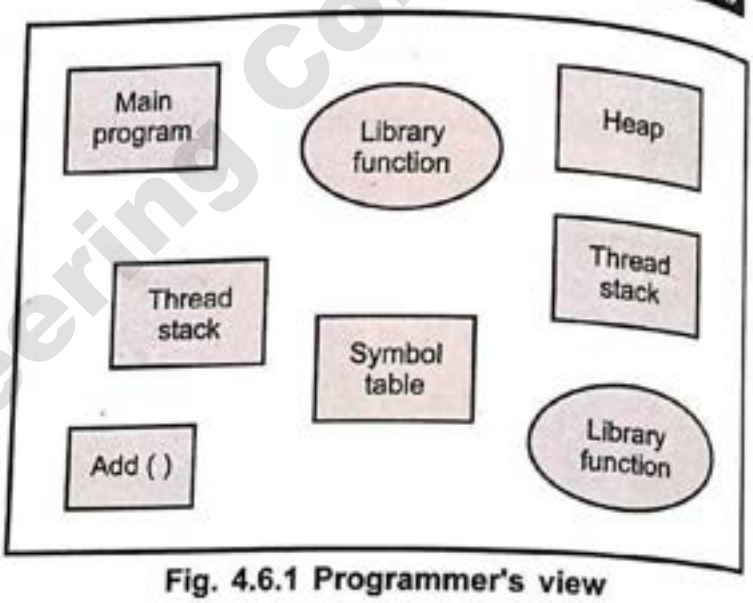


Fig. 4.6.1 Programmer's view

- All segment size may be equal or may not be equal. Segmentation support user view of memory. Fig. 4.6.1 shows programmer's view of a program.
- Segmentation can be implemented with or without paging.
- Collection of segment is called as logical address space. Each segment is identified by its name.
- Processor generates logical addresses. These addresses consists of a segment number and an offset into the segment. Segment number is used as an index to segment page table.
- Fig. 4.6.2 shows logical address.
- Segment names are normally symbolic names.

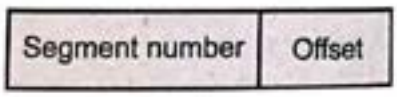


Fig. 4.6.2 Logical address

- Operating system maintains a segment table for each process. It is usually stored in main memory as a segment that is not to be loaded as long as the process can run.
- Each entry in the segment table has a segment base and a segment limit. The base field contains the segment relocation register for the target segment. The segment limit field contains the length of the segment.
- Fig. 4.6.3 shows an address translation in segmentation.

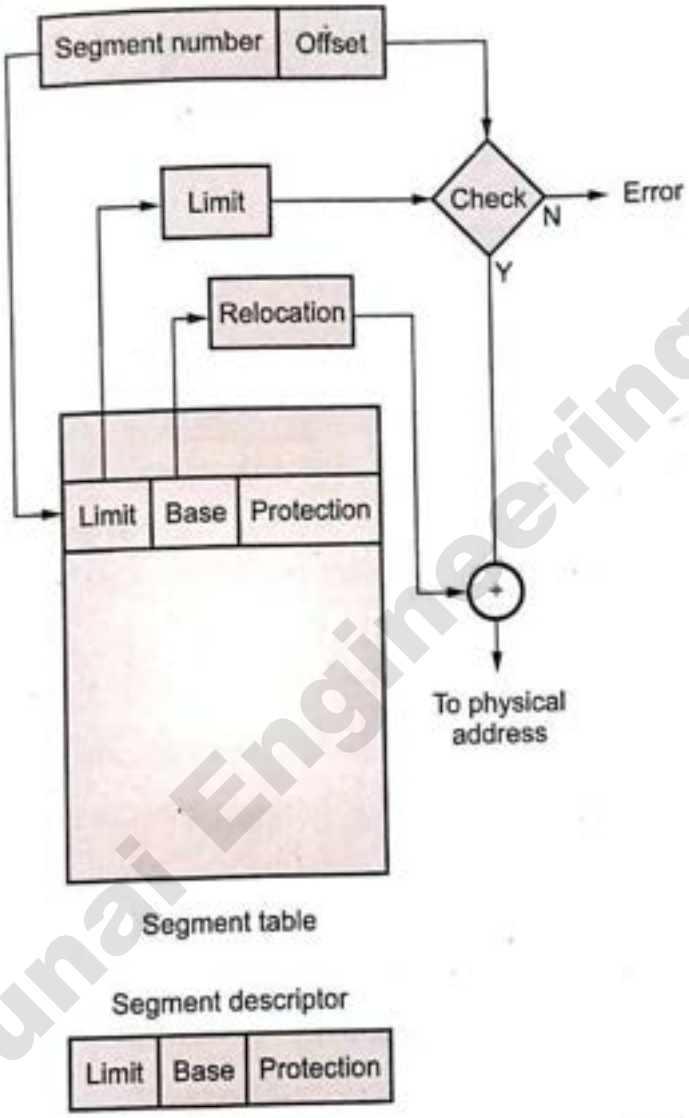


Fig. 4.6.3 Address translation in segmentation

- Segment table contains the physical address of the start of the segment. Then add the offset to the base and generate the physical address.
- If the required reference is not found in one of the segment registers, then error is generated. At the same time, operating system does lookup in segment table and loads new segment descriptor into the register.

4.6.1 Protection and Sharing

- Sharing of segment provides less overhead than the pure paging segment. Multiple processes can share a segment.
- Two processes share a segment when their segment table entries point to the same segment in main memory. Operating system maintains global segment table for everyone and also used by OS.
- Access rights for segment are usually included in table entry. An illegal access of memory is protected by memory mapping hardware. It checks the protection bits associated with each segment table.
- Fig. 4.6.4 shows sharing in segmentation. Access rights for segments are : Read, write, append and execute.
- Segmentation suffer from external fragmentation because segment sizes vary. Entire segment is either memory or on secondary storage disk.

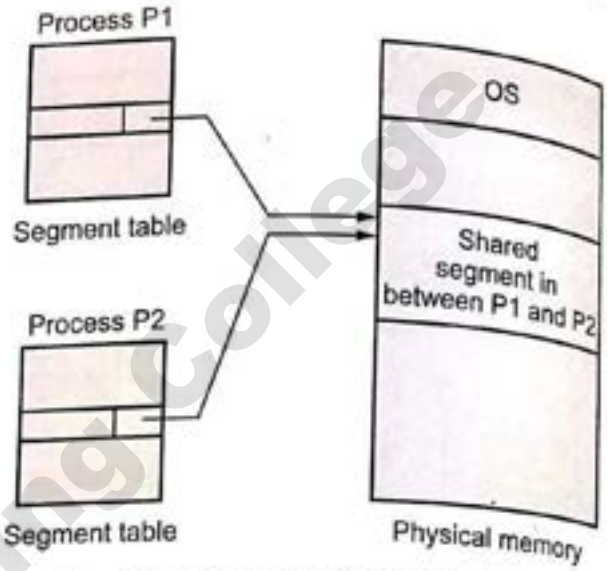


Fig. 4.6.4 Segmentation with sharing

4.6.2 Advantages

1. It provides virtual memory.
2. Allows dynamic segment growth.
3. Segmentation assists dynamic linking.
4. Segmentation is visible.

4.6.3 Disadvantages

1. Maximum size of a segment is limited by the size of main memory.
2. Difficulty to manage variable size segments on secondary storage.
3. Fragmentation and complicated memory management.

4.6.4 Difference between Segmentation and Paging

Sr. No.	Segmentation	Paging
1.	Program is divided into variable size segments.	Program is divided into fixed size pages.
2.	User (or compiler) is responsible for dividing the program into segments.	Division into pages is performed by the operating system.

3.	Segmentation is slower than paging.	Paging is faster than segmentation.
4.	Segmentation is visible to the user.	Paging is invisible to the user.
5.	Segmentation eliminates internal fragmentation.	Paging suffers from internal fragmentation.
6.	Segmentation suffers from external fragmentation.	There is no external fragmentation.
7.	Processor uses page number, offset to calculate absolute address.	Processor uses segment number, offset to calculate absolute address.
8.	OS maintain a list of free holes in main memory.	OS must maintain a free frame list.

Example 4.6.1 *Describe a mechanism by which one segment could belong to the address space of two different processes.* **AU : May-16, Marks 8**

Solution : Segment tables are a collection of base-limit registers, segments can be shared when entries in the segment table of two different jobs point to the same physical location. The two segment tables must have identical base pointers, and the shared segment number must be the same in the two processes.

Review Questions

1. *Draw the diagram of segmentation memory management scheme and explain its principle.* **AU : Dec.-17, Marks 13**
2. *Explain why sharing a reentrant module is easier when segmentation is used than when pure paging is used with example.* **AU : May-18, Marks 13**

4.7 Segmentation with Paging

AU : May-16

- Most of the architecture support paging and segmentation. All the pages of segment need not be in main memory.
- It simplify the memory allocation and speed increases. It requires a high speed register to store the base address of the segment map table.
- For each segment, page table is created by OS. A pointer to the page table is kept in the segment's entry in the segment table.
- Segments are typically larger than pages.
- The base address get from the segment descriptor table is concatenated with the offset. This new address is referred to as a **linear address.** Linear address is generated by paging hardware.

Advantages

Combines all advantages of paging and segmentation.

Disadvantages

1. It increases hardware cost. 2. It increases processor overheads.

3. Dangers of thrashing.

Example 4.7.1 *Paging system consists of physical memory 2^{24} bytes, pages of logical address space is 256. Page size of 2^{10} bytes, how many bits are in a logical address.*

Solution : Logical address space = $256 = 2^8$

Page size = 2^{10} bytes

So, Total logical address space = $2^8 \times 2^{10} = 2^{18}$ bytes

For 2^{18} byte address space - 18-bit address is required.

Example 4.7.2 *On a system using simple segmentation, compute the physical address for each of the logical addresses, logical address is given in the following segment table. If the address generates a segment fault, indicate so.*

Segment	Base	Length
0	330	124
1	876	211
2	111	99
3	498	302

a) 0, 99 b) 2, 78 c) 1, 265 d) 3, 222 e) 0, 111

Solution : a) 0, 99

Offset = 99; Segment length = 124; Segment = 0

* Offset 99 is less than segment length 124.

* Starting location of segment 0 is start from 330.

* Physical address = Offset + Segment base

= 99 + 330 = 429

b) 2, 78

Segment = 2

Offset = 78

Segment length = 99

• Offset 78 is less than segment length 99

• Starting location of segment 2 is 111

• Physical address = Segment base + Offset

= 111 + 78 = 189

c) 1, 265

Segment = 1

Offset = 265

Segment length = 211

• Offset 265 is greater than segment length 211.

• This address results in a segment fault.

d) 3, 222

Segment = 3

Offset = 222

Segment length = 302

• Offset 222 is less than segment length 302.

• Starting location of segment 3 is 498

• Physical address = Segment base + Offset

= 498 + 222 = 720

e) 0, 111

Segment = 0

Offset = 111

Segment length = 124

• Offset 111 is less than segment length 124.

• Starting address of segment 0 is 330

• Physical address = Segment base + Offset

= 330 + 111 = 441

Example 4.7.3 *System using a paging and segmentation, the virtual address space consists of upto 8 segments where each segment can be up to 2^{29} byte long. The hardware pages each segment into 256 bytes pages. How many bits in the virtual address specify the i) Segment number ii) Page number iii) Offset within page iv) Entire virtual address*

Solution : i) Segment number : Virtual address space consists of upto 8 segments. So

$$8 = 2^3$$

$\therefore$ 3 bits are needed to specify segment number.

ii) Page number : Hardware pages each segment into 256 byte pages. So

$$256 = 2^8 \text{ byte pages}$$

Size of segment is 2^{29} bytes.

$$\therefore \quad 2^{29}/2^8 = 2^{29-8} = 2^{21} = 21 \text{ pages}$$

$\therefore$ 21 bits are required to specify the page number.

iii) Offset within the page : For 2^8 byte page, 8 bits are needed.

iv) Entire virtual address = Segment number + Page number + Offset

$$= 3 + 21 + 8 = 32$$

University Question

1. *Why are segmentation and paging sometimes combined into one scheme ? Explain them in detail with example.* **AU : May-16, Marks 8**

4.8 Virtual Memory

- Virtual memory is a method of using hard disk space to provide extra memory. It simulates additional main memory.
- In Windows operating system, the amount of virtual memory available equals the amount of free main memory plus the amount of disk space allocated to the swap file.

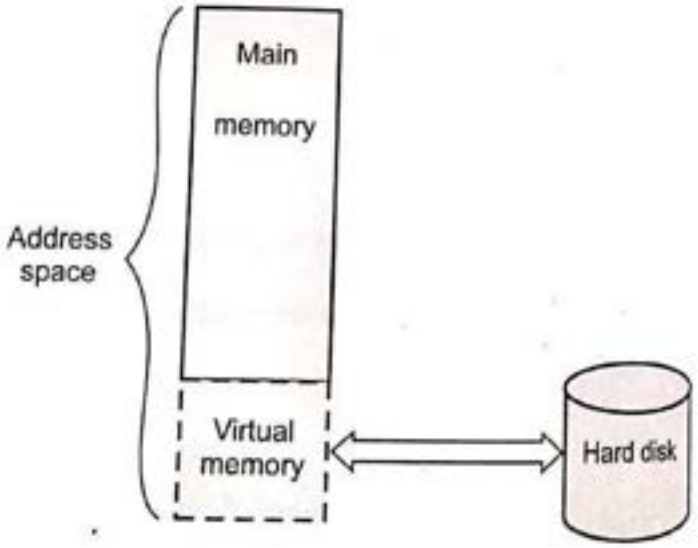


Fig. 4.8.1 Logical view of virtual memory

- Fig. 4.8.1 shows logical view of virtual memory concept.
- A swap file is an area of your hard disk that is set aside for virtual memory. Swap files can be either temporary or permanent.
- Virtual memory is stored in the secondary storage device. It helps to extend additional memory capacity and work with primary memory to load applications. The virtual memory will reduce the cost of expanding the capacity of physical memory. The implementations of virtual memory will different for operating systems to operating system.
- Each process address space is partitioned into parts that can be loaded into primary memory when they are needed and written back to secondary storage otherwise. Address space partitions have been used for the code, data and stack identified by the compiler and relocating hardware. The portion of the process that is actually in main memory at any time is defined to be the **resident set** of the process.
- The logical addressable space is referred to as virtual memory. The virtual address space is much larger than the physical primary memory in a computer system. The virtual memory works with the help of secondary storage device and its speed is low compared to the physical storage location.
- Virtual memory uses two types of addresses : ***Virtual address and physical address.***
- Virtual address is referred by process and physical address is actual address of the main memory.
- Whenever a process accesses a virtual address, the system must translate it to a physical address. Virtual memory system uses special purpose hardware. This hardware is called as memory management unit.
- Most virtual memory system use a technique called paging. The virtual address space is divided into fixed size units called pages. The corresponding units in the physical memory are called page frames. The pages and page frames are normally the same size. The area on a hard disk that stores page frames is usually called the paging file or the swap file.
- Fig. 4.8.2 shows concept of virtual memory.
- When the system is ready to run a process, the system loads the process code and data from secondary storage into primary memory. Only a small portion of these needs to be in primary memory at once for the process to execute. The success of implementing virtual memory system is how to map virtual addresses to physical addresses. Because processes reference only virtual addresses.

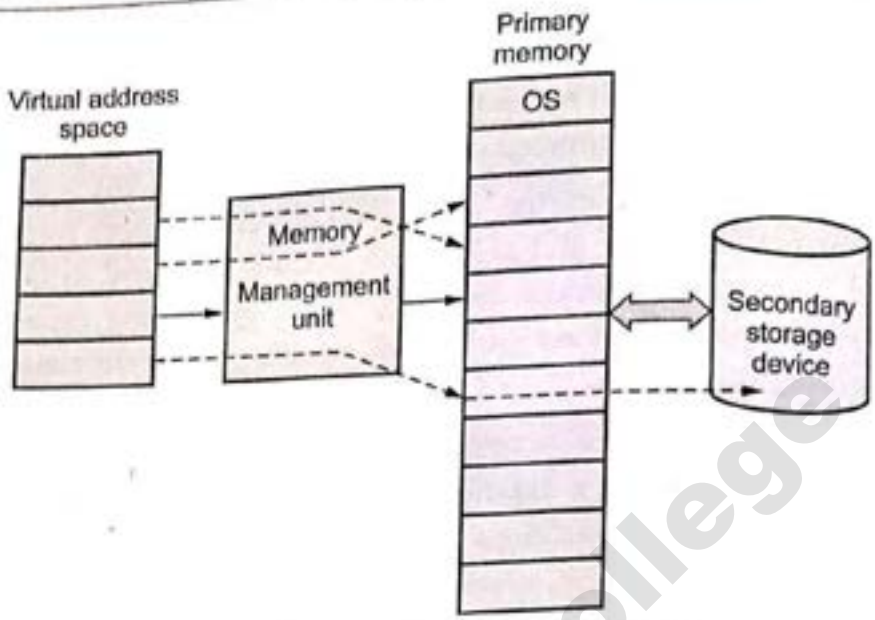


Fig. 4.8.2 Concept of virtual memory

- Virtual memory can be implement in one of two ways :
  1. Paging
  2. Segmentation

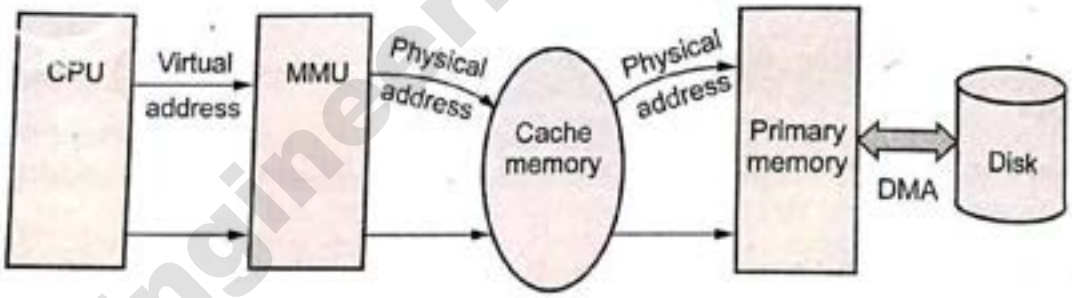


Fig. 4.8.3 Virtual memory organization

- Following are the situations, when entire program is not required to load fully.

1. User written error handling routines are used only when an error occurs in the data or computation.
2. Certain options and features of a program may be used rarely.
3. Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
4. Many routines are commonly used at mutually exclusive times during a run.

- Virtual memory makes the task of programming much easier. Virtual memory is commonly implemented by **demand paging**. Virtual memory mechanism bridges the size and speed gaps between the main memory and secondary storage and is usually implemented in part by software technique.

Benefits

1. Less I/O needed to load/swap a process.
2. Degree of multiprogramming can be increased.
3. A program's logical address space can be larger than physical memory.

4.9 Demand Paging

AU : May-16

- In demand paging, a page is loaded into primary memory only when the process references it. It is the simplest fetch policy implemented in virtual memory. Fig. 4.9.1 shows concept of demand paging.
- Demand paging guarantees that the system brings into primary memory only those pages that processes actually need. It involves interaction between hardware and software component of the virtual memory.
- Paging with swapping method will become as demand paging concept. Process's logical address space is stored on the secondary storage device.
- Demand paging requires that the page map table for each process keep track of each page as it is loaded or removed from primary memory.
- When user want to execute a process, user swap a process into a memory. **Lazy swapper** is used for swapping the process into memory. Lazy swapper swaps the pages whenever needed.

Working of demand paging

1. Program attempts to find a page.
2. If the page is located in the primary memory, then the program executes.
3. If the page cannot be found in the primary memory, then the page fault occurs.
4. Memory reference is checked for required page to determine it is a valid reference to a location on a secondary storage, if it is then the page will be required page. If not, the process is terminated.
5. Schedule disk operation to read the required page into primary memory.
6. Restart the instruction that was interrupted by the OS trap.

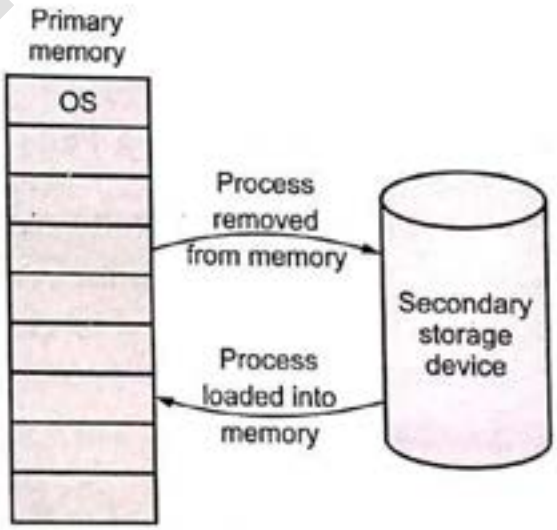


Fig. 4.9.1 Demand paging

- Demand paging takes support from hardware to distinguish the pages. The valid-invalid bit method is used. This method gives the idea about page location. Page may be in the primary memory or in the secondary storage.
- If valid bit is set, then required page is stored in the primary memory. When an invalid bit is set, then page is on the secondary storage disk.
- Valid-invalid bit field is attached with page table. Fig. 4.9.2 shows page table entry.

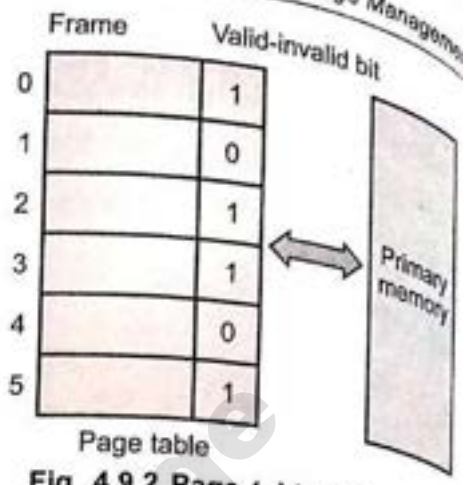


Fig. 4.9.2 Page table entry with valid-invalid

- The valid-invalid bit field contain a boolean value to indicate whether the page exists in primary memory.

  0 = Not in memory

  1 = In memory

- Initially all valid-invalid entries are set to 0. This information is used for page replacement algorithm. Page fault leads to the page replacement operation to load the required pages in primary memory.
- A failure to find a page in memory is often called a page fault.
- Demand paging is most efficient when users are aware of the page size used by operating system.

4.9.1 Steps in Handling a Page Fault

- Fig. 4.9.3 shows page fault handling steps in demand paging.
- When page fault occurs, the system performs following steps :

1. Check the process control block table for reference which bit is set i.e. valid bit or invalid bit for memory access.
2. If invalid bit is set, user will terminate the process. If valid bit is set, then the page is loaded into memory.
3. Check for free frame for new **page loading.**
4. Start reading secondary storage device for desired page for allocated frame.
5. Required page is loaded into the memory after completion of read operation. PCB table is updated.
6. System restart the instruction that was interrupted by the error. Page is loaded in memory so process can use it.

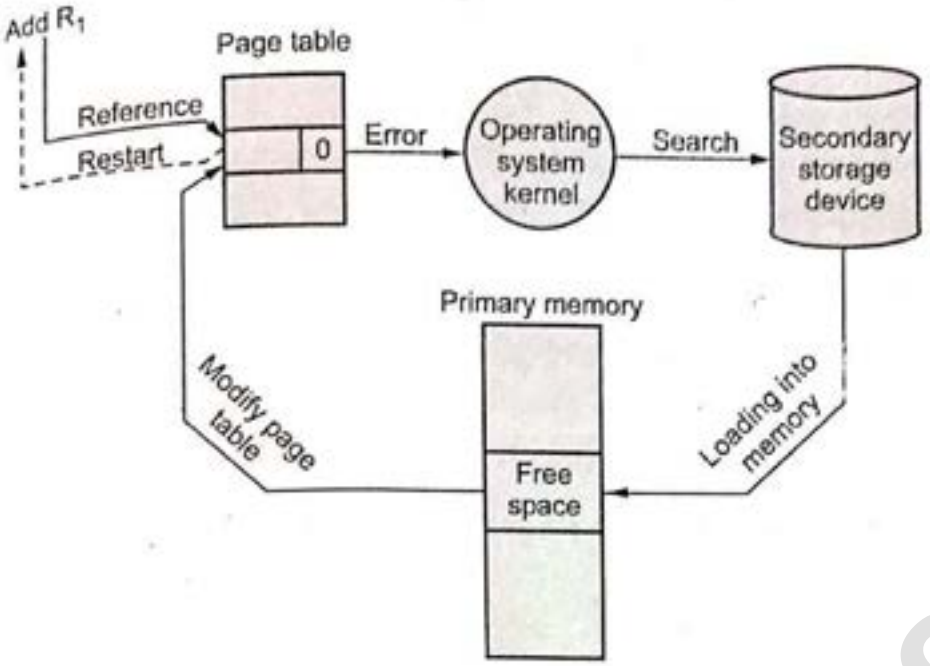


Fig. 4.9.3 Handling of page fault

4.9.2 Demand Paging Performance

- Demand paging is a solution to inefficient memory utilization. Performance of the computer system is affected by demand paging. Effective access time for a demand paged memory is calculated as follows :
- Since demand paging like caching, can compute average access time. Memory access time (m) for most computers now ranges from 10 to 200 nanoseconds. If there is no page fault, then

  Effective access time = Memory access time

- If there is page fault, then

  Effective access time = Hit rate × Hit time + Miss rate × Miss time

  = Hit time + Miss rate × Miss penalty

  Effective access time = Memory access time * (1 – p) + p * page fault service time

where p is the probability of a page fault

- Effective access time is directly proportional to page fault rate.
- Suppose memory access time = 100 ns, Page fault service time = 25 ms then

  Effective access time = Memory access time * (1 – p) + p * page fault service time

  = 100 (1 – p) + p * 25,000,000

  = 100 + 24,999,900 p

4.9.3 Advantages of Demand Paging

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

4.9.4 Disadvantages of Demand Paging

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. Due to the lack of an explicit constraints on a jobs address space size.

4.9.5 Comparison of Demand Paging with Segmentation

Sr. No.	Segmentation	Demand paging
1.	Segments may of different size.	Pages are of same size.
2.	Segment can be shared.	Pages can not be shared.
3.	It allows dynamic growth of segments.	Page size is fixed.
4.	Segment map table indicates the address of each segment in memory.	Page map table keeps track of pages in memory.
5.	Segments are allocated to the program while compilation.	Pages are loaded in memory on demand.
6.	Provides virtual memory.	Also provides virtual memory.

University Question

1. *Under what circumstances do page faults occur ? Describe the actions taken by the operating system when a page fault occurs.* **AU : May-16, Marks 8**

4.10 Page Replacement

AU : Dec.-12, 16, 17, May-15, 16, 18

- When a page fault occurs, page replacement algorithms are used for loading the page in memory and no free page frame exist in memory.
- Page fault occurs if a running process references a nonresident page.
- The goal of a replacement strategy is to minimize the fault rate. To evaluate a replacement algorithm, following parameters are used :
  1. The size of a page 2. A set of reference strings 3. The number of page frames.
- Given these inputs, we can determine the number of page faults and hence the page-fault rate. A reference string is a list of pages in order of their reference by the processor.

- When we find that a page frame reference is in main memory then we have a page hit and when page fault occurs we say it is page miss. A poor choice of page replacement algorithm may result in lot of page misses.
- Page replacement algorithm deals with how the kernel decides which page reclaim. Operating systems selects the local or global page replacement policy. Local replacement policy allocates a certain number of pages to each process. If a process needs new page, it must replace one of its own pages. If the OS uses a global policy, it can take a page from any process, using global selection criteria.
- UNIX OS uses global page replacement policy.

4.10.1 First-In-First-Out

- FIFO page replacement algorithm removes the pages that have been in memory the longest. Here system keeps track of the order in which pages enter primary memory.
- When page must be replaced, the oldest page is selected. System maintain a FIFO queue of pages. Pages are inserted at the tail of the queue.
- FIFO algorithms interfere with the locality of the process.
- Consider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9
- Page frame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	4	4	4	4	4	4	6	6	6	6	9	9	9	9	9	4	4
Page frame 2		2	2	2	2	5	5	5	5	5	7	7	7	7	7	7	2	2	2	2
Page frame 3			3	3	3	3	3	3	2	2	2	8	8	8	8	8	8	5	5	9
Page fault	PF	PF	PF	PF		PF			PF	PF	PF	PF		PF			PF	PF	PF	PF

Total number of page fault = 14

- Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	1	5	5	5	5	5	5	5	5	9	9	9	9	9	9	9
Page frame 2		2	2	2	2	2	2	2	2	6	6	6	6	6	6	6	2	2	2	2
Page frame 3			3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	5	5	5
Page frame 4				4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	4	4
Page fault	PF	PF	PF	PF		PF				PF	PF	PF		PF			PF	PF	PF	

Total number of page fault = 12

4.10.3.1 Belady's Anomaly

- Using FIFO algorithm for some reference strings actually generate more page faults when more page frames are allotted. This truly unexpected result was first demonstrated by Belady in 1970 and is known as Belady's anomaly.
- FIFO does not satisfy the inclusion property and is not a stack algorithm.
- A page replacement algorithm that satisfies the inclusion property is free from Belady anomaly.
- Consider the reference string : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- Page frame size = 3

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Page frame 2	-	2	2	2	1	1	1	1	1	3	3	3
Page frame 3	-	-	3	3	3	2	2	2	2	2	4	4
Page fault	PF	PF	PF	PF	PF	PF	PF			PF	PF	

Total number of page fault = 9

Page frame size = 4

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Page frame 1	1	1	1	1	1	1	5	5	5	5	4	4
Page frame 2	-	2	2	2	2	2	2	1	1	1	1	5
Page frame 3	-	-	3	3	3	3	3	3	2	2	2	2
Page frame 4	-	-	-	4	4	4	4	4	4	3	3	3
Page fault	PF	PF	PF	PF			PF	PF	PF	PF	PF	PF

Total number of page fault = 10

- Paging algorithm has worse performance when the amount of primary memory allocated to the process is increased. The problem creates because the set of pages loaded with a memory allocation of 3 is not same as the with a memory allocation of 4.
- There are a set of demand paging algorithms whereby the set of pages loaded with an allocation of m frames is always a subset of the set of pages loaded with an allocation of m + 1 frame. This property is called the inclusion property.
- FIFO does not satisfy the inclusion property and is not a stack algorithm. LRU can be a stack algorithm.
- FIFO algorithm uses only the modified and status bits when swapping is used.

Advantage of FIFO :

1. FIFO is very simple and easy to implement.

Disadvantages of FIFO :

1. It suffers from Belady's anomaly.
2. It is not very effective.
3. System needs to keep track of each frame.

4.10.2 LRU Page Replacement Algorithm

- LRU stands for least recently used. This replacement policy chooses to replace the page which has not been referenced for the longest time. This policy assumes the recent past will approximate the immediate future. The operating system keeps track of when each page was referenced by recording the time of reference or by maintaining a stack of references.
- System must maintain a time stamp for every page that indicates the time of the last access. When deciding which page to remove, the operating system must scan all the pages of memory for finding the oldest time stamp. Instead of keeping an extra time stamp bits, LRU is implement by maintaining pages in an ordered list.
- System can manage LRU with a list called the LRU stack or the paging stack. In the LRU stack, the first entry describes the page referenced least recently, the last entry describes to the last page referenced. If a page is referenced, move it to the end of the list.
- LRU algorithm does not interfere with the locality of the process.
- Consider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9
- Page frame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	4	4	4	3	3	3	6	6	6	6	9	9	9	2	2	2	9
Page frame 2	-	2	2	2	2	2	2	4	4	4	7	7	7	7	7	7	7	5	5	5
Page frame 3	-	-	3	3	3	5	5	5	2	2	2	8	8	8	8	8	8	8	4	4
Page fault	PF	PF	PF	PF		PF	PF	PF	PF	PF	PF	PF		PF	.		PF	PF	PF	PF

Total number of page fault = 16

- Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	1	5	5	5	5	6	6	6	6	6	6	6	2	2	2	2
Page frame 2	-	2	2	2	2	2	2	2	2	2	2	2	2	9	9	9	9	5	5	5
Page frame 3	-	-	3	3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	4	4
Page frame 4	-	-	-	4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	9
Page fault	PF	PF	PF	PF		PF				PF	PF	PF		PF			PF	PF	PF	PF

Total number of page fault = 13

- LRU is a stack algorithm. Increase in memory size will never cause an increase in the number of page interrupts.

Advantages :

1. It is quite good algorithm.
2. It is amenable to full statistical analysis.
3. Never suffers from Belady's anomaly.

Disadvantages :

1. Problem is how to implement.
2. It requires hardware assistance.

4.10.3 LRU Approximation Algorithms

- LRU is a desirable algorithm to use, but it is expensive to implement directly. It is often approximated. By using one reference bit, user can do a second-chance or clock replacement algorithm.
- Page frames are arranged in circular list. Initially each reference bit is set to 0. It is set to 1 any time the page is referenced. Algorithm uses pointer to points to the next candidate for a page replacement.
- When a page replacement is needed, the frame pointed to is considered as a candidate. If its reference bit is 0, it is selected. If its bit is 1, the bit is set to 0, and the pointer is incremented to examine the next frame. This continues until a frame is found with a reference bit of 0. This may require that all frames be examined, bringing us around to the one we started with.

Second chance replacement

- In some books, the second chance replacement policy is called the clock replacement policy. In the second chance page replacement policy, the candidate

pages for removal are consider in a round robin matter and a page that has been accessed between consecutive considerations will not be replaced.

- The page replaced is the one that, considered in a round robin matter. It has not been accessed since its last consideration.

- Working :

1. Add a "second chance" bit to each memory frame.
2. Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
3. A new page read into a memory frame has the second chance bit set to ZERO (0)
4. When you need to find a page for removal, look in a round robin manner in the memory frames :
   i. If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
   ii. If the second chance bit is ZERO, replace the page in that memory frame.

String	0	4	1	4	2	4	3	4	2	4	0	4	1	4	2	4	3	4
Frame 1	0	0	0	0	0	0	0	0	2	0	2	0	2	0	2	1	2	1	2	0	2	0	2	0	1	0	1	0	1	0	1	0	3	0	3
Frame 2	-	4	0	4	0	4	1	4	1	4	0	4	1	4	1	4	1	4	1	4	0	4	1	4	1	4	1	4	0	4	1	4	1	4	1	4
Frame 3	-	-	1	0	1	0	1	0	1	0	3	0	3	0	3	0	3	0	0	0	0	0	0	0	0	0	2	0	2	0	2	0	2		
PF	PF	PF	PF		PF		PF				PF		PF		PF		PF																		

Total number of page fault = 9

4.10.4 Optimal Page Replacement

- The data which will not be accessed permanently or for a longest time should be replaced in optimal page replacement algorithm.
- Optimal page replacement algorithm gives less page fault as compared to FIFO and LRU. This algorithm never suffer from Belady's anomaly.
- Use of this page-replacement algorithm guarantees the lowest possible page-fault rate for a fixed number of pages
- The key distinction between FIFO and optimal is that FIFO uses the time when a page was brought into memory whereas optimal uses the time when a page is to be used.
- It is difficult to implement because of future reference.
- Consider the reference string : 1, 2, 3, 4, 2, 5, 3, 4, 2, 6, 7, 8, 7, 9, 7, 8, 2, 5, 4, 9

- Page frame size = 3

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	4	4	4	4	4	4	6	7	7	7	7	7	7	2	2	4	4
Page frame 2		2	2	2	2	5	5	5	5	5	5	8	8	8	8	8	8	5	5	5
Page frame 3			3	3	3	3	3	3	2	2	2	2	2	9	9	9	9	9	9	9
Page fault	PF	PF	PF	PF		PF			PF	PF	PF	PF		PF			PF	PF	PF	

Total number of page fault = 13

- Page frame size = 4

Reference string	1	2	3	4	2	5	3	4	2	6	7	8	7	9	7	8	2	5	4	9
Page frame 1	1	1	1	1	1	5	5	5	5	5	5	5	5	9	9	9	9	9	9	9
Page frame 2	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	5	5	5
Page frame 3	-	-	3	3	3	3	3	3	3	6	7	7	7	7	7	7	7	7	4	4
Page frame 4	-	-	-	4	4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	8
Page fault	PF	PF	PF	PF		PF			PF	PF	PF		PF					PF	PF	

Total number of page fault = 11

Advantages :

1. It has lowest page fault rate.
2. Optimal never suffer from Belady's anomaly.
3. Used for comparison studies.

Disadvantages :

1. Difficult to implement.
2. It requires future reference string.

4.10.5 Difference between FIFO, LRU and Optimal

Sr. No.	FIFO	LRU	Optimal
1.	Number of page fault is more than LRU and optimal.	Number of page fault is more than optimal and less than FIFO.	Number of page fault is less both.
2.	Suffer from Belady's anomaly.	Does not suffer from Belady's anomaly.	Does not suffer from Belady's anomaly.

3.	Used in OS.	Used in OS.	Not used in OS.
4.	Future predication is not required.	Future predication is not required.	Future predication is required.
5.	Simple to implement.	Considered to be good.	Difficult to implement.

Example 4.10.1 *Consider the string 0, 1, 2, 3, 0, 1, 2, 3, 0 ,1 2, 3, 4, 5, 6, 7. With frame size 3 and 4. Calculate page fault.*

Solution : 1) FIFO

Page frame size = 3

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	3	3	3	2	2	2	1	1	1	4	4	4	7
1		1	1	1	0	0	0	3	3	3	2	2	2	5	5	5
2			2	2	2	1	1	1	0	0	0	3	3	3	6	6
Page fault	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

* Page fault

This example incurs 16 page faults in FIFO algorithm,

FIFO algorithm with four page frames

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1		1	1	1	1	1	1	1	1	1	1	1	1	5	5	5
2			2	2	2	2	2	2	2	2	2	2	2	2	6	6
3				3	3	3	3	3	3	3	3	3	3	3	3	7
Page fault	*	*	*	*									*	*	*	*

Number of page fault is 8.

2) LRU

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	3	3	3	2	2	2	1	1	1	4	4	4	7
1		1	1	1	0	0	0	3	3	3	2	2	2	5	5	5
2			2	2	2	1	1	1	0	0	0	3	3	3	6	6
Page fault	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Number of page fault = 16.

Consider the same reference string with 4 page frames

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1		1	1	1	1	1	1	1	1	1	1	1	1	5	5	5
2			2	2	2	2	2	2	2	2	2	2	2	2	6	6
3				3	3	3	3	3	3	3	3	3	3	3	3	7
Page fault	*	*	*	*								*	*	*	*	*

Number of page fault = 8.

3) Optimal

Frame	0	1	2	3	0	1	2	3	0	1	2	3	4	5	7
0	0	0	0	0	0	0	0	0	0	1	1	1	4	4	7
1		1	1	1	1	1	2	2	2	2	2	2	2	5	5
2			2	3	3	3	3	3	3	3	3	3	3	3	6
Page fault	*	*	*	*			*			*			*	*	*

Number of page fault = 10.

With page frame = 4 for same reference string.

Frame	0	1	2	3	0	1	2	3	0	1	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
1		1	1	1	1	1	1	1	1	1	1	1	5	5	5
2			2	2	2	2	2	2	2	2	2	2	2	6	6
3				3	3	3	3	3	3	3	3	3	3	3	7
Page fault	*	*	*	*								*	*	*	*

Number of page fault = 8.

Example 4.10.2 *Consider the following page-reference string :*

2 3 2 1 5 2 4 5 3 2 5 2

How many page faults occur for the following replacement algorithms, assuming three frames. i) FIFO ii) LRU iii) Optimal.

Solution : i) FIFO

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	5	5	5	5	3	3	3	3
1		3	3	3	3	2	2	2	2	2	5	5
2				1	1	1	4	4	4	4	4	2
Page fault	*	*		*	*	*	*		*		*	*

Number of page fault = 9.

ii) LRU

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	2	2	3	3	3	3
1		3	3	3	5	5	5	5	5	5	5	5
2				1	1	1	4	4	4	2	2	2
Page fault	*	*		*	*		*		*	*		

Number of page fault = 7.

iii) Optimal

Frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	4	4	4	2	2	2
1		3	3	3	3	3	3	3	3	3	3	3
2				1	5	5	5	5	5	5	5	5
	*	*		*	*		*			*		

Number of page fault = 6.

Example 4.10.3 *Consider the following page reference string :*

1, 2, 3, 2, 5, 6, 3, 4, 6, 3, 7, 3, 1, 5, 3, 6, 3, 4, 2, 4, 3, 4, 5, 5, 1.

Indicate page fault and calculate total number of page faults and successful ration for FIFO, Optimal and LRU algorithms. Assume there are four frames and initially all the frames are empty. **AU : Dec.-15, Marks 12**

Solution : i) FIFO

	1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
1	1	1	1	1	1	6	6	6	6	6	6	6	1	1	1	1	1	1	2	2	2	2	2	2
2	-	2	2	2	2	2	2	4	4	4	4	4	4	5	5	5	5	5	5	5	3	3	3	3
3	-	-	3	3	3	3	3	3	3	3	7	7	7	7	7	6	6	6	6	6	6	6	5	5
4	-	-	-	-	5	5	5	5	5	5	5	3	3	3	3	3	3	4	4	4	4	4	4	2
Page Fault	PF	PF	PF		PF	PF		PF			PF	PF	PF	PF		PF		PF	PF		PF		PF	PF

Total number of page fault = 16

ii) Optimal

	1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	1
2	-	2	2	2	2	6	6	6	6	6	6	6	6	6	6	6	6	4	4	4	4	4	4	4
3	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	-	-	-	-	5	5	5	4	4	4	7	7	7	5	5	5	5	5	5	5	5	5	5	5
Page Fault	PF	PF	PF		PF	PF		PF			PF			PF				PF	PF					PF

Total number of page fault = 11

iii) LRU

	1	2	3	2	5	6	3	4	6	3	7	3	1	5	3	6	3	4	2	4	3	4	5	1
1	1	1	1	1	1	6	6	6	6	6	6	6	6	5	5	5	5	5	2	2	2	2	2	1
2	-	2	2	2	2	2	2	4	4	4	4	4	1	1	1	1	1	4	4	4	4	4	4	4
3	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	-	-	-	-	5	5	5	5	5	5	7	7	7	7	7	6	6	6	6	6	6	6	5	5
Page Fault	PF	PF	PF		PF	PF		PF			PF		PF	PF		PF		PF	PF				PF	PF

Total number of page falult = 11

Example 4.10.4 *Consider the following page reference string :*

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

How many page faults would occur for the following replacement algorithms, assuming 1, 2, 3, 4, 5, 6 or 7 frames ? Initially all frames are empty :

i) LRU ii) FIFO iii) Optimal.

Solution :

i) LRU

No. of page frame	1	2	3	4	5	6	7
No. of page fault	20	18	15	10	9	6	7

ii) FIFO

No. of page frame	1	2	3	4	5	6	7
Page fault	20	18	16	14	10	10	7

iii) Optimal

No. of page frame	1	2	3	4	5	6	7
Page fault	20	15	11	8	7	7	7

Example 4.10.5 *Consider the following reference string :*

A B C D A B E A B C D E

How many page faults would occur for the following replacement algorithms with 3 and 4 frames ?

i) FIFO replacement ii) Optimal replacement iii) LRU replacement Whether this reference string suffers from Belady's anomaly ? Explain.

Solution :

i) FIFO : (Page frame = 3)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	D	D	D	E	E	E	E	E	E
2	-	B	B	B	A	A	A	A	A	C	C	C
3	-	-	C	C	C	B	B	B	B	B	D	D
Page Fault	*	*	*	*	*	*	*			*	*	

Total number of page fault = 9

(Page frame = 4)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	E	E	E	E	D	D
2	-	B	B	B	B	B	B	A	A	A	A	A
3	-	-	C	C	C	C	C	C	B	B	B	B
4	-	-	-	D	D	D	D	D	D	C	C	C
Page Fault	*	*	*	*			*	*	*	*	*	*

Total number of page fault = 10

FIFO algorithm for this reference string suffers from Belady's anomaly.

ii) Optimal (Page frame = 3)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	A	A	A	C	C	C
2	-	B	B	B	B	B	B	B	B	B	D	D
3	-	-	C	D	D	D	E	E	E	E	E	E
Page Fault	*	*	*	*			*			*	*	

Total number of page fault = 7

(Page frame = 4)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	A	A	A	A	D	D
2	-	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	C	C	C	C	C	C
4	-	-	-	D	D	D	E	E	E	E	E	E
Page fault	*	*	*	*			*				*	

Total no. of page fault = 6

iii) LRU (Page frame = 3)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	D	D	D	E	E	E	C	C	C
2	-	B	B	B	A	A	A	A	A	A	D	D
3	-	-	C	C	B	B	B	B	B	B	B	E
Page fault	*	*	*	*	*	*	*			*	*	*

Total number of page fault = 10

(Page frame = 4)

	A	B	C	D	A	B	E	A	B	C	D	E
1	A	A	A	A	A	A	A	A	A	A	A	E
2	-	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	E	E	E	E	D	D
4	-	-	D	D	D	D	D	D	D	C	C	C
Page fault	*	*	*	*			*			*	*	*

Example 4.10.6 *If the reference string is 1, 2, 3, 4, 5, 2, 5, 3, 2, 5, 4, 1, 5 and the maximum number of pages which can be stored at a time in memory is 3 then calculate the number of page fault when LRU (least recently used) page replacement algorithm is used.*

Solution :

LRU

	1	2	3	4	5	2	5	3	2	5	4	1	5
1	1	1	1	4	4	4	4	3	3	3	4	4	4
2	-	2	2	2	5	5	5	5	5	5	5	5	5
3	-	-	3	3	3	2	2	2	2	2	2	1	1
Page Fault	*	*	*	*	*	*		*			*	*	

Total number of page fault = 9.

Example 4.10.7 *Consider the following page reference string :*

A B C D B A E F B A B C E G C B A B

How many page faults will occur for LRV, FIFD and optimal page replacement algorithms, assuming four available frames ?

Solution :

i) LRU (page frame = 4)

	A	B	C	D	B	A	E	F	B	A	B	C	E	G	C	B	A	B
1	A	A	A	A	A	A	A	A	A	A	A	A	A	G	G	G	G	G
2	-	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	E	E	E	E	E	C	C	C	C	C	C	C

4	-	-	-	D	D	D	D	F	F	F	F	F	E	E	E	E	A	A
Page Fault	*	*	*	*			*	*				*	*	*			*	

Total number of page fault = 10

ii) FIFO (page frame = 4)

	A	B	C	D	B	A	E	F	B	A	B	C	E	G	C	B	A	B
1	A	A	A	A	A	A	E	E	E	E	E	C	C	C	C	C	C	C
2	-	B	B	B	B	B	B	F	F	F	F	F	E	E	E	E	E	E
3	-	-	C	C	C	C	C	C	B	B	B	B	B	G	G	B	B	B
4	-	-	-	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A
Page Fault	*	*	*	*			*	*	*	*		*	*	*		*		

Total number of page fault = 12

iii) OPTIMAL (page frame = 4)

	A	B	C	D	B	A	E	F	B	A	B	C	E	G	C	B	A	B
1	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	-	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
3	-	-	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
4	-	-	-	D	D	D	E	F	F	F	F	F	E	G	G	G	G	G
Page Fault	*	*	*	*			*	*					*	*				

Total number of page fault = 8

Example 4.10.8 *Consider the following reference string for pages :*

0, 1, 1, 2, 3, 4, 3, 3, 1, 0, 2

Apply page replacement policies-LRU and optimal on this reference string assuming three and four frames respectively.

