UNIT-I

PART-B

1. Enumerate the different operating system structure and explain with neat sketch. (APRIL/MAY 2019, APRIL/MAY 2018, APRIL/MAY 2017, Nov/Dec 2015, NOV/DEC 2013, APRIL/MAY 2010)

An operating system is a construct that allows the user application programs to interact with the system hardware. Since the operating system is such a complex structure, it should be created with utmost care so it can be used and modified easily. An easy way to do this is to create the operating system in parts. Each of these parts should be well defined with clear inputs, outputs and functions.

Simple Structure

There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope. A common example of this is MS-DOS. It was designed simply for a niche amount for people. There was no indication that it would become so popular.

Layered Structure

One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.

An image demonstrating the layered approach is as follows:

As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc from their upper layers.

One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.

2. Discuss multiprocessor systems in detail. (APRIL/MAY 2017, MAY/JUNE 2013)

Multiprocessor Operating System refers to the use of two or more central processing units (CPU) within a single computer system. These multiple CPUs are in a close communication sharing the computer bus, memory and other peripheral devices. These systems are referred as *tightly coupled systems.*

These types of systems are used when very high speed is required to process a large volume of data. These systems are generally used in environment like satellite control, weather forecasting etc.

Multiprocessing system is based on the symmetric multiprocessing model, in which each processor runs an identical copy of operating system and these copies communicate with each other. In this system processor is assigned a specific task. A master processor controls the system. This scheme defines a master-slave relationship. These systems can save money in compare to single processor systems because the processors can share peripherals, power supplies and other devices. The main advantage of multiprocessor system is to get more work done in a shorter period of time. Moreover, multiprocessor systems prove more reliable in the situations of failure of one processor. In this situation, the system with multiprocessor will not halt the system; it will only slow it down.

The whole task of multiprocessing is managed by the operating system, which allocates different tasks to be performed by the various processors in the system.

Applications designed for the use in multiprocessing are said to be threaded, which means that they are broken into smaller routines that can be run independently. This allows the operating system to let these threads run on more than one processor simultaneously, which is multiprocessing that results in improved performance.

Multiprocessor system supports the processes to run in parallel. Parallel processing is the ability of the CPU to simultaneously process incoming jobs. This becomes most important in computer system, as the CPU divides and conquers the jobs. Generally the parallel processing is used in the fields like artificial intelligence and expert system, image processing, weather forecasting etc.

In a multiprocessor system, the dynamically sharing of resources among the various processors may cause therefore, a potential bottleneck. There are three main sources of contention that can be found in a multiprocessor operating system:

Locking system: In order to provide safe access to the resources shared among multiple processors, they need to be protected by locking scheme. The purpose of a locking is to serialize accesses to the protected resource by multiple processors. Undisciplined use of locking can severely degrade the performance of system. This form of contention can be reduced by using locking scheme, avoiding long critical sections, replacing locks with lock-free algorithms, or, whenever possible, avoiding sharing altogether.

Shared data: The continuous accesses to the shared data items by multiple processors (with one or more of them with data write) are serialized by the cache coherence protocol. Even in a moderate-scale system, serialization delays can have significant impact on the system performance. In addition, bursts of cache coherence traffic saturate the memory bus or the interconnection network, which also slows down the entire system. This form of contention can be eliminated by either avoiding sharing or, when this is not possible, by using replication techniques to reduce the rate of write accesses to the shared data.

False sharing: This form of contention arises when unrelated data items used by different processors are located next to each other in the memory and, therefore, share a single cache line: The effect of false sharing is the same as that of regular sharing bouncing of the cache line among several processors. Fortunately, once it is identified, false sharing can be easily eliminated by setting the memory layout of non-shared data.

Apart from eliminating bottlenecks in the system, a multiprocessor operating system developer should provide support for efficiently running user applications on the multiprocessor. Some of the aspects of such support include mechanisms for task placement and migration across processors, physical memory placement insuring most of the memory pages used by an application is located in the local memory, a

3. Explain the different types of system call?

The interface between a process and an operating system is provided by system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

System calls are usually made when a process in user mode requires access to a resource. Then it requests the kernel to provide the resource via a system call.

Types of System Calls

There are mainly five types of system calls. These are explained in detail as following

Here are the types of system calls:

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file, reading a file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

Some of the examples of all the above types of system calls in Windows and Unix are given as follows:

Types of S		
Process Control	ExitProcess() WaitForSingleObject()	exit() wait()
File Management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()

Device Management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()

There are many different system calls as shown above. Details of some of those system calls are as follows:

wait()

In some systems, a process may wait for another process to complete its execution. This happens when a parent process creates a child process and the execution of the parent process is suspended until the child process executes. The suspending of the parent process occurs with a wait() system call. When the child process completes execution, the control is returned back to the parent process.

exec()

This system call runs an executable file in the context of an already running process. It replaces the previous executable file. This is known as an overlay. The original process identifier remains since a new process is not created but data, heap, stack etc. of the process are replaced by the new process.

fork()

Processes use the fork() system call to create processes that are a copy of themselves. This is one of the major methods of process creation in operating systems. When a parent process creates a child process and the execution of the parent process is suspended until the child process executes. When the child process completes execution, the control is returned back to the parent process.

exit()

The exit() system ltithreaded
environment, this reclaims
resources that were

kill()

The kill() system call is used by the operating system to send a termination signal to a process that urges the process to exit. However, kill system call does not necessary mean killing the process and can have various meanings.

System Program:

System programs provide an environment where programs can be developed and executed. In the simplest sense, system programs also provide a bridge between the user interface and system calls. In reality, they are much more complex. For example, a compiler is a complex system program.

System Programs Purpose

The system program serves as a part of the operating system. It traditionally lies between the user interface and the system calls. The user view of the system is actually defined by system programs and not system calls because that is what they interact with and system programs are closer to the user interface.

An image that describes system programs in the operating system hierarchy is as follows:

In the above image, system programs as well as application programs form a bridge between the user interface and the system calls. So, from the user view the operating system observed is actually the system programs and not the system calls.

Types of System Programs

System programs can be divided into seven parts. These are given as follows:

Status Information

The status information system programs provide required data on the current or past status of the system. This may include the system date, system time, available memory in system, disk space, logged in users etc.

Communications

These system programs are needed for system communications such as web browsers. Web browsers allow syst ired.

File Manipulation

These system prog ng various commands like create, delete, copy, rename, print etc. These commands can create files, delete files, copy the contents of one file into another, rename files, print them etc.

Program Loading and Execution

The system programs that deal with program loading and execution make sure that programs can be loaded into memory and executed correctly. Loaders and Linkers are a prime example of this type of system programs.

File Modification

System programs that are used for file modification basically change the data in the file or modify it in some other way. Text editors are a big example of file modification system programs.

Application Programs

Application programs can perform a wide range of services as per the needs of the users. These include programs for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.

Programming Language Support

These system programs provide additional support features for different programming languages. Some examples of these are compilers, debuggers etc. These compile a program and make sure it is error free respectively.

OS generation:

Operating Systems have evolved over the years. So, their evolution through the years can be mapped using generations of operating systems. There are four generations of operating systems. These can be described as following

The First Generation (1945 - 1955): Vacuum Tubes and Plugboards

.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were no operating systems so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plugboards, programs were written on cards and read into the system.

The Second Generation (1955 - 1965): Transistors and Batch Systems

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as mainframes and were locked in air-conditioned computer rooms with staff to operate them.

The Batch System was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the magnetic tape. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and ran it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

Until the 1960's, there were two types of computer systems i.e the scientific and the commercial computers. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced multiprogramming. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This created network operating systems and distributed operating systems. The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

4. Write short notes on operating system services and components.(NOV/DEC 2019, MAY/JUNE 2012)

An Operating System provides services to both the users and to the programs.

- It provides programs an environment to execute.
- It provides users the services to execute the programs in a convenient manner.

Following are a few common services provided by an operating system −

- Program execution
- I/O operations
- File System manipulation
- Communica
- Error Detect
- Resource Al
- Protection

Program execution

Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management −

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

File system manipulation

A file represents a collection of related information. Computers can store files on the disk (secondary storage), for long-term storage purpose. Examples of storage media include magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management −

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication

In case of distribu memory, peripheral devices, en all the processes. Multiple nes in the network.

The OS handles routing and connection strategies, and the problems of contention and security. Following are the major activities of an operating system with respect to communication −

- Two processes often require data to be transferred between them
- Both the processes can be on one computer or on different computers, but are connected through a computer network.
- Communication may be implemented by two methods, either by Shared Memory or by Message Passing.

Error handling

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

Resource Management

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities.

Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system. Following are the major activities of an operating system with respect to protection −

- The OS ensures that all access to system resources is controlled.
- The OS ensures that external I/O devices are protected from invalid access attempts.
- The OS provides authentication features for each user by means of passwords.

Components of Operating System

- Kernel.
- Process Execution.
- Interrupt.
- Memory Management.
- Multitasking.
- Networking.
- Security.
- User Interface.

5. Discuss about the functionality of system boot with respect to operating system. (APR/MAY 2015)

The BIOS, operating system and hardware components of a computer system should all be working correctly for it to boot. If any of these elements fail, it leads to a failed boot sequence.

System Boot Process

The following diagram demonstrates the steps involved in a system boot process:

Here are the steps:

- The CPU initializes itself after the power in the computer is first turned on. This is done by triggering a series of clock ticks that are generated by the system clock.
- After this, the CPU looks for the system's ROM BIOS to obtain the first instruction in the start-up program. This first instruction is stored in the ROM BIOS and it instructs the system to run POST (Power on Self Test) in a memory address that is predetermined.
- POST first checks the BIOS chip and then the CMOS RAM. If there is no battery failure detected by POST, then it continues to initialize the CPU.
- POST also checks the hardware devices, secondary storage devices such as hard drives, ports etc. And other hardware devices such as the mouse and keyboard. This is done to make sure they are working properly.
- After POST makes sure that all the components are working properly, then the BIOS finds an operating system to load.
- In most computer systems, the operating system loads from the C drive onto the hard drive. The CMOS chip typically tells the BIOS where the operating system is found.
- The order of the different drives that CMOS looks at while finding the operating system is known as the boot sequence. This sequence can be changed by changing the CMOS setup. · After finding the appropriate boot drive, the BIOS first finds the boot record which tells it to find the beginning of the operating system.
- After the initialization of the operating system, the BIOS copies the files into the memory. Then the operating system controls the boot process.
- In the end, the operating system does a final inventory of the system memory and loads the device drive
- The users ca

Without the syste e software components, including the ones not frequently required. With the system boot, only those software components need to be downloaded that are legitimately required and all extraneous components are not required. This process frees up a lot of space in the memory and consequently saves a lot of time.

6. Sketch the structure of Direct Memory Access in detail. (APR/MAY 2017, APR/MAY 2015)

For the execution of a computer program, it requires the synchronous working of more than one component of a computer. For example, Processors - providing necessary control information, addresses…etc, buses - to transfer information and data to and from memory to I/O devices…etc. The interesting factor of the system would be the way it handles the transfer of information among

processor, memory and I/O devices. Usually, processors control all the process of transferring data, right from initiating the transfer to the storage of data at the destination. This adds load on the processor and most of the time it stays in the ideal state, thus decreasing the efficiency of the system. To speed up the transfer of data between I/O devices and memory, DMA controller acts as station master. DMA controller transfers data with minimal intervention of the processor.

DMA Controller:

The term DMA stands for direct memory access. The hardware device used for direct memory access is called the DMA controller. DMA controller is a control unit, part of I/O device's interface circuit, which can transfer blocks of data between I/O devices and main memory with minimal intervention from the processor.

DMA Controller Diagram in Computer Architecture

DMA controller provides an interface between the bus and the input-output devices. Although it transfers data without intervention of processor, it is controlled by the processor. The processor initiates the DMA controller by sending the starting address, Number of words in the data block and direction of transfer of data .i.e. from I/O devices to the memory or from main memory to I/O devices. More than one external device can be connected to the DMA controller.