Solution :

i) LRU (Page frame = 3)

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	3	3	3	3	3	3	2
2	-	1	1	1	1	4	4	4	4	0	0
3	-	-	-	2	2	2	2	2	1	1	1

Page Fault	*	*		*	*	*			*	*	*

Total number of page fault = 8

Page frame = 4

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	0	4	4	4		4	2
2	-	1	1	1	1	1	1	1		1	1
3	-	-	-	2	2	2	2	2	2	0	0
4	-	-	-	-	3	3	3	3	3	3	3
Page Fault	*	*		*	*	*				*	*

ii) Optimal (Page frame = 3)

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	0	4	4	4	4	0	0
2	-	1	1	1	1	1	1	1	1	1	2
3	-	-	-	2	3	3	3	3	3	3	3
Page Fault	*	*		*	*	*				*	*

Total number of page fault = 7

Page frame = 4

	0	1	1	2	3	4	3	3	1	0	2
1	0	0	0	0	0	0	0	0	0	0	0
2	-	1	1	1	1	1	1	1	1	1	1
3	-	-	-	2	2	4	4	4	4	4	2
4	-	-	-	-	3	3	3	3	3	3	3
Page Fault	*	*		*	*	*					*

Total number of page fault = 6

Example 4.10.9 *Assume a reference string of (7012030423). With page frame size 03, calculate number of page faults with the following algorithms :*
i) LRU ii) FIFO.

Solution :

i) LRU (Page frame = 3)

	7	0	1	2	0	3	0	4	2	3
1	7	7	7	2	2	2	2	4	4	4
2	-	0	0	0	0	0	0	0	0	3
3	-	-	1	1	1	3	3	3	2	2
Fault	*	*	*	*		*		*	*	*

Total number of page fault = 8

ii) FIFO

	7	0	1	2	0	3	0	4	2	3
1	7	7	7	2	2	2	2	4	4	4
2	-	0	0	0	0	3	3	3	2	2
3	-	-	1	1	1	1	0	0	0	3
Fault	*	*	*	*		*	*	*	*	*

Total number of page fault = 9

Example 4.10.10 *Consider the following page reference string :*
1, 2, 3, 1, 4, 5, 6, 2, 1, 3, 2, 7, 6, 3, 4, 1, 2, 6
How many page faults would occur for the LRU replacement algorithm assuming six frames ? All frames are initially empty.

Solution :

LRU

	1	2	3	1	4	5	6	2	1	3	2	7	6	3	4	1	2	6
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	-	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	-	-	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

4	-	-	-	-	4	4	4	4	4	4	4	7	7	7	7	7	7	7
5	-	-	-	-	-	5	5	5	5	5	5	5	5	5	4	4	4	4
6	-	-	-	-	-	-	6	6	6	6	6	6	6	6	6	6	6	6
Page fault	*	*	*		*	*	*					*			*			

Total number of page fault = 8.

4.10.6 Second Chance Algorithm

- Second chance algorithm is also called clock algorithm. It maintains reference bit per page.
- Goal : Remove a page that has not been referenced recently.
- Second chance algorithm is actually a FIFO replacement algorithm with a small modification that causes it to approximate LRU. It uses reference bit. When a page is selected according to a FIFO order, user checks its reference bit.
- It uses a circular buffer with as many slots as there are frames in memory. Each buffer slot contains a record {page number; reference bit}, for a page currently in memory.
- If reference bit is set then page was referenced and this page is not removed otherwise page is removed. When a page is replaced, a pointer is set to point to the next frame in buffer.
- Second chance is looking for an old page that has not been referenced in the most recent clock interval. If all the pages have been referenced, second chance degenerates into pure FIFO.
- It requires hardware support for reference bit.
- A reference bit for each frame is set to 1 in following conditions :
  1. Page is first time loads into the frame memory.
  2. When page frame is referenced.
- When it is time to replace a page, the first frame encountered with the reference bit set to 0 is replaced. During the search for replacement, each reference bit set to 1 is changed to 0.

Example :

	0	1	3	6	2	4	5	2	5	0	3	1	2	5	4	1	0
0	0	0	0	0	2	2	2	2	2	2	3	3	3	3	4	4	4
1	-	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1
2	-	-	3	3	3	3	5	5	5	5	5	5	2	2	2	2	0
3	-	-	-	6	6	6	6	6	6	0	0	0	0	5	5	5	5
Page Fault	PF	PF	PF	PF	PF	PF	PF			PF	Pf	PF	PF	PF	PF		PF

Total number of page fault = 14

Example 4.10.11 *Consider the following reference string:*

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 assume there are three frames which are initially empty. Using FIFO page replacement algorithm determine the number of page fault for the reference string above. Provide necessary diagram for page frames.

Solution : FIFO page replacement algorithm

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
2	-	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
3	-	-	1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
Page Fault	PF	PF	PF	PF		PF	PF	PF	PF	PF	PF			PF	PF			PF	PF	PF

Number of page fault = 15

Example 4.10.12 *Discuss situations in which the least frequently used (LFU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm. Also discuss under what circumstances the opposite holds good.*

AU : May-16, Marks 8

Solution : Consider the following sequence of memory accesses in a system that can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1, and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence "1 2 3 4 5 2," the least recently used algorithm performs better.

University Questions

1. *Consider the following page reference string 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8 9, 7, 8, 9, 5, 4, 4, 5, 3. How many page faults would occur for the following replacement algorithms ? Assume four frames and all frames are initially empty.*
   i) LRU replacement ii) FIFO replacement iii) Optimal replacement.
   AU : CSE/IT : May-15, Marks 16
2. *Discuss situations in which the most frequently used (MFU) page replacement algorithm generates fewer page faults than the least recently used (LRU) page-replacement algorithm. Also discuss under what circumstances the opposite holds.* **AU : Dec.-16, Marks 8**
3. *When do page faults occur ? Consider the reference string :*
   1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6,
   how many page faults and page fault rate occur for the FIFO, LRU and optimal replacement algorithms, assuming one, two, three, four page frames ? **AU : Dec.-17, Marks 13**
4. *Discuss situation under which the most frequently used page replacement algorithm generates fewer page faults than the least recently used page replaement algorithm. Also discuss under which circumstances the opposite holds.* **AU : May-18, Marks 13**

4.11 Allocation of Frames

- In the last section, we discussed local page replacement policy. Now we have considered how to assign pages of a process logical memory to a fixed set of allocated frames. We also need to decide how many frames to allocate to each process. It is also possible to select the victim from a frame currently allocated to another process.
- There are fixed amount of free memory with various processes at different time in a system. The question is how this fixed amount of free memory is allocated among the different processes.
- Let us consider the single process system. All free memory for user programs can initially be put on the free frame list. When the user starts executing his program, it will generate a sequence of page faults. The user program would get all free frames from the free frame list.
- When this list was exhausted and the more free frames are required, the page replacement algorithm can be used to select one of the in-used pages to be replaced with the next required page and so on. After the program was terminated, all used pages are put on the free frame list again.
- Each process needs a certain minimum number of pages.

  i. Pages for instructions. ii. Pages for local data. iii. Pages for global data.

4.11.1 Equal Allocation

- Allocation may be fixed. Here "m" frames are splits among the "p" number of process.

$$\text{Frame allocation} = \frac{\text{Number of free frame}}{\text{Number of process}} = \frac{m}{p}$$

- For example, if there are 180 frames and 6 processes, give each process 30 frames.
- High priority jobs have same number of page frames and low priority jobs. Degree of multiprogramming might vary in equal allocation.
- Problem with equal allocation is that, there may be the wastage of free frames. Some process may require less number of free frames than allocated one. To solve this problem, proportional allocation method is used.
- **Proportional allocation :** Processes that have more logical memory get more frames.

Priority allocation

- In priority allocation, higher priority processes get more frames. System uses a proportional allocation scheme using priorities rather than size.
- If process P_i generates a page fault, then select for replacement one of its frames or select for replacement a frame from a process with lower priority number.

4.11.2 Global Vs Local Allocation

- If a process needs frames, should pages from other processes be discarded or just pages from the process which wants frames ? Local policy means giving each process a share of the physical memory and swapping pages in and out on a per process basis.
- **Global allocation :** One process can select a replacement frame from the set of all frames. Process may not be able to control its page fault rate.
- Advantages of global allocation :
  i. Flexibility of allocation. ii. Minimize total number of page faults
- Disadvantages of global allocation
  1. One memory-intensive process can hog memory, hurt all processes
- **Local allocation :** Each process can only select from its own set of allocated frames. Process slowed down even if other less used pages of memory are available.
- **Per process local replacement :**

Advantage : No interference across processes.

Disadvantage : Potentially inefficient allocation of memory.

- **Per user local replacement :** Each user has separate pool of pages

Advantage : Fair across different users.

Disadvantage : Allocation is inefficient.

- Local page replacement is more predictable; depends on no external factors. A process which uses global page replacement cannot predict the page fault rate; may execute in 0.5 seconds once and 10.3 on another run. Overall, global replacement results in greater system throughput.

Example 4.11.1 *Consider a system having 64 frames and with 4 processes. The virtual memory size is as follows :*

v(1) = 16, v(2) = 128, v(3) = 64, v(4) = 48. Allocate free page frames by using equal allocation and proportional allocation policy.

Solution :

$$\textbf{Frame allocation} = \frac{\text{Number of free frame}}{\text{Number of process}} = \frac{64}{4} = 16 \text{ frames per process}$$

Proportional allocation :

Sum of all virtual memory size (v) = v(1) + v(2) + v(3) + v(4)
= 16 + 128 + 64 + 48
= 256

Then it allocates is following ways :

Process 1 = (16 / 256) * 64
= 4 frames

Process 2 = (128 / 256) * 64
= 32 frames

Process 3 = (64 / 256) * 64
= 16 frames

Process 4 = (48 / 256) * 64
= 12 frames

4.12 Thrashing

AU : Dec.-15

- Thrashing occurs when a process does not have "enough" frames allocated to store the pages it uses repeatedly, the page fault rate will be very high.
- A process is thrashing if it is spending more time for paging in/out than executing. Since thrashing leads to low CPU utilization, the operating system's medium or long term scheduler may step in, detect this and increase the degree of multiprogramming.

- Local replacement algorithms can limit the effects of thrashing. If the degree of multiprogramming is increased over a limit, processor utilization falls down considerably because of thrashing. Fig. 4.12.1 shows thrashing.

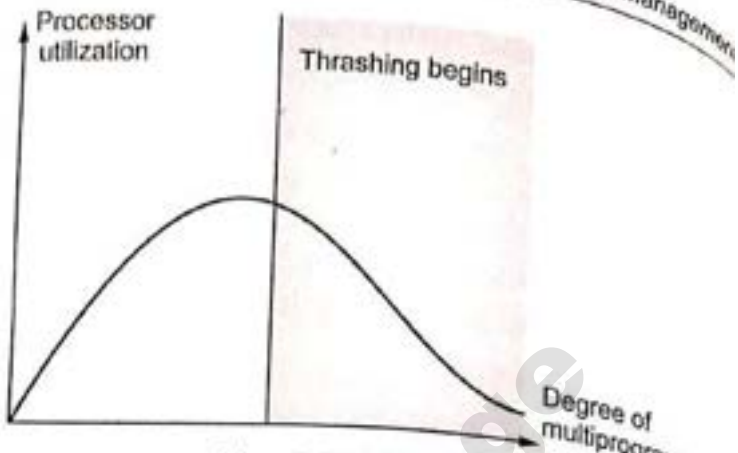


Fig. 4.12.1 Thrashing

- The selection of a replacement policy to implement virtual memory plays an important part in the elimination of the potential for thrashing. A policy based on the local mode will tend to limit the effect of thrashing. Replacement policy based on the global mode is more likely to cause thrashing. Since all pages of memory are available to all transactions, a memory-intensive transaction may occupy a large portion of memory, making other transactions susceptible to page faults and resulting in a system that thrashes.
- Thrashing is solved by using working set model and page fault frequency.

4.12.1 Working Set Model

- Working sets model is proposed by Peter Denning to prevent thrashing.
- We define the working set of information $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t-\tau, t)$. Fig. 4.12.2 shows working set model.

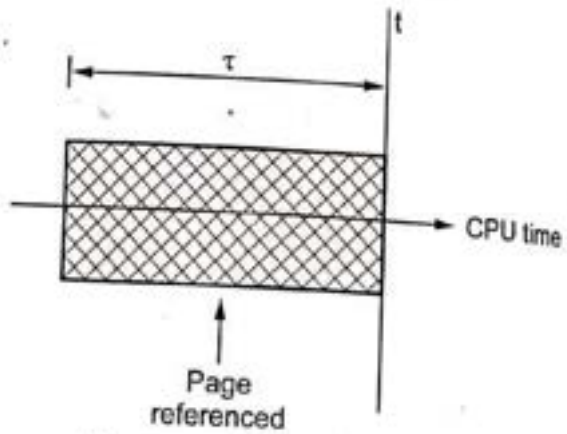


Fig. 4.12.2 Working set model

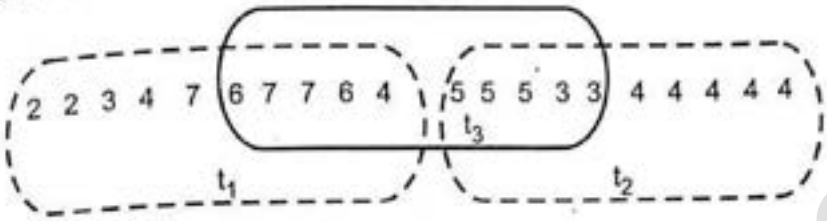
- Here τ is the working set parameter. The elements of $W(t, \tau)$ are pages but they can be anything else used by a process.
- If the sum of all working sets of all runnable threads exceeds the size of memory, then stop running some of the threads for a while. Divide processes into two groups : active and inactive.
- When a process is active its entire working set must always be in memory : never execute a thread whose working set is not resident. When a process becomes inactive, its working set can migrate to disk.

- Threads from inactive processes are never scheduled for execution. The collection of active processes is called the balance set. The system must have a mechanism for gradually moving processes into and out of the balance set. As working sets change, the balance set must be adjusted.

How to compute working sets ?

Δ = Working set windows size.

Example : Assume windows size is 10 and the reference string given below, on which the window is shown at different time instants.



Working set for process

$WS(t_1) = \{2, 3, 4, 6, 7\}$ $WS(t_2) = \{5, 3, 4\}$

$WS(t_3) = \{6, 7, 4, 5, 3\}$

Page Fault Frequency

- Page fault frequency is used to preventing thrashing. Page fault rate is controlled by using this method. In the per-process replacement policy, each process is allocated a fixed number of physical page frames. Then monitor the rate at which page faults are occurring for each process.

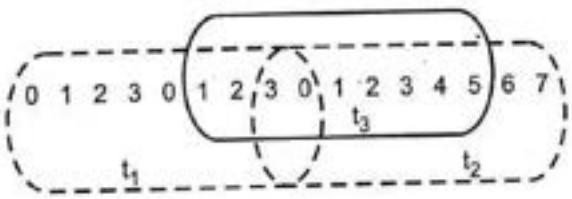
1. If the rate gets too high for a process, assume that its memory is overcommitted; increase the size of its memory pool.
2. If the rate gets too low for a process, assume that its memory pool can be reduced in size.
3. If the sum of all memory pools doesn't fit in memory, deactivate some processes.

Example 4.12.1 *Given the following reference string :*

0 1 2 3 0 1 2 3 0 1 2 3 4 5 6 7

Determine $WS(t_i)$ with $\Delta = 9$.

Solution :



$WS(t_1) = \{0, 1, 2, 3\}$

$WS(t_2) = \{3, 0, 1, 2, 4, 5, 6, 7\}$

$WS(t_3) = \{1, 2, 3, 0, 4, 5\}$

4.12.2 Locality of Reference

- Locality of reference is also known as the principle of locality. Many applications continually reference large amounts of data.
- Locality of reference is observes that an application does not access all of its data at once with equal probability. Instead, it accesses only a small portion of it at any given time.
- There are two basic types of reference locality.
  1. Temporal locality 2. Spatial locality

  1. Temporal locality : Temporal locality is locality over time. If at one point in time a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future.

  2. Spatial locality : If a particular memory location is referenced at a particular time, then it is likely that nearby memory locations will be referenced in the near future.

University Question

1. *Explain the effect of thrashing.* **AU : Dec.-15, Marks 4**

4.13 Allocating Kernel Memory

AU : May-17

- Kernel creates and destroys many data structure during its operation. Kernel makes efficient use of memory.
- The kernel reserves part of physical memory for its own text and static data structures.
- Page-level allocator : This does the job of allocating memory. The page-level allocator has two principle clients, paging system and kernel memory allocator.
- Paging system : Is a part of virtual memory system. It allocates pages to user process to hold portions of their address space.
- Kernel memory allocator : Provides buffers of memory to various kernel subsystems.
- Kernel allocator services requests for dynamic memory allocation for its clients not for user processes. Must use the allocated space efficiently - allocation may be fixed or not, it may be possible to exchange it with paging system. If it runs out of memory it blocks the caller, until more memory is free. It must monitor which parts of its pool are allocated and which are free.

4.13.1 Buddy System

- Linux operating system manages memory using buddy algorithm. The buddy system is a simple dynamic storage allocation method. It combines free buffer coalescing with a power of 2 allocator. It is described by Knowlton and Knuth.
- Single allocation block to be split, to form two blocks half the size of the parent block. These two blocks are known as 'buddies'.
- Initially memory consists of a single contiguous area of 128 pages. When user send request for memory, it is first rounded up to a power of 2.
- For example : If the minimum allocation size is 8 bytes and the memory size is 1 MB then we can create a list for 8 bytes hole, a list for 16 byte holes, one for 32-bytes holes, 64, 128, 256, 512, 1 K, 2 K, 4 K, 8 K, 16 K, 32 K, 64 K, 128 K, 256 K, 512 K and one list for 1 MB holes.
- All the lists are initially empty, except for the 1 MB list, which has one hole listed. All allocations are rounded up to a power of two. Suppose the request for memory is 75 K then allocations rounded up to 128 K, 13 K allocations rounde up to 16 K, etc.
- If a block of that size is free, it is taken; otherwise, the smallest free block larger than the desired size is found and split in two halves. This splitting continuous until the appropriate size is reached.
- When memory is deallocated, the buddy system groups contiguous free pages. When process free a block of memory, the memory manager checks the bitmap for checking block size. If adjacent block is also free, the memory manager combines the two blocks into a larger blocks.

Example

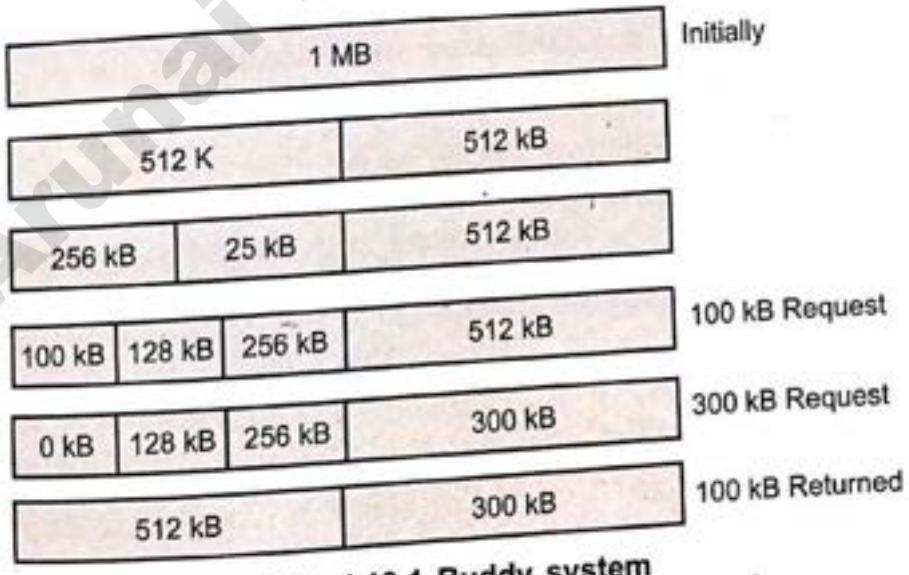


Fig. 4.13.1 Buddy system

- This method makes deallocation fast. If a block of size 2^K is free then the memory manager checks only the list of 2^K holes to merge them into a 2^{K+1} sized partition. Internal fragmentation is caused since memory requests are fitted in 2^k sized partitions.

Advantages

1. Allow memory to be reused.
2. Easy exchange of memory between the allocator and the paging system.
3. It provides flexibility.

Disadvantages

1. It wastes lot of space in internal fragmentation.
2. Programming interface is one more drawback.
3. Merging of holes is recursive, so poor worst case behaviour.

4.13.2 Slab Allocator

- Slab allocator allocates memory from any one of a number of available slab cache. A slab is made up of one or more physically contiguous pages.
- Slab cache consists of number of objects. A slab contains objects of only one kind. Each object is a kernel data structure. The slab allocator allocates all kernel objects of same class together in a pool.
- When the kernel requests memory for a new structure, the slab allocator returns a portion of a slab in the slab cache for that structure. If all of the slabs in a cache are occupied, the slab allocator increases the size of the cache to include more slabs.
- A slab is organized in a standard sized area allocated by the paging system. The slabs of a pool are entered in doubly link list for addition and deletion of slabs.
- Slab allocator divides the slab into three parts :
  1. Kernel memory slab structure
  2. Set of objects
  3. Unused space.
- Fig. 4.13.2 shows slab organization.
- Slab states are as follows :
  1. Full
  2. Partially empty
  3. Empty.

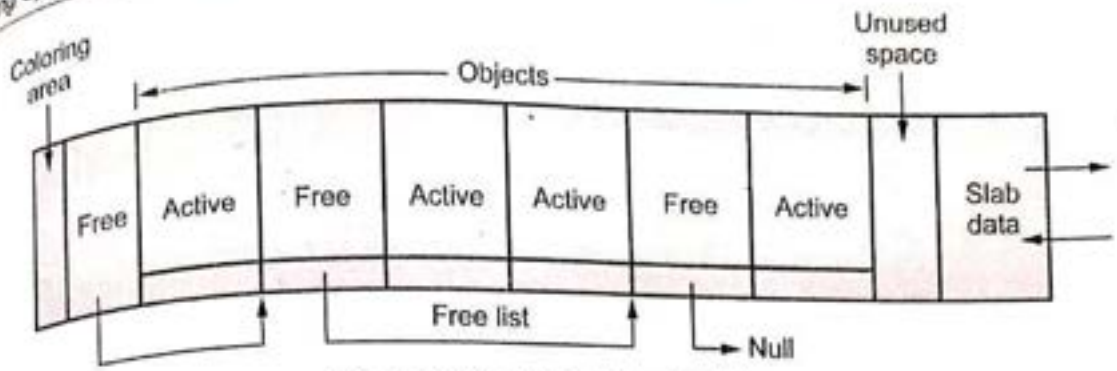


Fig. 4.13.2 : Slab organization

- These three states depends upon number of active objects.
- The kernel allocates an object from a slab by removing the first element from the free list and incrementing the slab's in-use count. When freeing the object, it identifies the slab by a simple calculation :

$$\text{Slab address} = \frac{\text{Object address}}{\text{Slab size}}$$

- When in-use count becomes zero, the slab is free.
- Slab allocator introduced in Solaris 2.4 operating system.

Advantages / Benefits

1. Slab allocator provides better memory utilization.
2. Requests are serviced quickly by removing an object from free list.
3. It is space efficient.

Drawback

1. Overhead for having separate cache for each type of object.

University Question

1. *Discuss the concept of buddy system allocation with neat sketch.* **AU : May-17, Marks 6**

4.14 32 and 64 Bit Architecture Examples

- Intel 80 × 86 has a 4 gigabyte address space, with a page size of 4096 bytes. It supports paging and segmentation.
- Fig. 4.14.1 shows address translation on the Intel 8086.
- Virtual address consists of a 16-bit segment selector and 32-bit offset. Process address space may contain up to 8192 segments. Each process consists of local descriptor table with one entry for each of its segments.

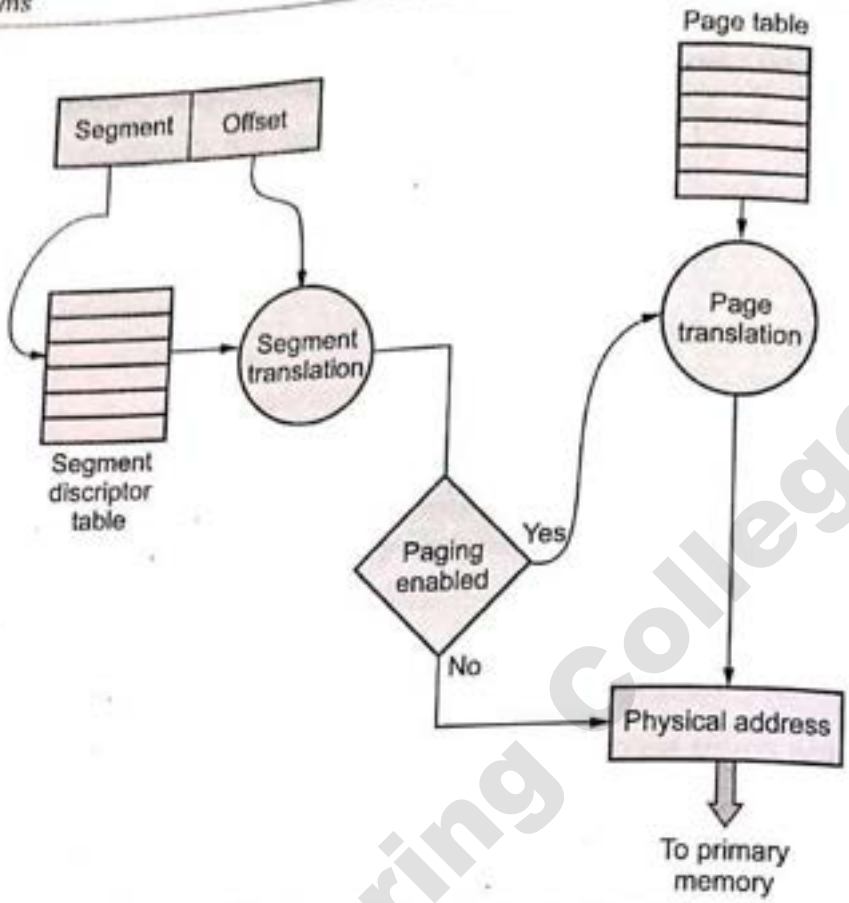


Fig. 4.14.1 Intel 8086 address translation

- Each segment contains segment descriptor. Base, size and protection bit are the components of segment descriptor.
- System also maintains global descriptor table. GDT maintains information about kernel code, stack segment, data etc.
- Intel 8086 uses a two level page table structure. Each process maintain small page table. Each page table is one page in size and holds 1024 PTEs. It maps a contiguous region 4 MB is size.
- Each process also has a page table which contains PTE. Page directory is the level-1 page table and page tables themselves are the level-2 tables.
- Each page table entry contains the physical page number, protection field, valid, modified and reference bits. Fig. 4.14.2 shows page table entry.

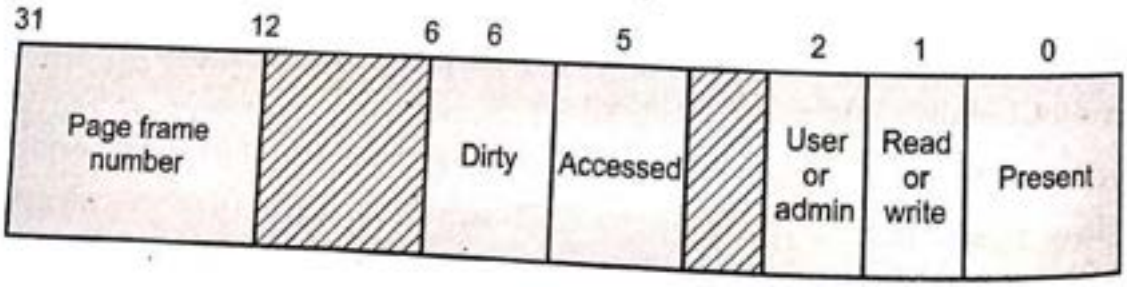


Fig. 4.14.2 Page table entry

4.15 OS Examples

AU : May-15

4.15.1 Windows 8 Memory Management

- Virtual memory allows multiple processes to share physical memory without being able to corrupt one another's data. In an OS with virtual memory, each program has its own virtual address space, a logical region of addresses whose size is dictated by the address size on that system. Regions in a process's virtual address space can be mapped to physical memory, to a file, or to any other addressable storage. The OS can move data held in physical memory to and from a swap area when it is not being used, to make the best use of physical memory.
- Each process address space is private and cannot be accessed by other processes unless it is shared. The virtual address space for a process is the set of virtual memory addresses that it can use.
- A virtual address does not represent the actual physical location of an object in memory; instead, the system maintains a page table for each process, which is an internal data structure used to translate virtual addresses into their corresponding physical addresses.
- Each time a thread references an address, the system translates the virtual address to a physical address.
- Windows 8 has a better scheme for the prioritization of memory allocations made by applications and system components. This means that Windows can make better decisions about what memory to keep around and what memory to remove sooner.

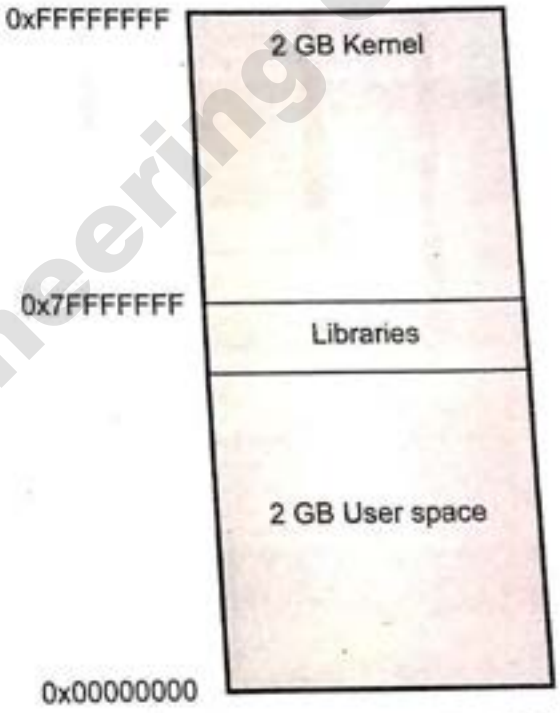


Fig. 4.15.1 Windows default 32-bit address space

- In Windows 8, any program has the ability to allocate memory as "low priority." This is an important signal to windows that if there is memory pressure, windows can remove this low priority memory to make space, and it does not affect other memory required to sustain the responsiveness of the system.
- The virtual address space for 32-bit Windows is 4 GB in size and divided into two partitions. Fig. 4.15.1 shows Windows default 32-bit address space.

- The process address space must contain everything a program requires including the program itself and the shared libraries that it uses. The 2 GB of space is the same for all processes and is shared by all processes.

Memory combining

- Memory combining is a technique in which windows efficiently assesses the content of system RAM during normal activity and locates duplicate content across all system memory. Windows will then free up duplicates and keep a single copy.
- Fig. 4.15.2 shows the windows API provides different levels of memory management for versatility in application programming.

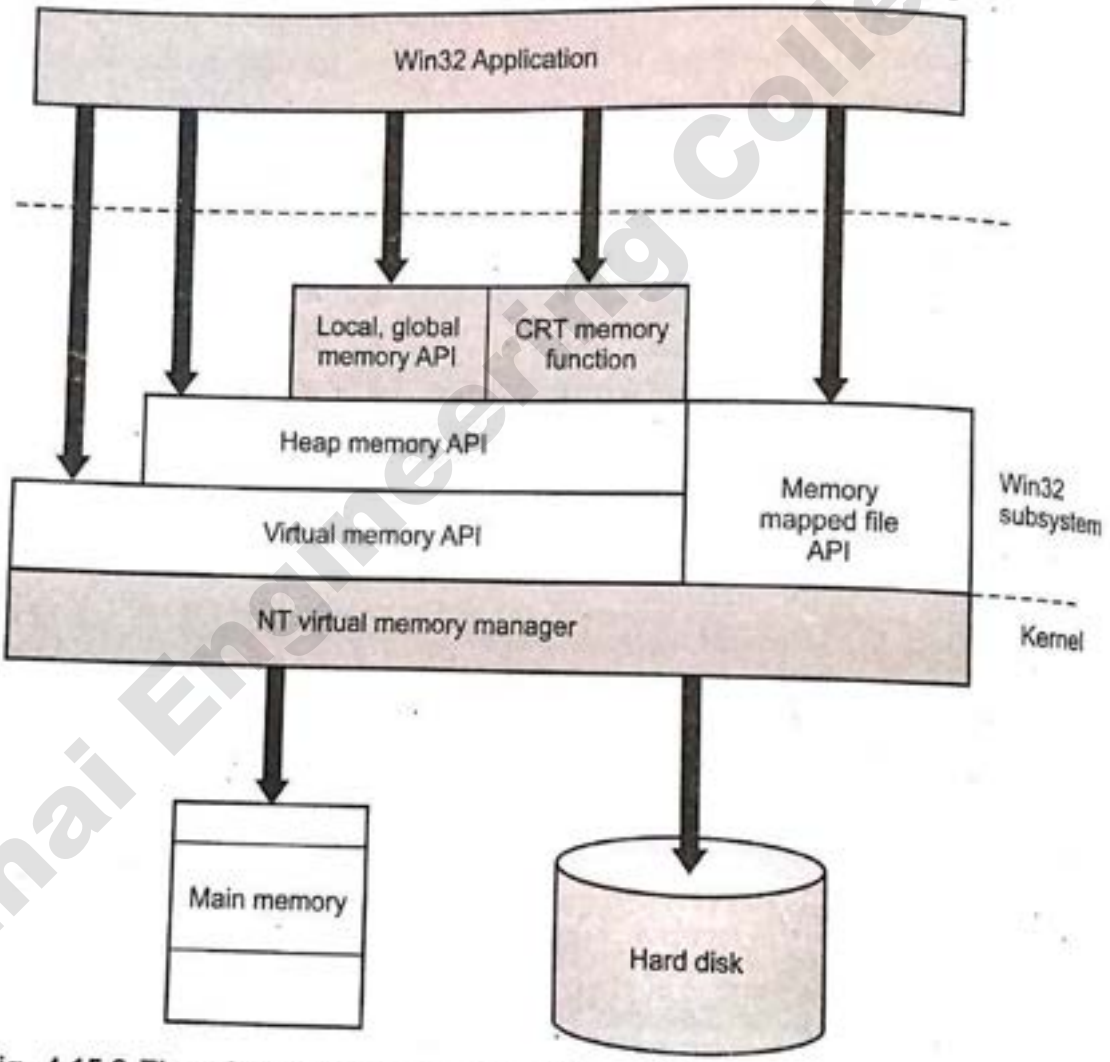


Fig. 4.15.2 The windows API provides different levels of memory management

- If the application tries to write to the memory in future, windows will give it a private copy. All of this happens under the covers in the memory manager, with no impact on applications.

- The memory manager creates the following memory pools that the system uses to allocate memory : *Non-paged pool* and *Paged pool*.
- Both memory pools are located in the region of the address space that is reserved for the system and mapped into the virtual address space of each process. The non-paged pool consists of virtual memory addresses that are guaranteed to reside in physical memory as long as the corresponding Kernel objects are allocated. The paged pool consists of virtual memory that can be paged in and out of the system.
- Windows offers three groups of functions for managing memory in applications : *Memory-mapped file functions, heap memory functions,* and *virtual memory functions.*

Paging for windows

- Process address space is upto 2 GB. This space is divided into fixed size pages. Operating system manages these pages into regions. The regions are as follows :
  1. **Available :** Currently not used by process.
  2. **Reserved :** Virtual memory manager reserved for process and it can not allocated to another use.
  3. **Committed :** Addresses for which the virtual memory manager has initialized for use by the process to access virtual memory pages.

University Question

1. *Write short note on memory mapped files.* **AU : CSE/IT : May-15, Marks 6**

4.16 Copy on Write

AU : Dec.-16

- The fork ( ) system call works by creating a copy of parent's address space for the child process. Instead of copying the entire memory, the child copies just the page tables of the parent process and points to all of the same pages. The pages are marked with a special **Copy on Write (COW) bit**.
- Fig. 4.16.1 shows copy on write implementation.
- When the contents of memory are to be updated, the COW bit is checked. If it is set, a new page is allocated, the data from the old page copied, the update is made on the new page and the "copy-on-write" bit is cleared for the new page.
- If one process tries to alter a page, a memory error occurs. The operating system receives an interrupt, sees that the page is shared, makes a copy of the page by physically copying the contents to another page frame and mapping the second page frame to the current process and gets the process to retry the memory access.

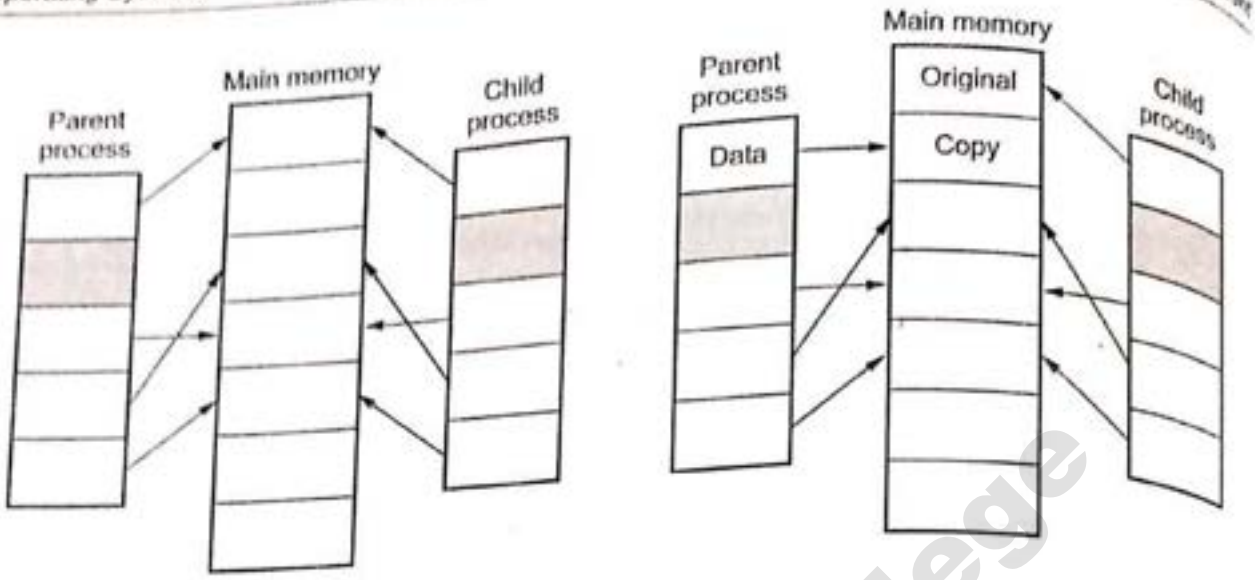


Fig. 4.16.1 CoW implementation

- In this way the simple case of fork( ) and then exec of a simple program does not use any more memory resource except that needed to hold the page tables which is a small percentage of the total virtual memory used.
- Linux, Solaris and Windows operating system used copy on write method.

University Question

1. *What is the copy-on-write feature and under what circumstances is its use beneficial ? What hardware support is required to implement this feature ?* **AU : Dec.-16, Marks 8**

Two Marks Questions with Answers

Q.1 What is internal fragmentation ?

Ans. : Internal fragmentation exists when the smallest available block is larger than the requested memory. It is the memory which is internal to a partition, but is not being used.

Q.2 What is an overlay ? What is the use of it ?

Ans. : It allows a process to execute despite to the system having insufficient physical memory. The idea of overlay is to keep in memory only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded into space that was occupied previously by instructions that are no longer needed. Overlay do not require any special support from the operating system.

Q.3 Distinguish between internal and external fragmentation.

Ans. : Internal fragmentation is the area in a region or a page that is not used by the process it is allocated to. The space is wasted until the process terminates. External

fragmentation occurs when there is enough free space to satisfy a request for memory, but none of the free holes between processes in memory is large enough to satisfy the request.

Q.4 Bring out the disadvantages of fixed partition multiprogramming.

Ans. : Disadvantages of fixed partition multiprogramming

a. Limited degree of multiprogramming.

b. Internal fragmentation.

c. Placement policies.

Q.5 State the function of memory manager.

Ans. : Functions of memory manager

- Allocates primary memory to processes.
- Maps process address space to primary memory.
- Minimizes access time using cost-effective memory configuration.
- May use static or dynamic techniques

Q.6 What are memory pages and segments ?

Ans. : Logical memory is also broken into blocks of the same size called **pages**. It divides a program into a smaller block called segments, each of which is allocated to memory independently.

Q.7 What is quick fit memory allocation ?

Ans. : The quick fit algorithm takes a different approach to those we have considered so far. Separate lists are maintained for some of the common memory sizes that are requested. For example, we could have a list for holes of 4 K, a list for holes of size 8 K etc. One list can be kept for large holes or holes which are not fit into any of the other lists.

Quick fit allows a holes of the right size to be found very quickly, but it suffers in that there is even more list maintenance.

Q.8 Define zero level paging.

Ans. : There is no paging is used. The TLB is large enough to hold all the entries which is required for processing.

Q.9 Name the dynamic memory allocation techniques based on size.

Ans. : Segmentation.

Q.10 Specify the three ways used to keep track of the memory usage by O.S.

Ans. : MMU, Overlays, Swapping.

Q.11 Name two differences between logical and physical addresses. **AU : May-16**

Ans. : A logical address does not refer to an actual existing address; rather, it refers to an abstract address in an abstract address space. Contrast this with a physical address that refers to an actual physical address in memory. A logical address is generated by the CPU and is translated into a physical address by the memory management unit (MMU). Therefore, physical addresses are generated by the MMU.

Q.12 Differentiate segmentation from paging.

Ans. : In segmentation, program is divided into variable size segments whereas program is divided into fixed size pages in paging. Segmentation is visible to user and paging is invisible.

Q.13 Explain the advantages of inverted page tables.

Ans. : Advantages

1. It decreases the amount of memory needed to store each page table.
2. It limit the search using TLB.
3. Separate hardware is not required.

Q.14 What is the purpose of paging the page tables ? AU : EIE : Dec.-16

Ans. : In certain situations the page tables could become large enough that by paging the page tables, one could simplify the memory allocation problem (by ensuring that everything is allocated as fixed size pages as opposed to variable sized chunks) and also enable the swapping of portions of page table that are not currently used.

Q.15 What is a translation look aside buffer used for ?

Ans. : Problem with paging is that, extra memory references to access translation tables can slow programs down by a factor of two or three. Too many entries in translation tables to keep them all loaded in fast processor memory. To solve this problem TLB is used. TLB is used to store a few of the translation table entries.

Q.16 What is segmentation ?

Ans. : Segmentation divides a program into a number of smaller blocks called segments.

Q.17 Why should paging be used by operating systems ?

Ans. : Paging permits the physical address space of process to be non-contiguous.

Q.18 Explain associative mapping.

Ans. : A main memory block can be placed into any cache block position. As there is no fix block, the memory address has only two fields : word and tag. This technique is called associative mapping.

Q.19 Explain the type of fragmentation present in paging and segmentation systems.

Ans. : Paging : Internal fragmentation.

Segmentation : External fragmentation.

Q.20 Discuss the salient features and merits of multilevel paging and inverted page tables.

Ans. : Features and merits of multilevel paging

1. It solves the problem of large logical address space.
2. Operating system can leave partitions unused until a process needs them.
3. It reduces main memory use.

Features and merits of inverted page tables

1. It decreases the amount of memory needed to store each page table.
2. Associative memory access is very fast which is used in inverted page table.

Q.21 What are the consequences of multiprogramming with fixed partitioning and variable partitioning ?

Ans. : 1. Fixed partitioning

- **Advantages**

i) Simple to implement.

ii) Overhead is less.

- **Disadvantages**

i) Suffer from internal fragmentation.

ii) Inefficient use of memory.

2. Variable partitioning

- **Advantages**

i) Multiprogramming.

ii) Better utilization of processor and I/O devices.

iii) Easy to implement.

- **Disadvantages**

i) Fragmentation.

ii) Require contiguous memory location.

iii) Some memory may never be utilized.

Q.22 Explain how memory can be dynamically allocated using first fit, best fit and worst fit strategies.

Ans. : First fit : Take the first available hole in memory that is of adequate size. Fast but leads to fragmentation.

Best fit : Take the hole that most closely matches (atleast as big as) the size of the program. Usage efficient but slow.still leads to fragmentation.

Worst fit : Take the biggest possible hole. Aims to reduce the number of little and wasted memory holes. Still leads to fragmentation.

Q.23 Explain what the use of a page table is and how it is used.

Ans. : Use of page table,

1. It shows the frame location for each page of the process.
2. Processor uses page table to produce a physical address.

Q.24 Describe non-contiguous memory allocation.

Ans. : For N memory blocks, the loss of 0.5 N blocks is possible due to external fragmentation. This means that one-third of memory is not usable. But compaction is too costly to perform regularly. External fragmentation arises because we are trying to allocate memory contiguously. We can deal with external fragmentation if we can allow process memory to be non-contiguous.

Q.25 Explain the consequences of swapping.

Ans. :

a. Context switching time is fairly high.
b. Only idle process must swap.
c. Transfer time is directly proportional to the amount of memory swapped.
d. It increase the O.S. overheads.

Q.26 What is address binding ? AU : CST/IT : Dec.-10

Ans. : The process of associating program instructions and data to physical memory addresses is called address binding, or relocation.

Q.27 What are the disadvantages of single contiguous memory allocation ? AU : CST/IT :Dec.-11

Ans. : Disadvantages :

1. Memory is not fully utilized. 2. Poor utilization of processors.

Q.28 What is page frame ? AU : CST/IT : Dec.-11

Ans. : Physical memory unit is called page frame.

Q.29 Why are segmentation and paging sometimes combined into one scheme ? AU : CST/IT : May-12

Ans. : Segmentation and paging are often combined in order to improve upon each other. Segmented paging is helpful when the page table becomes very large. A large contiguous section of the page table that is unused can be collapsed into a single segment table entry with a page table address of zero. Pages segmentation handles the case of having very long segments that require a lot of time for allocation. By paging the segments, we reduce wasted memory due to external fragmentation as well as simplify the allocation.

Q.30 Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames. How many bits are there in the logical address and in the physical address. AU : CST/IT : May-12, 13

Ans. : Addressing within a 1024-word page requires 10 bits because $1024 = 2^{10}$. Since the logical address space consists of $8 = 2^3$ pages, the logical addresses mut be $10 + 3 = 13$ bits. Similary, since there are $32 = 2^5$ physical pages, physical addresses are $5 + 10 = 15$ bits long.

Q.31 What is memory ?

Ans. : Memory is a device used to store the data and instructions required for any operation.