Working of DMA Controller

DMA controller has to share the bus with the processor to make the data transfer. The device that holds the bus at a given time is called bus master. When a transfer from I/O device to the memory or vice verse has to be made, the processor stops the execution of the current program, increments the program counter, moves data over stack then sends a DMA select signal to DMA controller over the address bus.

If the DMA controller is free, it requests the control of bus from the processor by raising the bus request signal. Processor grants the bus to the controller by raising the bus grant signal, now DMA controller is the bus master. The processor initiates the DMA controller by sending the memory addresses, number of blocks of data to be transferred and direction of data transfer. After assigning the data transfer task to the DMA controller, instead of waiting ideally till completion of data transfer, the processor resumes the execution of the program after retrieving instructions from the stack.

7. **Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems? (MAY/JUNE 2016).**

Symmetric Multiprocessing system: in this case each processor runs an identical copy of the OS, and hence they can ndows NT, UNIX, LINUX

Asymmetric Mult s the system, the other processor either look to the master for instruction or have predefined task assigned . Example SunOS v4.

Advantages of multiprocessor systems:

1. Increased throughput
2. economy of scale
3. reliability more

Disadvantages:

1. Common computer bus, clock , memory and peripheral devices

2. cost is more

BASIS FOR COMPARISON	SYMMETRIC MULTIPROCESSING	ASYMMETRIC MULTIPROCESSING
Basic	Each processor run the tasks in Operating System.	Only Master processor run the tasks of Operating System.
Process	Processor takes processes from a common ready queue, or there may be a private ready queue for each processor.	Master processor assign processes to the slave processors, or they have some predefined processes.
Architecture	All processor in Symmetric Multiprocessing has the same	All processor in Asymmetric Multiprocessing may have same or
Communication	All processors communicate with another processor by a shared memory.	Processors need not communicate as they are controlled by the master processor.

BASIS FOR COMPARISON	SYMMETRIC MULTIPROCESSING	ASYMMETRIC MULTIPROCESSING
Failure	If a processor fails, the computing capacity of the system reduces.	If a master processor fails, a slave is turned to the master processor to continue the execution. If a slave processor fails, its task is switched to other processors.
Ease	Symmetric Multiprocessor is complex as all the processors need to be synchronized to maintain the load balance.	Asymmetric Multiprocessor is simple as master processor a

UNIT-II

PART-B

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

Process	Burst time (B.T)	Arrival time(A.T)
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. (APRIL/MAY 2017)

Solution:

FCFS

Process	Burst time(B.T)	Arrival time(A.T)
P1	**8**	**0.00**
P2	**4**	**1.001**
P3	**9**	**2.001**
P4	**5**	**3.001**
P5	**3**	**4.001**

Gantt Chart

P1	P2	P3	P4	P5

0 8 12 21 26 29

Process	C.T	T.A=C.T-A.T	W.T=T.A-B.T
P1	**8**	**8**	**0**
P2	**4**	**10.999**	**6.99**
P3	**9**	**18.999**	**9.999**
P4	**5**	**22.999**	**17.999**
P5	**3**	**24.999**	**21.999**

Where,

C.T -Completion time

T.A-Turnaround Time

A.T-Arrival Time

W.T-Waiting Time

Average Waiting Ti

Average Waiting Ti

Average Turnaround Time=(8+10.99+18.99+22.99+24.99)/5

Average Turnaround Time=17.199

SJF

Gantt Chart

P1	P5	P2	P4	P1	P3

0 1 4 8 13 20 29

Process	A.T	B.T	C.T	T.A.T	W.T
P1	**0.00**	**8**	**20**	**20**	**12**
P2	**1.001**	**4**	**8**	**6.99**	**2.99**
P3	**2.001**	**9**	**29**	**26.99**	**17.99**
P4	**3.001**	**5**	**13**	**9.99**	**4.99**
P5	**4.001**	**3**	**4**	**0**	**-3**

Average Turnaround Time=(20+6.99+26.99+9.99+0)/5

Average Turnaround Time=12.799

Average Waiting Time= (12+2.99+17.99+4.99-3)/5

Average Waiting Time=6.999

Where,

T.A.T=C.T-A.T

W.T=T.A.T-B.T

C.T -Completion time

T.A.T-Turnaround Time

A.T-Arrival Time

B.T-Burst Time

PRIORITY:

Here in the question, No priority is given, So assume the priority.

P4	**5**	**3.001**	**4**
P5	**3**	**4.001**	**5**

Gantt Chart

P1	P2	P3	P4	P5

0 8 12 21 26 29

Process	C.T	T.A.T	W.T
P1	**8**	**8**	**0**
P2	**12**	**10.999**	**6.99**

P3	**21**	**18.999**	**9.99**
P4	**26**	**22.999**	**17.99**
P5	**29**	**24.999**	**21.99**

Where,

T.A.T=C.T-A.T

W.T=T.A.T-B.T

Average Turnaround Time=(8+10.99+18.99+22.99+24.99)/5

Average Turnaround Time=17.192

Average Waiting Time=(0+6.99+9.99+17.99+21.99)/5

Average Waiting Time=11.392

ROUND ROBIN

Process	Burst time(B.T)	Arrival time(A.T)
P1	**8**	**0.00**
P2	**4**	**1.001**
P3	**9**	**2.001**
P4	**5**	**3.001**
P5	**3**	**4.001**

Quantum -2

P1	P2	P3	P4	P5	P1	P2	P3	P4	P5	P1	P3	P4	P1	P3	P3

0 2 4 6 8 10 12 14 16 18 19 21 23 24 26 28 29

Process	B.T	C.T	T.A.T	W.T
P1	**8**	**26**	**26**	**18**
P4	**5**	**24**	**20.99**	**15.99**
P5	**3**	**19**	**14.99**	**11.99**

Where,

Average Turnaround Time=(26+12.99+26.99+20.99+14.99)/5

Average Turnaround Time=20.392

Average Waiting Time=(18+8.99+17.99+15.99+11.99)/5

Average Waiting Time=14.592

2. **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 to races arise. (NOV/DEC 2019, APRIL/MAY 2017)**

Race Conditions

A **race condition** is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.

Avoiding Race Conditions:

Critical Section:

To avoid race condition we need Mutual Exclusion. Mutual Exclusion is some way of making sure that if one process is using a shared variable or file, the other processes will be excluded from doing the same things. The difficulty above in the printer spooler occurs because process B started using one of the shared variables before process A was finished with it. That part of the program where the shared memory is accessed is called the critical region or critical section. If we could arrange matters such that no two processes were ever in their critical regions at the same time, we could avoid race conditions. Although this requirement avoids race conditions, this is not sufficient for having parallel processes cooperate correctly and efficiently using shared data. (Rules for avoiding Race Condition) Solution to Critical section problem: 1. No two processes may be simultaneously inside their critical regions. (Mutual Exclusion) 2. No assumptions may be made about speeds or the number of CPUs. 3. No process running outside its critical region may block other processes. 4. No process should have to wait forever to enter its critical region.

Dining Philosophers Problem:

The dining philosopher problem is another classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

Problem Statement:

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle as shown in the below figure.

Dining Philosophers Problem

At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place.

Solution:

From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a hilosopher is in an endless cycl

An array of five sem

The code for each philosopher looks like:

```
while(TRUE) {
wait(stick[i]);
wait(stick[(i+1) % 5]);  // mod is used because if i=5, next
                         // chopstick is 1 (dining table is circular)
/* eat */
signal(stick[i]);

signal(stick[(i+1)  %  5]);
/* think */
}
```

When a philosopher wants to eat the rice, he will wait for the chopstick at his left and picks up that chopstick. Then he waits for the right chopstick to be available, and then picks it too. After eating, he puts both the chopsticks down.

But if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick forever. The possible solutions for this are:

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.

Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.

3. What is the average turnaround time for the following processes using (NOV/DEC 2017)

a) FCFS **(3)**

b) SJF non-preemptive **(3)**

c) Preemptive SJF. **(3)**

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

ii) With example elucidate livelock. **(4)**

a) FCFS

- Gantt Chart

P1	P2	

0 **8**

Process	A.T	B.T	C.T	T.A.T
P1	0.0	8	8	8
P2	0.4	4	12	11.6
P3	1.0	1	13	12

Where,

T.A.T=C.T-A.T

C.T -Completion time

T.A.T-Turn around Time A.T-Arrival Time

B.T-Burst Time

Average Turnaround Time=(8+11.6+12) / 3=31.6 / 3

=10.53

b) SJF non-preemptive

- Gantt Chart

P1	P2	P3

0 8 12 13

Process	**A.T**	**B.T**	**C.T**	**T.A.T**
P1	**0.0**	**8**	**8**	**8**
P2	**0.4**	**4**	**12**	**11.6**
P3	**1.0**	**1**	**13**	**12**

Where,

T.A.T=C.T-A.T

C.T -Completion time

T.A.T-Turnaround Time

A.T-Arrival Time

B.T-Burst Time

Average Turnaround Time=(8+11.6+12) / 3=31.6 / 3=10.53

a) Preemptive SJF

Gantt Chart

P1	P2	P3	P1	P2

0 0.4

P3	**1.0**	**1**	**2**	**1**

Where,

T.A.T=C.T-A.T

C.T -Completion time

T.A.T-Turnaround Time

A.T-Arrival Time

B.T-Burst Time

Average Turnaround Time=(9.6+12.2+1) / 3=22.8 / 3

Average Turnaround Time=7.6

4. Explain the FCFS, preemptive and non-preemptive versions of Shortest-Job First and Round Robin (time slice = 2) scheduling algorithms with Gantt charts for the four Processes given. Compare their average turnaround and waiting time. (NOV/DEC 2012)

Process	Arrival Time	Waiting Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Refer Notes(unit-2)

5. **Write in detail about several CPU scheduling algorithms. (APRIL/MAY 2018, MAY/JUNE 2014, APRIL/MAY2011)**
   - CPU scheduling is the basis of multi programmed operating systems.
   - The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization.
   - Scheduling is a fundamental operating-system function.

Scheduling Algorithms:

The following subsections will explain several common scheduling strategies, looking at only a single CPU burst each for a small number of processes. Obviously real systems have to deal with a lot more simultaneous processes executing their CPU-I/O burst cycles.

First-Come First-Serve Scheduling, FCFS:

- FCFS is very simple - Just a FIFO queue, like customers waiting in line at the bank or the post office or at a copying machine.
- Unfortunately, however, FCFS can yield some very long average wait times, particularly if the first process to get there takes a long time. For example, consider the following three processes:

P1	24
P2	3
P3	3

- In the first Gantt chart below, process P1 arrives first. The average waiting time for the three processes is (0 + 24 + 27) / 3 = 17.0 ms.

- In the second Gantt chart below, the same three processes have an average wait time of (0 + 3 + 6) / 3 = 3.0 ms. The total run time for the three bursts is the same, but in the second case two of the three finish much quicker, and the other process is only delayed by a short amount.
- FCFS can also block the system in a busy dynamic system in another way, known as the ***convoy effect***. When one CPU intensive process blocks the CPU, a number of I/O intensive processes can get backed up behind it, leaving the I/O devices idle. When the CPU hog finally relinquishes the CPU, then the I/O processes pass through the CPU quickly, leaving the CPU idle while everyone queues up for I/O, and then the cycle repeats itself when the CPU intensive process gets back to the ready queue.

Shortest-Job-First Scheduling, SJF:

- The idea behind the SJF algorithm is to pick the quickest fastest little job that needs to be done, get it out of the way first, and then pick the next smallest fastest job to do next.
- (Technically this algorithm picks a process based on the next shortest **CPU burst**, not the overall process time.)
- For example, the Gantt chart below is based upon the following CPU burst times, (and the assumption that all jobs arrive at the same time.)

Process	Burst Time
P1	6
P2	8

- In the case above the average wait time is (0 + 3 + 9 + 16) / 4 = 7.0 ms, (as opposed to 10.25 ms for FCFS for the same processes.)