Q.32 What is called pages ?

Ans. : The address space is usually broken into fixed-size blocks, called pages. Each page resides either in main memory or on disk.

Q.33 Why are pages sizes always powers of 2 ? AU : EIE : Dec.-16

Ans. : Paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

Q.34 Define external fragmentation ? AU : April / May-18

Ans. : Total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes.

Q.35 What is a page ?

Ans. : Divide logical memory into blocks of same size called pages.

Q.36 What is frame table ?

Ans. : Allocation and availability of the frame information is kept in a data structure called a frame table. Frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and if it is allocated, to which page of which process or processes.

Q.37 It is possible for a process to have two working sets, one representing data and another representing code ? Explain.

Ans. : Yes, in fact many processors provide two TLBs for this very reason. As an example, the code being accessed by a process may retain the same working set for a long period of time. However, the data the code accesses may change, thus reflecting a change in the working set for data accesses.

Q.38 Why should we use virtual memory ?

Ans. : Virtual memory allows execution of partially loaded processes.

Q.39 Explain why spin locks are not appropriate for uniprocessor system yet may be suitable for multiprocessor system.

Ans. : Using spin lock, no context switch is required when a process must wait on a lock and a context switch may take considerable time. When locks are expected to be held for short times, spin locks are useful.

Q.40 Define swap space.

Ans. : Secondary memory holds those pages that are not present in main memory. The secondary memory is usually a high speed disk. It is known as the swap device and the section of disk used for this purpose is known as swap space.

Q.41 What do you mean by page fault ? AU : CST/IT : Dec.-10

Ans. : A page fault is a trap to the software raised by the hardware when a program accesses a page that is mapped in the virtual address space, but not loaded in physical memory.

Q.42 Define TLB. AU : CST/IT : May-11

Ans. : TLB is the memory cache of the most recently used page table entires within the memory management unit.

Q.43 How do you limit the effects of thrashing ? AU : CST/IT : May-11

Ans. : To limit the effect of thrashing we can use local replacement algorithm. With local replacement algorithm, if the process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well. The problem is not entirely solved.

Q.44 What is demand paging ? AU : CST/IT : Dec.-11

Ans. : Determines when a page should be brough into memory. Demand paging only brings pages into main memory when a reference is made to a location on the page.

Q.45 What is virtual memory ? Mention its advantages. AU : CST/IT : Dec.-12

Ans. : Virtual memory is a technique that allows the execution of processes that are not completely in memory. Advantage of virtual memory is that programs can be larger than physical memory.

Q.46 Differentiate between global and local page replacement algorithms. AU : CST/IT : May-13

Ans. : Global page replacement considers all unblocked pages in main memory as candidates for replacement regardless for which process owns a particular page. Local page replacement chooses only among the resident pages of the process that generated the page fault in selecting a page to replace.

Q.47 What is meant by Belady's anomaly ? AU : CST/IT : May-13

Ans. : For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases.

Q.48 What is optimal page replacement ?

Ans. : The optimal policy selects for replacement the page that will not be used for longest period of time.

Q.49 What happens if the process tries to use a page that was not brought into memory ?

Ans. : Access to a page marked invalid causes a page-fault trap. Invalid bit indicates that the page either is not valid (not in logical address space) or is valid but is currently on the disk.

Q.50 What is the key distinction between FIFO and optimal algorithms ?

Ans. : The key distinction between FIFO and optimal algorithms is that FIFO uses the time when a page was brought into memory; the optimal uses the time when a page is to be used (future).

Q.51 Define a cache hit.

Ans. : When the CPU refers to memory and finds a required word in cache it is termed as cache hit.

Q.52 Define hit ratio.

Ans. : The ratio of the number of hits divided by the total CPU references to memory is the hit ratio.

Q.53 Define a miss.

Ans. : When the CPU refers to memory and if the required word is not found in cache it is termed as miss.

Q.54 What is temporal locality ?

Ans. : Recently referenced items are likely to be referenced again in the near future. This is often caused by special program constructs such as iterative loops, process stacks, temporary variables or subroutines.

Q.55 Define spatial locality.

Ans. : This refers to the tendency for a process to access items whose addresses are near one another.

Q.56 Write a formula for average memory access time.

Ans. : Average memory access time = Hit time + Miss rate × Miss penalty

Q.57 What do you mean by 'Thrashing' ? **AU : CSE/IT, May-15**

Ans. : It is a situation in which a process is spending more time paging than executing.

Q.58 Mention the two main approaches to identify and reuse free memory area in a heap.

Ans. : Two popular techniques to identify free memory areas as a result of allocation and deallocations in a heap are :

1. Reference count : The system associates a reference count with each memory area to indicate the number of its active users.
2. Garbage collection : In this technique two passes are made over the memory to identify unused areas.

Two main approaches to reuse free memory area in a heap are :

First-fit : Allocate the first hole that is big enough.

Best-fit : Allocate the smallest hole that is big enough.

Q.59 Mention the significance of LDT and GDT in segmentation. AU : CSE/IT : May-15

Ans. : A descriptor table is simply a memory array of 8-byte entries that contain descriptors. The Local Descriptor Table (LDT) is a memory table used in protected mode and containing memory segment descriptors. The Global Descriptor Table is a data structure used in 80286.

Q.60 Define demand paging in memory management. what are the steps required to handle a page fault in demand paging. AU : CSE/IT : Dec.-15

Ans. : Refer Q.44 and section 4.9.1.

Q.61 In memory management consider the program named as Stack 1 which size is 100 KB. This program is loaded in the main memory from 2100 to 2200 KB. Show the contents of the page map table for the given scenario. AU : CSE/IT : Dec.-15

Ans. : Refer section 4.3.

Q.62 How does the system detect thrashing ? AU : CSE/IT : June-16

Ans. : The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming.

Q.63 Consider a memory system with a cache access time of 10ns and a memory access time of 110ns - assume the memory access time includes the time to check the cache. If the effective access time is 10% greater than the cache access time, what is the hit ratio H ? AU : CSE : May-17

Ans. : Effective Access Time = $H \times T_{cache} + (1\text{-}H) \times T_{memory}$

$1.1 \times T_{cache} = H \times T_{cache} + (1\text{-}H) \times T_{memory}$

$1.1 \times 10 = H \times 10 + (1 - H)\ 110$

$11 = H \times 10 + 110 - 110 \times H$

$-99 = -100\ H$

$H = 99/100$

$H = 0.99$

Q.64 What is a difference between a user-level instruction and a privileged instruction? Which of the following instructions should be privileged and only allowed to execute in kernel mode ?
(a) User level instruction
(b) privileged instruction
(c) privileged instruction

AU : CSE : May-17

Ans. : A privileged instruction is an instruction that can only be executed in kernel mode. User level instruction is executed in user area.

Q.65 Will optimal page replacement algorithm suffer from Belady's anomaly? Justify your answer.

AU : CSE : May-17

Ans. : Optimal replacement never suffers from Belady's anomaly. It was always believed that an increase in the number of page frames would always result in the same number or fewer page faults.

Q.66 Consider the following page-reference string : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12. How many page faults ratio would occur for the FIFO page replacement algorithm ? Assuming three is four frames.

AU : Dec.-17

Ans. : FIFO

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	5	5	5	5	9	9	9	9
2	-	2	2	2	2	6	6	6	6	10	10	10
3	-	-	3	3	3	3	7	7	7	7	11	11
4	-	-	-	4	4	4	4	8	8	8	8	12
Page Fault	PF	PF	PF	PF	PF	PF	PF	PF	PF	PF	PF	PF

Page Fault = 12 Page fault ratio = 100%

Q.67 What are the counting based page replacement algorithms ?

AU : May-18

Ans. : Counting-based page replacement algorithms are least frequently used (LFU) and most frequently used (MFU).

UNIT - IV

5 I/O Systems

Syllabus

Mass Storage system - Overview of Mass Storage Structure, Disk Structure, Disk Scheduling and Management, swap - space management; I/O Systems - I/O Hardware, Application I/O interface, Kernel I/O subsystem, Streams, Performance.

Contents

5.1 Mass Storage Structure

I/O Systems

AU : Dec.-16

- A storage device is any device used in a computer to store information. It will retain this information when the computer is switched off.
- Type of storage device used varies based on the type of data and the rate at which it is created and used.
- Examples of storage devices are hard disk, CD-ROM, magnetic tape and removable media etc.
- The computer has many types of data storage devices. Some of them can be classified as the removable data storage devices and the others as the non-removable data storage devices.

1. **Hard disk :** The most common form of internal storage device used in a computer is a hard disk. A hard disk has a circular, magnetized surface on which the data is stored. A hard disk spins at a speed of between 60 and 120 revolutions per second. The data stored on the hard disk is read or written by a head that floats just above the disk (less than 0.1 mm) on a cushion of air. The surface of a hard disk is divided up into sectors and tracks. Data is stored in the 'blocks' created by the sectors and tracks. Moving data into a 'block' is called random access.
2. **Compact disk :** Compact disks can be used to store much more data than a floppy disk. Most compact disks can hold 650 megabytes of data. There are three main types of compact disk : CD-ROM, CD-R, CD-RW.
3. **Magnetic tape :** This is a cheap method of storing large amounts of data (typically 26 gigabytes) and is often used as a 'backing store' for large and mainframe computers.
4. **Removable media :** Zip drives are similar to floppy drives and use special (and rather expensive) floppy disks that can hold between 100 megabytes and 2 gigabytes of data.

5.1.1 Characteristics of Storage Device

1. When power to the device is shut off, data stored on the medium remains.
2. Storage devices are cheaper than memory.
3. They are capable of storing more data.
4. Rotational speeds have increased over time.
5. Storage devices are used for backup.

5.1.2 Magnetic Disk

- To store computer data, hard disks, flash memory, magnetic tape and optical media is used.
- Alternate storage for hard disk is Solid State Disks (SSD). These flash memory based devices offer a different set of tradeoffs from a standard disk. At the same time, capacity of the hard disk also increases.
- Hybrid category is introduced for storage media. It is hard disks with large flash memory buffers. Disk sizes are specified in gigabytes.
- Performances of hard disk and SSD technology is given below :

Characteristics	Hard disk	SSD
Size	Terabytes	Gigabytes
Random access time	8 ms	0.25 ms
Sequential read	100 MB/s	250 MB/s
Random read	2 MB/s	250 MB/s
Cost	$0.10 /GB	$3 /GB
Reliability	Moderate	Unknown
Limited writes	No	Yes

- Hard disk contains several rotating platters coated with magnetic film. They are read and written by tiny skating heads that are mounted on a metal arm that swings back and forth to position them. Heads float close to the surface of the platters but do not actually touch.
- A hard disk is really a set of stacked "disks," each of which, like phonograph records, has data recorded electromagnetically in concentric circles or "tracks" on the disk. A "head" writes or reads the information on the tracks. Two heads, one on each side of a disk, read or write the data as the disk spins. Each read or write operation requires that data be located, which is an operation called a "seek."
- A hard disk/drive unit comes with a set rotation speed varying from 4500 to 7200 rpm. Disk access time is measured in milliseconds. Although the physical location can be identified with cylinder, track and sector locations, these are actually mapped to a Logical Block Address (LBA) that works with the larger address range on today's hard disks.

- Mean Time Between Failures (MTBF) is the predicted elapsed time between inherent failures of a system during operation. MTBF can be calculated as the arithmetic mean (average) time between failures of a system.
- Each disk drive is managed by a controller. Each type of controller can support a fixed number of drives. SCSI controllers support up to seven disks per controller or up to 15 disks per controller.
- Each disk is assigned a drive address. This address is set by a switch, a dial or jumpers on the disk or by the physical location of the disk. Some SCSI devices, such as RAID's, have an additional identifying number called a logical unit number. It is used to address disks within the device.
- Fig. 5.1.1 shows physical disk.

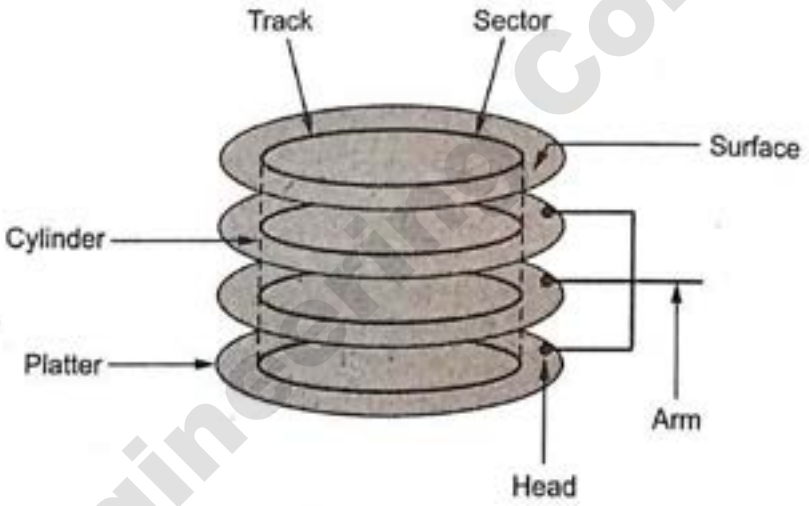


Fig. 5.1.1 Physical disk

- A disk consists of circular plates called **platters**. Each platter has an upper and lower oxide-coated surface. Recording heads, at least one per surface, are mounted on arms that can be moved to various radial distances from the center of the platters. The heads float very close to the surfaces of the platters, never actually touching them, and read and record data as the platters spin around.
- A ring on one surface is called a **track**. Each track is divided into disk blocks. Sometimes called sectors, these physical blocks on a disk are different from file system blocks.
- Formatting a disk divides the disk into tracks and disk blocks that can be addressed by the disk controller, writes timing marks, and identifies bad areas on the disk.
- SCSI disk drives are shipped preformatted. They do not require formatting at any time. Bad block handling is performed automatically by SCSI disks. Bad blocks are areas of a disk that cannot reliably store data.

Platter

- Platter is a circular, metal disk that is mounted inside a hard disk drive. Several platters are mounted on a fixed spindle motor to create more data storage surfaces in a smaller area.
- The platter has a core made up of aluminum or glass substrate, covered with a thin layer of Ferric oxide or cobalt alloy. On both sides of the substrate material, a thin coating is deposited by a special manufacturing technique. This, thin coating where actual data is stored is the media layer.
- Attributes of platter
  1. It is a rigid, round disk this is coated with magnetically sensitive material.
  2. Data is stored in binary code.
  3. It is encoded by polarizing magnetic areas on the disk surface.
  4. Data can be written to and read from both surfaces of a platter.
  5. A platter's storage capacity varies across drives.

Tracks

- Each platter is broken into thousands of tightly packed concentric circles, known as tracks. These tracks resemble the structure of annual rings of a tree.
- All the information stored on the hard disk is recorded in tracks. Starting from zero at the outer side of the platter, the number of tracks goes on increasing to the inner side.
- Each track can hold a large amount of data counting to thousands of bytes.

Sectors

- Each track is further broken down into smaller units called sectors. As sector is the basic unit of data storage on a hard disk. Disk contains concentric **tracks**. Tracks are divided into **sectors**. A sector is the smallest addressable unit in a disk.
- Fig. 5.1.2 shows surface of disk showing tracks and sectors.

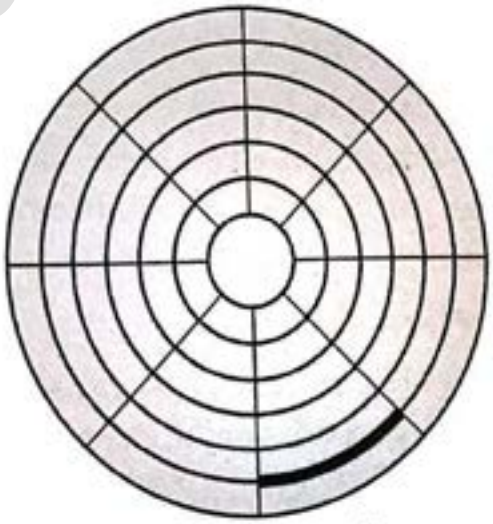


Fig. 5.1.2 Disk surface

- A single track typically can have thousands of sectors and each sector can hold more than 512 bytes of data. A few additional bytes are required for control structures and error detection and correction.

Clusters : Sectors are often grouped together to form clusters.

Read / Write Heads

- The heads are an interface between the magnetic media where the data is stored and electronic components in the hard disk. The heads convert the information, which is in the form of bits to magnetic pulses when it is to be stored on the platter and reverses the process while reading.
- The heads are the most sophisticated part of the hard disk. Each platter has two read/write heads, one mounted on the top and the other one at the bottom. These heads are mounted on head sliders, which are suspended at the ends of head arms.
- The head arms are all fused into a singular structure called actuator, which is responsible for their movement. Drives rotate at 60 to 200 times per second.
- **Transfer rate** is rate at which data flow between drive and computer.
- **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head.
- **Head crash** results from disk head making contact with the disk surface.
- "Each platter (disc-shaped) is coated with magnetic material on both surfaces. All platter surfaces has *arm* extended from fixed position. Tip of the arm contains read/write *head* for reading or writing data.
- The arm moves the heads from the spindle edge to the edge of the disc.
- When a program reads a byte from the disk, the operating system locates the surface, track and sector containing that byte, and reads the entire sector into a special area in main memory called **buffer**.
- The bottleneck of a disk access is moving the read/write arm.
- A **cylinder** is the set of tracks at a given radius of a disk pack. A cylinder is the set of tracks that can be accessed without moving the disk arm. All the information on a cylinder can be accessed without moving the read/write arm.
- Fig. 5.1.3 shows moving-head disk mechanism.
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a cylinder. Only one head reads/writes at any one time. Block size is a multiple of sector size.

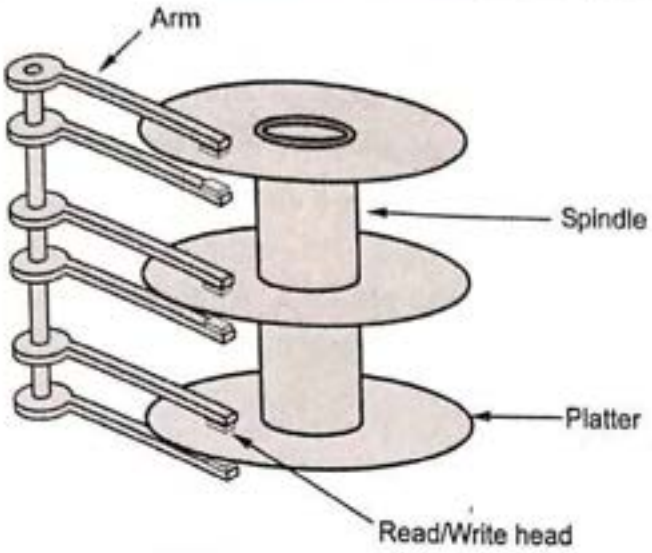


Fig. 5.1.3 Moving-head disk mechanism

- Disks can be removable. Drive attached to computer via I/O bus. Buses vary, including EIDE, ATA, SATA, USB, Fibre channel, SCSI etc.
- Host controller in computer uses bus to talk to disk controller built into drive or storage array.
- Disk controllers typically embedded in the disk drive, which acts as an interface between the CPU and the disk hardware. The controller has an internal cache that it uses to buffer data for read/write requests.

Zone Bit Recording

- Also known as multiple zone recording or zone-CAV recording. Disk divided into zones.

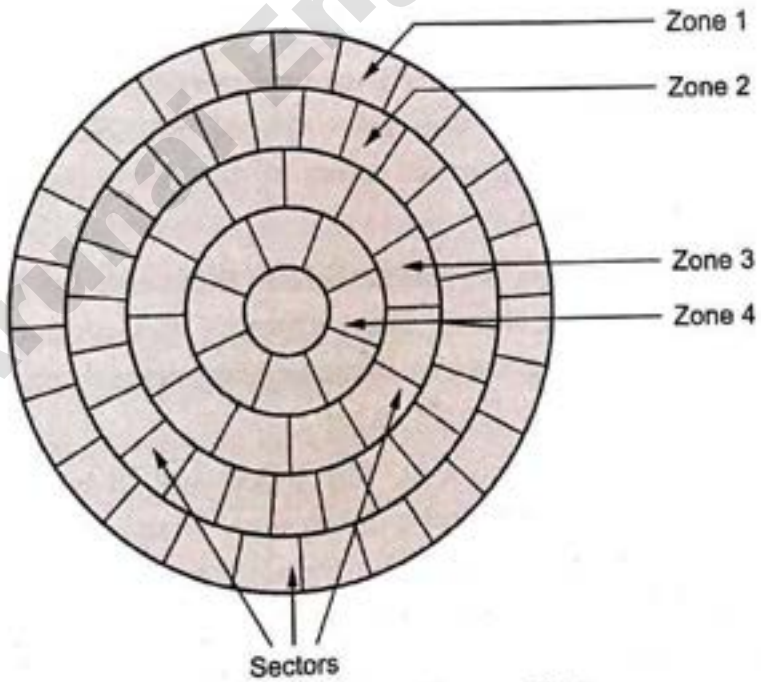


Fig. 5.1.4 Zone bit recording

- Cylinders in different zones have a different number of sectors. Number of sectors in a particular zone is constant. Data is buffered so the data rate to the I/O interface is constant. Fig. 5.1.4 shows zone bit recording.
- Zoned-bit recording uses the disk more efficiently. It groups tracks into zones that are based upon their distance from the center of the disk. Each zone is assigned an appropriate number of sectors per track. This means that a zone near the center of the platter has fewer sectors per track than a zone on the outer edge.

5.1.3 Solid State Disks

- SSD is a storage device that is based on semiconductors rather than rotating magnetic platters. Most SSDs are based on NAND Flash chips because they are fast, highly reliable, widely available and are non-volatile, meaning they save data even without a power source.
- SSDs use the same Serial ATA (SATA) or IDE interface as hard drives, making them functionally identical.
- Solid-state storage technology typically provides faster system performance than traditional magnetic media drives. In addition, there are no moving parts in SSDs and therefore the risk of mechanical failure is near zero. Solid-state drives also provide improved overall system responsiveness while consuming much less power than a traditional hard disk drive.
- A flash-based SSD typically uses a small amount of DRAM as a cache, similar to the cache in hard disk drives. A directory of block placement and wear leveling data is also kept in the cache while the drive is operating.
- Each page of flash memory in an SSD can be rewritten only a limited number of times. To limit the wear on any given page, the SSD firmware maintains a mapping table and distributes writes across all the drive' pages. This remapping is invisible to operating system.
- Flash memory pages must be erased before they can be rewritten. Erasing is separate operation that is slower than writing. Rebuilding a buffer of erased pages is harder than it might seem because filesystems typically do not mark or erase data blocks they are no longer using.
- Standard size of the disk block is 512 bytes, but that size is too small for filesystems to deal with efficiently. Filesystem manages the disk in terms of clusters of 1 KiB to 8 KiB in size. The translation layer maps filesystem clusters into ranges of disk blocks for reads and writes.
- SSD only can read or write data in 4 KiB pages, file system cluster boundaries and SSD page boundaries should coincide.

- **Solid state disks** (SSDs) mirror the functionality of the existing standard of hard disk drives. SSD is volatile or non-volatile depending upon memory technology.
- Because SSDs do not have moving parts and therefore performance is insensitive to issues such as seek time and rotational latency. Therefore, a simple FCFS policy will suffice.
- Solid state disks commonly use the FCFS disk scheduling policy.
- SSDs have the advantage of being faster than magnetic disks as there are no moving parts and therefore do not have seek time or rotational latency.
- In an SSD the LBAs are actually inside of the flash media. SSDs contain a number of NAND flash components.
- **Solid-State Disks(SSDs) features :**
  1. Low power consumption.
  2. Faster random access.
  3. Greater shock resistance.
- SSD performance is being increased due to the exploitation of parallel I/O architectures. An SSD interacts with the host computer via standard interface such as PATA or SATA and behaves much like a standard hard drive.
- Operating systems use storage devices to provide file systems and virtual memory. SSD controller is used to translate read/write requests into flash memory operations.
- The controller exploits RAM to temporarily buffer write requests or accessed data during handling read/write requests.
- To increase the read/write bandwidth of SSD, many SSDs use an interleaving technique that exploits the parallelism of accessing multiple NAND chips simultaneously.
- If there are multiple independent channels, the read/write bandwidth of SSDs can be accelerated further by exploiting inter-channel and intra-channel parallelism.
- SSD challenges :
  1. Reliability for large scale flash storage.
  2. The balance between cost, performance and lifetime.
  3. Cost per bit of NAND flash memory is still high.

University Question

1. *Describe some advantages and disadvantages of using SSDs as a caching tier and as a disk-drive replacement compared with using only magnetic disks ?* **AU : Dec.-16, Marks 8**

5.2 Disk Structure

- Magnetic disk drives are addresed as large 1- dimensional arrays of logical bolcks, where the logical block is the smallest unit of transfer. The smallest addressable unit on a disk storage is a block. Logical block size is normally 512 bytes. It also possible that, block size may change with formatting.
- Modern magnetic disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer.
- 1-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder.
- In theory, this mapping support for conversion of logical block number into an old-style disk address. That type of address consists of a cylinder number, track number within that cylinder, and a sector number within that track.
- Rotational speed is measured by two ways: CAV and CLV.
- **Constant angular velocity(CAV)** : The rotational speed of the disk is constant. To use the platter in an efficient way, the outer tracks have more sectors than the inner tracks. Used in hard disks.
- **Constant Linear velocity (CLV)** : The density of bits per track is uniform. To get constant data rate the rotation speed is increased as the head moves from the outer to the inner tracks. Used in CD and DVD.
- CD and DVD differ from hard disks in that they use a single track that spirals out from the centre to the periphery.

5.3 Disk Performance Parameters

- The actual details of disk I/O operation depend on the :
  1. Computer system
  2. Operating system
  3. Nature of the I/O channel and disk controller hardware.
- Currently, disks are at least four orders of magnitude slower than main memory. Fig. 5.3.1 shows a general timing diagram of disk I/O transfer.
  (See Fig. 5.3.1 on next page).
- The disk is rotating at constant speed when it perform read or write operation. To read or write the head must be positioned at the desired track and at the beginning of the desired sector on that track. Head is moving from one track to other track for selecting proper track.
- Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system.

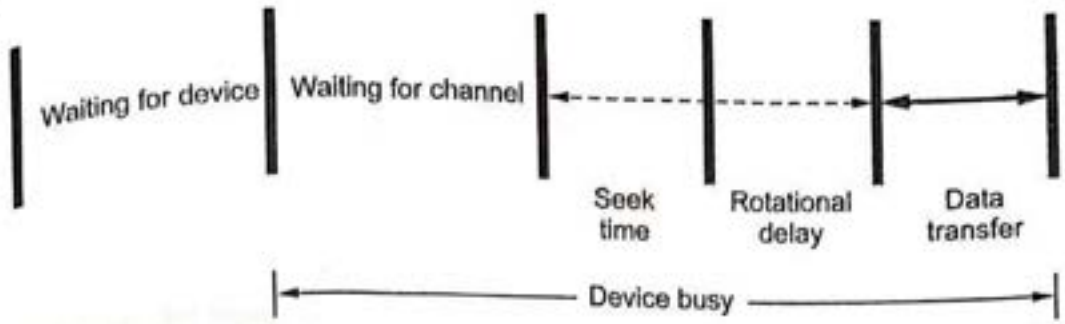


Fig. 5.3.1 : Disk I/O transfer timing

- Following parameters are used for disk performance :
  1. Seek time 2. Rotational delay 3. Access time

1. **Seek time :** The time it takes to position the head at the track is known as **seek time.**

   Seek time = Number of track traversed × Disk drive constant + Startup time

   Seek time do not apply to device with fixed read/write heads.

2. **Rotational delay :** The time it takes for the beginning of the sector to reach the head is known as rotational delay. It is also known as search time.

3. **Access time :** The sum of the seek time and the rotational delay equals the **access time.**

   Access time = Seek time + Rotational latency

- **Transfer time :**

$$\text{Transfer time} = \frac{\text{Number of bytes to be transferred}}{\text{Number of bytes on the track} \times \text{Rotation speed}}$$

- **Disk capacity :**

Disk capacity = Number of cylinders × Number of heads × Number of sectors per track × Number of bytes per track

Example 5.3.1 *What is the maximum size of disk which contains following parameters ? Cylinder = 1024, heads = 16 and sectors per track is 63.*

Solution : Total size of disk = Number of cylinders × Number of heads × Number of sectors/track × Number of bytes/sector.

= 1024×16×63×512

= 528 48

Example 5.3.2 *A disk has 19456 cylinders, 16 heads and 63 sectors/tracks. The disk spins at 5400 r.p.m. Seek time between adjacent tracks is 2 ms. Assuming the read/write head is already positioned at track 0. How long does it take to read the entire disk ?*

Solution : 1) Each track can be read in one revolution.

2) 11.11 ms is required to read one track.

3) For reading all 19456×16 tracks, approximately 3459 seconds is required.

4) Seek time = 19456 – 1×2 = 39 sec.

5) Total time is 3498 sec.

5.4 Disk Scheduling

AU : May-15, 17, 18, Dec.-17

- Disk scheduling algorithms are used to reduce the total seek time of any request. I/O request issues a system call to the operating system. If desired disk drive or controller is idle then the request is served immediately. If device is bust then the request for service will be placed in the queue of pending requests.
- When one request is completed, the operating system has to choose which pending request to service next.
- The processor is much faster than the disk, so it's highly likely that there will be multiple outstanding disk requests before the disk is ready to handle them. Because disks have non-uniform access times, re-ordering disk requests can greatly reduce the latency of disk requests.
- Multiple requests to disk will arrive while one is being serviced. Disk scheduling increases the disk's bandwidth.
- To service a request, a disk system requires that the head be moved to the desired track, then a wait for latency and finally the transfer of data. In the following section we will examine several scheduling algorithms.
  1. FIFO 2. SSTF 3. SCAN 4. C-SCAN
  5. LOOK 6. C-LOOK

5.4.1 First In First Out

- FIFO is also called first come first served method. Requests are processed in queue order. Fig. 5.4.1 shows FCFS algorithm. (Fig. 5.4.1 see on next page).
- FCFS has a fair policy in the sense that once a request has arrived, its place in the schedule is fixed irrespective of arrival of a higher priority request.
- The requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38 and 184. Starting track at 100 and total number of track is 200. FCFS process the entire request in sequential order.
- FCFS would begin at track 100, move 45 tracks to 55, and move 3 tracks to 58 and so on. The Y-axis corresponds to the tracks on the disk. The X-axis corresponds to time or the number of tracks traversed.
- Starting at track 100.

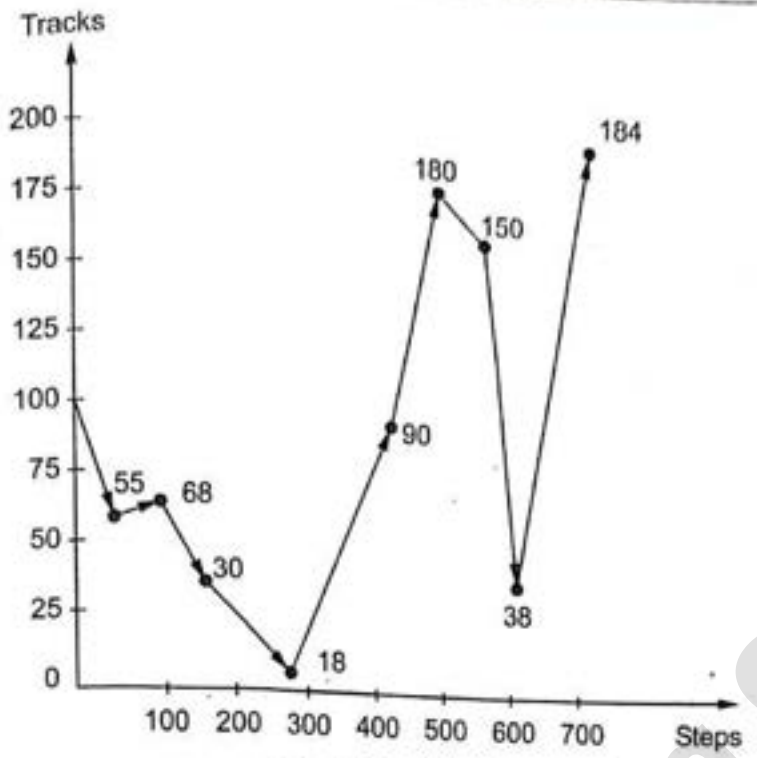


Fig. 5.4.1 : FCFS

Next accessed track is as below.

Next track accessed	Number of track traversed
55	45
58	03
39	19
18	21
90	72
160	70
150	10
38	112
184	146

Average seek length $= \frac{45+3+19+21+72+70+10+112+146}{9} = \frac{498}{9} = 55.333$

- FCFS is simple to implement. But it does not provide fast services. It can not take special action to minimize the overall seek time.

Advantages :

1. Simple and easy to implement.
2. Suitable for light loads.

Disadvantages :

1. FCFS does not provide fast services.
2. Do not maximize throughput.
3. May involve lots of unnecessary seek distance

5.4.2 Shortest Seek Time First

- SSTF select the next request at the one requiring the minimum seek time from the current position. Fig. 5.4.2 shows SSTF algorithm.
- In Shortest-Seek-Time-First (SSTF) scheduling priority is given to those processes which need the shortest seek, even if these requests are not the first ones in the queue. It means that all requests nearer to the current head positions are serviced together before moving head to distant tracks.
- Select the disk I/O request that requires the least movement of the disk arm from its current position and always choose the minimum seek time.

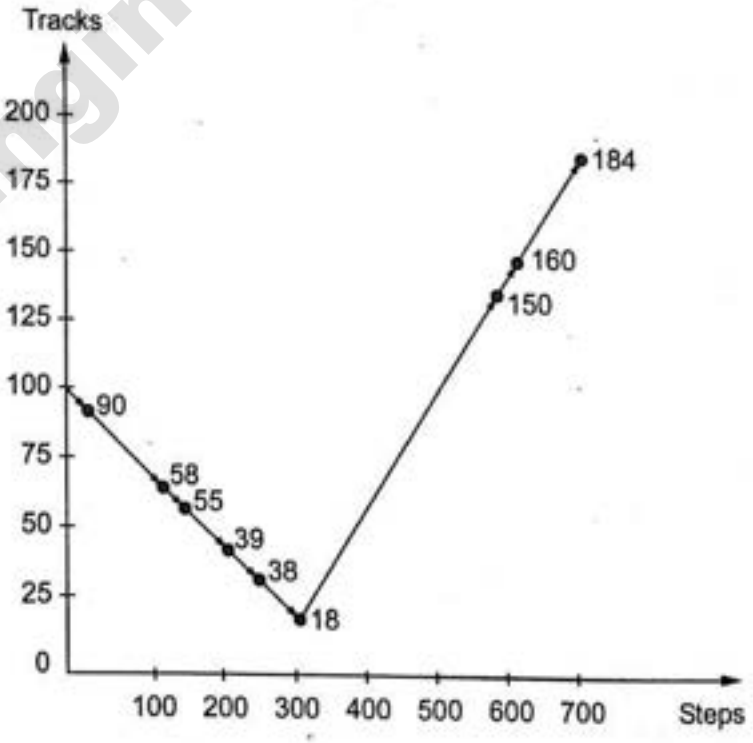


Fig. 5.4.2 : SSTF

Next track accessed	Number of track traversed
90	10
58	32
55	03
39	16
38	01
18	20
150	132
160	10
184	24

Average seek length $= \frac{10+32+3+16+1+20+132+10+24}{9} = \frac{248}{9} = 27.55$

- The only tricky part is if there are two jobs with the same distance. In this case, some kind of tie-breaking needs to be employed to pick one. For instance, you could just use a random number to effectively flip a coin and pick one.
- SSTF does not ensure fairness and can lead to indefinite postponement because its seek pattern tends to be highly localized.
- Under heavy load, SSTF can prevent distant requests from ever being serviced. This phenomenon is known as starvation. SSTF scheduling is essentially a form of shortest job first scheduling. SSTF scheduling algorithm are not very popular because of two reasons.
  1. Starvation possibly exists. 2. It increases higher overheads.

Advantages :

1. Throughput is better than FCFS.
2. SSTF minimize the response time.
3. Less number of head movements.

Disadvantages :

1. One major issue is STARVATION.
2. Localize under heavy load.

5.4.3 SCAN

- SCAN is also called elevator algorithm.
- The next request scheduled is closest to current request but in one particular direction. All requests in other direction are put at the end of the list. SCAN services tracks in only one direction i.e. either increasing or decreasing track number.
- When SCAN reaches the edge of the disk (or track 0), it reverses direction.

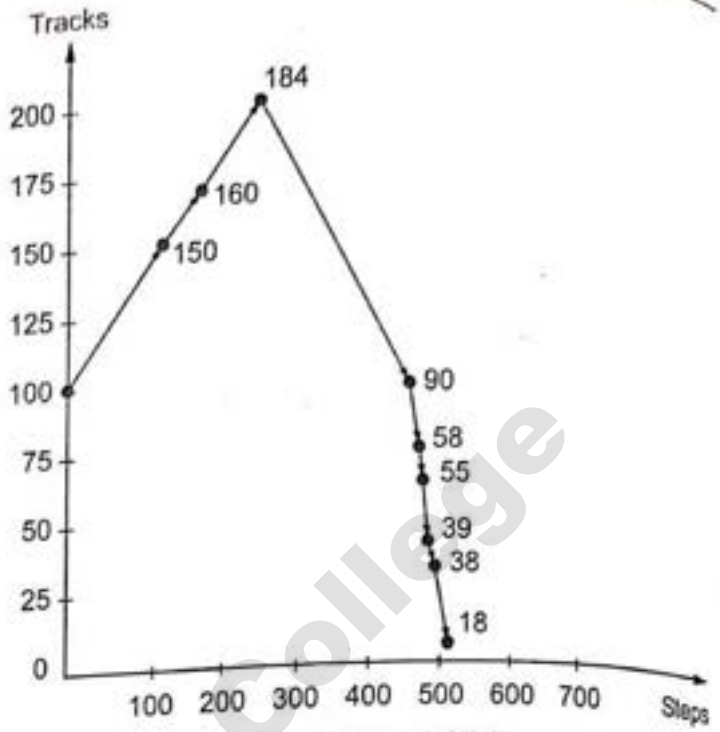


Fig. 5.4.3 : SCAN

- If any request comes on its way it will be serviced immediately, while request arriving just behind the head will have to wait until disk head moves to the end of the disk, reverses direction and returns before being serviced.
- SCAN is called elevator because an elevator continues in one direction servicing requests before reversing direction.
- Consider the example in which the requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38, and 184. Starting track at 100 and total number of track is 200. Head is moving towards increasing track number.

Current head position = 100

Head is moving towards increasing number of track.

Next track accessed	Number of track traversed
150	50
160	10
184	24
90	94
58	32
55	03
39	16
38	01
18	20

$$\text{Average seek length} = \frac{50+10+24+94+32+3+16+01+20}{9} = \frac{250}{9} = 27.77$$

Advantages :

1. SCAN eliminates starvation.
2. Throughput similar to SSTF

Disadvantages :

1. Increase overhead
2. Needs directional bit

5.4.4 Circular SCAN

- C-SCAN also moves the head in one direction, but it offers fairer service with more uniform waiting times.
- It moves the head from one end of the disk to the other end, servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- C-SCAN treats the cylinders as a circular list that wraps around from the last cylinder to the first one. It scans in cycles, always increasing or decreasing order.
- Fig. 5.4.4 shows C-SCAN. Consider the example in which the requested tracks in the order received are : 55, 58, 39, 18, 90, 160, 150, 38 and 184. Starting track at 100 and total number of track is 200. Head is moving towards increasing track number.

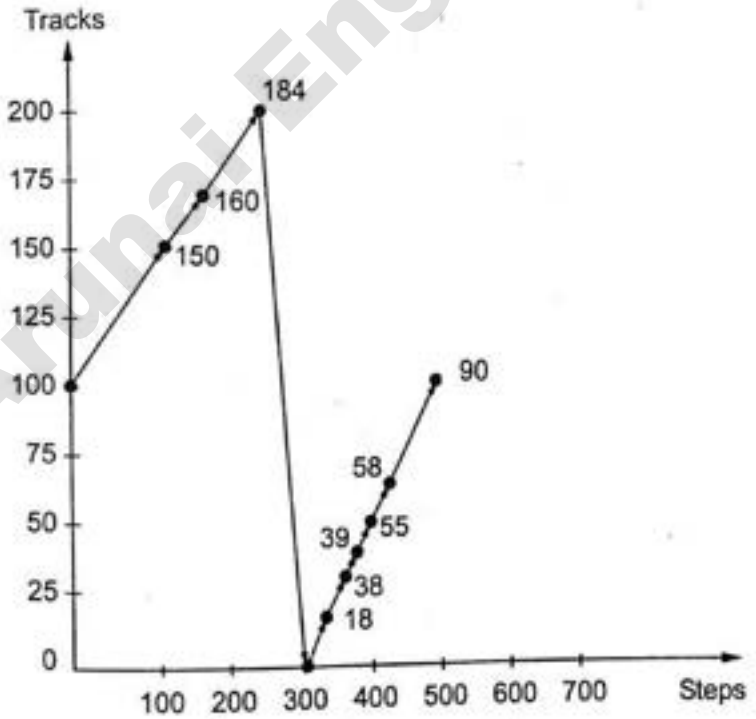


Fig. 5.4.4 : C-SCAN

Next track accessed	Number of track traversed
150	50
160	10
184	24
0	184
18	18
38	20
39	1
55	16
58	3
90	32

Average seek length $= \frac{50+10+24+184+18+20+1+16+3+32}{10} = \frac{358}{10} = 35.8$

5.4.5 LOOK Scheduling

- Start the head moving in one direction. Satisfy the request for the closest track in that direction when there is no more request in the direction, the head is traveling reverse direction and repeat.
- This algorithm is similar to SCAN, but unlike SCAN, the head does not unnecessarily travel to the innermost and outermost track on each circuit.
- Fig. 5.4.5 shows the look scheduling algorithm.
- For example, the disk request queue contains a set of references for blocks on tracks 76, 124, 17, 269, 201, 29, 137 and 12.
- Current head position = 76

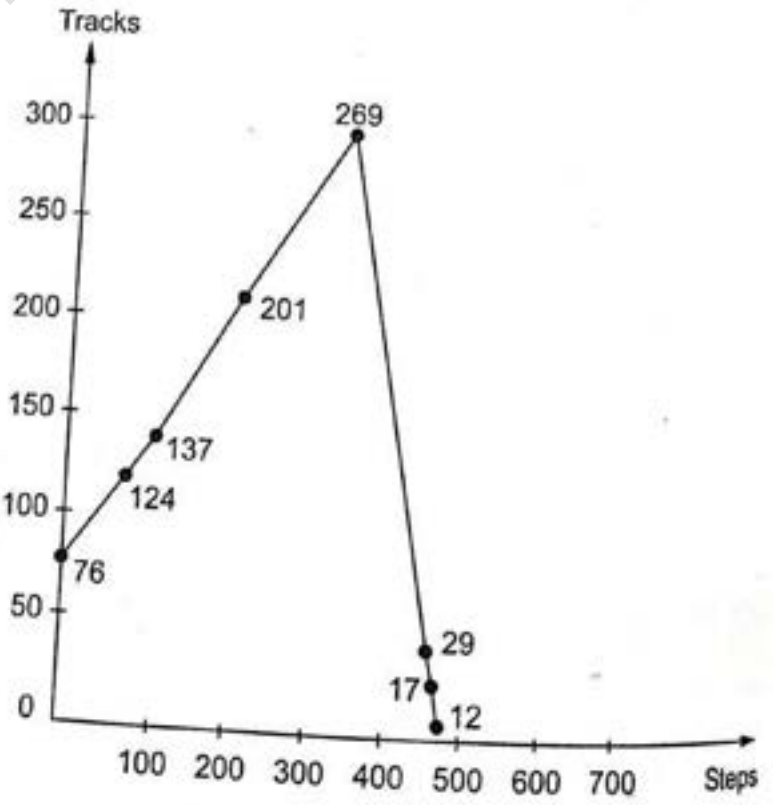


Fig. 5.4.5 : LOOK scheduling

Next track accessed	Number of track traversed
124	48
137	13
201	64
269	68
29	240
17	12
12	5

Average seek length $= \frac{48+13+64+68+240+12+5}{10} = \frac{450}{7} = 64.28$

5.4.6 C-LOOK

- C-LOOK is same as C-SCAN except the head stops after servicing the last request in the preferred direction, then services the request to the cylinder nearest the opposite side of the disk.
- **Example :** Consider the example in which the requested tracks in the order received are : 23, 89, 132, 42, 187. Starting track at 100 and there are 200 cylinders numbered from 0 - 199.

Next track accessed	Number of track traversed
89	11
42	47
23	19
187	164
132	55

Average seek length $= \frac{11+47+19+164+55}{5} = \frac{296}{5} = 59.2$

Solved Examples

Example 5.4.1 *The requested tracks, in the order received are : 55, 58, 39, 18, 90, 160, 150, 38, 184. AppIBA the following disk scheduling algorithms. Starting track at 100.*
1) FCFS 2) SSTF 3) SCAN 4) C-SCAN.