- SJF can be proven to be the fastest scheduling algorithm, but it suffers from one important problem: How do you know how long the next CPU burst is going to be?
  - For long-term batch jobs this can be done based upon the limits that users set for their jobs when they submit them, which encourages them to set low limits, but risks their having to re-submit the job if they set the limit too low. However that does not work for short-term CPU scheduling on an interactive system.
  - Another option would be to statistically measure the run time characteristics of jobs, particularly if the same tasks are run repeatedly and predictably. But once again that really isn't a viable option for short term CPU scheduling in the real world.
  - A more practical approach is to ***predict*** the length of the next burst, based on some historical measurement of recent burst times for this process. One simple, fast, and relatively accurate method is the ***exponential average***, which can be defined as follows. (The book uses tau and t for their variables, but those are hard to distinguish from one another and don't work well in HTML.)

$$estimate[i + 1] = alpha * burst[i] + (1.0 - alpha) * estimate[i]$$

  - In this scheme the previous estimate contains the history of all previous times, and alpha serves as a weighting factor for the relative importance of recent data versus past history. If alpha is 1.0, then past history is ignored, and we assume the next burst will be the same length as the last burst. If alpha is 0.0, then all measured burst times are ignored, and we just assume a constant burst time. Most commonly alpha is set at 0.5, as illustrated in Figure a.

- SJF can be either preemptive or non-preemptive. Preemption occurs when a new process arrives in the ready queue that has a predicted burst time shorter than the time remaining in the process whose burst is currently on the CPU. Preemptive SJF is sometimes referred to as ***shortest remaining time first scheduling.***
- For example, the following Gantt chart is based upon the following data:

P2	1	4
P3	2	9
p4	3	5

- The average wait time in this case is ((5 - 3) + (10 - 1) + (17 - 2)) / 4 = 26 / 4 = 6.5 ms. (As opposed to 7.75 ms for non-preemptive SJF or 8.75 for FCFS.)

Priority Scheduling:

- Priority scheduling is a more general case of SJF, in which each job is assigned a priority and the job with the highest priority gets scheduled first. (SJF uses the inverse of the next expected burst time as its priority - The smaller the expected burst, the higher the priority.)
- Note that in practice, priorities are implemented using integers within a fixed range, but there is no agreed-upon convention as to whether "high" priorities use large numbers or small numbers. This book uses low number for high priorities, with 0 being the highest possible priority.
- For example, the following Gantt chart is based upon these process burst times and priorities, and yields an average waiting time of 8.2 ms:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

- Priorities can be assigned either internally or externally. Internal priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, and other factors available to the kernel. External priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.
- Priority scheduling can be either preemptive or non-preemptive.
- Priority scheduling can suffer from a major problem known as ***indefinite blocking***, or ***starvation***, in which a low-priority task can wait forever because there are always some other jobs around that have higher priority.

- If this problem is allowed to occur, then processes will either run eventually when the system load lightens (at say 2:00 a.m.), or will eventually get lost when the system is shut down or crashes. (There are rumors of jobs that have been stuck for years.)
- One common solution to this problem is ***aging***, in which priorities of jobs increase the longer they wait. Under this scheme a low-priority job will eventually get its priority raised high enough that it gets run.

Round Robin Scheduling

- Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called ***time quantum***.
- When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.
  - If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.
  - If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.
- The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on.
- RR scheduling can give the effect of all processors sharing the CPU equally, although the average wait time can be longer than with other scheduling algorithms. In the following example the average wait time is 5.66 ms.

Process	Burst Time
P1	24
P2	3
P3	3

- The perform s large enough, then ess gets 1/nth of the processor time and share the CPU equally.
- **BUT**, a real system invokes overhead for every context switch, and the smaller the time quantum the more context switches there are. Most modern systems use time quantum between 10 and 100 milliseconds, and context switch times on the order of 10 microseconds, so the overhead is small relative to the time quantum.

- Turnaround time also varies with quantum time, in a non-apparent manner.

- In general, turnaround time is minimized if most processes finish their next cpu burst within one time quantum. For example, with three processes of 10 ms bursts each, the average turnaround time for 1 ms quantum is 29, and for 10 ms quantum it reduces to 20. However, if it is made too large, then RR just degenerates to FCFS. A rule of thumb is that 80% of CPU bursts should be smaller than the time quantum.

Multilevel Queue Scheduling

- When processes can be readily categorized, then multiple separate queues can be established, each implementing whatever scheduling algorithm is most appropriate for that type of job, and/or with different parametric adjustments.
- Scheduling must also be done between queues, that is scheduling one queue to get time

relative to other queues. Two common options are strict priority (no job in a lower priority queue runs until all higher priority queues are empty) and round-robin (each queue gets a time slice in turn, possibly of different sizes.)

Multilevel Feedback-Queue Scheduling

- Multilevel feedback queue scheduling is similar to the ordinary multilevel queue scheduling described above, except jobs may be moved from one queue to another for a variety of reasons:
  - If the ive, then it ma
  - Agin e can get bum
- Multilevel feedback queue scheduling is the most flexible, because it can be tuned for any situation. But it is also the most complex to implement because of all the adjustable parameters. Some of the parameters which define one of these systems include:
  - The number of queues.
  - The scheduling algorithm for each queue.
  - The methods used to upgrade or demote processes from one queue to another. (Which may be different.)
  - The method used to determine which queue a process enters initially.

6. What is critical section? Specify the requirements for a solution to critical section problem. (MAY/JUNE 2019, NOV/DEC 2012)

The Critical-Section Problem:

- There are n processes that are competing to use some shared data
- Each process has a code segment, called critical section, in which the shared data is accessed.
- Problem - ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Requirements to be satisfied for a Solution to the Critical-Section Problem

1. **Mutual Exclusion -** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress -** If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting -** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

```
do {
        entry section critical
              section

        exit section

              remainder section }

      while (true);
```

- Two general approaches are used to handle critical sections in operating systems: **preemptive kernels** and **nonpreemptive kernels**.
- A preemptive kernel allows a process to be preempted while it is running in kernel mode.
- A non-preemptive kernel does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU.

7. How mon

- A high-le
  for proce
- Only one process may be active within the monitor at a time monitor monitor-name

{

// shared variable declarations

procedure body P1 (…) { …. }

…

procedure body Pn (…) {……}

{

initialization code

}

- To allow a process to wait within the monitor, a condition variable must be declared as o condition x, y;
- Two operations on a condition variable:
- x.wait () -a process that invokes the operation is suspended.
- x.signal () -resumes one of the suspended processes(if any)

8. Consider the following system snapshot using data structures in the banker's algorithm, with resources A, B, C and D and process P0 to P4.

	Max	**Allocation**	**Need**	**Available**
	A B C D	**A B C D**	**A B C D**	**A B C D**
P0	**6 0 1 2**			
P1	**1 7 5 0**			
P2	**2 3 5 6**			
P3	**1 6 5 3**	**0 6 3 3**		
P4	**1 6 5 6**	**0 2 1 2**		

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 requests from process P4 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)

Solution:

a) How many resources of type A, B, C and D are there?

A-9; B-13;C-10;D-11

b) What are the contents of the need matrix?

Need [i, j] = Max [i, j] - Allocation [i, j]

So, the content of Need Matrix is:

Process	Need A B C D
P0	2 0 1 1
P1	0 6 5 0
P2	1 1 0 2
P3	1 0 2 0
P4	1 4 4 4

c) Is the system in a safe state? Why?

The system is in a safe state as the processes can be finished in the sequence P0, P2, P4, P1 and P3.

d) If a requests from process P4 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.

If a request from process P4 arrives for additional resources of (1,2,0,0,), and if this request is granted, the new system state would be tabulated as follows.

9. Write in detail about deadlock avoidance and Bankers algorithm in detail. (NOV/DEC 2009) (APRIL/MAY2010, NOV/DEC 2012) (NOV/DEC 2013)

Deadlock Avoidance

Requires that the system has some additional *a priori* information available.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
  - System is in safe state if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes is the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$.
- That is:

  – If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.

  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.

- When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Avoidance algorithms

- Single instance of a resource type. Use a resource-allocation graph

- Multiple instances of a resource type. Use the banker's algorithm

Resource-Alloca

- *Claim edge* P_i o R_j indicated that process P_j may request resource R_j; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.

- Resources must be claimed *a priori* in the system.

Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.
- Let n = number of processes, and m = number of resources types.
- ***Available****:* Vector of length m. If available [j] = k, there are k instances of resource type R_j available.
- ***Max****: n x m* matrix. If *Max* [*i,j*] = *k*, then process P_i may request at most k instances of resource type R_j.
- ***Allocation****: n* x *m* matrix. If Allocation[*i,j*] = *k* then P_i is currently allocated k instances of *Rj.*

- ***Need****: n* x *m* matrix. If *Need*[*i,j*] = *k*, then P_i may need k more instances of R_j to complete its task.

Nee **Example of Banker's Algorithm**

- 5 processes
  3 resource types

  A (10 instan

- Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
*P*0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

- The content of the matrix *Need* is defined to be *Max* - *Allocation*.

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0

P_3 0 1 1

P_4 4 3 1

- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria.

d [*i,j]* = *Max*[*i,j*] - *Allocation* [*i,j*].

10. Write in detail about deadlock recovery. (APRIL/MAY2011)

Deadlock recovery performs when a deadlock is detected.

When deadlock detected, then our system stops working, and after the recovery of the deadlock, our system start working again.

Therefore, after the detection of deadlock, a method/way must require to recover that deadlock to run the system again. The method/way is called as deadlock recovery.

Here are various ways of deadlock recovery that we will discuss briefly in this tutorial.

- Deadlock recovery through preemption
- Deadlock recovery through rollback
- Deadlock recovery through killing processes

Let's discuss about all the above three ways of deadlock recovery one by one.

Deadlock Recovery

The ability to take a ive it back without the process noticing. It is highly dependent on the nature of the resource.

Deadlock recovery through preemption is too difficult or sometime impossible.

Deadlock Recovery through RollBack

In this case of deadlock recovery through rollback, whenever a deadlock is detected, it is easy to see which resources are needed.

To do the recovery of deadlock, a process that owns a needed resource is rolled back to a point in time before it acquired some other resource just by starting one of its earlier checkpoints.

Deadlock Recovery through Killing Processes

This method of deadlock recovery through killing processes is the simplest way of deadlock recovery.

Sometime it is best to kill a process that can be return from the beginning with no ill effects.

11. Consider the following set of processes, with the length of the CPU - burst time given in Milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

The processes are arrived in the order P1, P2, P3, P4, P5, all at time 0.

- **Draw 4 Gantt charts illustrating the execution of these processes using FCFS, SJF Priority and RR (Time Slice = 1) scheduling**
- **What is the turnaround time of each process for each of the scheduling?**
- **Calculate the waiting time for each of the process (APRIL/MAY 2018, MAY/JUNE 2012, NOV/DEC 2015)**

Refer Notes (Unit-2)

13. Discuss in detail the critical section problem and also write the algorithm for Readers-Writers Problem with semaphores (NOV/DEC 2013)

The Critical-Section Problem:

- There are n processes that are competing to use some shared data
- Each process has a code segment, called critical section, in which the shared data is accessed.
- Problem - e er process is allowed to

READERS-WRITERS PROBLEM:

The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object.

The readers-writers problem is used to manage synchronization so that there are no problems with the object data. For example - If two readers access the object at the same time there is no problem. However if two writers or a reader and writer access the object at the same time, there may be problems.

To solve this situation, a writer should get exclusive access to an object i.e. when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the same time.

This can be implemented using semaphores. The codes for the reader and writer process in the reader-writer problem are given as follows:

Reader Process

The code that defines the reader process is given below:

```
wait (mutex);

rc ++;

if (rc == 1)

wait (wrt);

signal(mutex);

. READ THE OBJECT

wait(mutex);

rc --;
```

```
if (rc == 0)

signal (wrt);

signal(mutex);
```

In the above code, mutex and wrt are semaphores that are initialized to 1. Also, rc is a variable that is initialized to 0. The mutex semaphore ensures mutual exclusion and wrt handles the writing mechanism and is common to the reader and writer process code.

The variable rc denotes the number of readers accessing the object. As soon as rc becomes 1, wait operation is used on er the read operation is done, r wrt. So a writer can access th

Writer Process

The code that defines the writer process is given below:

```
wait(wrt);

. WRITE INTO THE OBJECT

signal(wrt);
```

If a writer wants to access the object, wait operation is performed on wrt. After that no other writer can access the object. When a writer is done writing into the object, signal operation is performed on wrt.

14. Discuss how deadlocks could be detected in detail. (APR/MAY 2015)

Deadlock Detection

- **Allow system to enter deadlock state**
- **Detection algorithm**
- **Recovery scheme**

Single Instance of Each Resource Type

- Maintain *wait-for* graph
  Nodes are processes.
  - P_i o P_j if P_i is waiting for P_j.
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.

Several Instances of a Resource Type

- *Available:* A vector of length m indicates the number of available resources of each type.
- *Allocation:* An n x m matrix defines the number of resources of each type currently allocated to each pr
- *Request:* An [i_j] = k, then process P_i is

15. 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 pseudo code for implementing the operations. (APR/MAY 2015)

Semophores

- It is a synchronization tool that is used to generalize the solution to the critical section problem in complex situations.

- A Semaphore s is an integer variable that can only be accessed via two indivisible (atomic) operations namely

wait (s)

{

1. wait or P operation (to test)
2. signal or V operation (to increment)

while(s 0);

s--;

}

signal (s)

{

s++;

}

Mutual Exclusion Implementation using semaphore

do

{

wait(mutex);

critical section

remainder section } while (1);

signal(mutex);

Semaphore Implementation

- The semaphore discussed so far requires a busy waiting. That is if a process is in critical-section, the other process that tries to enter its critical-section must loop continuously in the entry cod
- To overcom ns wait and signal sh
  - x Wh value is not positive, the process can block itself. The block operation places the process into a waiting queue associated with the semaphore.
  - A process that is blocked waiting on a semaphore should be restarted when some other process executes a signal operation. The blocked process should be restarted by a wakeup operation which put that process into ready queue.
- x To implemented the semaphore, we define a semaphore as a record as:

typedef struct {

int value;

struct process *L;

} semaphore;

Deadlock & starvation:

Example: Consider a system of two processes , P0 & P1 each accessing two semaphores ,S & Q, set to the value 1.

P0 P1

Wait (S) Wait (Q)

Wait (Q) Wait (S)

Signal(S) Signal (Q)

Signal (Q) Signal(S)

- Suppose that P0 executes wait(S), then P1 executes wait(Q). When P0 executes wait (Q), it must wait until P1 executes signal(Q).Similarly when P1 executes wait(S), it must wait until P0 executes signal(S). Since these signal operations cannot be executed, P0 & P1 are deadlocked.
- Another problem related to deadlock is indefinite blocking or starvation, a situation where a process wait indefinitely within the semaphore. Indefinite blocking may occur if we add or remove processes from the list associated with a semaphore in LIFO order.

Types of Semaphores

- *Counting* semaphore - any positive integer value
- *Binary* semaphore - integer value can range only between 0 and 1

16.Discuss about the issues to be considered in the multithreaded program. (APR/MAY 2015): (MAY/JUNE 2014)(APRIL/MAY2011, MAY/JUNE 2012)

Some of the issues with multithreaded programs are as follows

1. **Increased Complexity**: Multithreaded processes are quite complicated. Coding for these can only be handled by expert programmers.
2. **Complications due to Concurrency**: It is difficult to handle concurrency in multithreaded processes. This may lead to complications and future problems.
3. **Difficult to Identify Errors**: Identification and correction of errors is much more difficult in multithreaded processes as compared to single threaded processes.
4. **Testing Complications**: Testing is a complicated process i multithreaded programs as compared to single threaded programs. This is because defects can be timing related and not easy to identify.
5. **Unpredictable results**: Multithreaded programs can sometimes lead to unpredictable results as they are essentially multiple parts of a program that are running at the same time.
6. **Complications for Porting Existing Code**: A lot of testing is required for porting existing code in multithreading t thread safe need to be rep

17.(i) Describe th

(ii) Provide two programming examples in which multithreading does not provide better performance than a single- threaded solution. (MAY/JUNE 2016)

Context Switch

x When CPU switches to another process, the system must save the state of the old

process and load the saved state for the new process

- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Multithreading Models

Multithreading allows the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process. Therefore, multithreading leads to maximum utilization of the CPU by multitasking.

The main models for multithreading are one to one model, many to one model and many to many model. Details about these are given as follows:

One to One Model

The one to one model maps each of the user threads to a kernel thread. This means that many threads can run in parallel on multiprocessors and other threads can run when one thread makes a blocking system call.

A disadvantage of the one to one model is that the creation of a user thread requires a corresponding kernel thread. Since a lot of kernel threads burden the system, there is restriction on the number of threads in the system.

A diagram that demonstrates the one to one model is given as follows:

Many to One Model

The many to one model maps many of the user threads to a single kernel thread. This model is quite efficient as the user space manages the thread management.

A disadvantage of the many to one model is that a thread blocking system call blocks the entire process. Also, multiple threads cannot run in parallel as only one thread can access the kernel at a time.

A diagram that dem

Many to Many Model

The many to many model maps many of the user threads to a equal number or lesser kernel threads. The number of kernel threads depends on the application or machine.

The many to many does not have the disadvantages of the one to one model or the many to one model. There can be as many user threads as required and their corresponding kernel threads can run in parallel on a multiprocessor.

A diagram that demonstrates the many to many model is given as follows:

UNIT-III
PART-B

1. **Describe the hierarchical paging technique for structuring page tables. (8) (MAY/JUNE 2013)**

Multilevel Paging is a paging scheme which consist of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table are stores actual frame information. Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).

Virtual address:

In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory. So it requires more than one memory access to get the physical address of page frame. One access for each level needed. Each page table entry **except** the last level page table entry contains base address of the next level page table.

Reference to actual page frame:

- Reference to PTE in level 1 page table = PTBR value + Level 1 offset present in virtual address.
- Reference to PTE in level 2 page table = Base address (present in Level 1 PTE) + Level 2 offset (present in VA).
- Reference to PTE in level 3 page table= Base address (present in Level 2 PTE) + Level 3 offset (present in VA).

- Actual page frame address = PTE (present in level 3).
- Generally the page table size will be equal to the size of page.

Assumptions:

Byte addressable memory, and n is the number of bits used to represent virtual address.

Important formula:

Number of entries in page table:
= (virtual address space size) / (page size)
= Number of pages

Virtual address space size:
= 2^n B

Size of page table:

<>= (number of entries in page table)*(size of PTE)

If page table size > desired size then create 1 more level.

Disadvantage:

Extra memory references to access address translation tables can slow programs down by a factor of two or more. Use translation look aside buffer (TLB) to speed up address translation by storing page table entries.

2. **What is the cause for thrashing? How does the system detect thrashing? Once it detects, what can the system do to eliminate this problem? (NOV/DEC 2017, MAY/JUNE 2009)**

Cause for thrashing:

Thrashing is **caused** by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault. The **system can detect thrashing** by evaluating the level of CPU utilization as compared to the level of multiprogramming. It **can** be **eliminated** by reducing the level of multiprogramming.

How does the system detect thrashing?

Thrashing occurs when too many processes are run on a processor at a given time **.....** This **can be detected** by monitoring the page fault frequency and CPU utilization. If increasing the number of processes results in increasing page fault rate and decreasing CPU utilization, then the system is **thrashing.**

3. **Write in detail about Segmentation. (NOV/DEC 2009)**

Segmentation:

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

4. **Write in detail about Segmentation with Paging. (APRIL/MAY 2018, APRIL/MAY2010)**

Segmentation and Paging-

- Paging and Segmentation are the non-contiguous memory allocation techniques.
- Paging divides the process into equal size partitions called as pages.
- Segmentation divides the process into unequal size partitions called as segments.

Segmented Paging-

In segmented paging,

- Process is first divided into segments and then each segment is divided into pages.
- These pages are then stored in the frames of main memory.
- A page table exists for each segment that keeps track of the frames storing the pages of that segment.
- Each page table occupies one frame in the main memory.
- Number of entries in the page table of a segment = Number of pages that segment is divided.
- A segment table exists that keeps track of the frames storing the page tables of segments.
- Number of entries in the segment table of a process = Number of segments that process is divided.

- The base address of the segment table is stored in the segment table base register.

Translating Logical Address into Physical Address-

- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of three parts-

1. Segment Number
2. Page Number
3. Page Offset

- Segment Number a.
- Page Number spe the data.
- Page Offset speci

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Segment table provides the frame number of the frame storing the page table of the referred segment.
- The frame containing the page table is located.

Step-03:

- For the generated page number, corresponding entry is located in the page table.
- Page table provides the frame number of the frame storing the required page of the referred segment.
- The frame containing the required page is located.

Step-04:

- The frame number combined with the page offset forms the required physical address.
- For the generated page offset, corresponding word is located in the page and read.

The following diagram illustrates the above steps of translating logical address into physical address-

Advantages-

The advantages of s

- Segment table co
- It reduces memor
- The size of Page Table is limited by the segment size.
- It solves the problem of external fragmentation.

Disadvantages-

The disadvantages of segmented paging are-

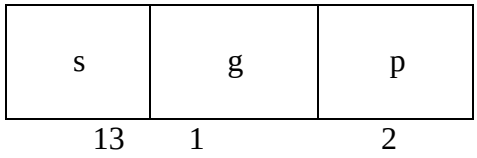
- Segmented paging suffers from internal fragmentation.
- The complexity level is much higher as compared to paging.

5. Explain the segmentation with paging implemented in OS/2 32-bit IBM system. Describe the following algorithms: (APRIL/MAY2010)(April/May2019)

a. First fit
b. Best Fit
c. Worst Fit

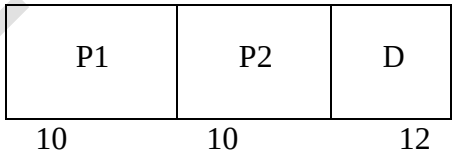
Segmentation with Paging(32 and 64 bit architecture) :

- The IBM OS/2,32 bit version is an operating system running on top of the Intel 386 architecture. The 386 uses segmentation with paging for memory management. The maximum number of segments per process is 16 KB, and each segment can be as large as 4 gigabytes.
- The local-address space of a process is divided into two partitions. o The first partition consists of up to 8 KB segments that are private to that process. o The second partition consists of up to 8KB segments that are shared among all the processes.
- Information about the first partition is kept in the local descriptor table (LDT), information about the second partition is kept in the global descriptor table (GDT).
- Each entry in the LDT and GDT consist of 8 bytes, with detailed information about a particular segment including the base location and length of the segment. The logical address is a pair (selector, offset) where the selector is a16-bit number:



Where s designates the segment number, g indicates whether the segment is in the GDT or LDT, and p deals with protection.

- The base and limit information about the segment in question are used to generate a linear-address.
- First, the limit is used to check for address validity. If the address is not valid, a memory fault is generated, resulting in a trap to the operating system. If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address. This address is then translated into a physical address.
- The linear address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits. Since we page the page table, the page number is further divided into a 10-bit ddress is as follows



- To improve the efficiency of physical memory use. Intel 386 page tables can be swapped to disk. In this case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is in memory or on disk.
- If the table is on disk, the operating system can use the other 31 bits to specify the disk location of the table; the table then can be brought into memory on demand.

(a) First-fit: Allocate the *first* hole that is big enough.

(b) Best-fit: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.

(c) Worst-fit: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole

First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

Fragmentation

External Fragmentation - This takes place when enough total memory space exists to satisfy a

request, but it is n all holes scattered througho

Internal Fragmen ory.

Example: hole = 184 bytes
Process size = 182 bytes.
We are left with a hole of 2 bytes.

6. **Explain how paging supports virtual memory. With a neat diagram explain how logical address is translated into physical address. (NOV/DEC 2012)**

Virtual Memory:

- It is a technique that allows the execution of processes that may not be completely in main memory.
- In practice, most real processes do not need all their pages, or at least not all at once, for several reasons:
  1. Error handling code is not needed unless that specific error occurs, some of which are quite rare.
  2. Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
  3. Certain features of certain programs are rarely used.