Solution :

Disk request	FCFS		SSTF		SCAN		C-SCAN	
	Next track	No. of track traversed	Next track	No. of track traversed	Next track	No. of track traversed	Next track	No. of track traversed
55	55	45	90	10	150	50	150	50
58	58	03	58	32	160	10	160	10
39	39	19	55	03	184	24	184	24
18	18	21	39	16	90	94	18	166
90	90	72	38	01	58	32	38	20
160	160	70	18	31	55	03	39	01
150	150	10	150	132	39	16	55	16
38	38	112	160	10	38	01	58	03
184	184	146	184	24	18	20	90	32
Total track traversed		498		259		250		322

1) FCFS

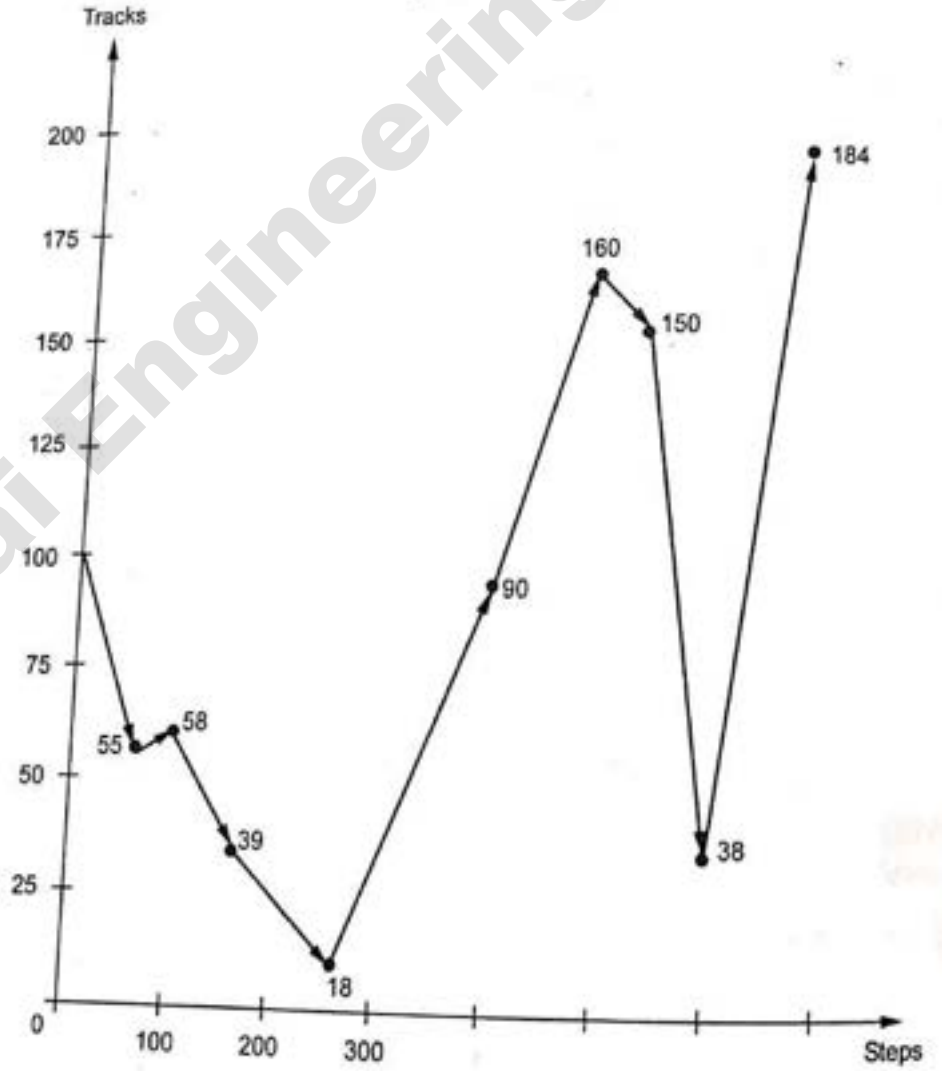


Fig. 5.4.6

2) SSTF

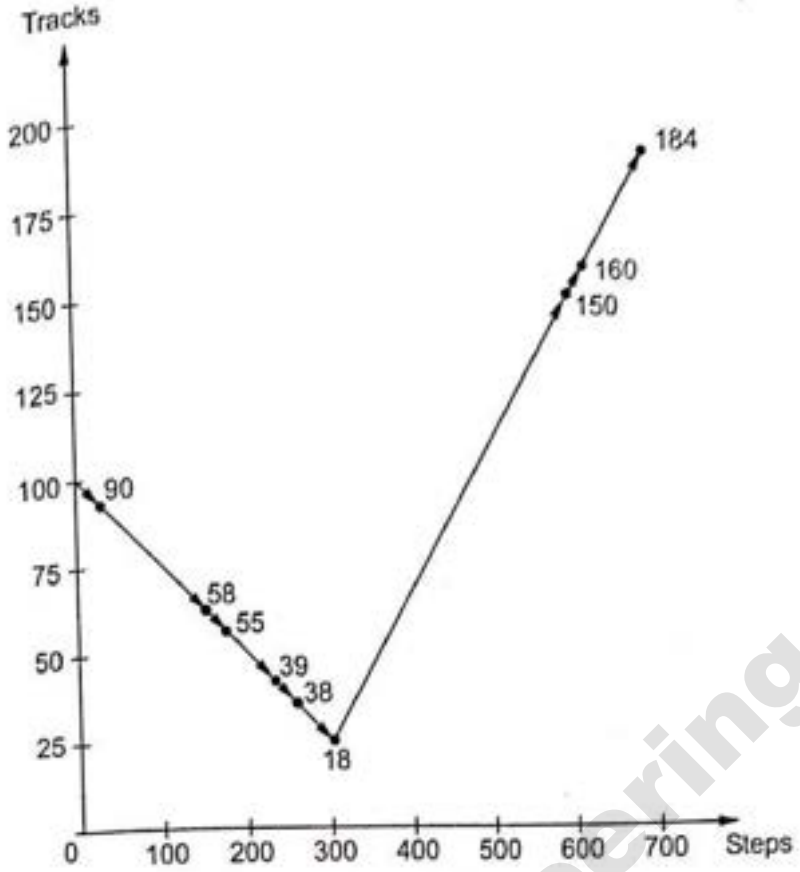


Fig. 5.4.7

3) SCAN

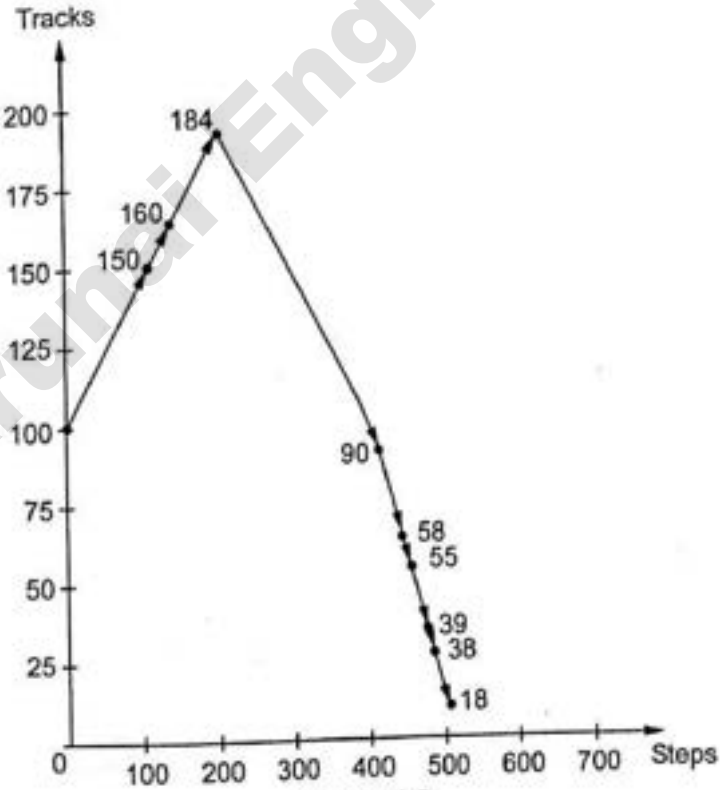


Fig. 5.4.8

4) C-SCAN

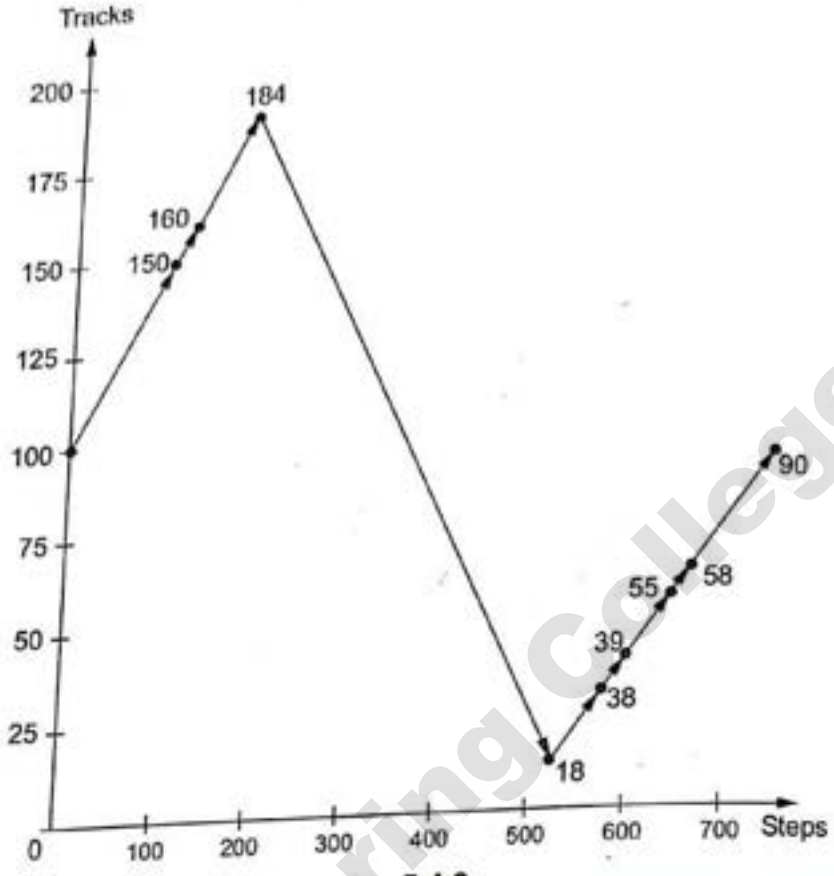


Fig. 5.4.9

Example 5.4.2 *On a disk with 1000 cylinders, numbers 0 to 999, compute the number of tracks the disk arm must move to satisfy all the request in the disk queue. Assume the last request received was at track 345 and the head is moving towards track 0. The queue in FIFO order contains requests for the following tracks. 123, 874, 692, 475, 105, 367. Perform the computation for the following scheduling algorithm.*

1) FIFO 2) SSTF 3) SCAN 4) LOOK 5) C-SCAN 6) C-LOOK.

Solution : 1) FIFO (First In First Out)

Next track accessed	Number of tracks traversed
123	222
874	751
692	182
475	217
105	370
376	271

Seek length = 2013

2) SSTF

Next track accessed	Number of tracks traversed
376	31
475	99
692	217
874	182
123	751
105	18

Seek length (Head movement) = 1298

3) SCAN

Next track accessed	Number of tracks traversed
123	222
105	18
0	105
376	376
475	99
692	217
874	182

Seek length = 1219

4) LOOK

Next track accessed	Number of tracks traversed
123	222
105	18
376	271
475	99
692	217
874	182

Seek length = 1009

5) C-SCAN

Next track accessed	Number of tracks traversed
123	222
105	18
0	105
999	999
874	125
692	182
475	217
376	99

Seek length = 1967

6) C-LOOK

Next track accessed	Number of tracks traversed
123	222
105	18
874	769
692	182
475	217
376	99

Seek length = 1507

Example 5.4.3 *Disk requests come in to the driver for cylinders 10, 22, 20, 2, 40, 6 and 38. A seek takes 6 msec per cylinder moved. How much seek time needed for FCFS, closet cylinder next algorithm? Initially arm is at cylinder 20.*

Solution : Initially arm is at cylinder 20.

First-Come First Serve		Closet cylinder next algorithm	
Next request accessed	Number of head movement	Next request accessed	Number of head movement
10	20 – 10 = 10	20	20 – 20 = 0
22	22 – 10 = 12	22	22 – 20 = 2
20	22 – 20 = 2	10	22 – 10 = 12
2	20 – 2 = 18	6	10 – 6 = 4
40	40 – 2 = 38	2	6 – 2 = 4
6	40 – 6 = 36	38	38 – 2 = 36
38	38 – 6 = 32	40	40 – 38 = 2
Total number of head movement	148		60

Total seek time for FCFS = 148 x 6 = **888 msec**

Total seek time for Closet cylinder next = 60 x 6 = **240 msec**

Example 5.4.4 *Assume that the disk head is initially positioned over track 100. For the disk space request of 27, 129, 110, 186, 147, 41, 10, 64 and 120 show how disk scheduling done for i) SSTF ii) C-SCAN iii) C-LOOK. Calculate the average seek length a show the tracking of the requests.*

Solution :

Disk request	SSTF		C-LOOK		C-LOOK	
	Next track	No. of track traversed	Next track	No. of track traversed	Next track	No. of track traversed
27	110	10	110	10	110	10
129	120	10	120	10	120	10
110	129	09	129	09	129	09
186	147	18	147	18	147	18
147	186	39	186	39	186	39
41	64	122	10	176	0	186
10	41	23	27	17	10	10

64	27	14	41	14	27	17
120	10	17	64	23	41	14
	Total	262		316	64	23
					Total	336

Average seek length

$$\text{SSTF} = \frac{262}{9} = 29.11$$

$$\text{C-LOOK} = \frac{316}{9} = 35.11$$

$$\text{C-SCAN} = \frac{336}{9} = 37.33$$

Example 5.4.5 *Consider a disk queue with requests for I/O to blocks on cylinders. 98, 183, 37, 122, 14, 124, 65, 67.*

If the disk head is start at 53, then find out the total head movement with respect to FCFS, SSTF, SCAN, C-SCAN and LOOK scheduling. **AU : Dec.-17, Marks 13**

Solution : Disk head start at 53

FCFS		SSTF		SCAN		C-SCAN		LOOK	
Next Request	Head Movement	Next Request	Head Movement	Next Request	Head Movement	Next Request	Head Movement	Next Request	Head Movement
98	45	65	12	37	16	65	13	37	16
188	90	67	2	14	23	67	2	14	23
37	151	37	30	0	14	98	31	65	51
122	85	14	23	65	65	122	24	67	2
14	107	98	84	67	2	124	2	98	31
124	110	122	24	98	31	188	64	122	24
65	59	124	2	122	24	0	188	124	2
67	2	188	64	124	2	14	14	188	64
				188	64	37	37		
Total head movement	649		241		241		375		213

University Questions

1. *Compare the functionalities of FCFS, SSTF, CSCAN and C-LOOK disk scheduling algorithms with an example for each.* **AU : CSE/IT : May-15, Marks 12**
2. *On a disk with 1000 cylinders, numbers 0 to 999, compute the number of tracks the disk arm must move to satisfy the entire request in the disk queue. Assume the last received was at track 345 and the head is moving towards track 0. The queue in FIFO order contains requests for the following tracks. 123, 874, 692, 475, 105 and 376. Find the seek length for the following scheduling algorithm. 1) SSTF 2) LOOK 3) CSCAN.* **AU : May-17, Marks 6**
3. *State and explainthe FCFS, SSTF and SCAN disk scheduling with examples.* **AU : May-18, Marks 13**

5.5 Disk Management

Operating system is responsible for disk management. Following are some activities discussed :

5.5.1 Disk Formatting

Disk formatting is of two types.

a) Physical formatting or low level formatting. b) Logical formatting.

Physical formatting :

- Disk must be formatted before storing a data.
- Disk must be divided into sectors that the disk controller can read/write.
- Low level formatting fills the disk with a special data structure for each sector.
- Data structure consists of three fields : header, data area and trailer.
- Header and trailer contain information used by the disk controller.
- Sector number and Error Correcting Codes (ECC) contained in the header and trailer.
- For writing data to the sector - ECC is updated.
- For reading data from the sector - ECC is recalculated.
- If there is mismatch in stored and calculated value then data area is corrupted.
- Low level formatting is done at factory.

Logical formatting :

- After disk is partitioned, logical formatting used.
- Operating system stores the initial file system data structures onto the disk.

5.5.2 Boot Block

- When a computer system is powered up or rebooted, a program in read only memory executes.
- Diagnostic check is done first.
- Stage 0 boot program is executed.
- Boot program reads the first sector from the boot device into main memory.
- Boot sector is the first sector of the boot device and contains stage-1 boot program.
- May be boot sector will not contain a boot program.
- PC booting from hard disk, the boot sector also contains a partition table.
- The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.
- Full boot strap program is more sophisticated than the bootstrap loader in the boot ROM.

5.5.3 Bad Blocks

- Data that resided on the bad blocks usually are lost. The controller maintains a list of bad blocks on the disk. The list is initialized during the low level format at the factory, and is updated over the life of the disk.
- Low level formatting also sets aside spare sectors not visible to the operating system. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as **sector sparing** or **forwarding.**
- A typical bad sector transaction might be as follows :
  1. The OS tries to read the data from logical block number 120.
  2. Device controller calculates the error correction code and it finds that required block data can not be read, so that sector is bad. It reports this information to OS.

5.6 Swap-Space Management

AU : Dec.-13

- Swap space management is low level task of OS.

5.6.1 Swap Space Use

- Swap space use is depends upon operating system. This is related to memory management of OS.
- Swap space cannot completely make up for a lack of physical RAM. Disk access is much slower than RAM access, by several orders of magnitude. Therefore, swap is

useful primarily as a means to run a number of programs simultaneously that would not otherwise fit into physical RAM.

- Virtual memory and swap space allows a large process to run even if the process is only partially resident. As "old" pages may be swapped out, the amount of memory addressed may easily exceed RAM as demand paging will ensure the pages are reloaded if necessary.
- Linux supports swap space in two forms : as a separate disk partition or a file somewhere on your existing Linux file systems. You can have up to 16 swap areas, with each swap area being a disk file or partition up to 128 MB in size.

5.6.2 Swap Space Location

- Location of the swap space is :
  1. Normal file system 2. Separate disk partition
- If it is a large file within the file system, normal file system operations are carry out on files. This method is simple and easy to implement.
- Separate disk partition: operating system can not keep any files or directory in this space. System creates separate swap space storage manager for allocating and de-allocating the block from this space.
- Manager always gives the preference for storage efficiency. It suffers from internal fragmentation.

University Question

1. *Explain how swap space is used located on the disk and maneged.* **AU : Dec.-13, Marks 8**

5.7 Input-Output Device

AU : May-16

- I/O devices are divided into three types : Human readable, Machine readable and Communication.

1. Human readable

- Computer user can communicate with computer using human readable device. These devices are help to user for giving input to computer and reading result.
- Example : Printer, display, keyboard and mouse.

2. Machine readable

- Machine readable is used for communicating with electronic devices and components.
- Example : USB device, disk drivers, controller, sensors etc.

3. Communication

- This type I/O devices used for communicating with remote devices.
- Example : modem, digital line drivers
- I/O devices are differentiating on following parameters.

1. Data rate
2. Application used
3. Complexity of control
4. Error conditions
5. Data representation
6. Unit of transfer

1. **Data rate** will change according to the input output devices. There may be differences of several orders of magnitude between the data transfer rates. Data transfer rate of keyboard is the lowest among the entire I/O device.
2. **Application used :** Different devices have different use in the system.
3. **Complexity of control :** It will change according to the input-output devices. Printer requires a relatively simple control interface. Disk is much more complex interface.
4. **Error conditions :** The nature of errors differ widely from one device to another.
5. **Data representation :** Different data encoding schemes are used for different devices.
6. **Unit of transfer :** Data may be transferred as a stream of bytes or characters or in larger blocks.

Example 5.7.1 *Why is it important to balance file-system I/O among the disks and controllers on a system in a multitasking environment?* **AU : May-16, Marks 8**

Solution : A system can perform only at the speed of its slowest bottle-neck. Disks or disk controllers are frequently the bottleneck in modern systems as their individual performance cannot keep up with that of the CPU and system bus. By balancing I/O among disks and controllers, neither an individual disk nor a controller is overwhelmed, so that bottleneck is avoided.

5.8 I/O Hardware

- Device management is the part of the operating system responsible for directly manipulating the hardware devices. Device management is implemented through the interaction of a device driver and interrupt routine. Fig. 5.8.1 shows the device management organization.

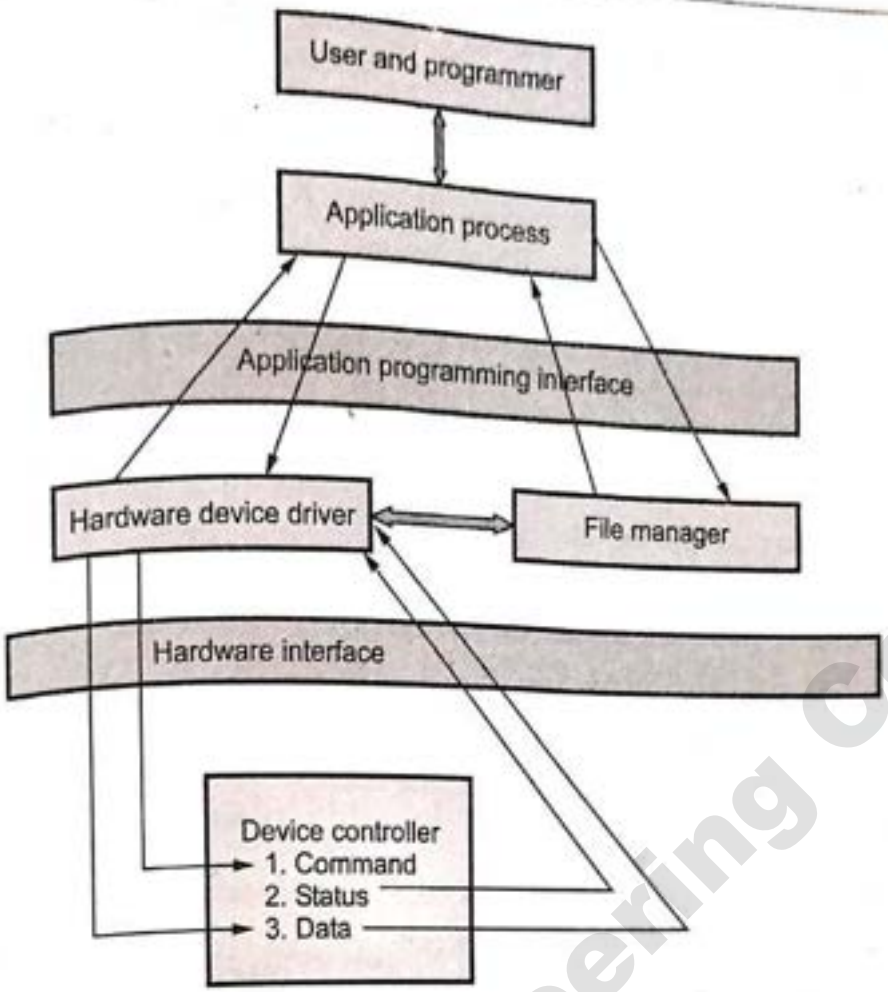


Fig. 5.8.1 : Device management organization

- To start I/O operation, the CPU loads appropriate instructions and values into the registers of the device controller via the device driver. The device controller examines the registers :
  1. The request may be a read or write instruction.
  2. The controller performs the appropriate actions.

Once finished, the controller triggers, an interrupt. The interrupt handler services the interrupt once it occurs.

Following techniques are used for performing I/O

1. Programmed I/O 2. Interrupt driven I/O 3. Direct Memory Access (DMA).

Evolution of I/O Function

1. Peripheral devices are directly controlled by CPU.
2. Input-output module or controller module is added. Processor uses programmed I/O with interrupts.
3. Same as step 2 but only interrupt are used.

4. The I/O takes direct control of memory by using DMA. It will transfer data by using DMA.
5. The I/O module looks like a separate processor with required instruction.
6. The I/O module has a local memory of its own. A large set of I/O devices can be controlled with minimal processor involvement.

5.8.1 DMA

- A special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continous intervention by the processor. This approach is called Direct Memory Access (DMA).
- DMA can be used with either polling or interrupt software. Fig. 5.8.2 shows the typical DMA block diagram.
- DMA is particularly useful on devices like disks, where many bytes of information can be transferred in single I/O operations.
- When used in conjuction with an interrupt, the CPU is notified only after the entire block of data has been transferred.
- For each byte or word transferred, it must provide the memory address and all the bus signals that control the data transfer.

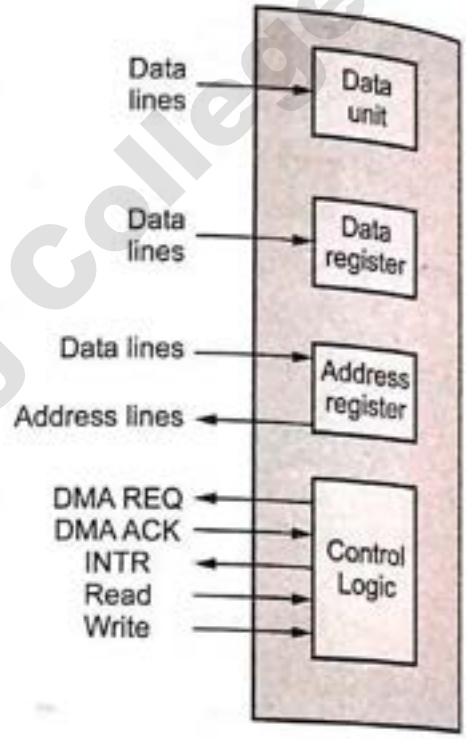


Fig. 5.8.2 : DMA block diagram

- Following Fig. 5.8.3 shows the difference between traditional I/O and DMA.

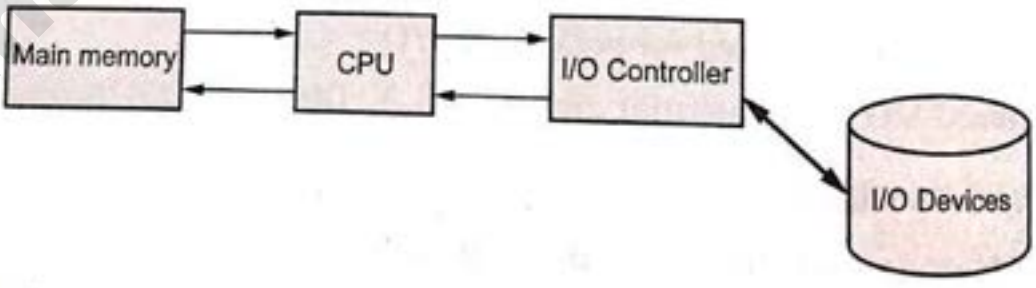


Fig. 5.8.3 (a) Traditional I/O operation

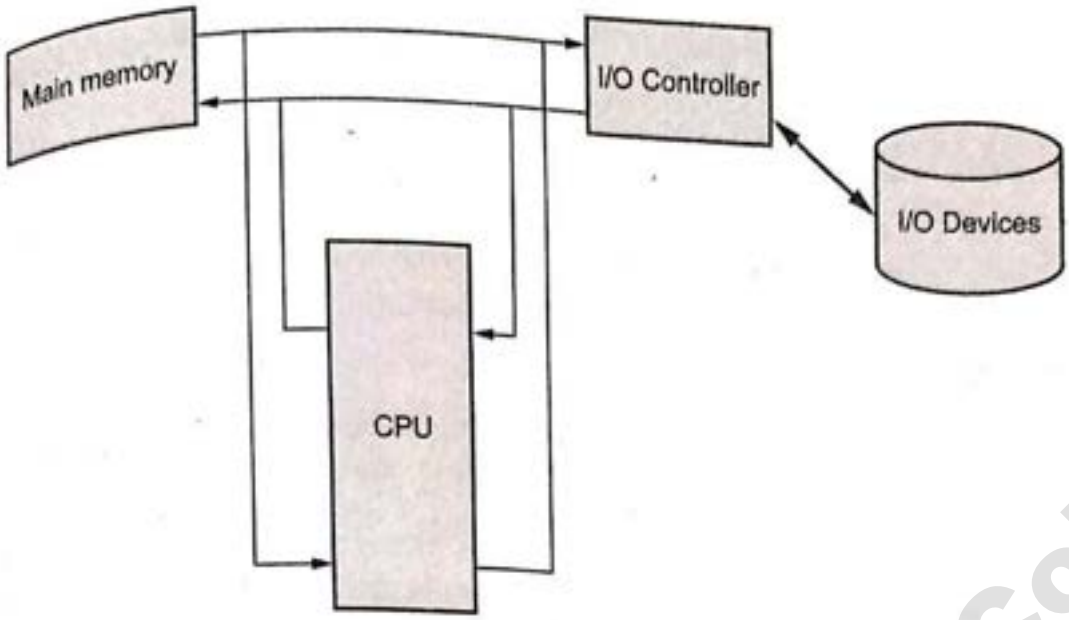


Fig. 5.8.3 (b) DMA operation

- DMA mechanism can be configured in a variety of ways.

1. Single bus, detached DMA

- All the modules use same system bus.
- This configuration is inefficient but inexpensive.
- It uses programmed I/O to exchange data between memory and an I/O module through the DMA module.

2. I/O bus

- I/O bus provide easily expandable configuration.
- It reduces number of I/O interfaces in the DMA module.
- Exchange of data between the DMA and I/O module takes place off the system bus.

DMA data transfer operation

Program → P

Device → D

1. Program makes a DMA setup request.
2. Program deposits the address value A and the data count (d).
3. Program also indicates the virtual memory address of the data on disk.
4. DMA controller records the receipt of relevant information and acknowledges the DMA complete.

5. Device communicates the data to the controller buffer.
6. The controller grabs the address bus and data bus to store the data, one word at a time.
7. Data count is decremented.
8. The above cycle is repeated till the desired data transfer is accomplished. At the same time a DMA data transfer complete signal is sent to the process.

5.8.2 Direct I/O with Polling

- CPU is responsible for data transfer. It transfers data between primary memory and device controller. Following are the steps for polling :
  1. Application process issues a read operation.
  2. Device driver check the status of device. If device is busy, the driver waits for it to become idle.
  3. Driver stores an input command into the controller command register. It starts the device.
  4. Driver continuously read the status register while waiting for the device to complete its operations.
  5. Driver copies the content of the controller data register into the user process space.

5.8.3 Interrupt Driven I/O

- When interrupt I/O is used, the CPU is free to ignore the I/O module until the interrupt signal a received. The only added overhead is processing a single interrupt when the I/O operation completes.
- In this method the program issues an I/O command and than continues to execute until it is interrupted by the I/O hardware to signal the end of I/O operation. Here the program enters a wait loop in which it repeatedly checks the device status. During this process the processor is not performing any useful computation.
- There are many situations where tasks can be performed while waiting for an I/O device to be ready, to allow this the I/O device should alert the processor when it becomes ready. It can be done by sending a hardware signal called an **interrupt**.
- The routine executed in response to an interrupt request is called Interrupt Service Routine (ISR). The processor first completes execution of instruction then it loads the program counter with the address of 1st instruction of ISR.

- In a multiprogramming system the wasted CPU time could be used by another process; because the CPU is used by other processes in addition to the one waiting for the I/O operation completion, in multiprogramming system may result a sporadic detection of I/O completion; this may be remedied by use of interrupts.
- The reason for incorporating the interrupts into computer hardware is to eliminate the need for a device driver to constantly poll the CSR.

Steps for performing an input operation

- The application process requests a read operation.
- The device driver queries the CSR to find out if the device is idle; if busy, then it waits until the device becomes idle.
- The driver stores an input command into the controller's command register, thus starting the device.
- When this part of the device driver completes its work, it saves information regarding the operation it began in the device status table; this table contains an entry for each device in system; the information written into this table contains the return address of the original call and any special parameters for the I/O operation; the CPU, after is doing this, can be used by other program, so the device manager invokes the scheduler part of the process manager. It then terminates.
- The device completes the operation and interrupts the CPU, therefore causing an interrupt handler to run.
- The interrupt handler determines which device caused the interrupt; it then branches to the device handler for that device.
- Device driver retrieves the pending I/O status information from the device status table. The device driver copies the content of the controller's data register(s) into the user process's space.
- The device handler returns the control to the application process.

5.9 Application I/O Interface

AU : May-13, June-14

- I/O devices are of two types :
  1. Block devices
  2. Character devices
- **Block device** stores information in fixed size blocks. Operating system assigns address to each block. A Block device is one with which the driver communicates by sending entire blocks of data. Examples for block devices : hard disks, USB cameras, Disk-On-Key.

- A block device is a device (e.g., a disk) that can host a file system. In UNIX systems, a block device can only handle I/O operations that transfer one or more whole blocks, which are usually 512 bytes in length.
- Every block device driver must provide an interface to the buffer cache as well as the normal file operations interface. Each block device is identified by major and minor numbers.
- Block devices are characterized by the capability to read and write blocks of data to and from random locations on an addressable medium. Examples of block devices include hard drives and USB Flash drives.
- Block and character devices are represented for user space applications as files than can be manipulated using the traditional file API.
- A **character device** is one with which the driver communicates by sending and receiving single characters. Examples for character devices : serial ports, parallel ports and sound cards.
- The only difference between a character device and a regular file is that, user can always move back and forth in the regular file, whereas most character devices are just data channels, which user can only access sequentially. User can perform mmap and lseek operation on the character device.
- Character devices can be thought of as serial streams of sequential data. Examples of character devices include serial ports and keyboards.
- Linux supports loadable device drivers, it is relatively easy to demonstrate a simple device driver skeleton. A device driver is a special kind of binary module. Unlike a stand-alone binary executable application, a device driver cannot simply be executed from a command prompt.
- Character device drivers are referenced through special files in the file system. This file is stored in the */dev* folder. The command "*ls -l*" is used to check the folder. The character "c" in the file listing indicates the file is a char device. It also contains device major and minor numbers.

5.9.1 Difference between Block and Character Device

Sr. No.	Character Device	Block Device
1.	A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets).	A block device is one with which the dirver communicates by sending entire blocks of data.
2.	Examples for character devices : serial ports, parallel ports, sounds cards.	Examples for block devices : hard disks, USB cameras, Disk-On-Key.

3.	The character devices provide for direct transmission between the disk and the user's read or write buffer.	The block devices access the disk using the system's normal buffering mechanism.
4.	Read the data character by character	Read only blocks of data.
5.	These devices are access only serially or sequentially.	These devices can be accessed randomly.

5.9.2 Network Device

- Network devices are deal differently from block and character devices. But these devices operate like I/O devices. User can not directly transfer data to network devices.
- Socket interface is used for communicating with network devices. Data can be put into the socket at one end, and read out sequentially at the other end. Sockets are normally full-duplex, allowing for bi-directional data transfer.
- UNIX uses pipes, FIFO, streams, queue and mailbox as a network device. Each network device is represented by a device data structure. The device data structure contains information about the device and the addresses of functions that allow the various supported network protocols to use the device's services.
- Network device drivers can be built into the Linux kernel. Each potential network device is represented by a device data structure within the network device list.

5.9.3 Clock and Timers

- Clocks are required in multiprogramming environment. Computers have hardware clocks and timers.
- Three types of time services are commonly needed in modern systems:
  1. current time of day
  2. elapsed time
  3. Set a timer for particular event
- Operating system used these function for timer applications. When user creates a file, operating system record it time and date.
- A Programmable Interrupt Timer (PIT) can be used to trigger operations and to measure elapsed time. User process uses timer by using operating system interface.
- Hardware clock is used to generate interrupt by operating system. On most systems the system clock is implemented by counting interrupts generated by the PIT.
- Clock and timer are used for preventing processes from running longer than allocated time period.

AU : Dec.-16

5.10 STEAMS

- STREAM is used as full duplex communication between a device driver and us level process. User process is interface with STREAM head.
- A STREAM consists of :
  1. STREAM head interfaces with the user process.
  2. Driver end interfaces with the device.
  3. Zero or more STREAM modules between them.
- Each module contains a **read queue** and a **write queue.** Message passing is use to communicate between queues.

5.10.1 Streams

- A stream is a full duplex connection between a process and a device driver. I consists of a set of linearly linked queue pairs, one member of each pair for inpu and other for output.
- Stream provide greater modularity and flexibility for the I/O subsystem.
- When a process writes data to a stream, the Kernel sends the data down the output queues.
- Each queue is a data structure that contains the following elements.
  1. An open procedure, called during an open system call.
  2. A close procedure, called during a close system call.
  3. A put procedure called to pass a message into the queue.
  4. A service procedure, called when a queue is scheduled to execute.
  5. A pointer to the next queue in the stream.
  6. A pointer to a list of message awaiting service.
  7. A pointer to a private data structure that maintains the state of the queue.
  8. Flags, high water mark and low water marks used for flow control scheduling and maintaining the queue state.
- Fig. 5.10.1 shows a stream after open. When another process opens the device, the Kernel finds the previously allocated stream via the inode pointer and invokes the open procedure of all modules on the stream. The Kernel assigns a special pointer in the in-core inode to indicate the stream head.

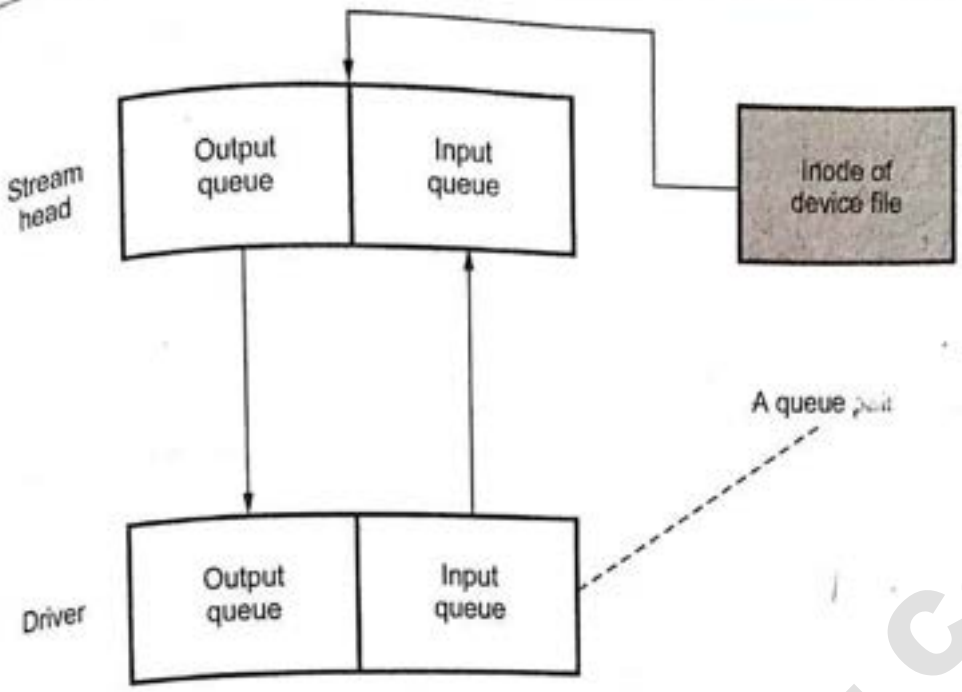


Fig. 5.10.1 Stream after open

- Modules communicate by passing messages to neighboring modules on a stream. A message consists of a linked list of message block headers. Each block header points to the start and end location of the block's data.
- Fig. 5.10.2 shows streams messages.

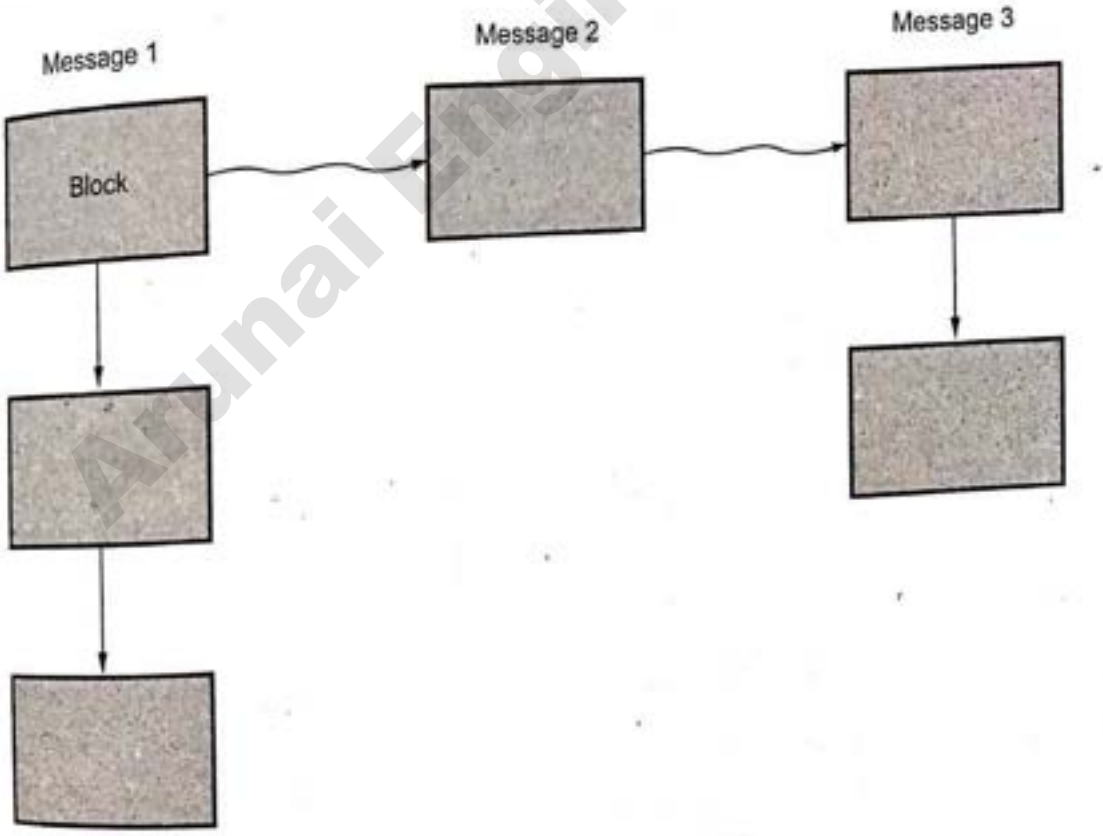


Fig. 5.10.2 Streams messages

- Messages are two types : Control and data. These are indicated by a type indicator in the message header.
- Control messages may result from ioctl system calls or from special conditions, such as a terminal hang up.
- Data messages may result from write system calls or the arrival of data from a device.
- Fig. 5.10.3 shows the pushing a module onto a stream.

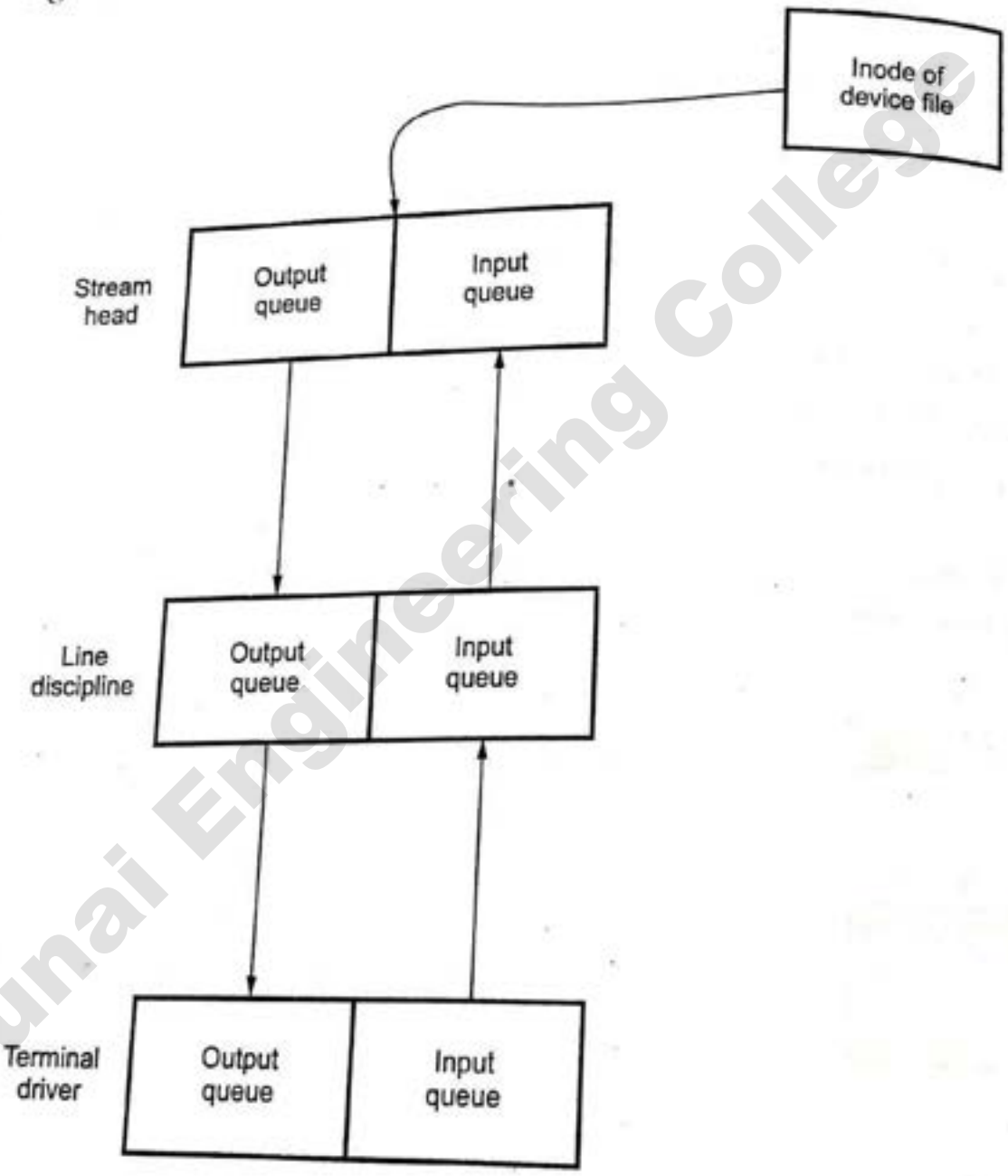


Fig. 5.10.3 Pushing a module onto a stream

- A process can push a line discipline module onto a terminal driver stream to do erase and kill character processing. The line discipline module does not have the same interfaces as the line discipline but its function is the same.

- A code segment that opens a terminal and pushes a line discipline may look like this :

```
fd = open ("/dev/ttyxy" O_RDWR);
ioctl (fd, PUSH, TTYLD);
```

where

PUSH = Command name

TTYLD = Number that identifies the line discipline module.

- There is no restriction to how many modules can be pushed onto a stream.
- A process can POP the modules off a stream in last in first out order, using another octl system call.

```
ioctl (fd, POP, 0);
```

University Question

1. *Distinguish between a STREAMS driver and a STREAMS module.*

AU : Dec.-16, Marks 8

5.11 Kernel I/O Subsystem

AU : May-17

- Kernel I/O subsystem provides various services like scheduling, buffering, caching, spooling, device reservation and error handling.

1. **I/O scheduling :** Some I/O request ordering via per-device queue. Scheduling can improve overall system performance. Operating-system developers implement scheduling by maintaining a queue of requests for each device.
2. **Buffering :** Buffer is a memory area that stores data while they are transferred between two devices or between a device and an application.
3. **Caching :** It is fast memory region which holds copies of data. Access to the cached copy is more efficient than access to the original. Cache files that are read/written frequently. Caching increases performance.
4. **Spooling and device reservation :** Spooling uses the disk as a large buffer for outputting data to printers and other devices. It can also be used for input, but is generally used for output. Its main use is to prevent two users from alternating printing lines to the line printer on the same page, getting their output completely mixed together. It also helps in reducing idle time and overlapped I/O and CPU.
5. **Error handling :** Most operating systems return an error number or code when I/O request fails. OS can recover from disk read, device unavailable, transient write failures.

6. **I/O protection :** User application may accidentally attempt to disrupt normal operation via illegal I/O instructions. To avoid this we can use all I/O instructions defined to be privileged and I/O must be performed via system calls.
7. **Kernel data structures :** Kernel keeps state info for I/O components, including open file tables, network connections, character device state. The kernel must create a data structure representing the new process. The scheduler can then use it if it decides to schedule that process to run when it is its turn.

5.11.1 Transferring I/O Requests to Hardware Operations

- Reading a file from disk, application refers to the data by a file name. On the secondary storage device, file system maps from the file name through the directory.
- In UNIX operating system, the name maps to an inode number and that inode contains the space-allocation information.
- UNIX represents device names in the regular file-system name space. To resolve a path name, UNIX looks up the name in the mount table to find the longest matching prefix; the corresponding entry in the mount table gives the device name.
- UNIX uses concept of major and minor number. Device files are found in the /dev directory. Each device is assigned a major and minor device number.
- The major device number identifies the type of device, i.e. all SCSI devices would have the same number as would all the keyboards. The minor device number identifies a specific device, i.e. the keyboard attached to this workstation.
- Consider reading a file from disk for a process :
  1. Determine device holding file.
  2. Translate name to device representation.
  3. Physically read data from disk into buffer.
  4. Make data available to requesting process.
  5. Return control to process.