Advantages:

- Allows the program that can be larger than the physical memory.
- Separation of user logical memory from physical memory
- Allows processes to easily share files & address space.
- Allows for more efficient process creation.

Virtual memory can be implemented using

- Demand paging
- Demand segmentation

Virtual Memory That is Larger than Physical Memory

Demand Paging:

A demand pen we want to execuentire process into m **per** - Never swaps a

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Transfer of a paged memory to contiguous disk space

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked by checking the bit. Marking a page will have no effect if the process never attempts to access the page. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Valid-Invalid bit

A valid - invalid bit is associated with each page table entry.
Valid bit represents the associated page is in memory.
In-Valid bit represents

(d) invalid page or

(e) valid page but is currently on the disk

Page table when some pages are not in main memory

Advantages

- Programs could be written for a much larger address space (virtual memory space) than physically exists on the computer.
- Because each process is only using a fraction of their total address space, there is more memory left for other programs, improving CPU utilization and system

throughput.

- Less I/O is needed for swapping processes in and out of RAM, speeding things up.

Handling Page Fault

If a page is needed that was not originally loaded up, then a page fault trap is generated, which must be handled in a series of steps:

1. Determine whether the reference is a valid or invalid memory access
2. a)If the reference is invalid then terminate the process.
   b)If the reference is valid then the page has not been yet brought into main memory.
3. Find a free frame.
4. Read the desired page into the newly allocated frame.
5. Reset the page table to indicate that the page is now in memory.
6. Restart the instruction that was interrupted.

Pure demand paging:

- Never
- We co
- When ocess, which faults for the page.
- After this page is bought into the memory, the process continue to execute, faulting as necessary until every page that it needs is in memory.

Performance of demand paging

Let p be the probability of a page fault ($0 \leq p \leq 1$)

- Effective Access Time (EAT)

EAT = (1 - *p*) x ma + *p* x page fault time.

Where ma ・ memory access, p ・ Probability of page fault ($0 \leq p \leq 1$)

- The memory access time denoted ma is in the range 10 to 200 ns.
- If there are no page faults then EAT = ma.

To compute effective access time, we must know how much time is needed to service a page fault.

A page fault causes the following sequence to occur:

1. Trap to the OS
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Check whether the reference was legal and find the location of page on disk.
5. Read the page from disk to free frame.
   a. Wait in a queue until read request is serviced.
   b. Wait for seek time and latency time.
   c. Transfer the page from disk to free frame.
6. While waiting, allocate CPU to some other user.
7. Interrupt from disk.
8. Save registers and process state for other users.
9. Determine that the interrupt was from disk.
7. Reset the page table to indicate that the page is now in memory.
8. Wait for CPU to be allocated to this process again.
9. Restart the instruction that was interrupted.

7. Explain the principles of segmented and paging implemented in memory with a diagram. (NOV/DEC2013)(MAY/JUNE2016)

Paging

(f) It is a memory management scheme that permits the physical address space of a process to be noncontiguous.

(g) It avoids the considerable problem of fitting the varying size memory chunks on to the backing store.

Basic Method

- Divide logical memory into blocks of same size called **"pages"**.
- Divide physical memory into fixed-sized blocks called **"frames"**
- Page size is

Address Translati

- Address ge
  - Pag
    address of each page in physical memory
  - Page offset (*d*) - combined with base address to define the physical memory address that is sent to the memory unit
  - For given logical address space 2*m* and page size *2*n

Paging Hardware

Every address generated by the CPU is divided into two parts: a **page number (p)** and a **page offset (d).** The page number is used as an index into a **page table.** The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

Paging model of logical and physical memory

Allocation

- When a process arrives into the system, its size (expressed in pages) is examined.
- Each page of process needs one frame. Thus if the process requires 'n' pages, at least 'n' frames must be available in memory.
- If 'n' frame
- The 1st pag
  number is p
- Repeat the

Paging example for a 32-byte memory with 4-byte pages

Page size = 4 bytes

Physical memory size = 32 bytes
i.e (4 X 8 = 32 so, 8 pages)

Logical address '0' maps to physical

address 20 i.e ((5 X 4) +0)

Where Frame no = 5, Page size = 4, Offset = 0

When we use a paging scheme, we have no external fragmentation: *Any* free frame can be allocated to a process that needs it. However, we may have some internal fragmentation.

Calculating internal fragmentation

- Page size = 2,048 bytes
- Process size = 72,766 bytes
- 35 pages + 1,086 bytes
- Internal fragmentation of 2,048 - 1,086 = 962 bytes.

Before Al

Frame table: It is able, how many total frameses in the physical memory.

Hardware implementation of Page Table

The hardware implementation of the page table can be done in several ways.

- The page table is implemented as a set of dedicated registers. These registers should be built with very high-speed logic to make the paging-address translation efficient. If the page table is very large (for example, 1 million entries). The use of fast registers to implement the page table is not feasible.

- The page table is kept in main memory. **Page-table base register (PTBR)** points to the page table. In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry - uniquely identifies each process to provide address-space protection for that process

If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory. In addition, add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, the operating system must select one for replacement.

Hit ratio: Percentage of times that a particular page is found in the TLB.
For example hit ra he
time.

Effective Access Ti

- Assume hit ratio is 80%.
- If it takes 20ns to search TLB & 100ns to access memory, then the memory access takes 120ns(TLB hit)
- If we fail to find page no. in TLB (20ns), then we must 1st access memory for page table (100ns) & then access the desired byte in memory (100ns). Therefore
  Total = 20 + 100 + 100
  = 220 ns(TLB miss).

Memory Protection

Memory protection implemented by associating protection bit with each frame

- One bit can define a page to be read-write or read-only. Every reference to memory goes through the page table to find the correct frame number. An attempt to write to a read-only page causes a hardware trap to the operating system
- **Valid-invalid** bit attached to each entry in the page table:
  "valid (v)" indicates that the associated page is in the process logical address space, and is thus a legal page
  "invalid (i)" indicates that the page is not in the process logical address space
- **page-table length register** (PTLR), to indicate the size of the page table. This value is checked against every logical address to verify that the address is in the valid range for the process.

Valid (v) or Invalid (i) Bit In A Page Table

Structures of the Page Table

a) Hierarchical Paging
b) Hashed Page Tables
c) Inverted Page Tables

a) Hierarchical Paging

Break up the Page table into smaller pieces. Because if the page table is too large then it is quite difficult to search the page number.

Example: "Two-Level Paging "

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset

Thus, a logical address is as follows:

Where *p1* is an index into the outer page table, and *p2* is the displacement within the page of the inner page table. This is also Known as forward-mapped page table

Address-Translation Scheme

Address-translation scheme for a two-level 32-bit paging architecture

It requires more number of memory accesses, when the number of levels is increased.

b) Hashed Page Tables

- Hashed page table is used if address spaces greater than 32 bits
- Each entry in hash table contains a linked list of elements that hash to the same location.
- Each entry consists of;
  (a) Virtual page numbers
  (b) Value of mapped page frame.
  (c) Pointer to the next element in the linked list.
- Working Procedure:
  - The virtual page number in the virtual address is hashed into the hash table.
  - Virtual page number is compared to field (a) in the 1st element in the linked list.
  - If there is a match, the corresponding page frame (field (b)) is used to form the desired physical address.

Clustered page table: It is a variation of hashed page table & is similar to hashed page table except that each entry in the hash table refers to several pages rather than a single page.

c)Inverted Page Table

It has one entry for each real page (frame) of memory & each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page. So, only one page table is in the system.

- When a memory reference occurs, part of the virtual address ,consisting of <Process-id, Page-no> is presented to the memory sub-system.
- Then the inverted page table is searched for match:
  (i) If a match is found, then the physical address is generated.
  (ii) If no match is found, then an illegal address access has been attempted.

Merit: Reduce the amount of memory needed.

Demerit: Improve the amount of time needed to search the table when a page reference occurs.

Shared Pages

One advantage of paging is the possibility of sharing common code.

Shared code

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Reentrant code (Pure code): Non-self modifying code. If the code is reentrant, then it never changes during execution. Thus two or more processes can execute the same code at the same time.

Drawback of Paging - Internal fragmentation

In the worst case a process would need n pages plus one byte. It would be allocated n+1 frames resulting in an **internal fragmentation** of almost an entire frame.

Example:

Page size = 2048 bytes

Process size= 72766 bytes

Process needs 35 pages plus 1086 bytes.

It is allocated 36 frames resulting in an internal fragmentation of 962 bytes.

Segmentation:

Memory-management scheme that supports user view of memory.

A program is a collection of segments. A segment is a logical unit such as: Main program, Procedure, Function, Method, Object, Local variables, global variables, Common block, Stack, Symbol table, arrays

Logical View of Segmentation

egmentation Hardware

(h) Logical address consists of a two tuple : **<Segment-number, offset>**

(i) Segment t :

- **Base** -
  memor
- **Limit**

(j) Segment-table base register (STBR) points to the segment table‘s location in memory

(k) Segment-table length register (STLR) indicates number of segments used by a program; Segment number ‘s‘ is legal, if s < STLR

To define an implementation to map two dimensional user-defined addresses into one-dimensional physical addresses. This mapping is effected by a segment table. Each entry in the segment table has a *segment base* and a *segment limit.* The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.

A logical address consists of two parts: a segment number, *s,* and an offset into that segment, *d.* The segment number is used as an index to the segment table. The offset *d* of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond, end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.

EXAMPLE:

For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353.

Sharing of Segments

- Another advan
- Each process has a segment table associated with it, which the dispatcher uses to define the hardware segment table when this process is given the CPU.
- Segments are shared when entries in the segment tables of two different processes point to the same physical location.

8. Explain the various page table structures in detail. **(APRIL/MAY2011)(MAY/JUNE 2014)**

Structures of the Page Table

d) Hierarchical Paging
e) Hashed Page Tables
f) Inverted Page Tables

d) Hierarchical Paging

Break up the Page table into smaller pieces. Because if the page table is too large then it is quite difficult to search the page number.

Example: "Two-Level Paging "

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset

Thus, a logical address is as follows:

Where *p1* is an index into the outer page table, and *p2* is the displacement within the page of the inner page table. This is also Known as forward-mapped page table

Address-Translati

Address-translation scheme for a two-level 32-bit paging architecture

It requires more number of memory accesses, when the number of levels is increased.

e) Hashed Page Tables

- Hashed page table is used if address spaces greater than 32 bits
- Each entry in hash table contains a linked list of elements that hash to the same location.
- Each entry consists of;
  (d) Virtual page numbers
  (e) Value of mapped page frame.
  (f) Pointer to the next element in the linked list.
- Working Procedure:
  - The virtual page number in the virtual address is hashed into the hash table.
  - Virtual page number is compared to field (a) in the 1st element in the linked list.
  - If there is a match, the corresponding page frame (field (b)) is used to form the desired physical address.
  - If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.

Clustered page table: It is a variation of hashed page table & is similar to hashed page table except that each entry in the hash table refers to several pages rather than a single page.

f) Inverted Page Table

It has one entry for each real page (frame) of memory & each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page. So, only one page table is in the system.

- When a memory reference occurs, part of the virtual address ,consisting of <Process-id, Page-no> is presented to the memory sub-system.
- Then the inverted page table is searched for match:
  (iii) If a match is found, then the physical address is generated.
  (iv)If no match is found, then an illegal address access has been attempted.

Merit: Reduce the amount of memory needed.
Demerit: Improve the amount of time needed to search the table when a page reference occurs.

Shared Pages

One advantage of paging is the possibility of sharing common code.

Shared code

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Reentrant code (Pure code): Non-self modifying code. If the code is reentrant, then it never

changes during execution. Thus two or more processes can execute the same code at the same time.

Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Drawback of Pagin

In the worst case frames resulting in an **internal fragmentation** of almost an entire frame.
Example:
Page size = 2048 bytes
Process size= 72766 bytes
Process needs 35 pages plus 1086 bytes.
It is allocated 36 frames resulting in an internal fragmentation of 962 bytes.

9. Write short notes on LRU, FIFO and clock replacement strategies? (APRIL/MAY2010, APRIL/MAY2011)(MAY/JUNE2016)

Page Replacement

- If no frames are free, we could find one that is not currently being used & free it.
- We can free a frame by writing its contents to swap space & changing the page table to indicate that the page is no longer in memory.
- Then we can use that freed frame to hold the page for which the process faulted.

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame
   - If there is a free frame, then use it.
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame
   - Write the victim page to the disk, change the page & frame tables accordingly.
3. Read the desired page into the (new) free frame. Update the page and frame tables.
4. Restart the process

Note:

If no frames are free, two page transfers are required & this situation effectively doubles the page- fault service time.

Modify (dirty) bit:

- It indicates t
- When we se
- If the bit is set, we know that the page has been modified & in this case we must write that page to the disk.
- If the bit is not set, then if the copy of the page on the disk has not been overwritten, then we can avoid writing the memory page on the disk as it is already there.

FIFO page replacement algorithm

- A simple and obvious page replacement strategy is FIFO, i.e. first-in-first-out.
- As new pages are brought in, they are added to the tail of a queue, and the page at the head of the queue is the next victim.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No.of available frames = 3 (3 pages can be in memory at a time per process

No. of page faults = 15

Drawback:

FIFO page replacement algorithm performance is not always good.

To illustrate this, consider the following example:

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- If No. of available frames = 3 then the no. of page faults =9
- If No. of available frames =4 then the no. of page faults =10
- Here the no. of page faults increases when the no. of frames increases .This is called as **Belady's Anomaly.**

LRU(Least Recently Used) page replacement algorithm

The Least Recently Used, algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No. of available frames = 3

No. of page faults = 12

LRU page replacement can be implemented using

1. Counters

- Every page table entry has a time-of-use field and a clock or counter is associated with the CPU.
- The counter or clock is incremented for every memory reference.
- Each time a page is referenced, copy the counter into the time-of-use field.
- When a page needs to be replaced, replace the page with the smallest counter value.

2. Stack

- Keep a stack of page numbers
- Whenever a page is referenced, remove the page from the stack and put it on top of the stack.
- When a page needs to be replaced, replace the page that is at the bottom of the stack.(LRU page)

Use of A Stack to Record The Most Recent Page References

10. Explain any four page replacement algorithms in detail? (NOV/DEC 2009) (NOV/DEC 2013) NOV/DEC2012) (MAY/JUNE2016)

Page Replacement

- If no frames are free, we could find one that is not currently being used & free it.
- We can free a frame by writing its contents to swap space & changing the page table to indicate that the page is no longer in memory.
- Then we can use that freed frame to hold the page for which the process faulted.

Basic Page Replacement

5. Find the location of the desired page on disk
6. Find a free frame
   - If there is a free frame, then use it.
   - If there **m**
     frame
   - Write th
7. Read the des
8. Restart the process

Note:

If no frames are free, two page transfers are required & this situation effectively doubles the page- fault service time.

Modify (dirty) bit:

- It indicates that any word or byte in the page is modified.
- When we select a page for replacement, we examine its modify bit.
  - If the bit is set, we know that the page has been modified & in this case we must write that page to the disk.
  - If the bit is not set, then if the copy of the page on the disk has not been overwritten, then we can avoid writing the memory page on the disk as it is already there.

Page Replacement Algorithms

1. FIFO Page Replacement
2. Optimal Page Replacement
3. LRU Page Replacement
4. LRU Approximation Page Replacement

- We evaluate an algorithm by running it on a particular string of memory references & computing the number of page faults. The string of memory reference is called a reference string. The algorithm that provides less number of page faults is termed to be a good one.

(b) FIFO page replacement algorithm

- A simple and obvious page replacement strategy is FIFO, i.e. first-in-first-out.
- As new pages are brought in, they are added to the tail of a queue, and the page at the head of the queue is the next victim.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No.of available frames = 3 (3 pages can be in memory at a time per process)

No. of page faults = 15

Drawback:

FIFO page replacement algorithm performance is not always good.

To illustrate this, consider the following example:

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- If No. of available frames = 3 then the no. of page faults =9
- If No. of available frames =4 then the no. of page faults =10
- Here the no. of page faults increases when the no. of frames increases .This is called as **Belady's Anomaly.**

(c) Optimal page replacement algorithm

- The Belady's anomaly lead to the search for an optimal page-replacement algorithm, which is simply that which yields the lowest of all possible page-faults, and which does not suffer from Belady's anomaly.
- This algorithm is simply "Replace the page that will not be used for the longest time in the future."

Example:

Reference string:

No.of available fr

No. of page faults = 9

Drawback:

It is difficult to implement as it requires future knowledge of the reference string.

(d) LRU(Least Recently Used) page replacement algorithm

The Least Recently Used, algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No. of available frames = 3

No. of page faults = 12

LRU page replacement can be implemented using

3. Counters

- Every page table entry has a time-of-use field and a clock or counter is associated with the CPU.
- The counter or clock is incremented for every memory reference.
- Each time a page is referenced, copy the counter into the time-of-use field.
- When a page needs to be replaced, replace the page with the smallest counter value.

4. Stack

- Keep a stack of page numbers
- Whenever a page is referenced, remove the page from the stack and put it on top of the stack.
- When a page needs to be replaced, replace the page that is at the bottom of the stack.(LRU page)

Use of A Stack to Record The Most Recent Page References

(e) LRU Approximation Page Replacement

- Reference bit
  - With each page associate a reference bit, initially set to 0
  - When page is referenced, the bit is set to 1
- When a page needs to be replaced, replace the page whose reference bit is 0

- The order of use is not known, but we know which pages were used and which were not used.

(i) Additional Reference Bits Algorithm

- Keep an 8-bit byte for each page in a table in memory.
- At regular intervals, a timer interrupt transfers control to OS.
- The OS shifts reference bit for each page into higher- order bit shifting the other bits right 1 bit and discarding the lower-order bit.

Example:

- If reference bit is 00000000 then the page has not been used for 8 time periods.
- If reference bit is 11111111 then the page has been used atleast once each time period.
- If the reference bit of page 1 is 11000100 and page 2 is 01110111 then page 2 is the LRU page.

(ii) Second Chance Algorithm

- Basic algorithm is FIFO

- When a page has been selected, check its reference bit.
  - If 0 proceed to replace the page
  - If 1 give the page a second chance and move on to the next FIFO page.
  - When a page gets a second chance, its reference bit is cleared and arrival time is reset to current time.
  - Hence a second chance page will not be replaced until all other pages are replaced.

(iii) Enhanced Second Chance Algorithm ·

Consider both reference bit and modify bit ·

There are four possible classes

1. (0,0) - neither recently used nor modified, Best page to replace
2. (0,1) - not recently used but modified, page has to be written out before replacement.
3. (1,0) - recently used but not modified, page may be used again
4. (1,1) - recently used and modified, page may be used again and page has to be written to disk

13. What is thrashing? Explain the working set model in detail. (MAY/JUNE 2009)
Given memory partitions of 100KB, 500KB, 200KB, 300KB and 600KB(in order), how would each of the first-fit, best-fit and worst-fit algorithms place processes of212KB, 417KB, 12KB and 426KB(in order)? Which algorithm makes the most efficient use of memory? (NOV/DEC 2008)
Refer Notes (Unit-3)

14. Explain in briefly and compare, fixed and dynamic memory partitioning schemes.(NOV/DEC2012)

Contiguous Allocation

(l) Main memory usually into two partitions:

- Resident operating system, usually held in low memory with interrupt vector.
- User processes then held in high memory.

(m) Single-partition allocation

- Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
- Relocation register contains value of smallest physical address; limit register contains range of logical addresses - each logical address must be less than the limit register.

Memory Protection

(n) It should consider;

- Protecting the OS from user process.
- Protecting user processes from one another.

(o) The above protection is done by **"Relocation-register & Limit-register scheme —**

(p) Relocation register contains value of smallest physical address i.e base value.

(q) Limit register contains range of logical addresses - each logical address must be less than the limit register

A base and a limit register define a logical address space

HW address protection with base and limit registers

Each process is contained in a single contiguous section of memory. There are two methods namely:

1. Fixed - Partition Method
2. Variable - Partition Method

Fixed - Partition Method

(r) Divide memory into fixed size partitions, where each partition has exactly one process.

(s) The drawback is memory space unused within a partition is wasted.(eg.when process size < partition size)

Variable-partition method:

(t) Divide memory into variable size partitions, depending upon the size of the incoming process.

(u) When a process terminates, the partition becomes available for another process.

(v) As processes complete and leave they create holes in the main memory.

(w) ***Hole*** - block of available memory; holes of various size are scattered throughout memory.

15. Consider the following page reference string (MAY/JUNE 2012) (APR/MAY 2015)
1,2,3,4,2,1,5,6,2,1,3,7,6,3,2,1,3,6.
How many page faults would occur for the following replacement algorithms, assuming one, two, three and four frames? LRU replacement, FIFO replacement Optimal replacement
Refer notes .. Page No.409-420

16. When do page faults occur? Consider the reference string: (NOV/DEC 2017)
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?

FIFO:

Frames	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
1	**1**	**1**	**1**	**4**	**4**	**4**	**4**	**6**	**6**	**6**	**6**	**3**	**3**	**3**	**3**	**2**	**2**	**2**	**2**	**6**
2		**2**	**2**	**2**	**2**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**7**	**7**	**7**	**7**	**1**	**1**	**1**	**1**
3			**3**	**3**	**3**	**3**	**5**	**5**	**5**	**1**	**1**	**1**	**1**	**6**	**6**	**6**	**6**	**6**	**3**	**3**

Page Hit=4

Total 14 page faults (Hit Time)

page fault rate (Miss rate) = 14 / 20 = 0.7

Frames	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
1	**1**	**1**	**1**	**1**	**1**	**1**	**5**	**5**	**5**	**5**	**5**	**3**	**3**	**3**	**3**	**3**	**1**	**1**	**1**	**1**
2		**2**	**2**	**2**	**2**	**2**	**2**	**6**	**6**	**6**	**6**	**6**	**7**	**7**	**7**	**7**	**7**	**7**	**3**	**3**
3			**3**	**3**	**3**	**3**	**3**	**3**	**2**	**2**	**2**	**2**	**2**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
4				**4**	**4**	**4**	**4**	**4**	**4**	**1**	**1**	**1**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**2**

Page Hit=6

Total page faults=14

Page fault rate = 14 / 20 = 0.7

No. of page faults does not increase / Decrease when more frames are allocated to the process.

LRU:

Frame=3

Frames	**1**	**2**	**3**	**4**	**2**															
0	**1**	**1**	**1**	**4**	**4**															
1		**2**	**2**	**2**	**2**	**2**	**2**	**6**	**6**	**6**	**6**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
2			**3**	**3**	**3**	**1**	**1**	**1**	**2**	**2**	**2**	**2**	**2**	**6**	**6**	**6**	**1**	**1**	**1**	**6**

No. of pages = 20

Page fault= 15

Page Hit = 5

Page fault rate = Page fault / No. of pages

=15 / 20 = 0.75

OPTIMAL:

Frame=3

Frames	**1**	**2**	**3**	**4**	**2**	**1**	**5**	**6**	**2**	**1**	**2**	**3**	**7**	**6**	**3**	**2**	**1**	**2**	**3**	**6**
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**6**
1		**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**7**	**7**	**7**	**2**	**2**	**2**	**2**	**2**
2			**3**	**4**	**4**	**4**	**5**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**1**	**1**	**1**	**1**

Page fault= 11

Page Hit = 9

17. Explain the concept of demand paging in detail with neat diagram (MAY/JUNE 2014) (NOV/DEC 2013)

Demand Paging

A demand paging system is quite similar to a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper called pager. **Lazy Swapper** - Never swaps a page into memory unless that page will be needed

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Transfer of a paged memory to contiguous disk space

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked by checking the bit. Marking a page will have no effect if the process never attempts to access the page. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Valid-Invalid bit

A valid - invalid bit is associated with each page table entry. Valid bit represents the associated page is in memory.