University Question

1. *Explain about Kernel I/O subsystem and transforming I/O to hardware operations.*

AU : May-17, Marks 7

5.12 Performance

- I/O devices play important role in system performance. Different I/O devices send request to CPU to execute device driver code and CPU also take care of scheduling the processes fairly. This result in context switching.
- Because of context switching, overhead of CPU and hardware caches are increases.
- Programmed I/O can be more efficient interrupt request. Many I/O devices send interrupt request to the system but handling many interrupt request is an expansive task. Whenever interrupt occurs, system state changed, interrupt handlers execute and again system state is restored.
- Network traffic is also an example of high context switch. When user login on the remote machine, each typed user character is sent from client machine to remote server. User types the login name and password on the local machine so keyboard generates an interrupt for every typed key.
- Character is passed through interrupt handler to the device driver, to the kernel and then to the user process. To send a character to the remote machine, user process generates a network I/O system call. Then character is travel to local kernel, then network layer and network device driver etc.
- To reduce an interrupt overhead on the CPU, some operating system uses separate front end processors for terminal I/O.
- Following are the methods to improve the efficiency of I/O :
  1. Limit the number of context switches.
  2. Increase the concurrency operations.
  3. Reduce the use of memory while copying data in between device and user application.
  4. Use hardware for processing primitives.
  5. Reduce the overhead of CPU, memory subsystem, bus and I/O performance.

Two Marks Questions with Answers

Q.1 List out any two disk scheduling techniques.

Ans. : Disk scheduling techniques are FCFS, SSTF, SCAN and C-SCAN.

Q.2 Define seek-time and latency time for a hard-disk (HDD) mechanism.

Ans. : Latency time : It is the time spent waiting for the target sector to appear under the read and write head.

Seek time : Seek time is the time required to move the disk arm to the required track.

Q.3 In the context of disk scheduling define seek time.

Ans. : Seek time is the time required to move the disk arm to the required track.

Q.4 In the context of disk reliability define mirroring.

Ans. : A mirror set comprises two equal sized partitions on two disks. When an application writes data to a mirror set, FtDisk writes contents of the two partitions are identical.

Q.5 List the responsibilities of the operating system in connection with disk management.

Ans. : Responsibilities are

1. Free space management 2. Storage allocation 3. Disk scheduling

Q.6 What is the need for disk scheduling ? AU : CSE/IT : Dec.-12

Ans. : For a multiprogramming system with many processes, the disk queue may often have several pending requests. Thus, when one request is completed, the OS chooses which pending request to service next. To reduce seek time and increase disk bandwidth, disk scheduling is required.

Q.7 Briefly discuss the relative advantages and disadvantages of sector sparing and sector slipping.

Ans. : Sector sparing

Advantage

1. It highs bad sectors.

Disadvantage

1. Provides few spare sectors. So wasted of space.

Sector slipping

Advantage

1. Replaces the bad block.

Disadvantages

1. Data is lost 2. Requires manual intervention

Q.8 Why is rotational latency usually not considered in disk scheduling ? AU : May-16

Ans. : It is additional time waiting for the disk to rotate the desired sector to the disk head.

Q.9 What is rotational latency ? AU : CSE/IT : Dec.-11, May-13

Ans. : The time to wait for the target sector to rotate underneath the head.

Q.10 What is a swap space ? AU : CSE/IT : Dec.-10

Ans. : Secondary memory is swap device and the section of disk used for this purpose is known as swap space.

Q.11 Write the three basic functions which are provided by the hardware clocks and timers. AU : CSE/IT : May-11

Ans. : Three basic functions are :

1. Give the current time.
2. Give the elapsed time.
3. Set a timer to trigger operation X at time T.

Q.12 What are seek time and rotational latency ? AU : CSE/IT : Dec.-11

Ans. : Seek time : It is a time required to move the disk arm to the required track.

Rotational latency : It is time spent waiting for the target sector to appear under the read and write heads.

Q.13 Define mirroring and shadowing. AU : CSE/IT : Dec.-11

Ans. : Mirroring : Mirroring is the replication of logical disk volumes onto separate physical hard disks in real time to ensure continuous availability. Mirroring is also called shadowing.

Q.14 Which disk scheduling algorithm would be best to optimize the performance of a RAM disk ? AU : CSE/IT : Dec.-11

Ans. : RAM disk has uniform access times. So all disk scheduling algorithm gives same effect.

Q.15 Writable CD-ROM media are available in both 650 MB and 700 MB versions. What is the principle disadvantage, other than cost, of the 700 MB version ? AU : CSE/IT : Dec.-11

Ans. : Limited space is disadvantage of 700 MB versions.

Q.16 What is seek time ? AU : CSE/IT : May-11

Ans. : Seek time is the time required to move the disk arm to the required tracks.

Q.17 What characteristics determine the disk access speed ? AU : CSE/IT : May-11

Ans. : Rotational delay and seek time.

Q.18 Give the importance of swap-space management. AU : CSE/IT : Dec.-12

Ans. : Importance of swap space management is to provide the best throughput for the virtual memory system.

Q.19 Explain difference between latency and throughput.

Ans. : Latency is defined as the time required processing a single instruction, while throughput is defined as the number of instructions processed per second.

Q.20 What is cylinder ?

Ans. : Cylinder is used to refer to all the tracks under the arms at a given points on all surfaces.

Q.21 What is bandwidth ?

Ans. : The maximum amount of information that can be transferred to or from the memory per unit time is called bandwidth.

Q.22 Define constant linear velocity.

Ans. : The Constant Linear Velocity (CLV) format optimizes utilization of the usable recording area by having an uniform recording density throughout the disk. By maintaining the linear velocity constant, the recording density is kept constant. Data are stored in one continuous track that spirals from the circumference inwards to the centre. As the tracks moves inwards, the radius decreases and to maintain the same linear velocity the disk must spin faster.

Q.23 What is SCAN disk scheduling ?

Ans. : The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

Q.24 What are the two major components of access time ?

Ans. : Two components are :

a. Seek time is the time for the disk are to move the heads to the cylinder containing the desired sector.

b. Rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head

Q.25 What is the difference between block-oriented devices and stream-oriented devices ?

Ans. : Block-oriented devices stores information in blocks that are usually of fixed size, and transfers are made one block at a time. Generally, it is possible to reference data by its block number. Disks and tapes are examples of block-oriented devices. Stream-oriented devices transfer data in and out as a stream of bytes, with no block structure. Terminals, printers, communications ports, mouse and other pointing devices, and most other devices that are not secondary storage are stream oriented.

Q.26 What are the device drivers ?

Ans. : A device driver is a program that controls a particular type of device that is attached to your computer. There are device drivers for printers, displays, CD-ROM readers, diskette drives and so on.

Q.27 What is tertiary storage ?

Ans. : Tertiary storage is built from disk and tape drives that use removable media. Low cost is the defining characteristic of tertiary storage.

Q.28 What is trap ?

Ans. : A trap is a software generated interrupt caused either by an error or by a specific request from a user program that an operating system service be performed.

Q.29 What do you mean by WORM disk ?

Ans. : WORM means "Write Once, Read Many Times" disks can be written only once. Thin aluminum film sandwiched between two glass or plastic platters. To write a bit, the drive uses a laser light to burn a small hole through the aluminum; information can be destroyed by not altered.

Q.30 Give the various disk scheduling methods.

Ans. : 1. FCFS 2. SSTF 3. SCAN 4. C-SCAN 5. LOOK 6. C-LOOK

Q.31 What is HSM ? Where it is used ? AU : CSE/IT : May-15

Ans. : Hierarchical Storage Management (HSM) is a data storage techniques that automatically moves data between high-cost and low-cost storage media. HSM systems store the bulk of the enterprise's data on slower devices, and then copy data to faster disk drives when needed.

Q.32 A disk has 26310 cylinders, 16 tracks and 63 sectors. The disk spins at 7200 rpm. Seek time between adjacent track is 1 ms. How long does it take to read the entire disk ? (Refer section 5.6) AU : Dec.-15

Q.33 How does DMA increase system concurrency ? AU : May-16

Ans. : DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory buses.

Q.34 Define C-SCAN scheduling. AU : EIE : Dec.-16

Ans. : Head begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction

Q.35 Why is it important to scale up system-bus and device speeds as CPU speed increases ? AU : EIE : Dec.-16

Ans. : Consider a system which performs 50 % I/O and 50 % computes. Doubling the CPU performance on this system would increase total system performance by only 50 %. Doubling both system aspects would increase performance by 100 %. Generally, it is important to remove the current system bottleneck and to increase overall system performance rather than blindly increasing the performance of individual system components.

Q.36 Suppose that the disk rotates at 7200 RPM. What is the average rotational latency of this disk drive ? AU : CSE : May-17

Ans. : Disk rotates at 7200 rpm which gives 120 rotations per second.
One rotation = 60s / 7200 = 8.33 ms Thus, a full rotation takes 8.33 ms and the average rotational latency (a half rotation) takes 4.167 ms.

❑❑❑

UNIT - IV

6 File Systems

Syllabus

File System Interface - File concept, Access methods, Directory Structure, Directory organization, File system mounting, File Sharing and Protection; File System Implementation - File System Structure, Directory implementation, Allocation Methods, Free Space Management, Efficiency and Performance, Recovery.

Contents

6.1 Basic File Concepts

- Files are the building blocks of any operating system. Permanent storage of information and data is a file. File provides an easy means of information sharing. The operating system is not interested in the information stored in the files. Operating system map files with physical devices.
- An unstructured sequence of data means a file. It is also logical units of information created by processes. Information stored in the files must be persistent. File is any organized information and stored within the secondary storage. It is physical view of a file.
- A program which user prepare is a file. File represents programs and data. The output from complier may be an object code file or an executable file.
- Content or information/data of file is decided by the creator of the file. File contains various types of data like source program, object program, text and numeric data, images, graphs, sounds, payroll records, dead-stock records etc.

6.1.1 File Naming

- File store information on the disk. It is an abstract mechanism. File name is given, when process creates a file. File name continues to exist even after closing a file. It is accessed by other process.
- File name rule is changed according the operating system. All operating system allowed upto 8 string as a file name. Special characters and numbers are allowed to use as a file name. Many file systems support names as long as 255 characters.
- File names contains two parts. It is separated by a period. For example : **sorting.c**
- In the file name, after period is called file extension. File extension indicates something about a file. UNIX operating system support more than one file extension. It is up to the user, for extending the file extension.
- MS-DOS support only one file extension. File extension after period contains upto three characters. Following is the list of file extension.

Sr. No.	File extension	Remarks
1.	.c	C source program file
2.	.exe	Executable file
3.	.bak	Backup file
4.	.hlp	Help file

5.	.pdf	Portable document format file
6.	.txt	General text file
7.	.zip	Compressed file
8.	.gif	Graphics interchange format image file
9.	.html	WWW html file
10.	.mp3	Music encoded in MPEG layer 3 audio file format
11.	.ps	Postscript file

6.1.2 Types of File

- File type represents the internal structure of the file. According to the structure, file is divided into certain types : text, source, executable and object file.

1. **Text file :** It is sequence of character which organized into lines. Text file is a regular file.
2. **Source file :** It contains sequence of subroutines, procedure and functions.
3. **Executable file :** It contains a series of code sections. This file is input to the loader and loader loads this file into memory and then execute.
4. **Object file :** Object file is input to the linker. An Object file is a binary file that the compiler creates that converts the source to object code.

6.1.2.1 File Types

- File is divided into following types :
  1. Ordinary file
  2. Directory file
  3. FIFO file
  4. Character device file
  5. Block device file

Ordinary file

- Also called regular file. It contains only data as a stream of characters. An ordinary file cannot contain another file, or directory.
- It is either text file or a binary file.
- Examples of ordinary files include simple text files, application data files, files containing high-level source code, executable text files, and binary image files.
- **Text file :** It contains only printable characters.
- **Binary file :** It contains printable and unprintable characters that cover the entire ASCII range.
- Regular files may be created, browsed through, and modified by various means such as text editors.

Directory file

- A directory file is like a folder that contains other files, including subdirectory files.
- Directory files act as a container for other files, of any category. Directory files don't contain data in the user sense of data; they merely contain references to the files contained within them.
- Directory is created in UNIX by the *mkdir* command.
- The UNIX directory is considered empty if it contains no other files except the "." and ".." .
- Directory is removed by using *rmdir* command. Content of directory is displayed by ls command.

Device file

- Special files are also known as device files. In UNIX all physical devices are accessed via device files; they are what programs use to communicate with hardware.
- Files hold information on location, type, and access mode for a specific device.
- There are two types of device files; **character** and **block**.
- **Block device** files are used to access block device I/O. Block devices do buffered I/O, meaning that the data is collected in a buffer until a full block can be transferred.
- **Character device** files are associated with character or raw device access. They are used for un-buffered data transfers to and from a device. Rather than transferring data in blocks the data is transferred character by character.
- One transfer can consist of multiple characters.
- Some devices, such as disk partitions, may be accessed in block or character mode. Because each device file corresponds to a single access mode, physical devices that have more than one access mode will have more than one device file.
- Device files are found in the */dev* directory. Each device is assigned a major and minor device number.
- The major device number identifies the type of device, i.e. all SCSI devices would have the same number as would all the keyboards.
- The minor device number identifies a specific device, i.e. the keyboard attached to this workstation.
- Device files are created using the *mknod* command.
- Example of character device files is printers, modems and consoles.

FIFO file

- FIFO file is special pipe device file which provides temporary buffers for two or more processes to communicate by writing data to and reading data from the buffer.
- A FIFO is created by the *mknod* system call and may be opened and accessed by any process that knows the name and has the permissions.
- The buffer associated with a FIFO file is allocated when the first process opens the FIFO file for read or write. The buffer is discarded when all processes which are connected to the FIFO close their reference to the FIFO file.
- FIFO file may be removed like any regular file. FIFO files can be removed in UNIX via the *rm* command.

6.1.3 File Attributes

- One of the characteristics of file is a set of file attributes that give the operating system more information about the file and how it is intended to be used. For human users convenience bane is given to the file. File attributes are varies from system to system.
- File attributes are as follows :

  1. Name
  2. Identifier
  3. Type
  4. Location/place
  5. Size
  6. Protection
  7. Date and time

- **File name** is in human readable. User can perform any operation on file using its name.
- When file is created by user, operating system assigns unique identification number to each file. OS uses this **identifier** for its operation.
- **Type** information is required for systems that support different types of files.
- **Location/place** information is pointer to a device and actual location of files on the device. The information needed to locate the file on disk is kept in memory.
- **Size** is measured in bits or bytes. It gives idea about current size of the file.
- **Protection :** This information is stored on the per-process table so the operating system can allow or deny subsequent requests. Attributes like protection, password, creator and owner provides protection to a file.
- **Date and time :** This information is related to creation and modification of files. Protection, security and monitoring purposes this data is used by operating system.

6.1.4 Operation on File

- File operations are simply those things that user can perform on a file. For example, user can create a file, save a file, open a file, read a file and modify a file. There are many different types of file operations supported by operating system.
- Operating system can provides system call for performing various operations on the file. Six basic file operations are creating, write, read, delete, repositioning and truncating.
- Following are the some list of file operation :
  1. Create 2. Write 3. Close
  4. Read 5. Delete 6. Repositioning
- All these operations require that the directory structure be first searched for the target file.
- **File creation :** Any time user can create a file. File is also created without data. The steps for creating a file.
  a. Space : File system must provide a space for the file.
  b. New file entry is made in the directory.
- **Write :** To perform write operation on the file, file name and data is required. It updates a file by adding new data and changes some content of the file. For writing into a file, operating system searches the directory for particular file. Write pointer is kept at the location where the writing starts. Write pointer position is at middle of the file or end of the file. Write pointer is updated when writing to file is completed.
- **Close :** Process is not used a file after closing. Process can not perform any operation on file. Operating system remove it entry from open file table.
- **Read :** A process reads all or some portion of the data in a file. Read system call is used for reading a file. For reading a file, name and position of the memory is specified. Read pointer is kept into the file for next read operation. Read pointer is updated after completion of the read system call. Information about read pointer is stored in *per process current file position pointer.*
- **Delete :** Operating system remove the file from file structure. For deleting a file, directory is searched with particular name and file is deleted. Deleting means making space free from memory and disk. Operating system also removes the directory entry. This free space is used by another file.
- **Repositioning :** The directory is searched for the proper entry, and the current-file-position pointer is repositioned to a given value. Repositioning within

a file need not involve any actual I/O. This file operation is also known as a file seeks.

- Other file operations are open, append, seek, get and set attributes and rename.
- **Open :** Before performing any operation on file, it is necessary to open file in required mode.
- **Append :** Append is related to write operation of file. Append only add data to the end of the file.
- **Get and set attributes :** Get attribute is used before performing operation on file. When user request a file for any operation, get attributes is executed by the process. Depending upon the attributes, user can allow or deny the operation. User can set the attributes and it changed after creation of file.
- **Rename :** User can change the name of existing file.

6.1.5 File Structure

- File structure is represented by field, record, file and database. It is also represented by byte sequence, record sequence and tree. Fig. 6.1.1 shows the three different kinds of files. Unix and windows both use unstructured byte sequence

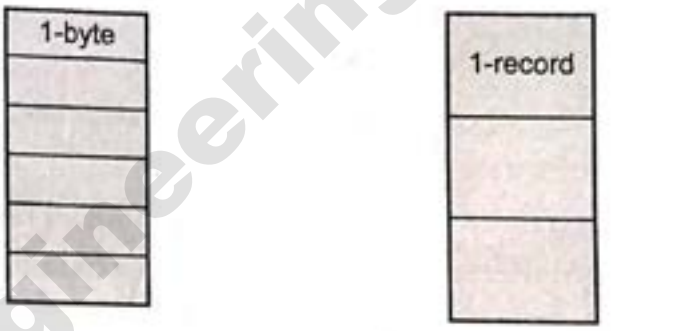


Fig. 6.1.1

- Byte sequence is unstructured file. Operating system is nothing to do with content of file. All content is in byte format. Windows and UNIX operating system uses this byte

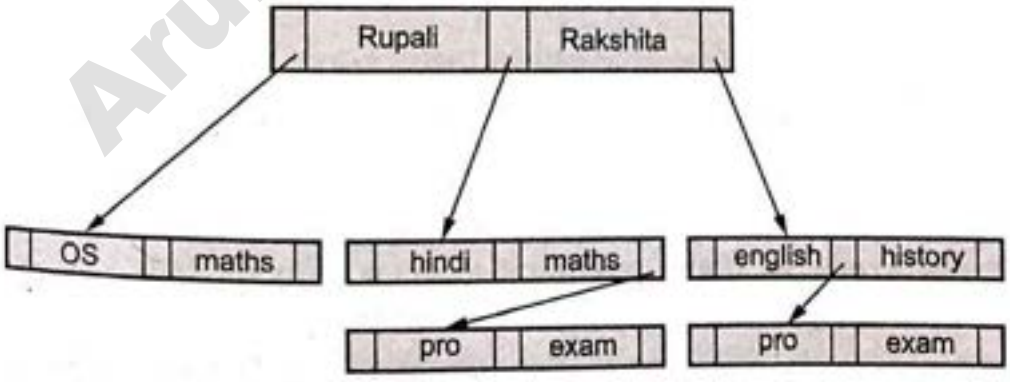


Fig. 6.1.1 (c) Tree structure

sequence method. It provides more flexibility to the user. User can put anything in the file and assign name whatever they want to the file.

- In record sequence model, file contains a sequence of fixed length record. Read and write operation on this type of file will perform one record at a time. It uses an idea of 80 column punched card. Most of the operating system based their file system on file consisting of 80 character records.
- Record sequence model also support 132 character records. Line printer uses 132 characters for printing.
- Last method is tree structure. File consists of tree of records. Each record contains key field in the fixed position. Key field is used to sort the tree record.
- Field, record, file and database is used in these file structure. Field is a basic element of data. Persons last name and date example of the field. It contains single value. Length and data type is characteristics of field.
- Collection of related field is records. A student record contains name, address, mobile number, university exam number, college name etc. Records may be fixed length or variable length. A record also contains sub-records. Sub-records may be one or more than one.
- File is collection of similar records. From user view, file is single entity and reference by name.
- Database is a collection of related data. All records have some relation in database. Engineering college database contains information about college, student, staff, university, syllabus, timetable, fee structure, various departments etc. One or more field is used in database.

6.2 File Access Methods

AU : May-17

- Information and data is kept in files. Files are stored on the secondary storage device. When this information is to be used, it has to be accessed and brought into primary main memory.
- Information in files could be accessed in many ways. It is usually dependent on an application. File organization checks how efficiently the input-output storage medium used.
- File access method defines the way processes read and write files. Different access methods are available. Some operating systems provide only one access method and other systems provide many access methods.

  1. Sequential access 2. Direct /Random access 3. Indexed sequential access
- Access method means accesses to files that use a particular file organization are implemented by an input output control system.

6.2.1 Sequential Access Method

- Sequential access method is simple method. The information in a file is accessed sequentially one record after another.
- In this method, a process could read all the records in a file in order, starting at the beginning. It can not skip any records and can not read out of order.
- Fig. 6.2.1 shows sequential access method.

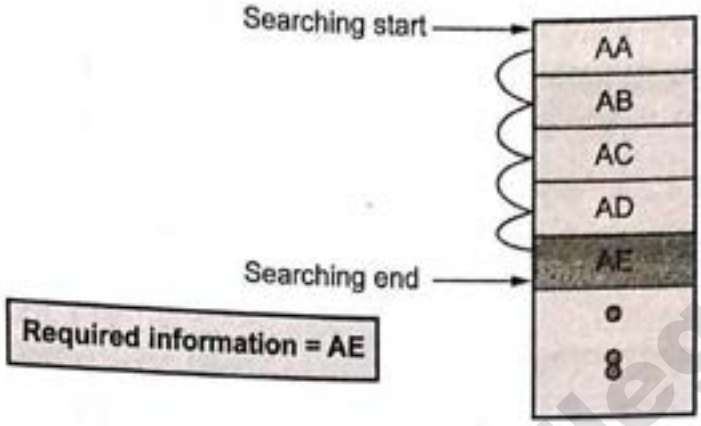


Fig. 6.2.1 Sequential file access

- Batch application uses sequential files. A byte stream file uses sequential access method.
- **Example :** Compiler and editor usually access files in this method. Transaction file is also example of sequential access method.
- Sequential file organization only method easily stored on tape and hard disk. Sequential access is best suited where most of the records in a file are to be processed.

Disadvantages :

- It provides poor performances.
- More efficient search technique is required.

6.2.2 Direct Access Method

- It is also called random access method. This access allows a user to position the read/write mark before reading or writing.
- Fig. 6.2.2 shows direct access method.
- Direct access file method provides, accessing the records directly. Direct access method is based on the hard disk that is a direct access device. It allows random access of any file block.

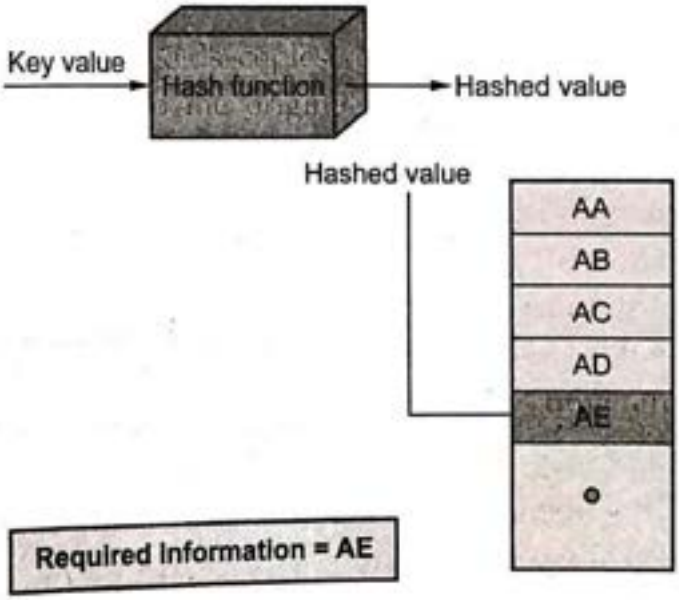


Fig. 6.2.2 Direct access method

- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing. This feature is used by editors. An editors need to randomly access the contents of the file.
- There is no restriction on the order of reading or writing for a direct access file. Operating system support is not required for direct access file.
- At the time of file creation, access method is defined. According to defined access method, file is accessed. Sequential access of a direct access file is possible but direct access of a sequential file is not.

Disadvantages :

1. Poor utilization of input-output device.
2. Consumes CPU time for record address calculation.

6.2.3 Indexed Sequential Access

- It is modified method of direct access method. This method is combination of direct and sequential access method.
- In simple indexed sequential method, simple single level of indexing is used. Sequential file is used as index.
- Fig. 6.2.3 shows indexed sequential file. Indexed sequential access contains key field and a pointer to the main file.

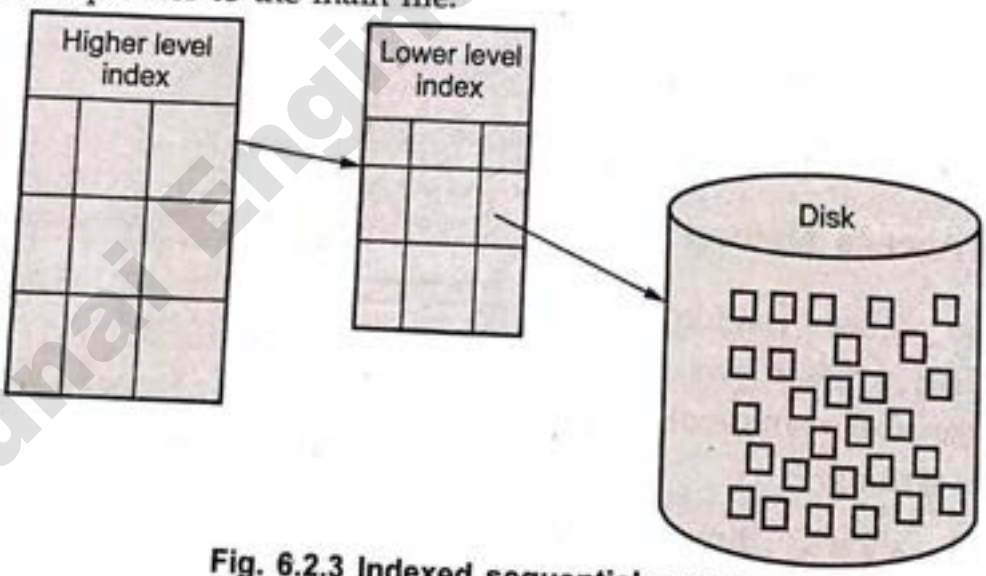


Fig. 6.2.3 Indexed sequential access

- Indexed sequential files are store records in the order that they are written to the disk. Records may be retrieved in sequential order or in random order using an index to represent the record number in the file.
- If file size is large, larger index is required and it contains large number of entries. CPU takes time to search in to the index. Higher level index is used to reduce the search time. An entry in the higher level index points to a section of the index.

This higher index contains the records. This index is searched to find the section of the disk that may contains a required record.

- File system maintains an overflow file. It adds new records to an overflow file. The record in the main file that immediately precedes the new record in logical sequence is updated to contain pointer to the new record in the overflow file.
- The overflow file is merged with the main file during a batch update. The multiple indexes for the same key field can be set up to increase efficiency.
- The lowest level of index file is treated as a sequential file and a higher level index file is created for that file. Higher index file would contain pointers to lower index files, which would point to the actual data items.

University Question

1. *Discuss about the various file access methods.* **AU : May-17, Marks 7**

6.3 File Directory and Directory Structure

AU : May-15,17

- User stores file in file system using directory. File system stores files of many users. Files are stored on the secondary storage device like hard disk, optical disk and memory disk. These device support random access method.
- Directory structure organizes and provides information about all files in the system. Every partition has a file system which consists of files and directory.
- Hard disks are split into one or more partitions. It is called as minidisks or volumes. User can assign names to volumes. Each disk contains minimum one partition.
- Operating systems support upto 24 partitions. Partitions can be larger than a disk i.e combing two hard disks. These partitions are called "virtual disks". Partition maintains information about files in "Device Directory".
- Files are stored in the directory. Directory is created by user. User is free to create as many as directory and assign suitable names. File system contains many directories. Directory maintains information about the file. File information includes type, size, date modified and name.
- In some operating system, directory itself is considered as a file. A set of logically associated files is called directory. Directory structure of the file system connects a directory to other directories in the system.
- Directory is a data structure that lists files and subdirectories in a collection. Directory structure contains a list of entries one for each file.
- Directory does not store any user data. Single dot (.) indicates a current directory and double dot (..) indicates parent directory. The root is identified by the directory '/'. The root file system has several subdirectories.

- Windows file system uses a letter followed by colon for specifying root directory. For example, root directory in windows is C: or D: or E:
- UNIX and Linux based file system uses slash (/) for specifying root directory.
- A directory can contain any number of files. A directory can also contain subdirectories. Because a directory is nothing more than a special kind of file, directories also have names.

Operation on directory :

1. **File searching :** Directory structure is searched for particular file entry. File uses symbolic names and similar names may indicate a relationship between files.
2. **Create a file :** User creates new files and added to the directory.
3. **Delete a file :** User remove the file after completing its work on files.
4. **Rename a file :** User change the file name if file content is change.
5. **Listing of directory :** Listing of files in the directory for some use. MS-DOS and windows uses "dir" command and Linux/UNIX uses "ls" command for this purposes.

6.3.1 Single Level Directory

- Single level directory is a simple structure and there are no sub-directories.
- Fig. 6.3.1 shows three files with single level directory. It is also called flat directory structure.
- Single level directory structure is suitable only for single user. If user increases, it creates the problem with assigning the name to files.

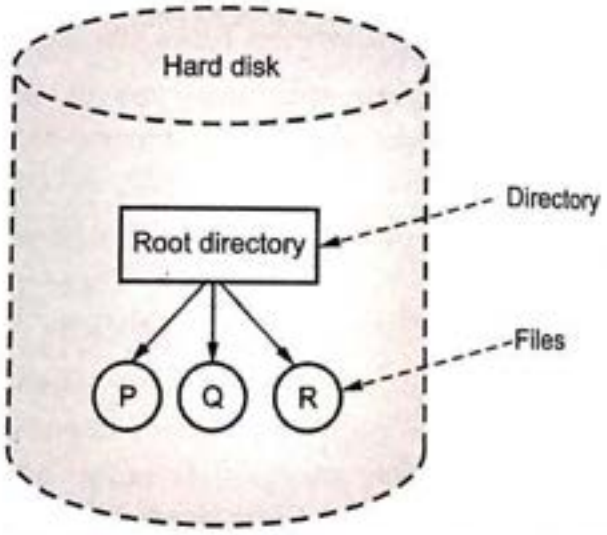


Fig. 6.3.1 Three files with single level directory

- In single level directory structure, no two files can have the same name.
- It is simple to implement and searching of file is faster. This type of directory system is used in cameras and phones.
- Because of limitation of file names, this structure is rarely implemented.
- File name size varies according to the operating system. MS-DOS operating system allows only 11 character file names and UNIX OS allows upto 255 characters.

Disadvantages of single level directory :

1. Not suitable for a large number of files
2. Limitation of unique file name.
3. For large number of file, recalling file name is difficult.

6.3.2 Hierarchical Directory System

- Here files are grouped together to form a sub directory. Each directory may contain several subdirectories. In hierarchical file system, a root directory points to other directories and files.
- Hierarchical directory system contains various forms of directory structures. They are as follows :
  1. Two-level directory structure
  2. Tree structured directories
  3. Acyclic graph directories

6.3.2.1 Two-level Directory Structure

- Two-level directory structure contains master file directory at the top level then user file directory at the second level. Actual user files are at the third level.
- File system maintains a master block for each user. Master block has one entry for each user.

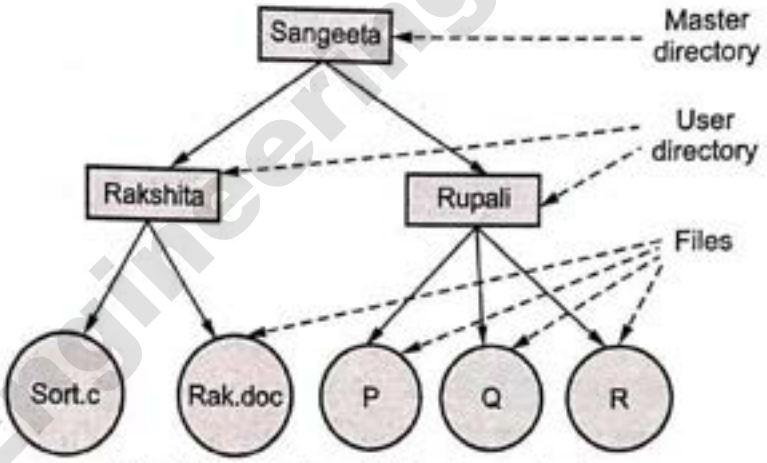


Fig. 6.3.2 Two-level directory structure

- User directory address is stored in the master block. Each user has a private directory. Different user can assign same names to their files as the directories for each user are now separate.
- Fig. 6.3.2 shows two-level directory structure.
- Here user can work only in own directory. Other user need not worry about deleting or modifying files by other users.
- When user wants to create a file, system search that particular file name in the directory. If same file name is not found, then it creates a file otherwise it gives error messages.
- In this structure, different users may have files with same name. It solves the name conflict problem. When user want to open a file, filename is searched for in users directory.
- A two level directory can be a tree or an inverted tree of height 2.

- There are still problems with two-level directory structure. It effectively isolates one user from another. This is some time advantage or some time disadvantages. For completely independent user is an advantages and disadvantages when cooperation between two users is required.

6.3.2.2 Tree Structured Directory

- Fig. 6.3.3 shows the tree structured directory.
- In this structure, directory itself is a file. A directory and sub directory contains a set of files. Internal format is same for all directories.
- Commonly used directory structure is tree structure. Tree has a root directory. All files in disk have a unique path name.
- A path name describes the path that the operating system must take to get from some starting point in the file system to the destination object. Each process has its own working directory. Path names are of two types : **relative and absolute.**

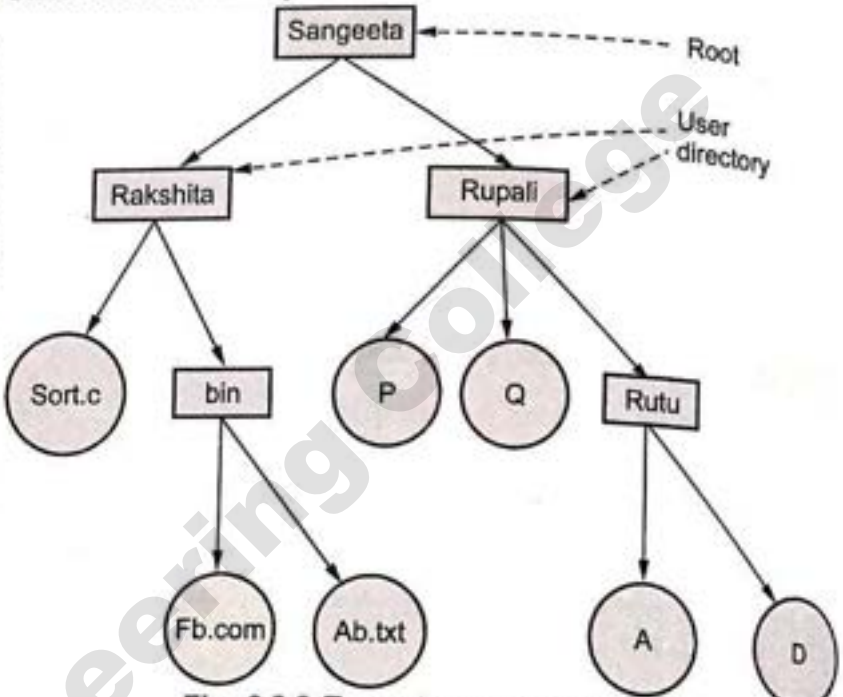


Fig. 6.3.3 Tree structured directory

- Relative path name : Its path starts from current working directory (may start with a "." or "..").
- Absolute path name : Its path starts from root directory (starts from "/").
- Tree structured directory differentiate file and subdirectory by using one bit representation. Zero (0) represents file and one (1) represents subdirectory.
- When process makes reference to a file for opening, file system search this file in the current directory. If needed file is not found in the current directory, then user either change the directory or give the full path of the file. System calls are used for performing any operation on the files or directory.
- Empty directory and its entry are easily deleted by operating system. If directory contains files and subdirectory, i.e. non-empty directory, some operating systems not delete the directory.

- To delete a directory, user must first delete all the files and subdirectory in that directory.

Merits

1. User can create his/her own directory.
2. User can access other user files by specifying path name.
3. Separate subdirectory for files associated with different topics
4. Searching id fast and efficient.

Demerits

1. Sub directory can not share between the user
2. Path is longer than two level directory structure

6.3.2.3 Acyclic Graph Directory

- Acyclic graph directory solve the problem of tree structure directory.
- It allows sharing the directory in between two users. At a time more than one places, shared directory or file will exist in the file system.
- Fig. 6.3.4 shows acyclic graph directory. Links can be used to construct acyclic graph structure. Acyclic graph is graph with no cycles.

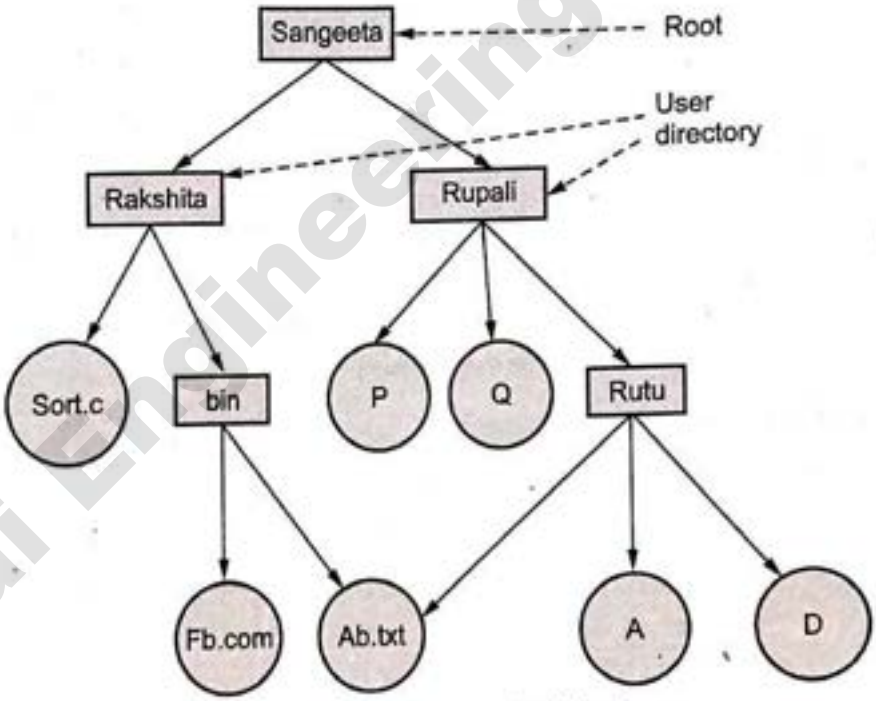


Fig. 6.3.4 Acyclic graph directory

- It is very interesting to note that a shared directory or file is not the same as two copies of the file. When there are two copies of files, each user can view the copy rather than the original, but if one user changes the file content, the changes will not appear in the other's copy.
- Only one original file exists for shared copy. Any changes made by one user are immediately visible to the other user. When user create file in shared directory, it automatically appear in all the shared subdirectories.

Implementation of shared files and directory :

1. Using link :

- Link is used for creating shared files and directory. A link is a pointer to another file or subdirectory. If reference to a file is made, the directory is first searched. A link is a logical relationship between an inode and a file that relates the name of a file to its physical location.
- UNIX operating system uses acyclic graph directory structure. Reference count is maintained by UNIX for files and directory. UNIX operating system provides two types of links : **hard link and symbolic link**
- A **hard link** contains multiple directory entries that both refer to the same file. Hard links are only used with ordinary files in the same file system.
- A symbolic link is also called soft link. It contains a special file which maintains the information about file where to find the linked file. Symbolic links may be used to link directories and files in other file systems.
- Soft links were designed for two specific situations: Links to directories, which must be symbolic, and links to files in other file systems.
- Windows operating system only supports symbolic links.
- Hard links require a reference count, or link count for each file, keeping track of how many directory entries are currently referring to this file. Whenever one of the references is removed the link count is reduced, and when it reaches zero, the disk space can be reclaimed.

2. Duplicating information :

- Duplicate all information about shared files in both sharing directories. Original file will remain as it is without modifying any content by shared user.
- Consistency is not supported by copied file and shared file. Shared files may have more than one absolute path.

Advantages :

1. Simple for file traverse.
2. Structure is more flexible than a simple tree structure.
3. It allows directories and files to be shared between other users.

Disadvantages :

1. Deletion of file is difficult.
2. Problem with duplicate directory entries is maintaining consistency if the file is modified.

University Questions

1. *Briefly discuss about the various directory structures.* **AU : CSE/IT : May-15, Marks 6**
2. *With neat sketch explain about the directory structure.* **AU : May-17, Marks 3**

6.4 File System Mounting

AU : May-16

- Files contained in a file system can be accessed only when the file system is mounted. Each device consists of its own local file system. Mounting is the operation that administrator/user perform to cause the file system on a physical storage device to appear as part of the file system.
- Mount system call is used on UNIX/Linux OS for mounting file system and unmount system call for disconnecting the file system. Mounting file system is used when there are more than one file system is used. Required file system is mounted in the host file system machine as a root directory.
- Some operating system support the mounting the file system at any point.
- File system stores directories and files. Each file system contains exactly one directory at the very top level, called the root directory for that file system. This root directory can contain other directories and files. One file system is designated the root file system or /. Every other file system is mounted under the root file system.

Procedure for mounting file system :

- To mount the file system, device name and location within the file structure is required. It is provides by operating system. A mount point is typically an empty directory where the mounted file system will be attached.
- The operating system verifies the file system. It reads the file system by using device driver and directory for proper format and valid file system.

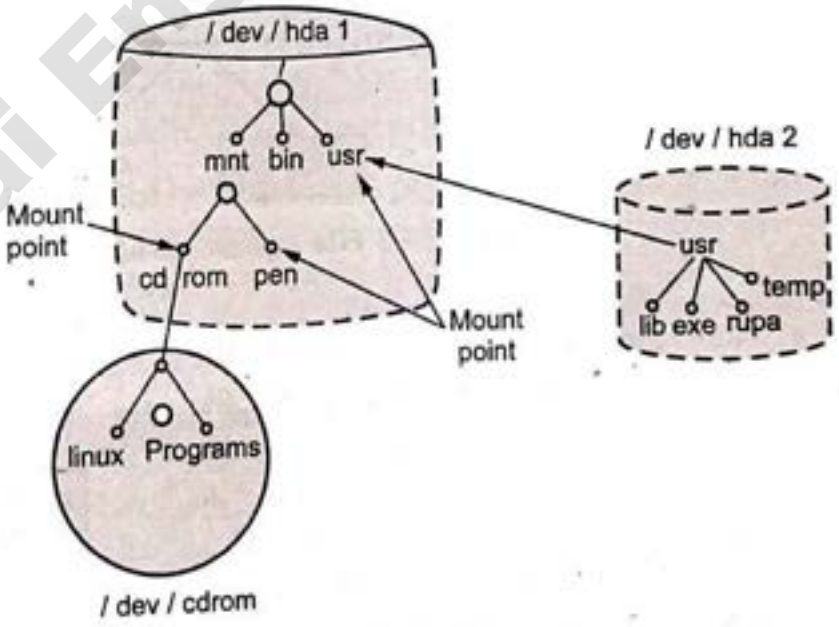


Fig. 6.4.1 Mounting file system

- Fig. 6.4.1 shows the mounting a file system on secondary storage device. File system manage mounted directories by using mount table. User can create soft links to files in mounted file system. Mount tables keep track of the actual physical location of the files.
- When a file system is mounted, the client can reference files by the local path name. There is no reference to remote host location, although files remain physically located at the remote site.
- Busy file system is can not unmount by using unmount command. A file system is considered busy in following situation :
  1. File system is shared in between the users.
  2. A program has a file open in that file system.
  3. A user is accessing a file/directory in the file system.
- **Example :** Consider a file structure of the pen drive. Following is Fig. 6.4.2 of file system before mounting the pen drive.

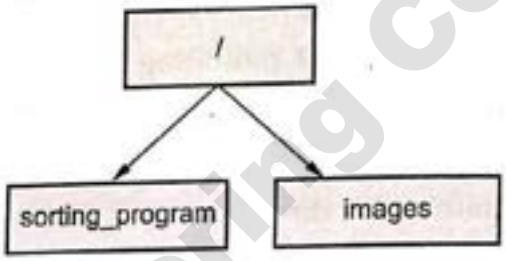


Fig. 6.4.2 File system before mounting

- The file system structure in the pen drive is as follows :

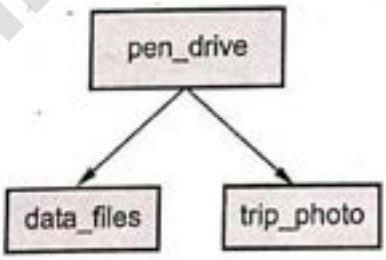


Fig. 6.4.3 File system structure in pen drive

- Now, the pen drive is mounted to the directory sorting_program. Here, the pen drive becomes the part of the file structure. To access a file in trip_photo, the path can be /sorting_program/ trip_photo. Fig. 6.4.4 shows file system after pen drive file system.

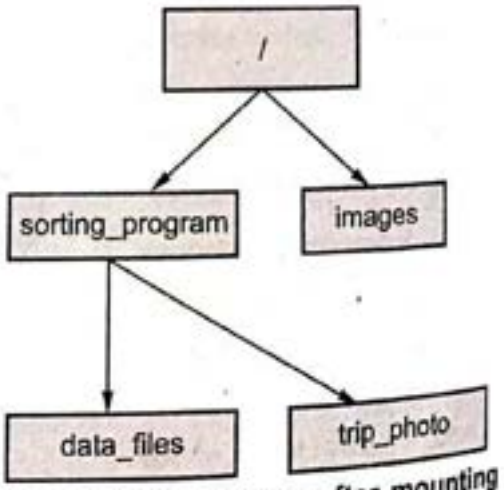


Fig. 6.4.4 File system after mounting

Advantages of mounting :

1. It is a user-oriented naming scheme
2. It makes sense to be able to mount at any level.

Disadvantages of mounting :

1. The hardware is less obvious since devices can be located anywhere in the tree structure

Example 6.4.1 *Discuss the advantages and disadvantages of supporting links to files that cross mount points.*

AU : May-16, Marks 8

Solution :

Advantages :

1. Greater transparency
2. User does not need to be aware of mount points

Disadvantages :

1. The error condition would expose to the user that a link is a dead link and that the link does indeed cross file system boundaries.
2. The file system containing the link might be mounted while the file system containing the target file might not be.

6.5 File Sharing

AU : May-17

- Multiprogramming and multitasking operating system support the multiple user. These users perform various operations on the files. Files are shared between numbers of users. File protection, naming and sharing is issue for file sharing.
- Directory structure may be allows files to be shared by users. Sharing must be done through a **protection** scheme.
- In single user system, operating system maintains attributes for all files and directory. According to the attributes, operating system allows to operate the files for users. But on the multi-user system, more attributes are needed. An attributes like owner, group and other user are required. These are the access permission for files and directory.
- Access permissions are of three types : Read, Write, and eXecute.
  1. An "r" indicates read permissions for a file or directory.
  2. A "w" indicates write permissions.
  3. An "x" indicates execution permissions if it is a file and search permissions if it is a directory.
  4. Owner is the file owner who created a file. Owner having all rights (read, write and execute).
  5. The groups of other user have some limited access permission. Mostly read and execute permission.

6. Last categories of user have only executed permission.

- **Example :** In UNIX operating system, file owner can perform all operation on a file. Group member can execute one subset of those operations and other than the group user can execute another subset of operations.
- When a user requests an operation on a file, the user ID can be compared with the owner attribute to determine if the requesting user is the owner of the file. Likewise, the group IDs can be compared.

Remote file systems

- Computer network is used to support remote file system. User can access remote file system. File Transfer Protocol (FTP) and Secure Shell protocol (SSH). Other supporting protocol like distributed file system and world wide web is also used for remote file system.
- SSH is a protocol for secure remote access to a machine over untrusted networks.
- A client-server model has three components :
  1. **Service :** A service is a software entity that runs on one or more machines. It provides an abstraction of a set of well-defined operations in response to applications' requests.
  2. **Server :** A server is an instance of a particular service running on a single machine.
  3. **Client :** A client is a software entity that exploits services provided by servers.
- Client application program running on the local machine requests a service from another application program, server running on the remote machine. Commonly server provides service to any client, not a particular client.
- Generally, a client application program that requests a service should run only when it is needed. A server program providing service should run all the time, as it does not know when its services will be needed.
- **Client-server** model allows clients to mount remote file systems from servers. The server is connected with multiple clients and it serve to all clients. Client and user-on-client identification is insecure or complicated.
- A client opens the communication channel using IP address of the remote host and the port address of the specific server program running on the host i.e. active open. Request-response may be repeated several times, the process is finite. The client closes the communication channel with an active close.
- A server program opens its door for incoming requests from clients but never initiates a service unless explicitly requested i.e. passive open. A server program is infinite and runs unless a problem occurs.
- Two or more clients can run at the same time on a machine is called concurrency in client. One client must start, run and terminate before another client may start. It is called iterative.
- Network File System (NFS) is standard for UNIX client-server file sharing protocol and Common Internet File System (CIFS) is standard Windows operating system protocol. The NFS client maintains a cache of file and directory attributes. The default settings will not ensure that files created or modified on one system will

be visible on another system within a minute of file creation or modification. Network file system is common distributed file sharing system.

- File data modifications might not be visible on any NFS client system other than the one where the modifications are being made until an NFS commit is executed. Most NFS clients will issue a commit as part of closing a file. If multiple systems are reading files that might be modified, file system locking should be used.
- NFS communication protocols lets processes running in different environments share a file system. NFS implements a file system model that is almost identical to a UNIX system.
- **Distributed Information Systems** such as LDAP, DNS, NIS, and Active Directory implement unified access to information needed for remote computing. It is also known as distributed naming services.
- A name in a distributed system is a string of bits or characters used to refer to an entity. To resolve names a naming system is needed. Names are used to share resources, uniquely identify entities and refer to locations.
- The naming and locating facilities jointly form a naming system that provides the users with an abstraction of an object that hides the details of how and where an object is actually located in the network.
- It provides a further level of abstraction when dealing with object replicas.
- Email and FTP is used for sending information to remote users. But these two protocols will not support true file sharing. Domain Name System (DNS) is used to map the host name to network address translation for the internet. DNS replace the Email and FTP. The DNS is a hierarchical distributed naming system.
- On windows operating system, user name and password is used to create login on the server. Using this authentication method, server will allow and deny access to a requested file system. CIFS is used in conjunction with login name and password.
- Industry is moving towards the new technology. So new protocol called Lightweight Directory Access Protocol (LDAP) is used for secure distributed naming service. LDAP allows user to search for an individual without knowing where they are located.
- An LDAP directory can be distributed among many servers. Each server can have a replicated version of the total directory that is synchronized periodically. An LDAP server is called a Directory System Agent. An LDAP server that receives a request from a user takes responsibility for the request, passing it to other DSAs as necessary, but ensuring a single coordinated response for the user.

Failure modes

- Reasons for local file system failures :
  1. Media fails where file system is stored.
  2. Corruption of directory structure
  3. Power cable and hard disk cable failure
  4. Disk controller failure
  5. Host adapter failure
- Apart from these reasons, there are many other reasons for remote file systems. Bandwidth and communication also cause the delay.

Consistency semantics

- Consistency semantics is related with file sharing on the network. When does the changes of the original file is reflected to other users. If there is difference in the content of the file; then it creates the problem.
- It deals with the consistency between the views of shared files on a networked system. When one user changes the file, when do other users see the changes of the content ?
- Define when modifications of the file data made by a user are observable by other users
  1. Unix semantics
  2. Session Semantics
  3. Immutable shared-files semantics
  4. Transaction-like semantics
- Difference operating systems consistency semantics is discussed here :

1. UNIX semantics

- Unix file system (UFS) implements :

a. Writes to an open file visible immediately to other users of the same open file.

b. Sharing file pointer to allow multiple users to read and write concurrently

- In UNIX semantics, a file is coupled with a single physical image that is associated as special resource. If there is conflict for single then it causes delays in user processes.
- Centralized system uses UNIX semantics. ' This is common for single processor systems, but difficult to achieve for distributed file systems.

2. Session semantics

- Andrew File System (AFS) implemented complex remote file sharing semantics.

1. Writes to a file by a user is not visible to other users.
2. Once the file is closed, the changes are visible only to new sessions.

- In this semantics, a file can be associated with multiple views. Almost no constraints are imposed on scheduling accesses. No user is delayed in reading or writing their personal copy of the file.
- AFS file systems may be accessible by systems around the world. Access control is maintained through complicated access control lists, which may grant access to the entire world or to specifically named users accessing the files from specifically named remote environments.
- Questions arise about how to handle reads/writes by multiple processes. The file-level transfer model should be used.

3. Immutable-shared-files semantics

- Under this system, when a file is declared as shared by its creator, it becomes immutable and the name cannot be re-used for any other resource. Hence it becomes read-only, and shared access is simple.
- Once a file is declared as shared by its creator, it cannot be modified.
- An immutable file has two key properties. Its name may not be reused and its contents may not be altered.

University Question

1. *With neat sketch explain about the file sharing.* **AU : May-17, Marks 3**

6.6 Protection

- User information and data is stored in the secondary storage. It is necessary to provide security from physical damage and improper access.

6.6.1 Type of Access

- Access is limited or full access of data. User having full access can perform read, write and modify operation on the data and information. In limited access, user only read and executes the information.
- Protection mechanism provides controlled access. Following are some of the operation performed on the files.
  1. Read : User can read a file.
  2. Write : User can rewrite a file.
  3. Delete : User can delete a file whenever space is required.
  4. Execute : User execute a file after loading into the main memory.
  5. List : User check attributes of the file and file/directory names.

6.6.2 Access Control

- Traditionally, a file object in Linux is associated with three sets of permissions. These sets assign read (r), write (w), and execute (x) permissions for the three user groups file owner group, and other.
- Nine bits are used to determine the characteristics of all objects in a Linux le system. Additionally, the set user id, set group id and sticky bits can be set for special cases.
- ACLs can be used for situations where the traditional file permission concept does not suffice. They allow the assignment of permissions to individual users or groups even if these do not correspond to the owner or the owning group. Access Control Lists are a feature of the Linux kernel.
- There are two types of ACLs : Access ACLs and default ACLs. An access ACL is the access control list for a specific file or directory. A default ACL can only be associated with a directory; if a file within the directory does not have an access ACL, it uses the rules of the default ACL for the directory. Default ACL's are optional.
- ACL's can be configured :
  1. Per user 2. Per group
  3. Via the effective rights mask
  4. For users not in the user group for the file

Access determination

- When a process attempts to access a file, its effective UID is compared to the UID that owns the file. If they are the same, access is determined by the ACL's user permissions. Otherwise, if a matching user specific ACL entry exists, permissions are determined by that entry in combination with the ACL mask.
- If no specific entry is available, the filesystem tries to locate a valid group related entry that provides the required access; these entries are processed in conjunction with the ACL mask. If no matching entry can be found, the other entry prevails.

NFSv4 ACL

- The NFSv4 (Network File System - Version 4 protocol introduces a new ACL format that extends other existing ACL formats. NFSv4 ACL is easy to work with and introduces more detailed file security attributes, making NFSv4ACLs more secure.
- NFSv4 ACL's are similar to Windows ACL's in structural perspective. In both the system, ACL stores this entity as a string.

- The Windows and NFSv4 permission model is more granular than the traditional UNIX read-write-execute model.

The main refinements are as follows:

1. NFSv4 distinguishes permissions to create files within a directory from permission to create subdirectories.
2. NFSv4 has a separate append permission bit.
3. NFSv4 has separate read and write permissions for data, file attributes, extended attributes, and ACLs.
4. NFSv4 controls a user's ability to change the ownership of a file through the standard ACL.

- **NFSv4 file permissions**

Code	Verbose name	Permission
r	read_data list_directory	Read sata or list directory contents
w	write_data add_file	Write data or create file
p	append_data add_subdirectory	Append data or create subdirectory
R	read_xattr	Read names attributes
W	write_xattr	Write names attributes
x	execute	Execute as a program
D	delete_child	Delete child within a directory
a	read_attributes	Read non-extended attributes
A	write_attributes	Write non-extended attributes
d	delete	delete
c	read_acl	Read access control list
C	write_acl	Write access control list
o	write_owner	Change ownership
s	synchronize	Force writes to complete synchronously

Access Determination in NFS

- In the POSIX ACL system, the file system attempts to match the user's identity to the single most appropriate access control entry. The access control entry (ACE) then provides a complete set of controlling permissions for the file.

- Each NFSv4 ACE is either an "allow" ACE or a "deny" ACE. When deciding whether to allow a particular operation, the filesystem reads the ACL in order, processing ACEs until either all requested permissions have been allowed or some requested permission has been denied.

ACL inheritance

- ACL inheritance is a mechanism that lets container objects pass access control information to their child objects. A container's child objects can be non-container objects as well as other container objects.
- From an administrator point of view, ACL inheritance simplifies access control management. An administrator can set the ACL on a parent object and if inheritance is enabled, he shouldn't need to set ACLs on each child object.

6.7 File System Structure

- File system is mounted and maintain by the secondary storage which provides by disk. Characteristics of disk are as follows :
  1. Same place is used for reading, writing and modification.
  2. User can access disk directly for any block of information.
- Block is used for data transfer. Each block contains one or more sectors. The I/O transfer between memory and disk are performed on block. Default sector size is 512 bytes.
- Concept of file system is used for the efficient and convenient access to the disk. File system allows data to be stored, located and retrieved easily. The file system is generally composed of many different levels.
- Fig. 6.7.1 shows layers of file system.
- There are six layers. Each layers give support to neighbour layers. It also provides function to above layer and below layer. Each layer uses the features of neighbour layer while creating new features for use by the higher levels.
- File system creates two main design problems :

1. Creation of algorithm and data structure for mapping of logical file system to physical secondary storage devices.
2. File system view for user.

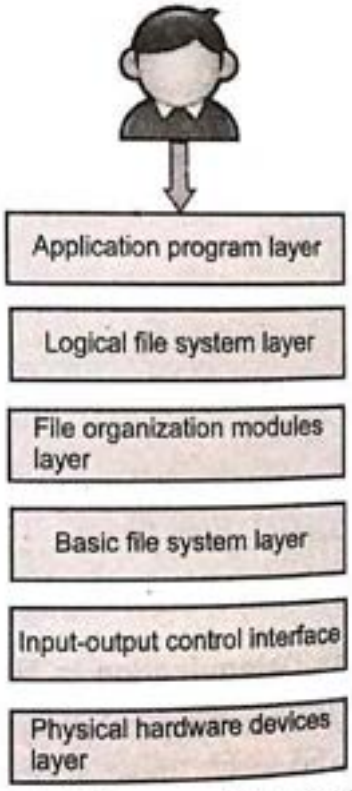


Fig. 6.7.1 Layers of file system

- **Input-output control interface :** It consists of device driver and interrupt handler. Both are used for data transfer between main memory and disk system. Device driver used as a translator. Device driver input is high level commands and output is low level hardware specific instructions.
- **Basic file system layer :** It generates the command for device drivers. Particular device driver read data and writes physical blocks on the disk. Basic file system also manages the memory buffer and caches that hold various file system. Cache memory is used to hold meta-data information of file system.
- **File organization modules layer :** This layer maintain information about files and their logical blocks and physical blocks. If translate the logical block address to physical block address by considering physical location of files and file allocation method. File logical numbers are do not match with physical block containing data, so translation is required. Free space manager is included in file organization modules.
- **Logical file system layer :** It manages metadata information. Actual data is not part of metadata information. But file system structure is part of metadata. Directory structure is used to organized file module. Operating system maintains file control block for each file. UNIX uses inode instead of file control block. File control block stored information about file, location, ownership and access permissions.
- **Application program layer :** User/programmer creates an application programs.
- **Physical hardware devices layer :** It contains actual hardware device link disk, memory etc.
- Following is the list of different operating systems file system :

Operating system	File system formats
Windows XP, NT and 2000	FAT, FAT32, NTFS
UNIX	UNIX File System (UFS)
Standard LINUX	Extended file system

- Google uses its own Google file system.

6.8 File System Implementation

AU : Dec.-15, 16, May-16

- File system is implemented on disk and memory. How to implement the file system, it varies according to operating system and file system. But all the operating system follow some general rules. If the file system is implemented on the disk, it contains the following information :

1. **Boot block control :** Boot block is maintained per volume. The boot block contains the initial bootstrap program used to load the operating system. Operating system requires some information at the time of booting. If the disk is divided into number of partitions, the operating system is stored in the first partition of the disk. If the operating system is not installed on the disk, then this block can be empty. In UNIX file system, this is called the book block and in NTFS, it is the partition boot sector.
2. **Volume control block :** It consists of volume or partitions detail information. The information like block size, number of blocks in the partition, free block count, free block pointer, free FCB count and FCB pointers. In UNIX operating system, each partition is a standalone file system. Superblock is name in UNIX file system. A super block describes the state of the file system : The total size of the partition, the block size, pointers to a list of free blocks, the inode number of the root directory, magic number, etc. In network file system, it is stored in the master file table.
3. **Directory structure :** It is used to organize the files. Directory structure is maintained per file system.
4. **Per-file PCB :** It contains information about files such as file size, file ownership, file permission and location of data blocks. In NTFS, master file table stored this information. Master table file uses a relational database structure.

- In-memory information is used for caching and files system management. Caching improve the performance.

1. **In-memory mount table :** It contains information about each mounted volume.
2. **In-memory directory structure cache :** It contains recently accessed directories information.
3. **System wide open table :** Open files FCB information is stored.
4. **Per-process open file table :** It maintains the pointer to the appropriate entry in the system wide open file tables.

- UNIX operating system treats a directory exactly as a file. Windows NT implements separate system calls for files and directories. It treats directory as entities separate from files. Fig. 6.8.1 shows a file control block.
- FCB specify the information that the system needs to manage a file. Sometimes it is called file attributes. These are highly system

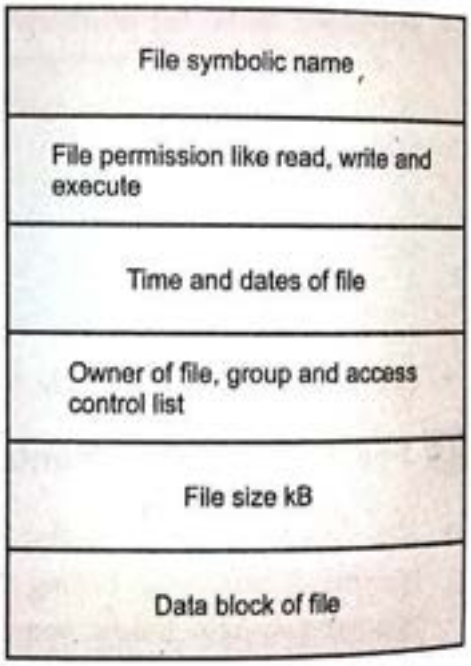


Fig. 6.8.1 File control block

dependent structures. For creating new file, an application program calls the logical file system.

Optimizations for file system :

- Delayed updates of data and metadata are main difficulty. Updates could be delayed in the hope that the same data might be updated in the future or that the updated data might be temporary and might be deleted in the near future. If computer were to crash without having committed the delayed updates, then the consistency of the file system is destroyed.

Metadata updates :

- For recoverable file system after crash, it must be consistent or must be able to be made consistent. So it is necessary to prove that logging metadata updates keeps the file system in a consistent. For inconsistent file system, the metadata must be written incompletely. Sometime, file system data structures is in wrong order. With metadata logging, the writes are made to a sequential log. The complete transaction is written there before it is moved to the file system structures. While updating data, file system crashed, the updates can be completed based on the information in the log. The logging ensures that file system changes are made completely. The order of the changes is guaranteed to be correct because of the sequential writes to the log. If a change was made incompletely to the log, it is discarded, with no changes made to the file system structures.

University Questions

1. *Discuss the functions of files and file implementation.* **AU : Dec.-15, Marks 8**
2. *Explain why logging metadata updates ensures recovery of a file system after a file-system crash.* **AU : May-16, Marks 8**
3. *Discuss how performance optimizations for file systems might result in difficulties in maintaining the consistency of the systems in the event of computer crashes.* **AU : Dec.-16, Marks 8**

6.9 Directory Implementation

- Directory is implemented by using linear list and hash table.

Linear list

- Linear list is simple method of directory implementation. It uses file names with pointers to data block. It is time consuming because of searching overheads.
- To create a new file, file name is searched in the directory to avoid the file name duplication. Then it adds new entry at the end of the directory. To delete a file,

again file name is searched in the directory. Space is released after deleting a file. Directory entry is also removed. Unused space is marked as free entry. Last entry of directory is copied into the free space list.

- For deleting a file, some OS uses linked list. Linked list decreases time required.
- Searching the directory entry is the major disadvantage. It is slow process. Because of this reason, many operating system uses software cache memory. It stores the most recently used directory information.

Hash table

- Hash table is one more data structure used for directory implementation. Hash table takes a value computed from the file name and returns a pointer to the file name in the liner list.
- Hash table reduces the searching time. It uses fixed size block.

6.10 Allocation Methods

AU : Dec.-17, May-18

- The primary issue of file implementing file storage is how to keep track of file blocks. File consists of a collection of blocks.

File allocation

- Following issue are considered for file allocation :

1. After creation of new file, whether the required maximum space for the file is allocated at once or not ?
2. Portion : Space is allocated to a file as one or more contiguous units.
3. Data structure or table used to keep track of the portion which is assigned to a file.

- Three major methods of allocating disk space are in wide use
  1. Contiguous 2. Linked 3. Indexed

6.10.1 Contiguous Allocation

- When user creates a file, system allocates a set of contiguous blocks on disk. This is a pre-allocation method that uses portion of variable size. Contiguous allocation is simple method to implement. It only search free list of correct number of consecutive blocks and mark them as used block.
- Disk address is a linear ordering on the disk. Because of linear ordering, accessing block b + 1 after block b normally requires no head movement. Here head movement is only one track. Fig. 6.10.1 shows contiguous allocation.
- Contiguous allocation of a file is defined by the disk address and the length of the first block.

File allocation table

File name	Start block number	Block length
File 1	1	3
File 2	18	2
File 4	6	6
File 5	20	1
File 6	25	3

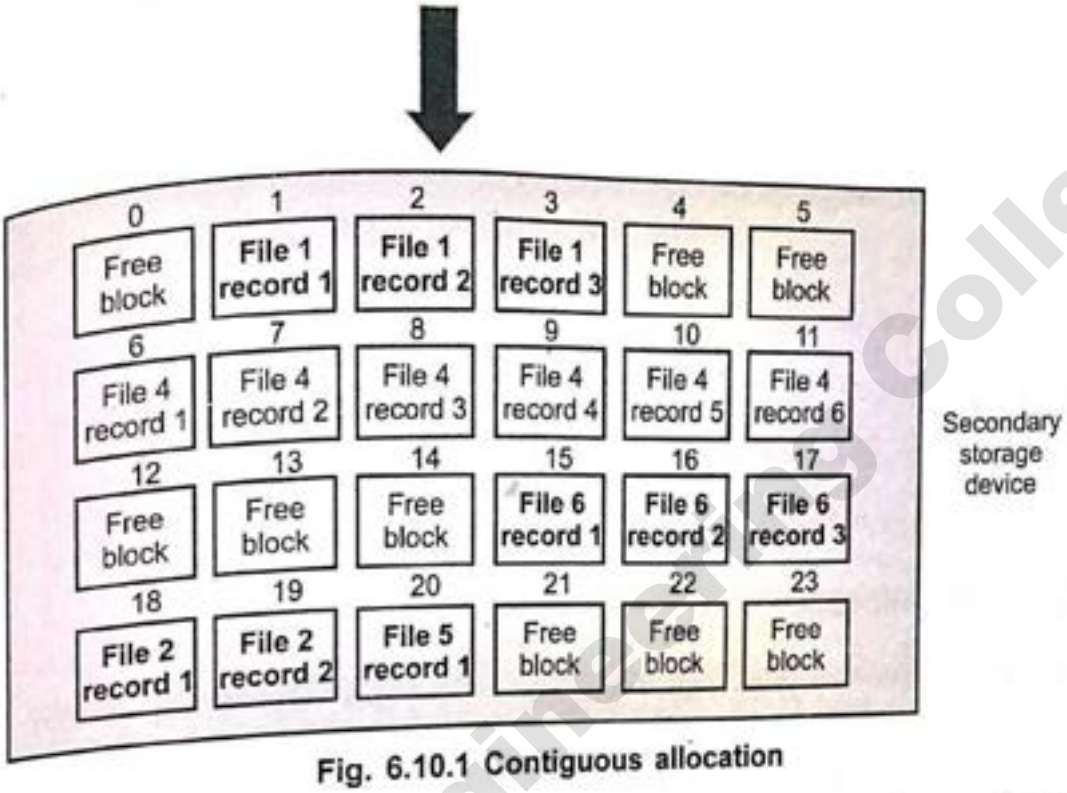


Fig. 6.10.1 Contiguous allocation

- If the file size is "n" blocks and starting location is "L", then it occupies blocks L, L+1, L+2, L+3, ..., L+(n–1). The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.
- Sequential and random access can be supported by contiguous allocation.
- It is easy to retrieve single block. To improve the I/O performance of sequential processing, multiple blocks can be brought in one at a time. It supports sequential access very well because files entire data is stored in adjacent block. This method also supports random access.
- Contiguous allocation also suffers from **external fragmentation.** Small free disk spaces are created after allocation free block and deleting files. External fragmentation means there will require free space available but that is not contiguous. To solve the problem of external fragmentation, compaction method is used.
- One more problem is that how to calculate the space needed for a file. It is difficult to estimate the required space.
- This method is good for storing data on CD-ROM.

Characteristic of contiguous file allocation :

1. It supports variable size portions.
2. Pre-allocation is required.
3. It requires only single entry for a file.
4. Allocation frequency is only once.

Advantages :

1. It supports variable size portion.
2. Easy to retrieve single block.
3. Accessing a file is easy.
4. It provides good performance.

Disadvantages :

1. It suffers from external fragmentation.
2. Pre-allocation is required.

6.10.2 Linked Allocation

- Linked allocation is also called chained allocation. Operating system keeps an ordered list of free blocks. File descriptor stores pointers to the first block and each block stores pointer to the nest block.
- Fig. 6.10.2 shows linked allocation. The disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. No space is lost to disk fragmentation.
- Creation of new file is easy. For new file, simply create new entry in the directory. Reading a file is straightforward. User simple read blocks by following pointers from block to block. There is no external fragmentation with linked allocation.
- To write to file, system finds a free block, and this new block is written to and linked to the end of the file.
- While creating a new file, it is not necessary to declare the size of the file. A file can contiguous to grow as long as free blocks are available.
- Compaction can be used so that blocks of one file are located continuously on the disk. It optimizes disk access.
- File allocation table is an extension of the linked allocation method. Instead of putting the pointers in the file, keep a table of the pointers around. This pointer table can be quickly searched to find ant random block in the file.

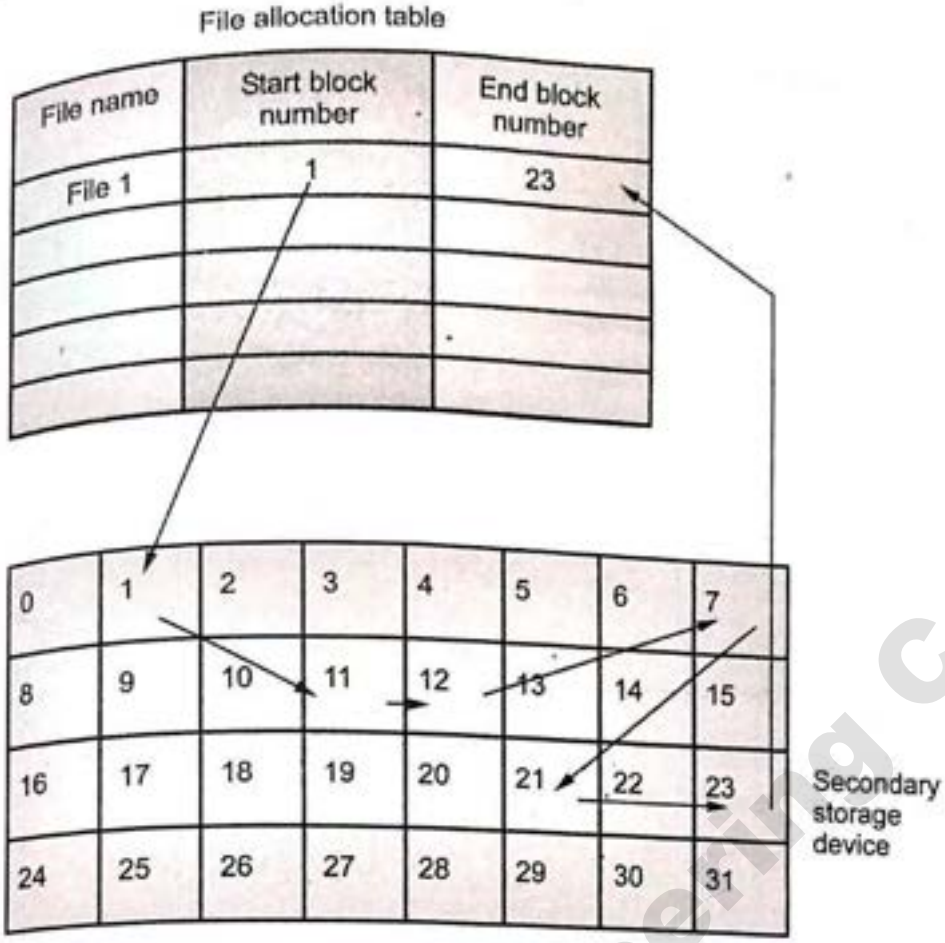


Fig. 6.10.2 Linked allocation

- Fig. 6.10.3 shows the file allocation table. All blocks on the disk must be included in the table. This method is used by windows operating system.
(Refer Fig. 6.10.3 on next page)

Characteristics

1. It supports fixed size portions.
2. Pre-allocation is possible.
3. File allocation table is one entry for a file.
4. Allocation frequency is low to high.

Advantages :

1. There is no external fragmentation.
2. It is never necessary to compact disk space.
3. Pre-allocation is not required.

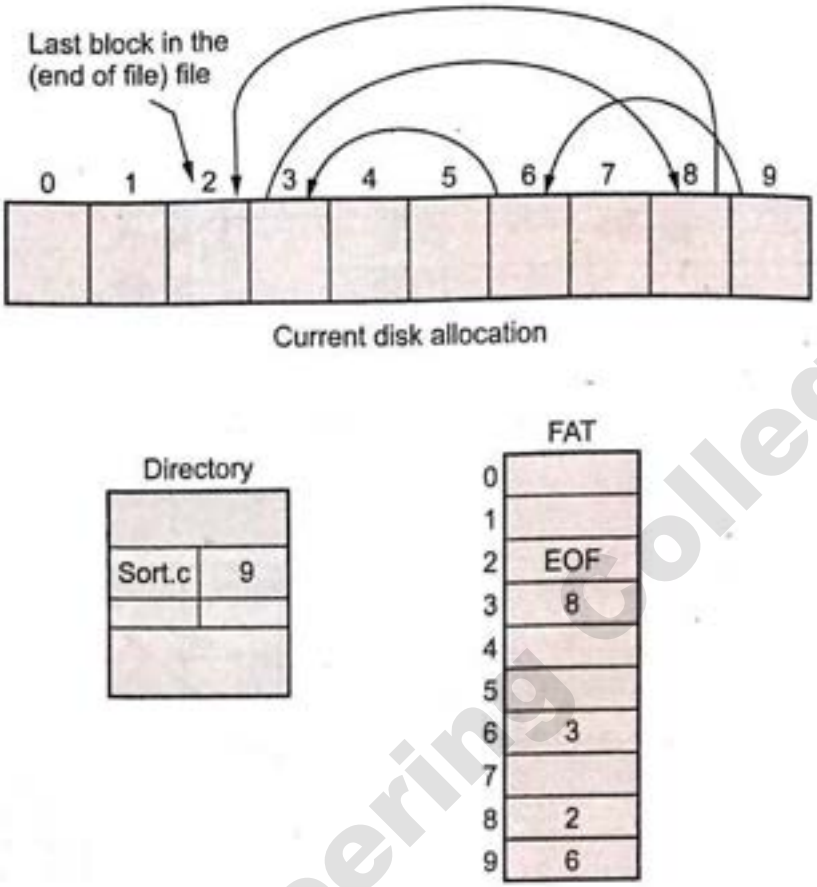


Fig. 6.10.3 FAT

Disadvantages :

1. Files are accessed only sequentially.
2. Space required for pointers.
3. Reliability is not good.
4. Can not support direct access.

6.10.3 Indexed Allocation

- Indexed allocation method solves the problem of both contiguous and linked allocation. It uses concept of index block. Index block stores the entire pointer in one location. But the index block will occupy some space and thus could be considered as an overhead of the method.
- OS keeps a list of free blocks. It allocates an array to hold pointers to all the blocks used by the file. It allocates blocks only on demand. Fig. 6.12.4 shows indexed allocation.

File allocation table

File name	Index block number
File 1	10
File 4	24

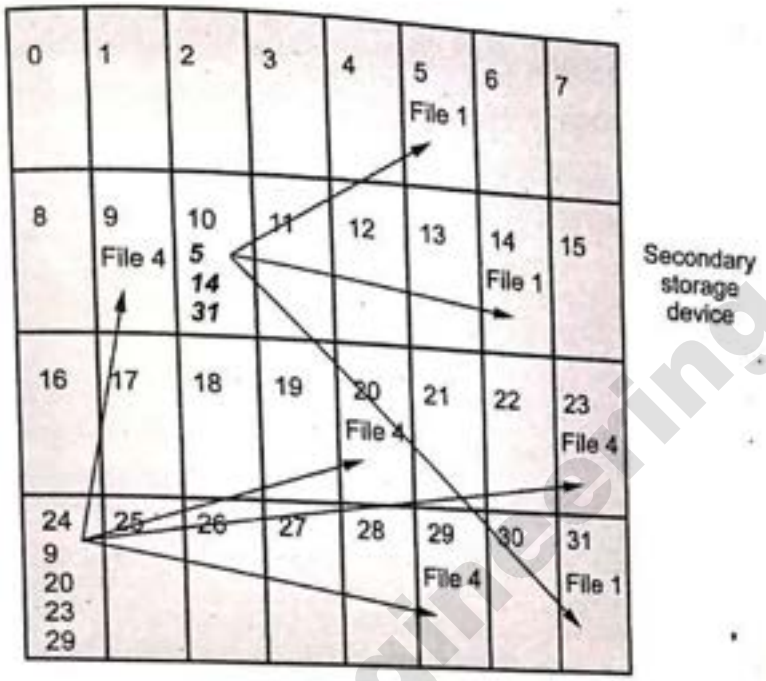


Fig. 6.10.4 Indexed allocation

- In indexed allocation, each file maintains its own index block. It contains an array of disk sector of addresses. For example : The n^{th} entry in the index block points to the n^{th} sector of the file. The directory contains the address of the index block of a file. To read the n^{th} sector of the file, the pointer in the n^{th} index block entry is read to find the desired sector.
- It supports direct access and without suffering from external fragmentation. Any free block anywhere on the disk may satisfy a request for more space.
- Indexed allocation does suffer from wasted space. The pointer overhead of the index block is generally greater than the pointer overhead of linked allocation.

Advantages :

1. It supports sequential and direct access.
2. No external fragmentation.
3. Faster than other two methods.
4. It supports fixed and variable size blocks.

Disadvantages :

1. Indexed allocation does suffer wasted space.
2. Pointer overhead is generally greater.

6.10.4 Comparison of Contiguous, Linked and Indexed File Allocation

Sr. No.	Contiguous	linked	Indexed
1	Allocate each file to contiguous blocks on disk	Allocate linked-list of fixed-sized blocks	Allocate fixed-sized blocks for each file
2	It suffers from external fragmentation	No external fragmentation	No external fragmentation
3	Pre-allocation is required	Pre-allocation is possible	Pre-allocation is possible
4	It support variable size portions	It support fixed size portions.	It supports fixed and variable size portions.
5	Simple to calculate random addresses	Cannot calculate random addresses without reading previous blocks	Supports random access
6	Allocation frequency is once	Allocation frequency is low to high	Allocation frequency is medium
7	No pointer overhead	Space required for pointer	Pointer overhead is generally greater

University Questions

1. *What are different allocation methods in disk storage ? Explain with neat sketch.*

AU : Dec.-17, Marks 8

2. *What are the various disk space allocation methods. Explain any two in detail.*

AU : May-18, Marks 13

6.11 Free Space Management

AU : Dec.-15

- Space that is allocated to files must be managed and space which is not allocated to any file must be managed. To keep track of free disk space, the system maintains a free space list. File allocation table and disk allocation both are required to manage free space properly.
- When a file is deleted, its disk space is added to the free-space list.
- Following are the techniques used for free space management :
  1. Bit tables or bit vector
  2. Chained free portions or linked list

3. Indexing or grouping
4. Free block list or counting

Bit tables or bit vector

- Each block on the disk is represented by bit. It uses vector chain for blocks. Free block is represented by o and used block represented by 1.
- For example, consider a disk where blocks 3, 4, 5, 7, 9, 14, 15, 19, 30, 31, 32, 35, 36, 37, 38 are free blocks and rest of the blocks are allocated. The free space is shown below :

  11100010101111001110111111111000110000

- The memory required for a block bitmap = Disk size / 8 × (block size of file system)
- Bit map requires extra space. For example :

  Block size = 2^{12} bytes

  Disk size = 2^{30} bytes

  $$\text{Block number (n)} = \frac{\text{Block size}}{\text{Disk size}} = \frac{230 \text{ bytes}}{212 \text{ bytes}} = 2^{18} \text{ bytes}$$

- When bit table is stored into the memory, then exhaustive search of the table can slow file system performance. Most of the file system use auxiliary data structure for bit table. File system also maintain summary table. Summary table contains sub-range, number of free blocks and maximum sized contiguous number of free blocks.
- Summary table is used to store information about contiguous free blocks. Whenever file system needs a number of contiguous blocks, it can scan the summary table to find an appropriate sub-range and then search that sub-range.
- This method is used in Apple Macintosh.

Advantage :

a. Easy to find a free blocks

b. It is as small as possible.

Disadvantage :

It may not be feasible to keep the bitmap in memory for large disks.

2. Link list

- In this method, all free space disk blocks are linked, keeping a pointer to the first free block. All file allocation methods used link list free space techniques.

- There is small space overhead because there is no need for a disk allocation table. In the above example, free blocks are 3, 4, 5, 7, 9, 14, 15, 19, 30, 31, 32, 35, 36, 37, 38. Here free block pointer is on block number 3. Block 3 contains pointer to block 4, block 4 contains pointer to block 5 and block 5 contains pointer to block 7 and so on.
- This method is not efficient because to reach free block 9, we have to traverse block 3, 4, 5 and 7 then we will reach to block 9.
- Disk will become quite fragmented after some use. Many portions will be a single block long.

3. Grouping

- First free block contains the addresses of n free blocks. The first n–1 of these blocks is actually free. The last block also contains the address of another n free block.
- Consider the free blocks : 3, 4, 5, 7, 9, 14, 15, 19, 30, 31, 32, 35, 36, 37, 38
  Then
- When all the blocks in the group have been allocated, then use the block that held the pointer.
- Because of grouping method, address of a large number of free blocks can be found quickly.

4. Counting

- It keeps the address of the first free block and the number *n* of free contiguous blocks that follow the first block. Each entry in the free space list then consists of a disk address and a count.

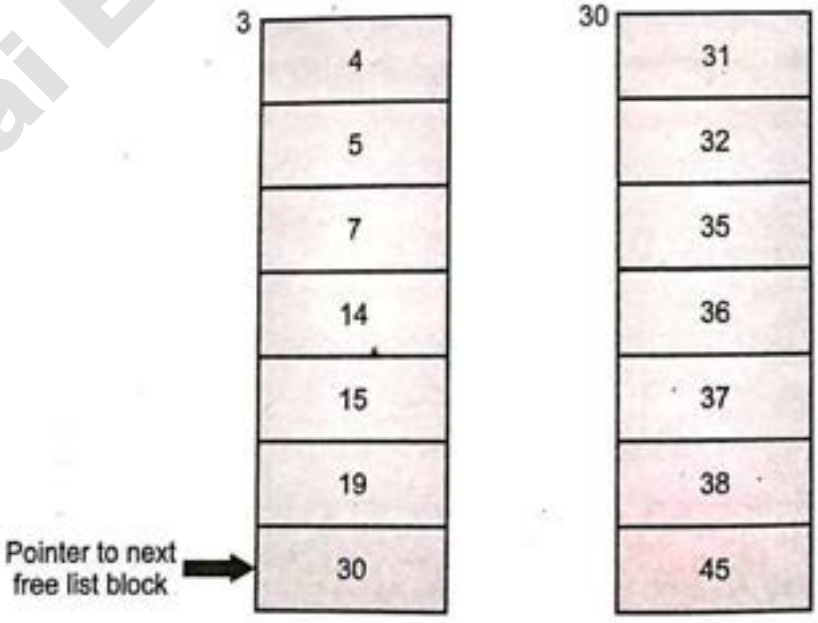


Fig. 6.11.1

University Question

1. *Explain free space management with neat example.* **AU : Dec.-15, Marks 8**

6.12 Efficiency and Performance

- To read a file may require many head seek movements and consequently take a much longer time than if its blocks were written one after another on the disk.
- Directories are searched as lists so that the average time to find a directory entry initially increases in direct proportion to the total number of entries. The blocks that a directory uses to store its entries are referenced from its inode. Searching for a directory entry therefore becomes slower when indirect blocks have to be accessed.
- For large numbers of files the only way to efficiently scale to support large numbers of files is to dynamically allocate index space for files.
- Many traditional file systems require a checking program to check file system consistency after a crash. On large, active file systems this type of checking can take a prohibitively long time to complete. Solving this problem must not degrade I/O performance.

6.12.1 Efficiency

- Disk allocation and directory algorithm are the main factor for efficient use of secondary storage disk. System efficiency and performance is depending upon the disk accessing method.
- UNIX operating system pre-allocates inode so it occupies space even before any files are created. It also distributes inode across the disk and tries to store data files near their inode, to reduce the distance of disk seeks between the inode and the data.
- Depending on the file size, some operating systems use variable size clusters. The large data that is stored in a directory so the more often the directory blocks have to be re-written. In year 1990, Sun implemented a new algorithm called clustering that allows for more extent-like behavior by gathering up to 56 kB of data in memory to allocate disk space contiguously. Sun also extended the maximum file size to 1 TB in Solaris 2.6.
- Kernel table sizes used to be fixed and could only be changed by rebuilding the kernels. Modern tables are dynamically allocated, but that requires more complicated algorithms for accessing them.

6.12.2 Performance

- Performance should not degrade as the size of the file system, an individual file or the total number of files stored grows.
- File system performance is often a major component of overall system performance and is heavily dependent on the nature of the application generating the load. To achieve optimal performance, the underlying file system configuration must be balanced to match the application characteristics. Fig. 6.12.1 shows file system caching via the page cache.

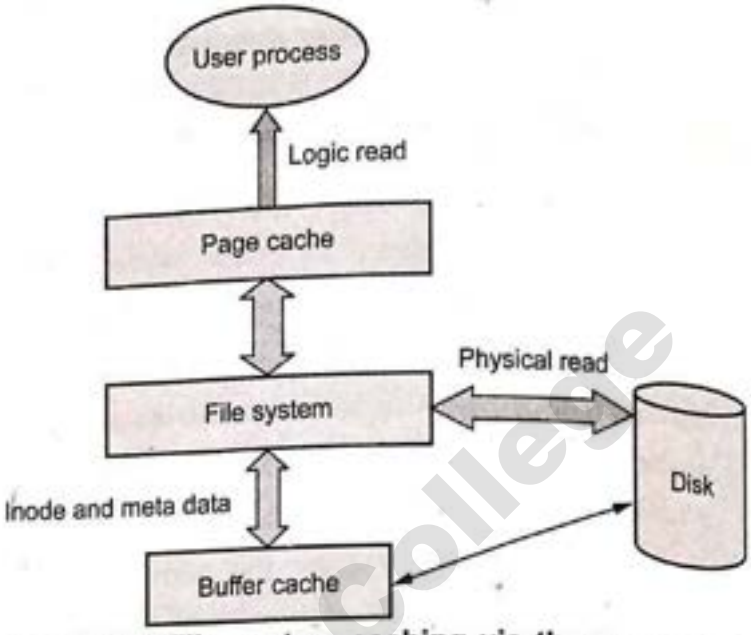


Fig. 6.12.1 File system caching via the page cache

- Traditional UNIX implements file system caching in the I/O subsystem by keeping copies of recently read or written blocks in a block cache. This block cache sits just above the disks and caches data corresponding to physical disk sectors.
- The file system is still involved in the page cache lookup, but the amount of work the file system needs to do is dramatically simplified. The page cache is implemented in the virtual memory system. In fact, the virtual memory system is architected around the page cache principle, and each page of the physical memory is identified the same way, by file and offset. Pages associated with files point to regular files, while pages of memory associated with the private memory space of processes point to the swap device.

6.13 Recovery

- File system have varying methods to deal with coruption, depending on the file system data structure and algorithm.

6.13.1 Log - Structured File Systems

- Rosenblum and Ousterhout introduced new type of file system i.e. log structured file systems. To improve write performance by buffering a sequence of file system changes to disk sequentially in a single disk write operation. Logs written include

all file system information, including file data, file inode, directory data, directory inode.

- When data is writing to disk, LFS stores all updates in in-memory segment. Data is written to disk when segment is full. Unused part of disk is used for writing the disk. LFS never overwrites existing data but it search free space and write to that position.
- New data is written sequentially in the log files free space. Writing to disk sequentially is not enough. Because in between first write and second write, disk may rotate from its position. To solve this problem, LFS uses write buffering.
- Before writing to the disk, LFS keeps track of updates in memory; when it has received a sufficient number of updates, it writes them to disk all at once, thus ensuring efficient use of the disk. When writing to disk, LFS buffers updates in an in-memory segment and then writes the segment all at once to the disk. As long as the segment is large enough, these writes will be efficient.

Two Marks Questions with Answers

Q.1 What are the essential requirements for long term information storage ?

Ans. : Name, type, location, size, protection and time, date for file.

Q.2 State the typical bad sector transactions. **AU : May-18**

Ans. :
a. O.S. tries to read logical block 87.
b. The controller calculates the ECC and finds that the sector is bad. It reports this finding to the O.S.
c. Next time the system is rebooted, a special command is run to tell the SCSI controller to replace the bad sector with a spare.
d. After that, whenever the system requests logical block 87, the request is translated into the replacement sectors address by the controller.

Q.3 What are link and unlink directory operations ?

Ans. : **Link :** Links a file to a new name in the file system directory structure, creating a new directory entry for an existing node.

Unlink : It removes a directory entry for a file.

Q.4 What data type is file ?

Ans. : A file is an abstract data type.

Q.5 Give an example of an application that could benefit from operating system support for random access to indexed files.

Ans : An application that maintains a database of entries could benefit from such support. For instance, if a program is maintaining a student database, then accesses to

the database cannot be modeled by any predetermined access pattern. The accesses to records are random and locating the records would be more efficient if the operating system were to provide some form of tree based index.

Q.6 What is the content of a typical file control block ? AU : CSE/IT, May-11

Ans. : FCB contains in formation about the file, including ownership, permissions and location of the file contents.

Q.7 What are the functions of file organization module in file system ?

Ans. : File organization module can translate logical block addresses to physical block addresses for the basic file system to transfer.

Q.8 Discuss about file descriptor and access control matrix.

Ans. : Protection mechanisms provide controlled access by limiting the types of file access that can be made. Protection can be viewed abstractly as a matrix, called an **access control matrix**. A **file descriptor** is an index into a small table of open files for the process. Descriptors start at 0 and seldom get higher than 6 or 7 for typical programs, depending on the maximum number of simultaneously open files.

Q.9 What is a file ? List some operations on it. AU : CSE/IT : Dec.-10

Ans. : File is an unstructured sequence of data, File operations are read, write, create, delete etc.

Q.10 What are the various file accessing methods ? AU : CSE/IT : Dec.-10

Ans. : Sequential access, direct, indexed etc.

Q.11 Write the attributes of a file. AU : CSE/IT : May-11

Ans. : Name, Identifier, Type, Location, Size, Time and date.

Q.12 Mention any four file attributes. AU : CSE/IT : Dec.-11

Ans. : Name, Type, Size and Location.

Q.13 Differentiate absolute path from relative path. AU : CSE/IT : Dec.-11

Ans. : An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name defines a path from the current directory.

Q.14 A direct or sequential access has a fixed file-size S-byte record. At what logical location, the first byte of record N will start ? (Refer section 6.2) AU : CSE/IT : Dec.-11

Q.15 What are the two types of system directories ? AU : CSE/IT : May-12

Ans. : A tree structured directory that allows a user to create subdirectories to organize the files.

A general graph structure that allows complete flexibility in the sharing of files and directories.

Q.16 What is garbage collection ? AU : CSE/IT : May-12

Ans. : Garbage collection is the process of automatically freeing objects that are no longer referenced by the program.

Q.17 Mention the major attributes and operations of a file. AU : CSE/IT : Dec.-12

Ans. : Major attributes of file is name, identification, type, location, size and protection. Operation on a file is to create, write, read, reposition, delete and truncate files.