In-Valid bit represents

(x) invalid page or

(y) valid page but is currently on the disk

Page table when some pages are not in main memory

Advantages

(a) Programs could be written for a much larger address space (virtual memory space) than physically exists on the computer.

(b) Because each process is only using a fraction of their total address space, there is more memory left for other programs, improving CPU utilization and system throughput. ·

(c) Less I/O is needed for swapping processes in and out of RAM, speeding things up. ·

Handling Page Fault

If a page is needed that was not originally loaded up, then a page fault trap is generated, which must be handled in a series of steps:

1. Determine wh
2. a)If the refere
   b)If the refere
3. Find a free frame.
4. Read the desired page into the newly allocated frame.
5. Reset the page table to indicate that the page is now in memory.
6. Restart the instruction that was interrupted.

Pure demand paging

- Never bring a page into memory until it is required.
- We could start a process with no pages in memory**.**
- When the OS sets the instruction pointer to the 1st instruction of the process, whichis on the non-memory resident page, then the process immediately faults for the page.
- After this page is bought into the memory, the process continue to execute, faulting as necessary until every page that it needs is in memory

Performance of demand paging

Let p be the probability of a page fault ($0 \leq p \leq 1$)

- Effective Access Time (EAT)
  EAT = (1 - *p*) x ma + *p* x page fault time.
  Where ma · memory access, p · Probability of page fault ($0 \leq p \leq 1$)

- The memory
- If there are no

To compute effective access time, we must know how much time is needed to service a page fault.

18.Why are translation look-aside buffers important? Explain the details stored in a TLB table entry**? (APRIL/MAY 2017, MAY/JUNE 2014) Refer notes ..Page No.364-377**

19. Write short notes on Memory Mapped Files. (APR/MAY 2015)

(i) Consider the following page reference string: 1,2,3,2,5,6,3,4,6,3,7,3,1,5,3,6,3,4,2,4,3,4,5,1 Indicate page faults and calculate total number of page faults and successful ratio for FIFO, optimal and LRU algorithms. Assume there are four frames and initially all the frames are empty. Refer notes ..Page No.409-420

(ii) Explain the effect of thrashing. (NOV/DEC 2015)

Discuss the given memory management techniques with diagrams

- **Partition Allocation Methods**
- **Paging and Translation Look-aside Buffer. (NOV/DEC 2015)**

Effect of thrashing

If any process that does not have "enough" frames, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.

High paging activity is called **thrashing**. A process is thrashing if it is spending more time paging than executing.

- If a process does not have —enough pages, the page-fault rate is very high. This leads to:

- low CPU utilization operating system thinks that it needs to increase the degree of multiprogramming
- another process is added to the system When the CPU utilization is low, the OS increases the degree of multiprogramming.
- If global replacement is used then as processes enter the main memory they tend to steal frames belonging to other processes.
- Eventually all processes will not have enough frames and hence the page fault rate becomes very high.
- Thus swapping in and swapping out of pages only takes place.
- This is the cause of thrashing.

To **limit thrashing**,
prevent thrashing, the

1. Working Set Strategy
2. Page Fault Frequency

Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses - each logical address must be less than the limit register.

Memory Protection

- It should consider;
  - Protecting the OS from user process.
  - Protecting user processes from one another.
- The above protection is done by **"Relocation-register & Limit-register scheme —**
- Relocation register contains value of smallest physical address i.e base value.
- Limit register contains range of logical addresses - each logical address must be less than the limit register

A base and a limit register define a logical address space

Each process is contai
two methods namely:

3. Fixed - Partition Method
4. Variable - Partition Method

Fixed - Partition Method

- Divide memory into fixed size partitions, where each partition has exactly one process.
- The drawback is memory space unused within a partition is wasted.(eg.when process size < partition size)

Variable-partition method:

- Divide memory into variable size partitions, depending upon the size of the incoming process.
- When a process terminates, the partition becomes available for another process.
- As processes complete and leave they create holes in the main memory.
- ***Hole*** - block of available memory; holes of various size are scattered throughout memory.

Dynamic Storage-Allocation Problem:

How to satisfy a request of size "n" from a list of free holes?

Solution:

- **First-fit**: Allocate the *first* hole that is big enough.
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole

First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

Fragmentation

External Fragmentation - This takes place when enough total memory space exists to satisfy a request, but it is not contiguous i.e., storage is fragmented into a large number of small holes scattered throughout the main memory.

Internal Fragmentation - Allocated memory may be slightly larger than requested memory.

Example: hole = 184 bytes Process
size = 182 bytes.
We are left with a hole

Solutions

1. **Coalescing:** Merge the adjacent holes together.

2. **Compaction:** Move all processes towards one end of memory, hole towards other end of memory, producing one large hole of available memory. This scheme is expensive as it can be done if relocation is dynamic and done at execution time.

3. Permit the logical address space of a process to be **non-contiguous**. This is achieved through two memory management schemes namely **paging** and **segmentation**.

UNIT-IV

PART-B

1. **Explain the different disk scheduling algorithms with examples. (APRIL/MAY 2018, NOV/DEC 2019, APRIL/MAY 2010, MAY/JUNE 2012, APRIL/MAY 2011, MAY/JUNE 2013, MAY/JUNE 2014)**

Disk Scheduling and Management

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives,

1. A fast access time and
2. High disk bandwidth.

- The **access time** has two major components;

□ The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.

□ The **rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head.

- The disk **bandwidths** the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives,
1. A fast access time and
2. High disk bandwidth.
□ The **access time** has two major components;
✓ The **seek tim** ining the desired sector.
✓ The **rotationa** sired sector to the disk head.
□ The disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

1. FCFS Scheduling:

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider, for example, a disk queue with requests for I/O to blocks on cylinders

I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67,

If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for **a total head movement of 640 cylinders**. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

2. SSTF (shortest-seek-time-first)Scheduling

Service all the requests close to the current head position, before moving the head far away to service other requests. That is selects the request with the minimum seek time from the current head position.

Total head movement = 236 cylinders

3. SCAN Scheduling

The disk head starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk

4. C-SCAN Scheduling

Variant of SCAN designed to provide a more uniform wait time. It moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

5. LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In this, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.

2. Explain and compare FCFS, SSTF, C-SCAN and C-LOOK disk scheduling algorithms with examples. (NOV/DEC 2012)

1. FCFS Scheduling:

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider, for example, a disk queue with requests for I/O to blocks on cylinders

I/O to blocks on cylinders

98, 183, 37, 122, 14, 124, 65, 67,

if the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for **a total head movement of 640 cylinders**. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

2. SSTF (shortest-seek-time-first)Scheduling

Service all the requests close to the current head position, before moving the head far away to service other requests. That is selects the request with the minimum seek time from the current head position.

Total head movement = 236 cylinders

3. C-SCAN Scheduling

Variant of SCAN designed to provide a more uniform wait time. It moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

4. LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In this, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.

3. Write short notes on disk management. (APRIL/MAY 2019, NOV/DEC 2009) (April/May 2019)

Disk Management

1. Disk Formatting:

Before a disk can store data, the sector is divided into various partitions. This process is called low- level formatting or physical formatting. It fills the disk with a special data structure for each sector. The data structure for a sector consists of

- ✓ Header,
- ✓ Data area (usually 512 bytes in size), and
- ✓ Trailer.

The header and trailer contain information used by the disk controller, such as a sector number and an

Error-correcting

This formattin

1. Test the disk and

2. To initialize the mapping from logical block numbers

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps.

(a) The first step is **Partition** the disk into one or more groups of cylinders. Among the partitions, one partition can hold a copy of the OS‘s executable code, while another holds user files.

(b) The second step is **logical formatting** .The operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory.

2. Boot Block:

For a computer to start running-for instance, when it is powered up or rebooted-it needs to have an initial program to run. This initial program is called bootstrap program & it should be simple. It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.

To do its job, the bootstrap program

1. Finds the operating system kernel on disk,

2. Loads that kernel into memory, and

3. Jumps to an initial address to begin the operating-system execution. The bootstrap is stored in read-only memory **(ROM).**

Advantages:

1. ROM needs no initialization.

2. It is at a fixed location that the processor can start executing when powered up or reset.
3. It cannot be infected by a computer virus. Since, ROM is read only.

The full bootstrap program is stored in a partition called the **boot blocks**, at a fixed location on the disk. A disk that has a boot partition is called a **boot disk or system disk**.

The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.

Bootstrap loader: load the entire operating system from a non-fixed location on disk, and to start the operating system running.

3. Bad Blocks:

The disk with defected sector is called as bad block.Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: "Handled manually

If blocks go bad during normal operation, a **special program** must be run manually to search for the bad b locks usually are lost.

Method 2: "sector sparing or forwarding"

The controller maintains a list of bad blocks on the disk. Then 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 operating system tries to read logical block 87.
2. The controller calculates the ECC and finds that the sector is bad.
3. It reports this finding to the operating system.
4. The next time that the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.
5. After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

Method 3: "sector slipping"

For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

4. Write short notes on file system in Linux. (NOV/DEC 2009) (NOV/DEC 2014)

File System Storage-File Concepts

File Concept

- A file is a named collection of related information that is recorded on secondary storage.
- From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.

Examples of files:

• A text file is a sequence of characters organized into lines (and possibly pages). A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements. An object file is a sequence of bytes organized into blocks understandable by the system's linker.

An executable file is a series of code sections that the loader can bring into memory and execute.

File Attributes

- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number identifies the file within the file system. It is the non-human readable
- **Type:** This inform
- **Location:** This inf device.
- **Size:** The current size of the file (in bytes, words or blocks)and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing and so on.
- **Time, date and user identification:** This information may be kept for creation, last modification and last use. These data can be useful for protection, security and usage monitoring.

File Operations

- Creating a file
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file
- Truncating a file

Access Methods

1. **Sequential Access**
    a. The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

The bulk of the operations on a file is reads and writes. A read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, a write appends to the end of the file and advances to the end of the newly written material (the new end of file). Such a file can be reset to the beginning and, on some systems, a program may be able to skip forward or back ward n records, for some integer n-perhaps only for n=1. Sequential access is based on a tape model of a file, and works as well on sequential-access devices as it does on random - access ones.

2. Direct Access

Another method is direct access (or relative access). A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order. The direct-access methods is based on a disk model of a file, since disks allow random access to any file block.

For direct access, the file is viewed as a numbered sequence of blocks or records. A direct-access file allows arbitrary blocks to be read or written. Thus, we may read block 14, then read block 53, and then write block7. There are no restrictions on the order of reading or writing for a direct-access file.

5. Write an elaborate note on RAID and RAID Levels. (APRIL/MAY 2010, MAY/JUNE 2012, NOV/DEC 2012, MAY/JUNE 2013)

RAID (Redundant Arrays of Independent Disks)

RAID, or "Redundant Arrays of Independent Disks" is a technique which makes use of a combination of multiple disks instead of using a single disk for increased performance, data redundancy or both.

Key evaluation points for a RAID System

- **Reliability:** How many disk faults can the system tolerate?

- **Availability:** What fraction of the total session time is a system in uptime mode, i.e. how available is the system for actual use?
- **Performance:** How good is the response time? How high is the throughput (rate of processing work)? Note that performance contains a lot of parameters and not just the two.
- **Capacity:** Given a set of N disks each with B blocks, how much useful capacity is available to the user?

RAID is very transparent to the underlying system. This means, to the host system, it appears as a single big disk presenting itself as a linear array of blocks. This allows older technologies to be replaced by RAID without making too many changes in the existing code.

Different RAID levels

RAID-0 (Striping)

- Blocks are "striped" across disks.
  In the figure, blocks "0,1,2,3" form a stripe.

Instead of placing just one block into a disk at a time, we can work with two (or more) blocks placed into a disk before moving on to the next one.