Q.18 What is meant by free-space management ? AU : CSE/IT : Dec.-12

Ans. : To keep track of free disk space, the system maintains a free space list. Free space list. Free space list records all free disks blocks i.e. those not allocated to some file or directory.

Q.19 What are the responsibilities of file manager ? AU : CSE/IT : May-13

Ans. : File manager is responsible for the maintenance of secondary storage. The file manager also provides a logical way for users to organize files in secondary storage.

Q.20 Name any four common file types.

Ans. : File types are text file, source file, object file and executable file.

Q.21 What is NFS ?

Ans. : The Network File System (NFS) is probably the most prominent network service using RPC. It allows you to access files on remote hosts in exactly the same way you would access local files. A mixture of kernel support and user-space daemons on the client side, along with an NFS server on the server side, makes this possible. This file access is completely transparent to the client and works across a variety of server and host architectures.

Q.22 What is virtual file system ?

Ans. : Virtual File Systems (VFS) provide an object-oriented way of implementing file systems. VFS allows the same system call interface to be used for different types of file systems.

Q.23 What is meant by mounting a file ?

Ans. : Mounting a filesystem, means taking that storage and connecting it to the operating system in a way that it's usable as a hierarchical storage device with directories and files. This could be the initial file system or another file system that connects to the root filesystem at a mount point.

Q.24 What is file management system ?

Ans. : File management system consists of system utility programs that run as privileged applications. The way a user or application may access files and programmer does not need to develop file management software.

Q.25 What are the disadvantages of log structured file systems ?

Ans. : It requires cleaning demon to produce clean space, which takes additional CPU time. Reads that are not handled by buffer cache are same performance as normal file system.

Q.26 Enlist different types of directory structure. AU : Dec.-17

Ans. : The most common schemes for defining the logical structure of a directory are :

a. Single-level directory
b. Two-level directory
c. Tree-structured directories
d. Acyclic-graph directories
e. General graph directory

Q.27 List the objectives for a file management system.

Ans. : Objectives are :

1. Meet the data management needs and requirements of the user
2. Guarantee that the data in the file are valid
3. Optimize performance
4. Provide I/O support for a variety of storage device types
5. Provide I/O support for multiple users.

Q.28 What are the various methods for free space management ?

Ans. : Bit vector , linked list, grouping and counting are various method for free space management.

Q.29 List the different operations that can be performed on a file. AU : CSE/IT : May-13

Ans. : Operations

1. Create a file
2. Writing a file
3. Deleting a file
4. Reading a file
5. Repositioning within a file.

Q.30 What is a file management system ?

Ans. : It is a set of system software that provide services to user and applications in the use of files.

Q.31 What are the disadvantages of log-structured file systems ?

Ans. : It decreases head contention and seek times.

Q.32 Define log structured file.

Ans. : Log structured file is used for file system meta data updates.

Q.33 List out the major attributes and operations of a file. AU : CSE/IT : May-15

Ans. : Attributes of file : Name, identifier, type, location and size.

Operation on file : read, write, create, delete.

Q.34 Do FAT file system is advantageous ? Why ? AU : CSE/IT : May-15

Ans. : The FAT file system protects files by storing two copies of the file allocation table on the FAT volume. FAT contains the FAT structure, which is a map of the data region. File size is no longer fixed. We do not need to know the size of the file at the beginning.

Q.35 Identify the two important functions of Virtual File System (VFS) layer in the concept of file system implementation. AU : Dec.-15

Ans. : Two important functions :

1. It separates file system generic operations from their implementation by defining a clean VFS interface.
2. VFS provides a mechanism for uniquely representing a file throughout a network.

Q.36 Differentiate between file and directory. AU : CSE : May-17

Ans. : The basic difference between the two is that files store data, while directory store files and other directory. File is a sequence of logical records. Directory lists the file by name and includes the file location on the disk, length, type etc.

Q.37 Write about swapping. Let us assume the user process is of size 1 MB and the backing store is a standard hard disk with a transfer rate of 6 MBPS. Calculate the transfer rate. AU : Dec.-17

Ans. :

$$\text{Transfer (time) rate} = \frac{\text{Process size}}{\text{Hard disk transfer rate}} = \frac{1000\text{ kB}}{5000\text{ kB}} = 200\text{ ms}$$

Q.38 What is the advantages of bit vector approach in free space management ? AU : May-18

Ans. : Advantages : It is simple and its efficiency in finding the first free block on the disk is high.

❑❑❑

UNIT - V

7 Linux Case Study

Syllabus

Linux System - Design Principles, Kernel Modules, Process Management, Scheduling, Memory Management, Input-Output Management, File System, Inter-process Communication

Contents

7.1 Introduction to Linux

AU : Dec.-15,17, May-16, 17

- Linux is developed in 1991 by Linus Torvalds. It is used in most of the computers, ranging from super computers to embedded system. Linux is a free operating system based on UNIX standards.
- Linux is multi user, multi tasking, time sharing and monolithic kernel etc.
- Linux is Free Open Source Software. Free means liberty and not related to price or cost and Open means source code is available and anybody can contribute to the development.
- Fig. 7.1.1 shows shell and kernel.
- The core Linux operating system kernel is entirely original, but it can run much existing free UNIX software, resulting in an entire UNIX-compatible operating system free from proprietary code.
- **Kernel** is a main program of UNIX system. It controls hardwares, CPU, memory, hard disk, network card etc.

Fig. 7.1.1 Shell and kernel

- Shell is an interface between user and kernel. Shell interprets your input as commands and passes them to kernel.

Design principles

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools.
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model.
- Main design goals are speed, efficiency and standardization.
- Linux is designed to be compliant with the relevant POSIX documents; at least two Linux distributions have achieved official POSIX certification.
- The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behaviour.

7.1.1 Components of Linux System

- Linux operating system consists of three components :
  1. Kernel 2. System library 3. System utility

Fig 7.1.2 shows Linux operating system components.

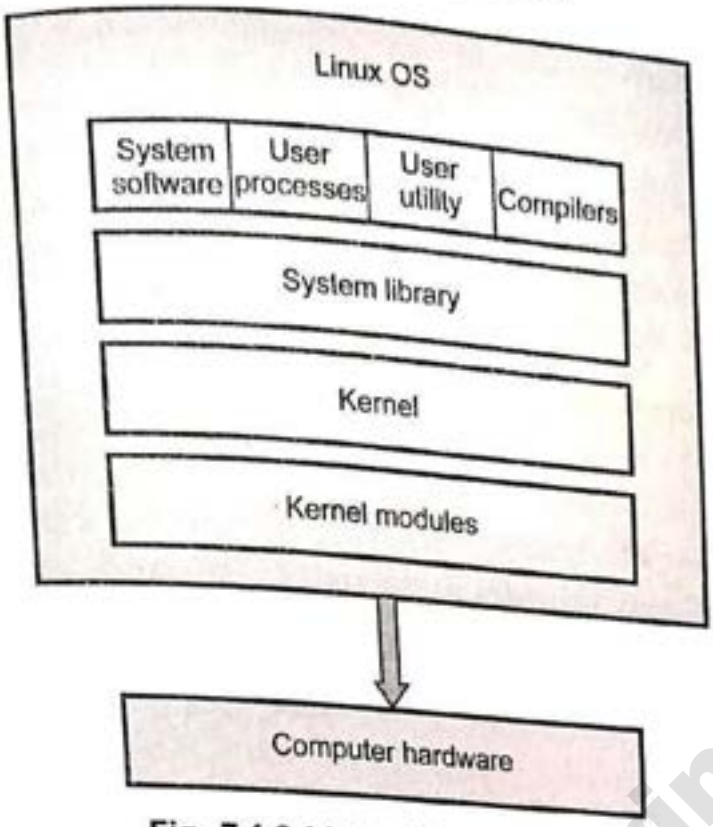


Fig. 7.1.2 Linux OS components

1. **Kernel :** Kernel is the heart of the Linux operating system. It is responsible for all major activities of Linux OS. It provides important abstraction to hide low level hardware information to user or application program. Kernel is combination of various modules and it interacts directly with the underlying hardware. The Linux kernel consists of several important parts : Process management, memory management, hardware device drivers, file system drivers, network management etc.

2. **System Library :** System libraries are special functions or programs. Libraries are used for accessing kernel's features. These libraries implements most of the functionalities of the operating system and do not requires kernel module's code access rights.

3. **System Utility :** System utility programs are responsible to do specialized, individual level tasks.

- Privileged mode is called kernel mode in Linux system. All the kernel code executes in the privileged mode with full access to the hardware resource of the computer. In Linux operating system, no user mode code is built into the kernel.

7.1.2 Linux Utilities

- From a certain point of view, Linux can be seen primarily as a set of utilities bonded together by a common kernel.

- Linux utilities come in two basic flavors : Open source and commercial. Some commercial products, such as RealPlayer, are available for free, while others, such as the BRU tape backup software, are available for a fee. The majority of Linux utilities are free, open-source packages.
- Traditional UNIX utilities evolved in the command-line environment. Because of that they can be joined together to form powerful shell scripts and time based jobs. Some of the newer, graphical utilities are designed to be clicked and pointed at in the X Window System.

Internal and External Commands :

- Linux commands are nothing but the text written in the shell or terminal that the terminal understands. Linux commands usually perform certain specific operations such as editing a text file, making, removing and moving a directory etc. The shell or the terminal of Linux provides a use set of commands and allows us to write the script that is able to perform various operations.
- Commands in Linux are usually divided into two sections internal commands and the external commands.

1. **External commands :** These are the most commonly used utilities and programs such as **cat, ls** and so forth. External commands are the commands that are executed by the kernel. These commands will have a process id running for it.
2. **Internal commands :** The shell has a number of built-in commands such as **cd** and **echo** which don't generate a process. Internal commands are the commands that are executed directly by the shell. These commands will not have a separate process running for each.

7.1.3 Kernel Module

- Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. These loadable kernel modules run in privileged kernel mode.
- Kernel module can implement a device driver, a file system or a networking protocol.
- The module interface allows third parties to write and distribute, on their own terms, device drivers or file systems that could not be distributed under the GPL.
- Kernel modules allow a Linux system to be set up with a standard, minimal kernel, without any extra device drivers built in.
- The Linux kernel is modular, which means it can extend its capabilities through the use of dynamically-loaded kernel modules. A kernel module can provide a device driver which adds support for new hardware; or support for a file system.

- On modern systems, kernel modules are automatically loaded by various mechanisms when the conditions call for it. However, there are occasions when it is necessary to load and/or unload modules manually, such as when a module provides optional functionality, one module should be preferred over another although either could provide basic functionality or when a module is misbehaving, among other situations.
- Kernel modules allow a Linux system to be set up with a standard minimal kernel, without any extra device drivers built in.
- The module support following Linux components :
  1. Module-management system
  2. Module loader and unloader
  3. Driver-registration system
  4. Conflict-resolution mechanism

Module-management system

- It allows modules to be loaded into main memory and to communicate with the rest of the kernel.
- Internal symbol table is stored in the kernel by Linux. This symbol table contains only explicitly exported entries. The set of exported symbols constitutes a well-defined interface by which a module can interact with the kernel.
- User can use these symbols by using an explicit request. Although exporting symbols from a kernel function requires an explicit request by the programmer.
- The loading of the module is performed in two steps :
  1. The kernel must reserve a continuous area of virtual kernel memory for the module. Address of the allocated memory is return by kernel. Loader uses this address for loading machine code.
  2. Module passes the system call and symbol table to kernel.

Driver Registration

- Kernel also maintains dynamic tables of all known drivers. At any time, these drivers are removed or added by using routines. These routines are responsible for registering the module's functionality.
- Allows modules to tell the rest of the kernel that a new driver has become available.
- Registration tables include following :
  1. Device drivers : These drivers include character devices, block devices and network interface devices.
  2. File systems : It contains format for storing files on a disk and other required information.

3. Network protocols : Module may implement all required networking protocols.
4. Binary format : This format specifies a way of recognizing, loading and executing a new type of executable file.

Conflict Resolution

- A mechanism that allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.
- The conflict resolution module aims to :
  1. Prevent modules from clashing over access to hardware resources.
  2. Prevent auto probes from interfering with existing device drivers.
  3. Resolve conflicts with multiple drivers trying to access the same hardware.

University Questions

1. *Write about Linux architecture and LINUX kernel with neat sketch.* **AU : Dec.-15, Marks 16**
2. *What are the primary goals of the conflict-resolution mechanism used by the Linux kernel for loading kernel modules ?* **AU : May-16, Marks 8**
3. *Write short note on LINUX kernel with neat sketch.* **AU : May-17, Marks 7**
4. *Explain the components of Linux system with neat sketch.* **AU : Dec.-17, Marks 6**

7.2 Process Management

AU : Dec.-16

- In Linux, tasks represent both processes and threads. Linux threads are really kernel threads.
- To manage multi-tasking, the OS needs to use a data structure which can keep track of every task's progress and usage of the computer's available resources. Such a data structure is called a 'process descriptor' and every active task needs one.
- Every task needs its own 'private' stack. So every task in addition to having its own code and data will also have a stack area that is located in user space plus another stack area that is located in kernel space.
- Each task also has a process descriptor which is accessible only in kernel space.
- The **task_struct** is used to represent a task.

1. **State :** Process execution states are executing, ready, suspended, stopped, zombie.
2. **Scheduling information :** Linux uses this information for scheduling processes.
3. **Identifiers :** Each process has a unique process identifier and also has user and group identifiers.

4. **IPC :** Linux supports the IPC mechanisms are pipes, shared memory, socket etc.

5. **Links :** In a Linux system no process is independent of any other process. Every process in the system, except the initial process has a parent process. New processes are not created, they are copied or rather cloned from previous processes. Every task_struct representing a process keeps pointers to its parent process and to its siblings as well as to its own child processes.

6. **Times and timers :** The kernel keeps track of processes creation time as well as the CPU time that it consumes during its lifetime. The kernel updates the amount of time.

7. **File system :** Processes can open and close files as they wish and the processes task_struct contains pointers to descriptors for each open file.

8. **Virtual memory :** Most processes have some virtual memory and the Linux kernel must track how that virtual memory is mapped onto the system's physical memory.

9. **Processor specific context :** When a process is suspended, all of that CPU specific context must be saved in the task_struct for the process. When a process is restarted by the scheduler its context is restored from here.

- Fig. 7.2.1 shows task_struct.

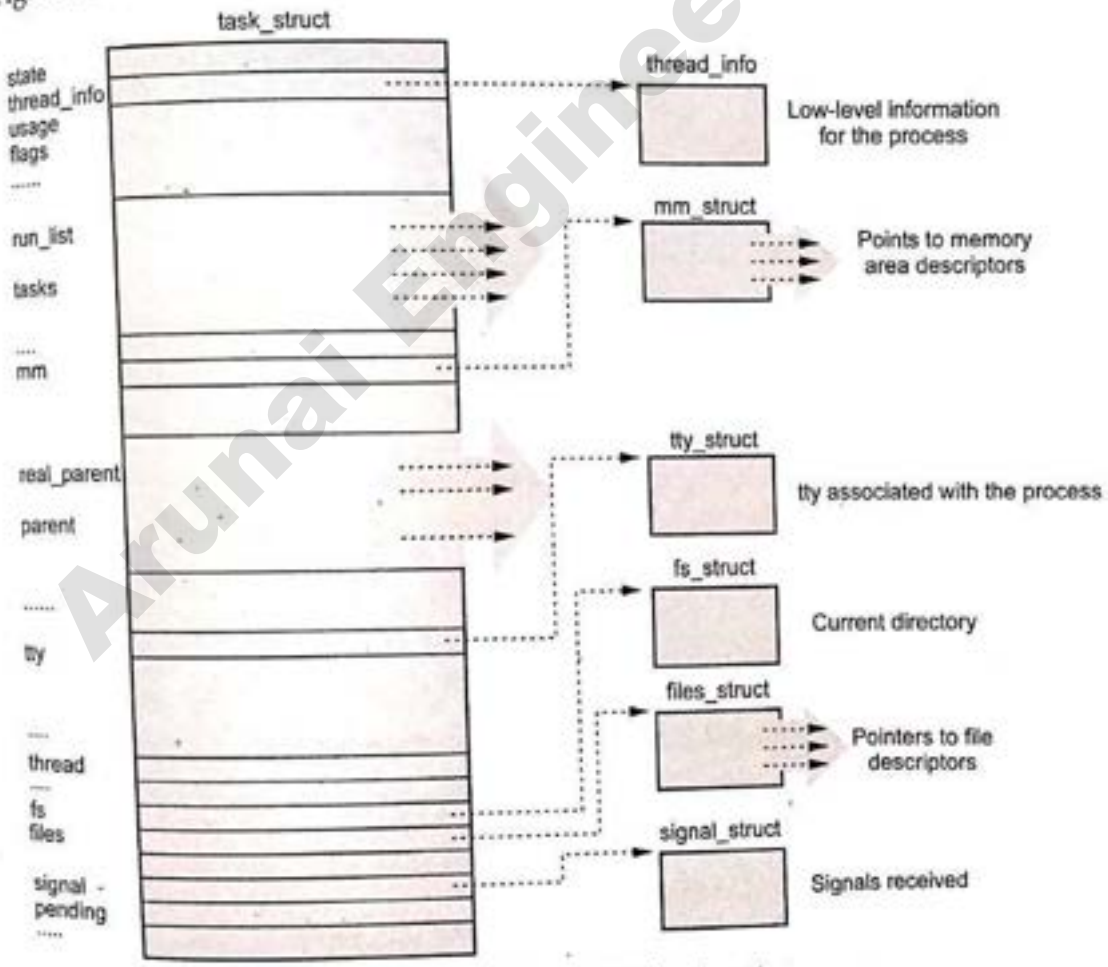


Fig. 7.2.1 Linux task_struct

1. Linux process

- Process in Linux is represented by task_struct data structure. It contain various information.

State	Various state of process execution : Ready, executing, suspended, zombie and stopped.
IPC	Linux support IPC mechanism
Scheduling information	Information needed by Linux to schedule processes. Process can be real time and a priority.
Links	Each process contains link to its parent and to its all children process.
Identifier	Each process has a unique process identifier (PID). It also user and group ID.
Times	It includes process creation time and time consumed by process.
File system	Includes pointers to any files opened by this process.

- Fig. 7.2.2 shows a process state transition diagram.

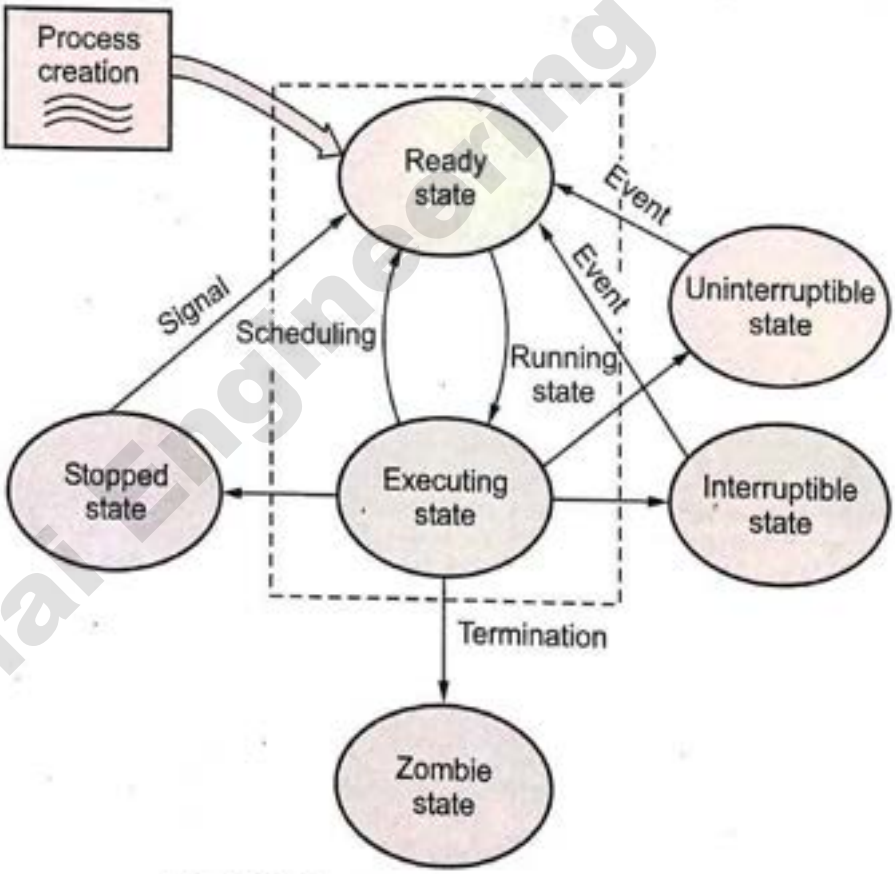


Fig. 7.2.2 Process/Thread state model

1. **Running :** Running state is two types : Executing or ready to execute.
2. **Interruptible :** It is blocked state of process. The process is waiting for an event or signal from another process.

3. **Uninterruptible :** It is also blocked state. Here process is waiting directly on hardware condition. Process can not handle any signals in this state.
4. **Stopped :** Process has been halted and can only resume by positive action from another process.
5. **Zombie :** Process has been terminated for some reason.

Linux threads

- Modern unix systems support for multiple Kernel level thread per process. Older version of Linux Kernel does not support multithreading.
- Pthread library is the set of user level library function and it mapps single Kernel level process.
- Linux does not recognize a distinction between thread and processes.
- Multiple user level threads that constitute a single user level process are mapped into Linux Kernel level processes that share the same group ID.
- New process is created in Linux by coping the attributes of the current process. New process shares its resources such as signal handlers, files and virtual memory.
- If two processes share the same virtual memory, they act like a threads within in a single process.
- Linux uses clone ( ) command for creating processes.

7.2.1 Completely Fair Scheduler

AU : Dec.-16, 17

- Linux kernel version 2.6.23, a new approach has been taken to the scheduling to runnable processes. The Completely Fair Scheduler (CFS) becomes the default Linux kernel scheduler.
- Completely Fair Scheduler does allocation of CPU resources fairly without compromising interactivity performance. Also CFS offers user fair scheduling, group scheduling and modular scheduler framework.
- CFS tries to assure that each process obtains its fair share of the processor time.
- Model process scheduling as if the system had an ideal, perfectly multitasking processor. In such a system, each process would receive 1/n of the processor's time, where n is the number of runnable processes, and we should schedule them for infinitely small durations, so that in any measurable period we'd have run all n processes for the same amount of time.
- CFS will run each process for some amount of time, round-robin, selecting next the process that has run the least. Rather than assign each process a time slice,

CFS calculates how long a process should run as a function of the total number of runnable processes. Instead of using the nice value to calculate a time slice, CFS uses the nice value to weight the proportion of processor a process is to receive.

- The CFS tries to keep track of the fair share of the CPU that would have been available to each process in the system. So, CFS runs a fair clock at a fraction of real CPU clock speed.
- The fair clock's rate of increase is calculated by dividing the wall time by the total number of processor waiting. The resulting value is the amount of CPU time to which each process is entitled.
- As a process waits for the CPU, the scheduler tracks the amount of time it would have used on the ideal processor.
- The wait time is represented by per-task wait_runtime. This is used to rank the processes for scheduling and to determine the amount of time the process is allowed to execute before being preempted.
- Scheduler selects the process with longest wait time and assigned to the CPU. Wait time decreases when process is running.
- At the same time, time of other waiting tasks increases. After some time, there will be another task with largest wait time and the currently running task will be preempted.
- Using this principle, CFS tries to be fair to all tasks and always tries to have a system with zero wait time for each process.

7.2.2 Linux Process Scheduling Policy

- Linux uses a timesharing technique. We know that this means that each process is assigned a small quantum or time slice that it is allowed to execute. This relies on hardware timer interrupts and is completely transparent to the processes.
- Linux schedule process according to a priority ranking this is a "goodness" ranking. Linux uses dynamic priorities, i.e. priorities are adjusted over time to eliminate starvation.
- Processes that have not received the CPU for a long time get their priorities increased, processes that have received the CPU often get their priorities decreased.
- Linux uses process preemption, a process is preempted when its time quantum has expired and new process enters TASK_RUNNING state and its priority is greater than the priority of the currently running process.

- The preempted process is not suspended, it is still in the ready queue, it simply no longer has the CPU.
- Consider a text editor and a complier, since the text editor is an interactive program, its dynamic priority is higher than the complier.
- The text editor will be block often since it is waiting for I/O. When the I/O interrupt receives a key-press for the editor is put on the ready queue and the scheduler is called since the editor's priority is higher than the complier. The editor getes the input and quickly blocks for more I/O.
- The Linux scheduling algorithm is not based on a continuous CPU time axis, instead it divides the CPU time into epochs. An epoch is a division of time or a period of time.
- In a single epoch, every process has a specified time quantum that is computed at the beginning of each epoch. This is the maximum CPU time that process can use during the current epoch.
- A process only uses its quantum when it is executing on the CPU, when the process is waiting for I/O its quantum is not used. As a result, a process, can get the CPU many times in one epoch, until its quantum is full used.
- A epoch ends when all runnable processes have used all of their quantum. A new epoch starts and all process get a new quantum.

University Questions

1. *How does Linux's Completely Fair Scheduler (CFS) provide improved fairness over a traditional UNIX process scheduler ? When is the fairness guaranteed ?*
   AU : Dec.-16, Marks 8
2. *Write the various system administrator roles in LINUX OS.*
   AU : Dec.-17, Marks 7

7.3 Memory Management

- In Linux, process divides address space logically in three segments : text, data and stack.
- **Text segment :** It contains read only and machine instruction of a program. Compiler and assembler produce a text segment by translating the C or C++ code. Only one copy of the instructions for the same program resides in memory at any time. The portion of the executable file containing the text segment is the text section.

- **Data segment :** Data segment support read and write operation. It contains the initialized and uninitialized data portions of a program. The initialized data segment contains variables and complier constant that need an initial value when the program is started. All the variables are initialized to zero after loading. Statically allocated and global data that are initialized with nonzero values live in the data segment. Each process running the same program has its own data segment. The portion of the executable file containing the data segment is the data section.
- Global and statically allocated data that are initialized to zero by default are kept in what is colloquially called the block started by symbol (BSS) area of the process. Each process running the same program has its own BSS area. When running, the BSS data are placed in the data segment. In the executable file, they are stored in the BSS section.
- System call brk is used to set the size of the program in the data segment.
- **Stack segment :** It support supports read and write operation. It holds the applications run time stack.
- When a program is running, the initialized data, BSS, and heap areas are usually placed into a single contiguous area : The data segment. The stack segment and code segment are separate from the data segment and from each other.
- The different memory areas can have different hardware memory protection assigned to them. For example, the text segment might be marked "execute only," whereas the data and stack segments would have execute permission disabled. This practice can prevent certain kinds of security attacks.

7.3.1 System Call

- System call for memory management is brk, mmap and unmap.

1. The brk( ) system call

Syntax : int brk(void *end_data_segment)

It returns 0 on success or –1 on failure. The brk( ) system call actually changes the process's address space. The address is a pointer representing the end of the data segment. Its argument is an absolute logical address representing the new end of the address space.

2. The system call mmap is used for mapping a file and unmap is for unmapping a file. These two system calls are control memory mapped files.

Syntax : void *mmap(void *addr, size_t lengthint " prot ", int " flags , int fd, off_t offset);
int munmap(void *addr, size_t length);

mmap( ) creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in addr. The length argument specifies the length of the mapping. The flags argument determines whether updates to the mapping are visible to other processes mapping the same region, and whether updates are carried through to the underlying file.

7.3.2 Implementation of Memory Management

- On 32-bit machine, each process can address 2^{32} bytes. It means that, each virtual address space is 4 GB. Linux uses paging to translate virtual addresses to physical addresses. It does not use segmentation.
- The kernel splits the 4 GB virtual address space between user level space and the kernel level space. A typical split dedicates 3 GB to user space, and 1 GB for kernel space.
- The kernel's code and data structures must fit into that space, but the biggest consumer of kernel address space is virtual mappings for physical memory. The kernel cannot directly manipulate memory that is not mapped into the kernel's address space.
- Linux uses different zones to handle memory.
  1. ZONE_ DMA : Some older I/O devices can only address memory up to 16 M
  2. ZONE_ NORMAL : Regular memory up to 896 M
  3. ZONE_ HIGHMEM : Memory above 896 M
- Memory up to 896 MB is addressed directly. Virtual addresses between 896 MB and 1 GB are constantly changed, to address physical memory at higher addresses. On 64-bit machines, this isn't an issue; all memory is directly addressable.
- Page descriptor is maintained by LINUX. It contains a pointer to the address space it belongs to.
- Physical memory is divided into zones and Linux maintains zone descriptor for each zone. Zone memory utilization is stored in the zone descriptor. It also contains array of free area.

7.3.3 Linux Virtual Memory

1. Virtual memory system maintain the address space which is visible to each process.
2. Virtual memory pages created by linux Kernel on demand. It manages the loading and swapping of pages.
3. Virtual memory manager maintains two separate view of a process's address space.
   a. Logical view - Set of separate region.

b. Physical view - Set of pages.

4. Logical view describing instructions concerning the layout of the address space.
5. Logical view address space consists of;
   a. Set of nonoverlapping regions.
   b. Page aligned subset of the address space.
   c. Region represents a continuous space.
6. Regions for each address space are linked into a balanced binary tree.
7. Physical view stored in the hardware page tables for the process.
8. Physical view is managed by set of routines.
9. Virtual memory region is implemented by linux in different way.
10. Virtual memory region is also defined by its reaction to writes.
11. Region mapping into process address space is private or shared.

7.3.4 Buddy Algorithm

- Linux operating system manages memory using buddy algorithm. The buddy system is a simple dynamic storage allocation method. It combines free buffer coalescing with a power of 2 allocator. It is described by Knowlton and Knuth.
- Single allocation block to be split, to form two blocks half the size of the parent block. These two blocks are known as 'buddies'.
- Initially memory consists of a single contiguous area of 128 pages. When user send request for memory, it is first rounded up to a power of 2.
- For example : If the minimum allocation size is 8 bytes and the memory size is 1 MB then we can create a list for 8 bytes hole, a list for 16 byte holes, one for 32-bytes holes, 64, 128, 256, 512, 1 K, 2 K, 4 K, 8 K, 16 K, 32 K, 64 K, 128 K, 256 K, 512 K and one list for 1 MB holes.
- All the lists are initially empty, except for the 1 MB list, which has one hole listed. All allocations are rounded up to a power of two. Suppose the request for memory is 75 K then allocations rounded up to 128 K, 13 K allocations rounde up to 16 K, etc.
- If a block of that size is free, it is taken; otherwise, the smallest free block larger than the desired size is found and split in two halves. This splitting continuous until the appropriate size is reached.
- When memory is deallocated, the buddy system groups contiguous free pages. When process free a block of memory, the memory manager checks the bitmap for

checking block size. If adjacent block is also free, the memory manager combines the two blocks into a larger blocks.

- This method makes deallocation fast. If a block of size 2^K is free then the memory manager checks only the list of 2^K holes to merge them into a 2^{K+1} sized partition. Internal fragmentation is caused since memory requests are fitted in 2^K sized partitions.

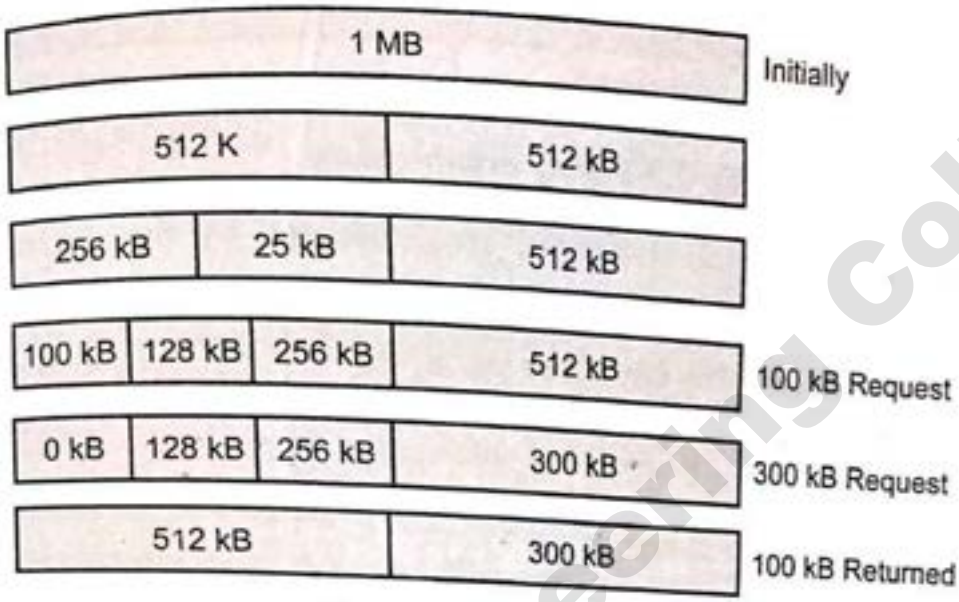


Fig. 7.3.1 Buddy system

Advantages

1. Allow memory to be reused.
2. Easy exchange of memory between the allocator and the paging system.
3. It provides flexibility.

Disadvantages

1. It wastes lot of space in internal fragmentation.
2. Programming interface is one more drawback.
3. Merging of holes is recursive, so poor worst case behaviour.

7.3.1 Slab Allocator

- Slab allocator allocates memory from any one of a number of available slab cache. A slab is made up of one or more physically contiguous pages.
- Slab cache consists of number of objects. A slab contains objects of only one kind. Each object is a kernel data structure. The slab allocator allocates all kernel objects of same class together in a pool.
- When the kernel requests memory for a new structure, the slab allocator returns a portion of a slab in the slab cache for that structure. If all of the slabs in a cache

are occupied, the slab allocator increases the size of the cache to include more slabs.

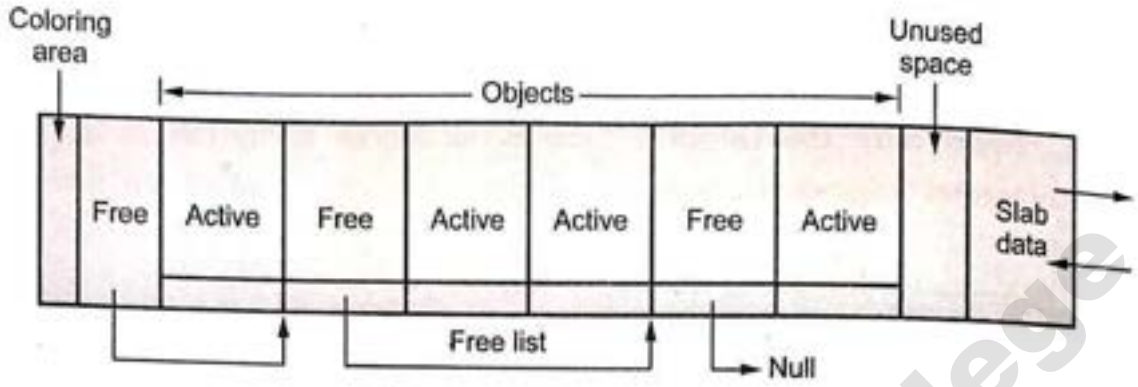


Fig. 7.3.2 Slab organization

- A slab is organized in a standard sized area allocated by the paging system. The slabs of a pool are entered in doubly link list for addition and deletion of slabs.
- Slab allocator divides the slab into three parts :
  1. Kernel memory slab structure 2. Set of objects
  3. Unused space.
- Fig. 7.3.2 shows slab organization.
- Slab states are as follows :
  1. Full 2. Partially empty 3. Empty.
- These three states depends upon number of active objects.
- The kernel allocates an object from a slab by removing the first element from the free list and incrementing the slab's in-use count. When freeing the object, it identifies the slab by a simple calculation :

$$\text{Slab address} = \frac{\text{Object address}}{\text{Slab size}}$$

- When in-use count becomes zero, the slab is free.
- Slab allocator introduced in Solaris 2.4 operating system.

Advantages / Benefits

1. Slab allocator provides better memory utilization.
2. Requests are serviced quickly by removing an object from free list.
3. It is space efficient.

Drawback

Overhead for having separate cache for each type of object.

7.3.5 Page Replacement Policy

- The Linux kernel goes through many extraordinarily complex operations to find a candidate set of pages that might be discarded.
- Policy that selects the page to be removed; crucial to system efficiency. Types include :
  1. First-In First-Out (FIFO) policy : Removes page that has been in memory the longest
  2. Least-Recently-Used (LRU) policy : Removes page that has been least recently accessed.
  3. Most Recently Used (MRU) policy
  4. Least Frequently Used (LFU) policy

7.3.6 Virtual Memory Regions

- There are several types of virtual memory region. Most memory regions are backed either by a file or by nothing.
- A region backed by nothing is the simplest type of virtual memory region. Such a region represents demand-zero memory.
- When a process tries to read a page in such a region, it is simply given back a page of memory filled with zeros.
- For example, a region may be taken from a file's contents, If it is a text region. When a region is backed by nothing, and when that region is accessed, a page filled with zeros is given.
- For regions having uninitialized data, the regions may not be backed by anything and may be filled with zeros.
- A virtual memory region is also defined by the region's is reaction to writes. If the region is a shared region, multiple processes can share the region at the same time.
- If the region is a private region, copy-on-write is set for the region. In this case, whenever one of the processes that is sharing the region tries to write to the region, a copy of the region is made for the writting; process.
- The kernel creates a new virtual address space in two stations. One is when a process runs a new program with the exec system call and the other is upon creation ofa new process by the fork system call when the fork system calls executed,a new process is created, but a new program is not run.
- Creating a new process with fork involves creating a complete copy of the existing process's virtual address space. The kernel copies the parent process's virtual memory address descriptors, then creates a new set of page tabes for the child.

- The parent's page tables are copied directly into the child's with the reference count of each page covered being incremented.
- After the fork, the parent and child share the same physical pages of memory in their address' spaces. If the virtual memory region is private,the pages for the regions are marked as read-only and copy-on-write is set.

7.4 Input-Output Management

- The Linux kernel can use several different I/O schedulers to prioritize disk input and output.
- The I/O scheduler schedules the pending I/O requests in order to minimize the time spent moving the disk head. This, in turn, minimizes disk seek time and maximizes hard disk throughput.
- Kernel does not issue block I/O requests to the disk in the order they are received or as soon as they are received. Instead, it performs operations called merging and sorting to greatly improve the performance of the system as a whole. The subsystem of the kernel that performs these operations is called the **I/O scheduler.**

7.4.1 Linux Elevator

- The first I/O scheduler is called the Linux Elevator.
- The Linux kernel 2.4 uses I/O scheduler is named as the Linux elevator.
- I/O schedulers often are called elevator algorithms, because they tackle a problem similar to that of keeping an elevator moving smoothly in a large building.
- The I/O scheduler found in the 2.4 Linux kernel is named the Linus Elevator. I/O schedulers often are called elevator algorithms, because they tackle a problem similar to that of keeping an elevator moving smoothly in a large building.
- The Linux Elevator performs both merging and sorting. When a request is added to the queue, it is first checked against every other pending request to see whether it is a possible candidate for merging. The Linux Elevator I/O scheduler performs both front and back merging.
- In summary, when a request is added to the queue, four operations are possible. In order, they are

1. If a request to an adjacent on-disk sector is in the queue, the existing request and the new request are merged into a single request.
2. If a request in the queue is sufficiently old, the new request is inserted at the tail of the queue to prevent starvation of the other, older, requests.

3. Next, if there is a suitable location sector-wise in the queue, the new request is inserted there. This keeps the queue sorted by physical location on disk.
4. Finally, if no such suitable insertion point exists, the request is inserted at the tail of the queue

7.4.2 Deadline Scheduler

- The deadline I/O scheduler sought to prevent the starvation caused by the Linux Elevator.
- The **Deadline scheduler** gives priority to read requests over write requests. It also imposes a deadline on all requests. After reaching the deadline, such requests gain priority over all other requests. This scheduling method helps prevent processes from becoming starved for I/O access. The Deadline scheduler is best used on physical media drives, since it attempts to group requests for adjacent sectors on a disk, lowering the time the drive spends seeking.
- I/O scheduler keeps the list of pending I/O requests sorted by block number. When a new I/O request is issued, it is inserted, block-wise, into the list of pending requests. This prevents the drive head from seeking all around the disk to service I/O requests.
- Instead, by keeping the list sorted, the disk head moves in a straight line around the disk. If the hard drive is busy servicing a request at one part of the disk and a new request comes in to the same part of the disk, that request can be serviced before moving off to other parts of the disk.

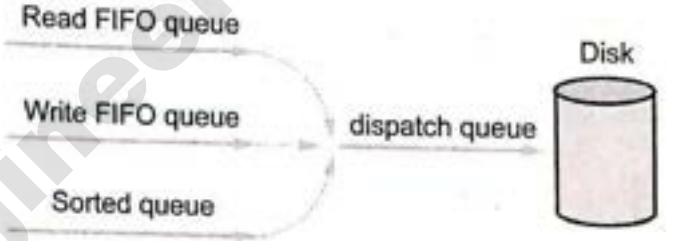


Fig. 7.4.1 Three queue of I/O scheduler

- Fig. 7.4.1 shows three queue of I/O scheduler.
- In the deadline I/O scheduler, each request is associated with an expiration time. By default, the expiration time is 500 milliseconds in the future for read requests and 5 seconds in the future for write requests.
- Under normal operation, the deadline I/O scheduler pulls requests from the head of the sorted queue into the dispatch queue. The dispatch queue is then fed to the disk drive. This results in minimal seeks.
- If the request at the head of either the write FIFO queue or the read FIFO queue expires, the deadline I/O scheduler then begins servicing requests from the FIFO queue. In this manner, the deadline I/O scheduler attempts to ensure that no request is outstanding longer than its expiration time.

7.4.3 Anticipatory I/O Scheduler

- **AIM :** To continue to provide excellent read latency, but also provide excellent global throughput.
- The anticipatory I/O scheduler starts with the deadline I/O scheduler as its base.
- When a read request is issued, it is handled as usual, within its usual expiration period. After the request is submitted, however, the anticipatory I/O scheduler does not immediately seek back and return to handling other requests.
- Instead, it does absolutely nothing for a few milliseconds. In those few milliseconds, there is a good chance that the application will submit another read request.
- Any requests issued to an adjacent area of the disk are immediately handled. After the waiting period elapses, the anticipatory I/O scheduler seeks back to where it left off and continues handling the previous requests.
- It is the default I/O scheduler in the Linux kernel. It performs well across most workloads. It is ideal for servers, although it performs very poorly on certain uncommon but critical workloads involving seek-happy databases.

7.4.4 Comparison between Windows and Linux I/O

Sr. No.	Windows	Linux
1.	It support advanced plug and play.	It support limited plug and play.
2.	Power management is good and based on CPU clock rate, sleep states.	Power management is very low and based on CPU clock rate management.
3.	I/O system uses layered approach.	I/O system uses a plug-in-model concept.
4.	Device drivers can be dynamically loaded and unloaded.	Device drivers can be dynamically loaded and unloaded.
5.	I/O is prioritized according to the thread priority.	I/O scheduling methods : deadline based, complete fairness queueing.
6.	System namespace is used for I/O devices and driver names.	File system is used for I/O devices names.

7.5 File System

- Even after a hard disk has been conceptually divided into partitions or logical volumes, it is still not ready to hold files.
- The File-system must be implemented in terms of raw disk blocks. The filesystem is the code that implements these, and it needs to add a bit of its own overhead and data.
- UNIX systems developed a well-defined Kernel interface that allowed multiple types of filesystems to be active at once.
- The filesystem interface also abstracted the underlying hardware, so filesystems see approximately the same interface to storage devices as do other UNIX programs that access the disks through device files *in /dev*.

7.5.1 Linux Filesystems : The ext Family

- The second extended file system was devised as an extensible and powerful file system for Linux.
- The Ext2fs supports standard UNIX file types : regular files, directories, device special files and symbolic links.
- Ext2fs is able to manage filesystems created on really big partitions. While the original kernel code restricted the maximal filesystem size to 2 GB, recent work in the VFS layer have raised this limit to 4 TB.
- Ext2fs provides long file names. It uses variable length directory entries. The maximal file name size is 255 characters. This limit could be extended to 1012 if needed.
- Ext2fs reserves some blocks for the super user. Normally, 5 % of the blocks are reserved. This allows the administrator to recover easily from situations where user processes fill up filesystems.
- Ext2fs implements fast symbolic links. A fast symbolic link does not use any data block on the filesystem.
- Ext2fs keeps track of the filesystem state. A special field in the super-block is used by the kernel code to indicate the status of the file system.
- When a filesystem is mounted in read/write mode, its state is set to "Not Clean". When it is unmounted or remounted in read-only mode, its state is reset to "Clean".

1. Ext2

- Ext2 stands for second extended file system. It was introduced in 1993. Developed by Rémy Card.
- This was developed to overcome the limitation of the original ext file system. Ext2 does not have journaling feature.
- On flash drives, USB drives, ext2 is recommended, as it does not need to do the over head of journaling.
- Maximum individual file size can be from 16 GB to 2 TB.
- Overall ext2 file system size can be from 2 TB to 32 TB.

2. Ext3

- Ext3 stands for third extended file system. It was introduced in 2001. Developed by Stephen Tweedie.
- Starting from Linux Kernel 2.4.15 ext3 was available.
- The main benefit of ext3 is that it allows journaling. Journaling has a dedicated area in the file system, where all the changes are tracked. When the system crashes, the possibility of file system corruption is less because of journaling.
- Maximum individual file size can be from 16 GB to 2 TB.
- Overall ext3 file system size can be from 2 TB to 32 TB.
- There are three types of journaling available in ext3 file system.
  a. Journal - Metadata and content are saved in the journal.
  b. Ordered - Only metadata is saved in the journal. Metadata are journaled only after writing the content to disk. This is the default.
  c. Writeback - Only metadata is saved in the journal. Metadata might be journaled either before or after the content is written to the disk.
- You can convert a ext2 file system to ext3 file system directly (without backup/restore).

3. Ext4

- Ext4 stands for fourth extended file system. It was introduced in 2008.
- Starting from Linux Kernel 2.6.19 ext4 was available.
- Supports huge individual file size and overall file system size.
- Maximum individual file size can be from 16 GB to 16 TB.
- Overall maximum ext4 file system size is 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).
- Directory can contain a maximum of 64,000 subdirectories.

- You can also mount an existing ext3 fs as ext4 fs (without having to upgrade it).
- Several other new features are introduced in ext4 : multiblock allocation, delayed allocation, journal checksum. fast fsck, etc. All you need to know is that these new features have improved the performance and reliability of the filesystem when compared to ext3.
- In ext4, you also have the option of turning the journaling feature "off".

HP-UX filesystems : VxFS and HFS

- VxFS is the mainstream HP-UX filesystem. It's based on a filesystem originally developed by Veritas Software, now part of Symantec.
- It includes a journal. HP sometimes refers to it as Journaled File System.
- VxFS is nearly unique among mainstream filesystems in that it supports clustering. By default, clustering features are turned off; use the **-o cluster** option to mount to turn them on.
- HFS is HP's previous mainstream filesystem. It's based on the UNIX File System and is now deprecated, though still supported.

AIX's JFS2

- Enhanced Journaled File System (JFS2) is a file system, introduced in AIX, that provides the capability to store much larger files than the existing Journaled File System (JFS).
- Summary for JSF2

Functions	JFS2	JFS
Fragments and block size	Block size (bytes) : 512, 1024, 2048, 4096. Maximum file system size in terabytes (TBs) : 4, 8, 16, 32	Fragment sizes (bytes) : 512, 1024, 2048, 4096 Maximum file system size in gigabytes (GBs) : 128, 256, 512, 1024
Maximum file system size	32 TBs	1 TB
Minimum file system size	16 TBs	Not applicable
Maximum file size	16 TBs	Approximately 63.876 GBs
Number of inodes	Dynamic, limited by disk space	Fixed, set at file system creation
Directory organization	B-tree	Linear
Compression	No	Yes
Quotas	Yes	Yes
Error logging	Yes	Yes

7.5.2 Filesystem Terminology

- **Inodes** are fixed-length table entries that each hold information about one file.
- For each file in the file system of the UNIX and POSIX.1 maintain the same set of file attributes. These attributes are stored not in the directory entry, but in an on-disk structure typically called an Inode. Attributes with their data are as follows :
  1. File type : Type of file
  2. Access permission : File access permission for owner, group and others.
  3. Hard link count : Number of hard link of a file.
  4. UID : The file owner user ID.
  5. GID : The file group ID.
  6. File size : File size in bytes.
  7. Last access time : Time the file was last accessed.
  8. Last modify time : Time the file was last modified.
  9. Inode number : The system inode number of the file.
  10. File system ID : The file system where the file is stored.
- Command **ls-l** is used to display file attributes. File attributes are required for the file Kernel to manage files. For example : When a user attempts to access a file, the kernel matches the user's UID and GID against those of the file to determine which category of access permission should be used for the access privilege of the user.
- An inode usually has an identifying number that you can see with **ls-i.**
- A **superblock** is a record that describes the characteristics of the filesystem. It contains information about the length of a disk block, the size and location of the inode tables, the disk block map and usage information, the size of the block groups, and a few other important parameters of the filesystem.
- Size of the file system represents the actual number of blocks (used + unused) present in the file system. The super block contains an array of **free disk block numbers,** one of which points to the next entry in the list.
- That entry in turn will be a data block, which contains an array of some other free blocks and a next pointer.
- When process requests for a block, it searches the free block list returns the available disk block from the array of free blocks in the super block.
- If the super block contains only one entry which is a pointer to a data block, which contains a list of other free blocks, all the entries from that block will be copied to the super block free list and returns that block to the process.

- Filesystems cache disk blocks to increase efficiency. All types of blocks can be cached, including superblocks, inode blocks and directory information.
- Caches are normally not "write-through," so there may be some delay between the point at which an application thinks it has written a block and the point at which the block is actually saved to disk.

7.5.3 Filesystem Polymorphism

- Filesystems are software packages with multiple components.
- The standard user-level commands knew about "the filesystem" that the system used and they simply implemented the appropriate functionality.
- The mkfs created new filesystems, **fsck** fixed problems and **mount** mostly just invoked the appropriate underlying system calls. These days' filesystems are more modular, so these commands call filesystem-specific implementations of each utility.

7.5.4 Filesystem Mounting

- A filesystem must be mounted before it becomes visible to processes. The mount point for a filesystem can be any directory, but the files and subdirectories beneath it are not accessible while a filesystem is mounted there.
- After installing a new disk, you should mount new filesystems by hand to be sure that everything is working correctly. For example, the command

$ sudo mount /dev/sda1 /mnt/temp

- Mounts the filesystem in the partition represented by the device file **/dev/sda1** on a subdirectory of **/mnt,** which is a traditional path used for temporary mounts.
- You can verify the size of a filesystem with the **df** command.

7.5.5 Setup for Automatic Mounting

- User will generally want to configure the system to mount local filesystems at boot time. A configuration file in **/etc** lists the device names and mount points of all the system's disks.
- On most systems this file is called **/etc/fstab,** but under both Solaris and AIX it has been restructured and renamed : **/etc/vfstab** on Solaris and **/etc/filesystems** on AIX.
- By default, ZFS filesystems mount themselves automatically and do not require *vfstab* entries. However, you can change this behavior by setting ZFS filesystem properties. Swap areas and non filesystem mounts should still appear in **vfstab.**

- **mount, umount, swapon** and **fsck** all read the filesystem catalog, so it is helpful if the data presented there is correct and complete. **mount** and **umount** use the catalog to figure out what you want done if you specify only a partition name or mount point on the command line.
- For example, with the Linux **fstab** configuration command :

  $ sudo mount /media/cdrom0

would have the same effect as typing

$ sudo mount -t udf -o user,noauto,exec,utf8 /dev/scd0 /media/cdrom0

- The conimand **mount -a** mounts all regular filesystems.
- For example :

  $ sudo mount -at ext4

- Mounts all local ext4 filesystems. The **mount** command reads **fstab** sequentially.
- The filesystems that are mounted beneath other filesystems must follow their parent partitions in the **fstab** file. For example, the line for **/var/log** must follow the line for **/var** *if /var* is a separate filesystem.

7.6 Inter-process Communication

- Exchange of data between two or more separate, independent processes/threads is possible using IPC. Operating systems provide facilities/resources for inter-process communications (IPC), such as message queues, semaphores, and shared memory.
- A complex programming environment often uses multiple cooperating processes to perform related operations. These processes must communicate with each other and share resources and information. The kernel must provide mechanisms that make this possible. These mechanisms are collectively referred to as **interprocess communication.**
- Distributed computing systems make use of these facilities/resources to provide application programming interface (API) which allows IPC to be programmed at a higher level of abstraction. (e.g., send and receive).
- Five types of inter-process communication are as follows :
  1. Shared memory permits processes to communicate by simply reading and writing to a specified memory location.
  2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
  3. Pipes permit sequential communication from one process to a related process.
  4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.
  5. Sockets support communication between unrelated processes even on different computers.

Name	Description	Scope	Use
File	• Data is written to and read from a typical UNIX file. • Any number of processes can interoperate.	Local	Sharing large data sets.
Pipe	• Data is transferred between two processes using dedicated file descriptors. • Communication occurs only between a parent and child process.	Local	Simple data sharing, such as producer and consumer.
Named pipe	• Data is exchanged between processes via dedicated file descriptors. • Communication can occur between any two peer processes on the same host.	Local	Producer and consumer, or command-and-control, as demonstrated with MySQL server and its command-line query utility.
Signal	• An interrupt alerts the application to a specific condition.	Local	Cannot transfer data in a signal, so mostly useful for process management.
Shared memory	• Information is shared by reading and writing from a common segment of memory.	Local	Cooperative work of any kind, especially if security is required.
Socket	• After special setup, data is transferred using common input/output operations.	Local or remote	Network services such as FTP, ssh, and the Apache Web Server.

Table 7.6.1 interprocess communication in UNIX

- **Purposes of IPC**
  1. Data transfer : One process may wish to send data to another process.
  2. Sharing data : Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it.
  3. Event modification : A process may wish to notify another process or set of processes that some event has occurred.

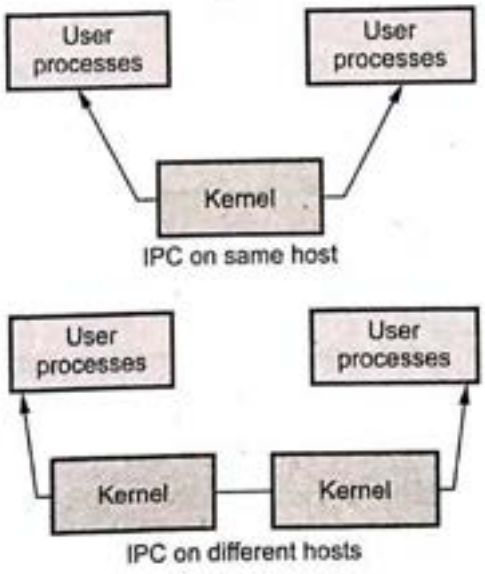


Fig. 7.6.1 IPC on hosts

4. Resource sharing : The kernel provides default semantics for resource allocation; they are not suitable for all application.
5. Process control : A process such as a debugger may wish to assume complete control over the execution of another process.

- IPC has two forms : IPC on same host and IPC on different hosts.

IPC is used for 2 functions :

1) **Synchronization :** Used to coordinate access to resources among processes and also to coordinate the execution of these processes. They are record locking, Semaphores, mutexes and condition variables.

2) **Message passing :** Used when processes wish to exchange information. Message passing takes several forms such as : Pipes, FIFOs, Message queues and Shared memory.

7.6.1 Pipes

- A pipe is a unidirectional, first-in first-out, unstructured data stream of fixed maximum size. Writers add data to the end of the pipe; readers retrieve data from the front of the pipe.
- Once read, the data is removed from the pipe and is unavailable to other readers. A pipe provides a simple flow control mechanism.
- A process attempting to read from empty pipe blocks until more data is written to the pipe. A process trying to write to a full pipe lock until another process reads data from pipe.
- The pipe system call creates a pipe and returns two file descriptors : One for reading and one for writing. These descriptors are inherited by child processes, which thus share access to the file.
- Each pipe can have several readers and writers. A give process may be a reader or writer or both. Fig. 7.6.2 shows data flow through a pipe.
- Pipes can be used only between processes that have a common ancestor. Normally, a pipe is created by a process, that process calls fork and the pipe is used between the parent and the child.

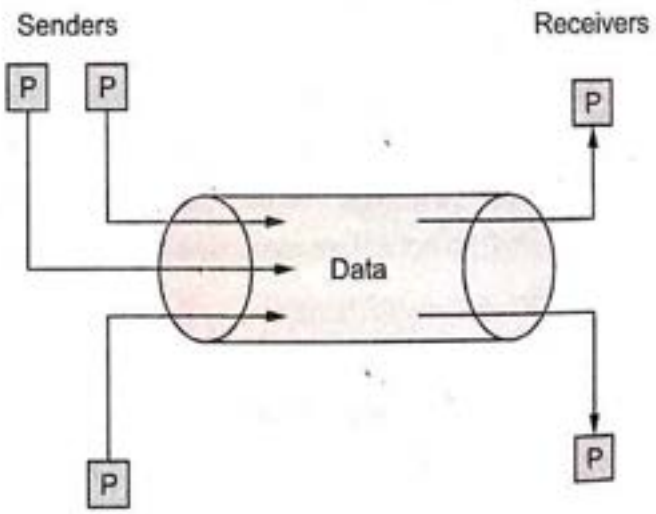


Fig. 7.6.2 Data flow through a pipe

- Example to show how to create and use a pipe :

```
main( )
{
  int pipefd[2], n;
  char buff[100];

  if (pipe(pipefd) < 0 )
  err_sys("pipe error");
  printf("read fd = %d, write fd = %d\n", pipefd[0], pipefd[1]);

 if (write(pipefd[1], "hello world\n", 12) != 12)
  err_sys("write error");
 if ( (n=read(pipefd[0], buff, sizeof(buff))) < =0)
  err_sys("read error");
 write (1, buff, n); /*fd=1=stdout*/
 }
```

Properties of Pipe :

1) Pipes do not have a name. For this reason, the processes must share a parent process. This is the main drawback to pipes. However, pipes are treated as file descriptors, so the pipes remain open even after fork and exec.
2) Pipes do not distinguish between messages; they just read a fixed number of bytes.
3) Pipes can also be used to get the output of a command or to provide input to a command.

- The most common use of pipes is to let the output of one program become the input for another. Users typically join two programs by a pipe using the shell's pipe operator (|).

Limitation of pipes :

1. Reading data removes it from the pipe, a pipe cannot be used to broadcast data to multiple receivers.
2. Data in a pipe is treated as a byte stream and has no knowledge of message boundaries.
3. If there are multiple readers on a pipe, a writer cannot direct data to a specific reader.

- **Program for sending data from parent process to child process over a pipe**

```
#include <stdio.h>
int main(void)
{
int n;
int fd[2];
pid_t pid;
char line[MAXLINE];

if (pipe(fd) < 0)
err_sys("pipe error");

if ((pid = fork( )) < 0)
{
  err_sys("fork error");

} else if (pid > 0)
 { /* parent */
 close(fd[0]);
 write(fd[1], "hello world\n", 12);

 } else { /* child */
  close(fd[1]);

n = read(fd[0], line, MAXLINE);
write(STDOUT_FILENO, line, n);

}
exit(0);
}
```

7.6.2 FIFO

- A FIFO (First In First Out) is a one-way flow of data. FIFOs have a name, so unrelated processes can share the FIFO.
- FIFOs are sometimes called named pipes. Pipes can be used only between related processes when a common ancestor has created the pipe. FIFOs can be used to duplicate an output stream in a series of shell commands.
- FIFOs is used to send data between a client and a server.

- Data written to a FIFO file are stored in a fixed-size buffer and retrieved in a first-in-first-out order.
- Function prototype for FIFO :

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo( const char* path_name, mode_t mode);
```

- The mode argument is just like in open ( ).
- When a process opens a FIFO file for read-only, the kernel will block the process until there is another process that opens the same file for write. If a process opens a FIFO for write, it will be blocked until another process opens the FIFO for read. This provides a method for process synchronization..
- If a process writes to a FIFO that is full, the process will be blocked until another process has read data from the FIFO to make room for new data in the FIFO.
- If a process attempts to read data from a FIFO that is empty, the process will be blocked until another process writes data to the FIFO.
- If a process does not desire to be blocked by a FIFO file, it can specify the O_NONBLOCK flag in the open call to the FIFO file.
- If two processes are to communicate via a FIFO file, it is important that the writer process closes its file descriptor when it is done, so that the reader process can see the end-of-file condition.
- There are two uses for FIFOs.
  1. FIFOs are used by shell commands to pass data from one shell pipeline to another without creating intermediate temporary files.
  2. FIFOs are used as rendezvous points in client server applications to pass data between the clients and the servers.
- FIFO offers some important **advantages** over pipes.
  1. FIFO may be accessed by unrelated processes.
  2. They are persistent, and hence are useful for data that must outlive the active users.
  3. They have a name in the file system name space.

Drawbacks :

1. Pipes must be explicitly deleted when not in use.
2. They are less secure than pipes.

- Comparison between FIFO and pipe

Normal operation

1. A FIFO is created by the *mknod* system call and may then be opened and accessed by any process that knows the name and has the permissions. FIFO continues to exist until explicitly deleted.
2. Pipe is created by the pipe system call, which returns a read and a write descriptor to it. A pipe may have multiple reader and writers. When no more readers or writers remain, the kernel automatically deletes the pipe.

I/O Operation

3. An I/O operation on FIFOs and pipes is quite similar. Write append data to the end, while reads remove data from the head. Once data has been read, it is removed from the pipe and is unavailable even to other readers.
4. In read operation, the size requested is greater than the amount of data currently in the pipe or FIFO, the kernel reads whatever data is available and returns the byte count to the caller. If no data is available, the reader will block until another process writes to the pipe.

7.6.3 Shared Memory

- A region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer. Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of kernel intervention.
- In contrast, in shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses and no assistance from the kernel is required.

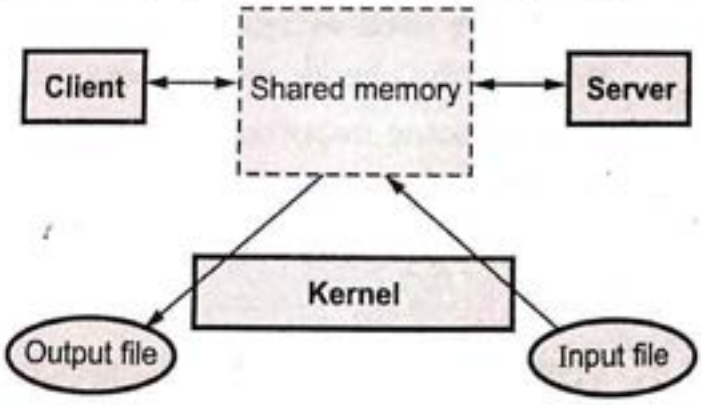


Fig. 7.6.3 Client/server with shared memory

- Fig. 7.6.3 shows client/server with shared memory.

Advantages

1. Good for sharing large amount of data
2. Very fast.

Limitations

1. No synchronization provided - applications must create their own.
2. Alternative to shared memory is *mmap* system call, which maps file into the address space of the caller.

Two Marks Questions with Answers

Q.1 State the components of LINUX system. AU : May-16

Ans. : Components of LINUX system is kernel, system libraries and system utilities.

Q.2 What is open source software ?

Ans. : Open source software is a type of software where user has access to the software's source code and can freely use, modify and distribute the software.

Q.3 What are the two types of Linux distribution ?

Ans. : Linux distribution are of two types : RPM based distribution and Debian based distribution.

Q.4 Explain difference between Linux and Windows OS.

Ans. :

Parameters	Linux OS	Windows OS
Name of developer company	Linus Torvalds	Microsoft
License	GNU/Free	Proprietary
File System Support	Ext2, Ext3, Ext4, JFS, FAT, FAT32 and NTFS	FAT, FAT32, NTFS
Power Management	Good	Very Low
Page Replacement Algorithm	Based on global clock algorithm	Based on the working set theory
Kernel	Kernel is non-pageable	Kernel is pageable
Cost	Freely download from the Internet	Not free.
Language support	Multilingual	Multilingual
User space and kernel space relation of memory	2GB:2GB	3GB:1GB

Q.5 What is data partition ?

Ans. : Data partition : Normal Linux system data, including the root partition containing all the data to start up and run the system.

Q.6 What is swap partition ?

Ans. : Swap partition : expansion of the computer's physical memory, extra memory on hard disk. Swap partition is an essential partition for installing Linux and enough space is to be allocated for this partition during the installation time.

Q.7 What is buddy algorithm in LINUX ?

Ans. : The buddy system is a simple dynamic storage allocation method. It combines free buffer with a power of 2 allocator. Single allocation block to be split, to form two blocks half the size of the parent block. These two blocks are known as 'buddies'.

Q.8 Define superblock ?

Ans. : A superblock is a record that describes the characteristics of the file system. It contains information about the length of a disk block, the size and location of the inode tables, the disk block map and usage information, the size of the block groups, and a few other important parameters of the file system.

Q.9 List the task of system administrator.

Ans. : Task includes starting up and shutting down the system, taking regular backup, keeping system logs, opening an account for new user etc.

Q.10 What is Red Hat Package Manager (RPM) ?

Ans. : RPM is a powerful command line driven package management system capable of installing, uninstalling, verifying, querying, and updating computer software packages. Each software package consists of an archive of files along with information about the package like its version, a description.

Q.11 What is virtualization ? AU : EIE : Dec.-16

Ans. : Virtualization allows multiple operating system instances to run concurrently on a single computer; it is a means of separating hardware from a single operating system.

Q.12 What is user space ?

Ans. : User space is the space in memory where user processes run. This space is protected. The system prevents one process from interfering with another process. Only kernel processes can access a user process.

Q.13 What is responsibility of Kernel in LINUX operating system ? AU : CSE/IT : May-15

Ans. : Kernel's responsibilities are to control hardware, enforce security and allocate resources such as CPU and memory.

Q.14 Enumerate the requirement for Linux system administrator. Brief any one. AU : Dec.-15

Ans. :

- System adminstrators are the people responsible for making computers work in the field. They are also responsible for the uninterrupted operation of the computers to take care of the business needs.
- Helping users to set up their working environment : Linux allows any user to customize his working environment. This is usually achieved by using.rc files.
- Installing and maintaining software : This may require installing software patches from time to time.
- Provisioning the mail and internet sevices : Users connected to any host shall seek mail and internet web access.
- Ensuring security of the system : The internet makes the task of system administration both interesting and challenging.

Q.15 Why virtualization is required ? AU : Dec.-15, 17

Ans. : Virtualization is a technology that allows operating systems to run as applications within other operating systems. Virtualization can improve not only resource utilization but also resource mangement.

Q.16 Define the function of caching-only server ? AU : May-16

Ans. : All servers cache answers to queries they receive from outside their own zone of authority. A cache-only server obtains all DNS information from other DNS servers. It does not store host information in domain files and does not perform zone transfers. A cache-only DNS server must have at least one root server or forwarder listed or it cannot resolve domain names. It stores the answer to each query in its cache for later use. A cache-only DNS server is not authoritative for any zone.

Q.17 Mention any two features of Linux file systems. AU : CSE : May-17

Ans. : The Linux file system is a hierarchically structured tree. Linux distinguishes between uppercase and lowercase letters in the file system.

Q.18 Enlist the advantage of using kernel modules in Linux. AU : CSE : May-17

Ans. :

1. Kernel modules are automatically loaded by various mechanisms.
2. It allows to setup with a standard minimum kernel without any extra device drivers built in.
3. It can implement device driver, file system and networking protocol.

Q.19 What scheduling algorithm is used in linux operating system to schedule jobs ? AU : EIE : Dec.-16

Ans. : Linux uses two scheduling policy for jobs. One is time sharing algorithm for preemptive scheduling and other is real time tasks.

Q.20 List the advantages of Linux OS. AU : Dec.-17

Ans. : 1. It is open source operating systems.

2. The security aspect of linux is much stronger than that of windows.

Q.21 What is handle ? How does a process obtain a handle ? AU : May-18

Ans. : User-mode code accesses objects using a opaque value called handle, which is returned by many APIs. Each process has a handle table containing entries that track the objects used by the process. The system process, which contains the kernel, has its own handle table, which is protected from user code. The handle tables in windows are represented by a tree structure, Kernel-mode code can access an object by using either a handle or reflenced pointer.

❑❑❑

UNIT - V

8 Mobile OS

Syllabus

iOS and Android - Architecture and SDK Framework, Media Layer, Services Layer, Core OS Layer, File System.

Contents

8.1 Android

- Android is an open source mobile OS developed by the Open Handset Alliance led by Google.
- Android is a software stack for mobile devices that includes an operating system, middleware and key applications.
- It is based on Linux 2.6 kernel.
- Android is an open source operating system, created by Google specifically for use on mobile devices (i.e. cell phones and tablets)
- It can be programmed in C/C++ but most app development is done in Java. It supports Bluetooth, Wi-Fi and 3G and 4G networking.

Android versions	Features
Android 1.1	1.API changes 2.Support for saving attachments for MMS
Android 1.5	1.Bluetooth A2DP and AVRCP support 2.Uploading video to You Tube and pictures
Android 1.6	1.Google free turn to turn support 2.WVGA screen resolution support
Android 2.0/2.1	1.HTML5 file support 2.Bluetooth 2.1 3.Microsoft exchange server
Android 2.2	1.Adobe flash 10.1 support 2.Wi-Fi hotspot support functionality
Android 2.3	1.Multi touch software keyboard 2.Support for extra large screen sizes and resolution
Android 3.0	1.3D desktop 2.Video chat 3.Optimized tablet support with new user interface

8.1.1 Android Architecture

- Fig. 8.1.1 shows Android software stack. Each layer of the stack and the corresponding elements within each layer are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.

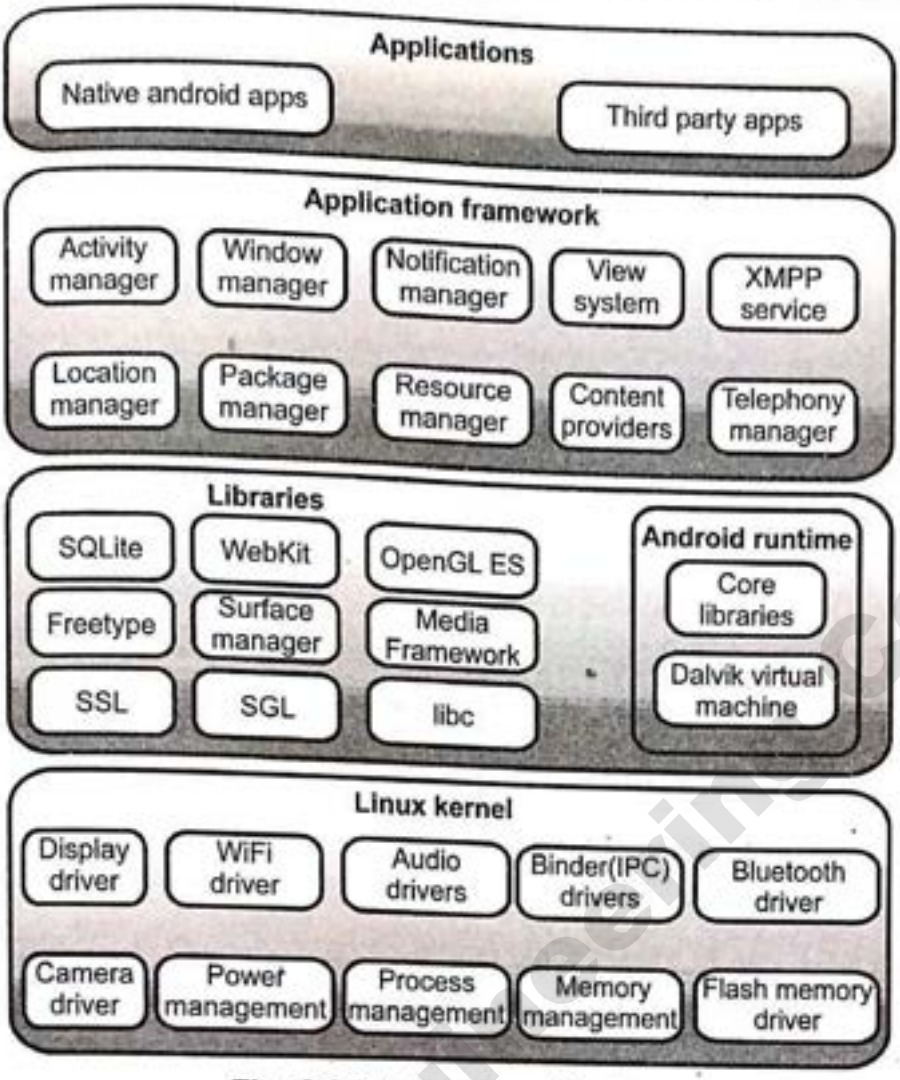


Fig. 8.1.1 Android architecture

- Android provides a set of core applications :
  1. Email client
  2. SMS program
  3. Calendar
  4. Maps
  5. Browser
  6. Contacts
  7. Etc
- All applications are written using the Java language.

App framework

- Used for enabling and simplifying the reuse of components

1. Developers have full access to the same framework APIs used by the core applications.

2. Users are allowed to replace components.

- App Framework features are as follows :

Feature	Role
View system	Used to build an application, including lists, grids, text boxes, buttons and embedded web browser
Content provider	Enabling applications to access data from other applications or to share their own data
Resource manager	Providing access to non-code resources (localized strings, graphics and layout files)
Notification manager	Enabling all applications to display customer alerts in the status bar
Activity manager	Managing the lifecycle of applications and providing a common navigation back stack

Libraries

- Libraries include a set of C/C++ libraries used by components of the Android system. It is exposed to developers through the Android application framework

Runtime

- Android run-time system provides core set of class libraries to ensure smooth platform for developers. With these libraries developers can easily import required libraries into their applications without doing any hard coding in applications.

Dalvik virtual machine

- Dalvik is a purpose built virtual machine designed specifically for android which was developed by Dan Bornstein and his team. Strictly it was developed for mobile devices. While developing Dalvik Virtual Machine Dan Bornstein and his team realize the constraints specific to mobile environment which is not going to change in future at least, like battery life,processing power and many more. So they optimized the dalvik virtual machine. Dalvik virtual machine uses register based architecture. With this architecture dalvik virtual machine has few advantages over java virtual machine such as :
  1. Dalvik uses its own 16 bit instruction set than java 8 bit stack instructions, which reduce the dalvik instruction count and raised its interpreter speed.
  2. Dalvik use less space, which means an uncompressed .dex file is smaller in size(few bytes) than compressed java archive file(.jar file).
- An open source software stack that includes operating system. Linux operating system kernel that provides low level interface with the hardware, memory management and process control.

- Middleware : A run time to execute Android applications including virtual machine and core libraries.

Important blocks in Android :

1. **Activity manager :** Manages the activity life cycle of applications
2. **Content providers :** Manage the data sharing between applications
3. **Telephony manager :** Manages all voice calls. We use telephony manager if we want to access voice calls in our application.
4. **Location manager :** Location management, using GPS or cell tower
5. **Resource manager :** Manage the various types of resources we use in our application

Android SDK features

The Android SDK includes

1. The Android APIs.
2. The core of the SDK.
3. Development tools.
4. No licensing, distributions, or development fees or release approval processes.
5. GSM, EDGE, and 3G networks for telephony and data transfer.
6. Full multimedia hardware control.
7. APIs for using sensor hardware including accelerometer and the compass.
8. APIs for location based services.

Application framework

1. Android offers developers the ability to build rich and innovative applications.
2. Developers have full access to the **same** framework APIs used by the core applications.
3. Underlying all applications is a set of services, including Views.
4. It can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser.

- Content providers enable applications to access data from other applications (such as Contacts), or to share their own data.
- A resource manager provides access to non-code resources such as localized strings, graphics and layout files.

- A notification manager enables all applications to display custom alerts in the status bar.
- An activity manager manages the lifecycle of applications and provides a common navigation backstack.

Libraries used in Android

- A set of C/C++ libraries used by various components of the Android system.
- System C library : Tuned for embedded Linux-based devices .
- Media Libraries : Based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files.
- Surface Manager : Manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.
- LibWebCore : A modern web browser engine which powers both the Android browser and an embeddable web view.
- SGL/ 3D libraries : SGL is underlying 2D graphics engine.
- SQLite : A powerful and lightweight relational database engine available to all applications.

Android run-time

- Android includes a set of core libraries that most of the functionality available in the core libraries of the Java programming language.
- Every Android app runs in its own process with its own instance of the Dalvik virtual machine. The Dalvik VM executes files in the Dalvik Executable (.dex) format.

Slow Android apps

1. By default on Android, all work is done in a single thread, the "main application" thread. If a component of the work takes a long time, the rest of the work will be "blocked". For example, a long time to access data across the network prevents responding to any GUI events.
2. In the Android OS, if a GUI doesn't respond to an input event in five seconds, then it is considered unresponsive and the OS will try to kill it.

Android thread design

1. Only perform GUI actions on main application thread. Spawn separate threads to perform data-intensive or slow actions. Make these threads asynchronous.

2. Main thread does not have to wait for/check on other threads. Instead, those threads run as they need to and report back to the original thread. Any changes made to the UI should go through the UI thread.

8.1.2 Comparison of Android OS Vs iPhone OS Features

Android OS	iPhone OS
Limited internal memory. Big headache because apart from photos and media content, the default memory is already limited	Good internal memory and user have choice of different internal memory sizes
It supports external memory. External SD card can be inserted to store photos, media, etc.	No external expandable memory. But no complaints here because the internal memory itself is huge and good enough
File Transfer/Synchronization is easy. Plug and transfer the data in flash drive mode.	File Transfer/ Synchronization is just like iPods.
It broadcasts the system-wide notifications and other application notifications in cascade windows which can be pulled down to see details. Register Listener, use callback to read the information	Push notifications and individual notifications on updates. It saves power because server pushes data. No need to poll and pull data
Hardware Design/Buttons : Menu and back buttons do not always do the same thing. The functionalities vary for different apps.	Hardware Design/Buttons : One button is used for one clear function. Menu and back always mean what they imply.

Android applications

a. Android applications get distributed in a .apk file. APK stands for "Android Package". It is simply a zip file that has a particular file structure. An APK contains :

1. The Android Manifest file (an XML file with lots of metadata).
2. A Resource bundle containing sounds, graphics, etc.
3. The Dalvik classes that make up user application.

8.1.3 Android Benefits

1. An open and free development platform. Handset makers can use it without royalty and customize to their hearts content.
2. Component-based architecture : Lots of default components can be replaced straightforwardly.

Proponents of Android point to the following benefits :

a. Lots of services : location, sql, maps, web, etc.

b. Well managed applications; isolated from each other to protect data and provide security;

c. Operating system can quit programs as needed to ensure good performance on mobile devices.

d. Portability : To support a new device, a company has to port the virtual machine; Android apps (Dalvik) then execute on the new device with little to no modification.

8.2 iOS Technology

- iOS is the operating system that runs on iPad, iPhone and iPod touch devices. The operating system manages the device hardware and provides the technologies required to implement native apps.
- The iOS Software Development Kit (SDK) contains the tools and interfaces needed to develop, install, run, and test native apps that appear on an iOS device's Home screen.

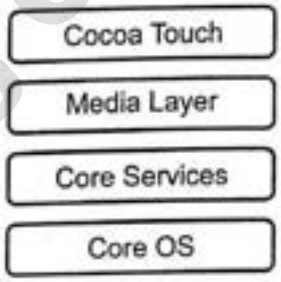


Fig. 8.2.1 iOS architecture

- Fig. 8.2.1 shows iOS architecture.
- The iOS Architecture is layered. At the highest level, iOS acts as an intermediary between the underlying hardware and the apps you create.
- Apps do not talk to the underlying hardware directly. Instead, they communicate with the hardware through a set of well-defined system interfaces. These interfaces make it easy to write apps that work consistently on devices having different hardware capabilities.
- The **Cocoa Touch layer** contains key frameworks for building iOS apps. These frameworks define the appearance of your app. They also provide the basic app infrastructure and support for key technologies such as multitasking, touch-based input, push notifications and many high-level system services.
- High-Level features of Cocoa touch layers are AirDrop, Multitasking, Auto Layout, Storyboards and Local Notifications And Apple Push Notification Service.
- Cocoa touch layer contains following frameworks for iPhone app development:
  a. UIKit framework
  b. Map kit framework
  c. Push notification service.
  d. Message UI framework
  e. Address book UI framework
  f. Game kit framework

g. iAd framework
h. Event kit UI framework
i. Accounts framework
j. Twitter framework

- The **Media layer contains** the graphics, audio and video technologies you use to implement multimedia experiences in your apps. The technologies in this layer make it easy for you to build apps that look and sound great.

6.2.1 Media Layer

- The role of the Media layer is to provide iOS with audio, video, animation and graphics capabilities.
- As with the other layers comprising the iOS stack, the media layer comprises a number of frameworks which may be utilized when developing iPhone apps.
- The technologies in this layer make it easy for you to build apps that look and sound great.
- Features of media layer :
  a. Graphics technologies
  b. Audio technologies
  c. Video technologies
  d. AirPlay
- Media layer contains following frameworks :

1. Core video framework : This framework provides buffering support for the Core media framework. Whilst this may be utilized by application developers it is typically not necessary to use this framework.
2. Core text framework : The iOS core text framework is a C-based API designed to ease the handling of advanced text layout and font rendering requirements.
3. Image I/O framework : The Image I/O framework, the purpose of which is to facilitate the importing and exporting of image data and image metadata, was introduced in iOS 4. The framework supports a wide range of image formats including PNG, JPEG, TIFF and GIF.
4. Assets library framework : The assets library provides a mechanism for locating and retrieving video and photo files located on the iPhone device. In addition to accessing existing images and videos, this framework also allows new photos and videos to be saved to the standard device photo album.
5. Core graphics framework : The iOS core graphics framework provides a lightweight two dimensional rendering engine. Features of this framework include PDF document creation and presentation, vector based drawing, transparent layers,

path based drawing, anti-aliased rendering, color manipulation and management, image rendering and gradients.

6. Core image framework : A new framework introduced with iOS 5 providing a set of video and image filtering and manipulation capabilities for application developers.
7. Quartz core framework : The purpose of the Quartz Core framework is to provide animation capabilities on the iPhone. It provides the foundation for the majority of the visual effects and animation used by the UIKit framework and provides an Objective-C based programming interface for creation of specialized animation within iPhone apps.
8. OpenGL ES framework : For many years the industry standard for high performance 2D and 3D graphics drawing has been OpenGL. OpenGL for Embedded Systems (ES) is a lightweight version of the full OpenGL specification designed specifically for smaller devices such as the iPhone.
9. GLKit framework : The GLKit framework is an Objective-C based API designed to ease the task of creating OpenGL ES based applications.
10. NewsstandKit framework : The Newsstand application is a new feature of iOS 5 and is intended as a central location for users to gain access to newspapers and magazines. The NewsstandKit framework allows for the development of applications that utilize this new service.
11. iOS audio support : iOS is capable of supporting audio in AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM, Linear PCM, μ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10 and AES3-2003 formats through the support provided by the following frameworks.
12. AV foundation framework : An Objective-C based framework designed to allow the playback, recording and management of audio content.

8.2.2 Core Services Layer

- The Core Services layer contains fundamental system services for apps.
- Key among these services are the core foundation and foundation frameworks, which define the basic types that all apps use.
- This layer also contains individual technologies to support features such as location, iCloud, social media and networking.
- Features :

1. Peer-to-Peer services
2. iCloud storage

3. Automatic reference counting
4. Block objects
5. Grand central dispatch
6. In-App purchase
7. SQLite
8. XML support
9. File-sharing support, data protection

- It consists of the following frameworks.
- Address book framework : This provides programmatic access to the iPhone Address Book contact database allowing applications to retrieve and modify contact entries.
- CFNetwork framework : The CFNetwork framework provides a C-based interface to the TCP/IP networking protocol stack and low level access to BSD sockets. This enables application code to be written that works with HTTP, FTP and domain name servers and to establish secure and encrypted connections using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).
- Core Data Framework : This framework is provided to ease the creation of data modeling and storage in Model-View-Controller (MVC) based applications. Use of the Core Data framework significantly reduces the amount of code that needs to be written to perform common tasks when working with structured data within an application.
- Core foundation framework : The core foundation framework is a C-based Framework which provides basic functionality such as data types, string manipulation, raw block data management, URL manipulation, threads and run loops, date and times, basic XML manipulation and port and socket communication.
- The core media framework is the lower level foundation upon which the AV foundation layer is built.
- Core telephony framework : The iOS core telephony framework is provided to allow applications to interrogate the device for information about the current cell phone service provider and to receive notification of telephony related events.
- EventKit framework : An API designed to provide applications with access to the calendar and alarms on the device.
- Most applications will use iCloud document storage to share documents from a user's iCloud account. This is the feature that users think of when they think of

iCloud storage. A user cares about whether documents are shared across devices and can see and manage those documents from a given device.

- Data protection allows applications that work with sensitive user data to take advantage of the built-in encryption available on some devices.
- When your application designates a specific file as protected, the system stores that file on-disk in an encrypted format. While the device is locked, the contents of the file are inaccessible to both your application and to any potential intruders.
- However, when the device is unlocked by the user, a decryption key is created to allow your application to access the file.

8.2.3 Core OS Layer

- The Core OS layer contains the low-level features that most other technologies are built upon
- Even if you do not use these technologies directly in your apps, they are most likely being used by other frameworks.
- And in situations where you need to explicitly deal with security or communicating with an external hardware accessory, you do so using the frameworks in this layer.
- This layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.
- The Core OS layer occupies the bottom position of the iOS stack and, as such, sits directly on top of the device hardware.
- The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.
- Accelerate framework : Introduced in iOS 4.0, the Accelerate framework , contains interfaces for performing DSP, linear algebra and image-processing calculations. The advantage of using this framework over writing your own versions of these interfaces is that they are optimized for all of the hardware configurations present in iOS, based devices. Therefore, you can write your code once and be assured that it runs efficiently on all devices.
- External accessory framework : It provides the ability to interrogate and communicate with external accessories connected physically to the iPhone via the 30-pin dock connector or wirelessly via Bluetooth.

- Security framework : The iOS Security framework provides all the security interfaces you would expect to find on a device that can connect to external networks including certificates, public and private keys, trust policies, key chains, encryption, digests and Hash-based Message Authentication Code (HMAC).
- The core bluetooth framework allows developers to interact specifically with Bluetooth Low-Energy ("LE") accessories. The Objective-C interfaces of this framework allow you to scan for LE accessories, connect and disconnect to ones you find, read and write attributes within a service, register for service and attribute change notifications and much more.
- System : The system level encompasses the kernel environment, drivers and low level UNIX interfaces of the operating system. The kernel itself is based on Mach and is responsible for every aspect of the operating system.
- It manages the virtual memory system, threads, file system, network and interprocess communication. The drivers at this layer also provide the interface between the available hardware and system frameworks.
- For security purposes, access to the kernel and drivers is restricted to a limited set of system frameworks and applications.
- iOS provides a set of interfaces for accessing many low-level features of the operating system. Your application accesses these features through the LibSystem library.

8.3 Android File Management

- Android uses a file system that's similar to disk-based file systems on other platforms. A file object is suited to reading or writing large amounts of data in start-to-finish order without skipping around.
- All Android devices have two file storage areas : "internal" and "external" storage.
- Android device may use an updated Linux file system, such as ext4 or a proprietary file system by a manufacturer, depending on who made the device and what has been done to it by the user.
- File system is the collection place on disk device for files. Visualize the file system as consisting of a single node at the highest level (ROOT) and all other nodes descending from the root node in a tree-like fashion.
- Samsung Galaxy S phones use the Samsung RFS proprietary file system while the Samsung Nexus S with Android 2.3 uses the Linux Ext4 file system.
- **< /mnt > :** This directory is used for mount points. The different physical storage devices (like the hard disk drives, floppies, CD-ROM's) must be attached to some directory in the file system tree before they can be accessed. This attaching is

called mounting and the directory where the device is attached is called the mount point.

- **SDCard :** The mounted SDCard is a storage device mounted to the file system in the typical Linux fashion. On the file system root the /sdcard is a symbolic link to /mnt/sdcard. /mnt/sdcard is where the SD card is actually mounted, but the same files can also be accessed in /sdcard.
- Fig. 8.3.1 shows typical directory structure of android file system.

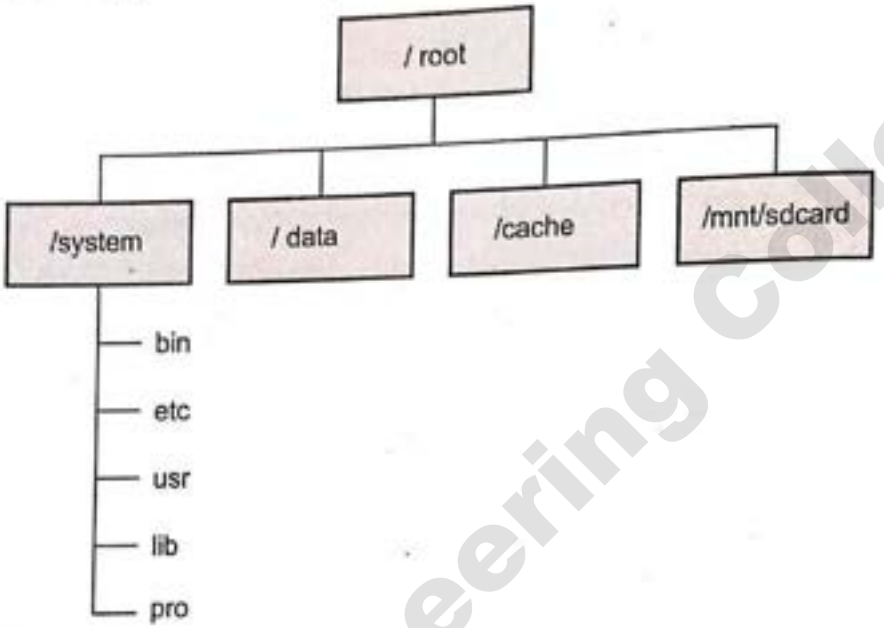


Fig. 8.3.1 Directory structure of Android

ro: mounted as read only : /root and /system
rw: mounted as read and write : /data, /cache and /mnt/sdcard

- The superblock is the key to maintaining the file system. It is an 8 kB block of disk space that maintains the current status of the file system. Because of its importance, a copy is maintained in memory and at each cylinder group within the file system.
- The copy in main memory is updated as events transpire. The update daemon is the actual process that calls on the kernel to flush the cached superblocks, modified inode and cached data blocks to disk.
- Usually Linux system assumes all file systems are read and writable.

1. ODEX FILE

- In Android file system, applications come in packages with the extension .apk. These application packages or APKs contain certain .odex files whose supposed function is to save space.

- These 'odex' files are actually collections of parts of an application that are optimized before booting. Doing so speeds up the boot process, as it preloads part of an application.
- On the other hand, it also makes hacking those applications difficult because a part of the coding has already been extracted to another location before execution.

2. DEODEX

- Deodexing is basically repackaging of these APKs in a certain way, such that they are reassembled into classes.dex files. All pieces of an application package are put together back in one place.
- Deodexed ROMs (or APKs) have all their application packages put back together in one place, allowing for easy modification such as theming. Since no pieces of code are coming from any external location, custom ROMs or APKs are always deodexed to ensure integrity.

8.3.1 SQLite

- SQlite is an open source embedded database. The resulting design goals of SQLite were to allow the program to be operated without a database installation or administration.
- It most widely deployed SQL database engine in the world. SQLite is based on the Structured Query Language (SQL). Android contains the SQLite database management classes that an application would use to manage its own private database.
- SQLite is open source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation. SQLite supports all the relational database features.
- It is designed to provide a streamlined SQL-based database management system suitable for embedded systems and other limited memory systems. The full SQLite library can be implemented in under 400 kB.
- In contrast to other database management systems, SQLite is not a separate process that is accessed from the client application. The library is linked in and thus becomes an integral part of the application program.

Unique Features

1. No configuration is required.
2. No server process to administer or user accounts to manage.
3. Easy to backup and transmit database.
4. Dynamic typing for column values, variable lengths for column records.

5. Query can reference multiple database files.
6. A few non-standard SQL extensions.

- SQLiteDatabase allows methods to open the database connection, perform queries and query updates and close the database. You can define keys and values for queries via the ContentValues object. This is necessary for Insert and Update calls. Delete only requires the Row Number.
- android.database.sqlite classes are as follows :

1. SQLiteCloseable - An object created from a SQLiteDatabase that can be closed.
2. SQLiteCursor - A cursor implementation that exposes results from a query on a SQLiteDatabase.
3. SQLiteDatabase - Exposes methods to manage a SQLite database.
4. SQLiteOpenHelper - A helper class to manage database creation and version management.
5. SQLiteProgram - A base class for compiled SQLite programs.
6. SQLiteQuery - A SQLite program that represents a query that reads the resulting rows into a CursorWindow.
7. SQLiteQueryBuilder - A convenience class that helps build SQL queries to be sent to SQLiteDatabase objects.
8. SQLiteStatement - A pre-compiled statement against a SQLiteDatabase that can be reused.

Two Marks Questions with Answers

Q.1 What is iOS SDK ?

Ans. : The iOS Software Development Kit (SDK) contains the tools and interfaces needed to develop, install, run and test native apps that appear on an iOS device's Home screen.

Q.2 What is media layer in iOS ?

Ans. : The media layer contains technologies to implement multimedia functions, such as graphics, Audio, Video, and Airplay. It contains frameworks for audio/video codecs and OpenGL features.

Q.3 Explain iOS architecture.

Ans. : iOS architecture is written in Objective-C language and comprised of four layers, Cocoa touch, Media, Core service and Core OS. System interfaces are provided by framework, which contains libraries and resources such as header and image.

Q.4 List framework used in Core OS Layer.

Ans. : Frameworks are accelerate framework, core bluetooth framework, external accessory framework, generic security services framework, security framework, system and 64-Bit Support

Q.5 What is Android ?

Ans. : Android is a Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it's free and any one can use it

Q.6 What is Dalvik Virtual Machine ?

Ans. : The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for memory, battery life and performance.

❑❑❑