Evaluation:

- Reliability:
  There is no duplication of data. Hence, a block once lost cannot be recovered.
- Capacity: N*B
  The entire space is being used to store data. Since there is no duplication, N disks each having B blocks are fully utilized.

RAID-1 (Mirroring)

- More than one copy of each block is stored in a separate disk. Thus, every block has two (or more)copies,lyingondifferentdisks.

  The above figure shows a RAID-1 system with mirroring level 2.
- RAID 0 was unable to tolerate any disk failure. But RAID 1 is capable of reliability.

Evaluation:

Assume a RAID system with mirroring level 2.

Reliability:1toN/2

- 1 disk failure can be handled for certain, because blocks of that disk would have duplicates on some other disk. If we are lucky enough and disks 0 and 2 fail, then again this can be handled as the blocks of these disks have duplicates on disks 1 and 3. So, in the best case, N/2 disk failures can be handled. ·

Capacity:N*B/2

Only half the space is being used to store data. The other half is just a mirror to the already stored data.

RAID-4 (Block-Level Striping with Dedicated Parity)

- Instead of duplicating data, this adopts a parity-based approach.
- Parity is calculated using a simple XOR function. If the data bits are 0,0,0,1 the parity bit is XOR(0,0,0,1) = 1. If the data bits are 0,1,1,0 the parity bit is XOR(0,1,1,0) = 0. A simple approach is that even number of ones results in parity 0, and an odd number of ones results in parity 1.
  Assume that in the above figure, C3 is lost due to some disk failure. Then, we can recompute the data bit stored in C3 by looking at the values of all the other columns and the parity bit. This allows us to recover lost data.

Evaluation:

- **Reliability:**1
  RAID-4 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data.

- **Capacity**:(N-1)*B

  One disk in the system is reserved for storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

RAID-5 (Block-Level Striping with Distributed Parity)

- This is a slight modification of the RAID-4 system where the only difference is that the parity rotates among the drives.
- This was introduc **Evaluation:**
- **Reliability**:1

  RAID-5 allows recovery of at most 1 disk failure (because of the way parity works). If more than one disk fails, there is no way to recover the data. This is identical to RAID-4.
- **Capacity:**(N-1)*B

  Overall, space equivalent to one disk is utilized in storing the parity. Hence, (N-1) disks are made available for data storage, each disk having B blocks.

6. Explain the services provided by Kernel I/O subsystem. (APRIL/MAY 2017, APRIL/MAY 2010, APRIL/MAY 2011, NOV/DEC2012, MAY/JUNE 2013)

Kernel I/O Subsystem

Kernels provide many services related to I/O.

- Oneway that the I/O subsystem improves the efficiency of the computer is by scheduling I/O operations.
- Another way is by using storage space in main memory or on disk, via techniques called buffering, caching, and spooling.

I/O Scheduling:

To determine a good order in which to execute the set of I/O requests.

Uses:

a) It can improve overall system performance,

b) It can share device access fairly among processes, and

c) It can reduce the average waiting time for 1/0 to complete.

Implementation: OS developers implement scheduling by maintaining a queue of requests for each device.

1. When an application issues a blocking I/O system call,

2. The request is placed on the queue for that device.

3. The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

Buffering:

Buffer: A memory area that stores data while they are transferred between two devices or between a device and an application.

Reasons for buffering:

a) To cope with a speed mismatch between the producer and consumer of a data stream.

b) To adapt between devices that have different data-transfer sizes.

c) To support copy semantics for application I/O.

number of bytes to write.

After the system call returns, what happens if the application changes the contents of the buffer? With copy semantics, the version of the data written to disk is guaranteed to be the version at the

time of the application system call, independent of any subsequent changes in the application's buffer.

A simple way that the operating system can guarantee copy semantics is for the write() system call to copy the application data into a kernel buffer before returning control to the application. The disk write is performed from the kernel buffer, so that subsequent changes to the application buffer have no effect.

Caching

A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original

Cache vs buffer: A buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere.

When the kernel receives a file I/O request,

1. The kernel first accesses the buffer cache to see whether that region of the file is already available in main memory.

2. If so, a physical disk I/O can be avoided or deferred. Also, disk writes are accumulated in the buffer cache for several seconds, so that large transfers are gathered to allow efficient write schedules.

Spooling and Device Reservation:

Spool: A buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. A printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together

The os provides a control interface that enables users and system administrators ;

a) To display the queue,

b) To remove unwanted jobs before those jobs print,

c) To suspend printing while the printer is serviced, and so on. Device reservation - provides exclusive access to a device

- System calls for allocation and de-allocation
- Watch out for deadlock

Error Handling:

- An operating system that uses protected memory can guard against many kinds of hardware and application errors.
- OS can recover from disk read, device unavailable, transient write failures
- Most return an erro

System error logs

7. Explain the file **MAY 2010)**

Allocation Methods

- The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- There are three major methods of allocating disk space:

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

1. Contiguous Allocation

- The contiguous - allocation method requires each file to occupy a set of contiguous blocks on the disk.
- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b,. b+1, b+2,…., b+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.

Disadvantages:

1. Finding space for a new file.

- The contiguous disk space-allocation problem suffer from the problem of external fragmentation. As file are allocated and deleted, the free disk space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, no one of which is large enough to store the data.

2. Determining how

- If we allocate too little space to a file, we may find that file cannot be extended. The other possibility is to find a larger hole, copy the contents of the file to the new space, and release the previous space. This series of actions may be repeated as long as space exists, although it can be time - consuming. However, in this case, the user never needs to be informed explicitly about what is happening; the system continues despite the problem, although more and more slowly.
- Even if the total amount of space needed for a file is known in advance pre-allocation may be inefficient.
- A file that grows slowly over a long period (months or years) must be allocated enough space for its final size, even though much of that space may be unused for a long time the file, therefore has a large amount of internal fragmentation.

To overcome these disadvantages:

- Use a modified contiguous allocation scheme, in which a contiguous chunk of space called as an **extent** is allocated initially and then, when that amount is not large enough another chunk of contiguous space an extent is added to the initial allocation.
- Internal fragmentation can still be a problem if the extents are too large, and external fragmentation can be a problem as extents of varying sizes are allocated and deallocated.

2. Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally bock 25.
- Each block contains a pointer to the next block. These pointers are not made available to the user.
- There is no external fragmentation with linked allocation, and any free block on the free space list can be used to satisfy a request.
- The size of a file does not need to the declared when that file is created. A file can continue to grow as long as free blocks are available consequently, it is never necessary to compacts disk space.

Disadvantages:

· **Used effectively o**

• To find the ith blo pointers until we get to the ith disk seek

consequently, it is inefficient to support a direct- access capability for linked allocation files.

· **Space required for the pointers**

• If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.

• Solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate the clusters rather than blocks. For instance, the file system may define a clusters as 4 blocks, and operate on the disk in only cluster units.

· **Reliability**

• Since the files are linked together by pointers scattered all over the disk hardware failure might result in picking up the wrong pointer. This error could result in linking into the free- space list or into another file. Partial solution are to use doubly linked lists or to store the file names in a relative block number in each block; however, these schemes require even more over head for each file.

File Allocation Table (FAT)

• An important variation on the linked allocation method is the use of a file allocation table(FAT).

• This simple but efficient method of disk- space allocation is used by the MS-DOS and OS/2 operating systems.

• A section of disk at beginning of each partition is set aside to contain the table.

• The table has entry for each disk block, and is indexed by block number.

• The FAT is much as is a linked list.

• The directory entry contains the block number the first block of the file.

• The table entry indexed by that block number contains the block number of the next block in the file.

• This chain continues until the last block which has a special end - of - file value as the table entry.

• Unused blocks are indicated by a 0 table value.

• Allocating a new block file is a simple matter of finding the first 0 - valued table entry, and replacing the previous end of file value with the address of the new block.

• The 0 is replaced with the end - of - file value, an illustrative example is the FAT structure for a file consisting of disk blocks 217,618, and 339.

3. Indexed Allocation

• Linked allocation solves the external - fragmentation and size- declaration problems of contiguous allocation.

• Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with
the blocks themselves all over the disk and need to be retrieved in order.

• Indexed allocation solves this problem by bringing all the pointers together into one location: the **index block**.

• Each file has its own index block, which is an array of disk - block addresses.

• The ith entry in the

• The directory cont

• To read the ith blo ad the desired block this scheme is similar to the paging scheme.

• When the file is created, all pointers in the pointers in the index block are set to nil. when the ith block is first written, a block is obtained from the free space manager, and its address is put in the ith index - block entry.

• Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space.

Disadvantages

1. Pointer Overhead

• Indexed allocation does suffer from wasted space. The pointer over head of the index block is generally greater than the pointer over head of linked allocation.

2. Size of Index block

If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue:

• **Linked Scheme:** An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we may link together several index blocks.

• **Multilevel index:** lock to point to a set of secon

• **Combined schem**

o Another alternative, k in the file's in node.

o The first 12 of these pointers point to direct blocks; that is for small (no more than 12 blocks) files do not need a separate index block

o The next pointer is the address of a single indirect block.

□ The single indirect block is an index block, containing not data, but rather the addresses of blocks that do contain data.

o Then there is a double indirect block pointer, which contains the address of a block that contain pointers to the actual data blocks. The last pointer would contain pointers to the actual data blocks. o The last pointer would contain the address of a triple indirect block.

8. Explain the role of Access Matrix for protection in files. (APRIL/MAY 2010) File Protection

(i) Need for file protection.

• When information is kept in a computer system, we want to keep it safe from **physical damage** (reliability) and **improper access** (protection).

• Reliability is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or though computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.

• File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally. Bugs in the file-system software can also cause file contents to be lost.

• Protection can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multi-user system, however, other mechanisms are needed.

(ii) Types of Access

• Complete protectio

• Free access is prov

• Both approaches a

• What is needed is **controlled access**.

• Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

1. **Read:** Read from the file.
2. **Write:** Write or rewrite the file.
3. **Execute:** Load the file into memory and execute it.
4. **Append:** Write new information at the end of the file.
5. **Delete:** Delete the file and free its space for possible reuse.
6. **List:** List the name and attributes of the file.

9. Explain directory subsystem (APRIL/MAY 2011)

A **directory** is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner.

There are several logical structures of a directory, these are given below.

Single-level directory -

Single level directory is simplest directory structure.In it all files are contained in same directory which make it easy to support and understand.

A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name . if two users call their dataset test, then the unique name rule violated.

1. **Advantages:**
   - Since it is a single directory, so its implementation is very easy.
   - If files are smaller in size, searching will faster.
   - The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

   Disadvantages:
   - There may change of name collision because two files can not have the same name.
   - Searching will become time taking if directory will large.
   - In this cannot group the same type of files together.

2. **Two-level**

directory -

As we have seen, a single level directory often leads to confusion of files names among different users. the solution to this problem is to create a separate directory for each user.

In the two-level directory structure, each user has there own *user files directory (UFD)*. The UFDs has similar structures, but each lists only the files of a single user. system's *master file directory (MFD)* is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

Advantages:
- We can give full path like /User-name/directory-name/.
- Different users can have same directory as well as file name.
- Searching of files become more easy due to path name and user-grouping.

Disadvantages:

1.
   - A user is not allowed to share files with other users.
   - Still it not very scalable, two files of the same type cannot be grouped together in the same user.

2. **Tree-structured directory -**

Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.

This generalizatio their files accordingly.

A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path.

Advantages:

- Very generalize, since full path name can be given.
- Very scalable, the probability of name collision is less.
- Searching becomes very easy, we can use both absolute path as well as relative.

Disadvantages:

- Every file does not fit into the hierarchical model, files may be saved into multiple directories.
- We can not share files.
- It is inefficient, because accessing a file may go under multiple directories.

Acyclic graph directory -

An acyclic graph is a graph with no cycle and allows to share subdirectories and files. The same file or subdirectories may be in two different directories. It is a natural generalization of the tree-structured directory.

- It is used in the situation like when two programmers are working on a joint project and they need to access files. m from other projects and file they want the subdirectories t be shared. So here we use Ac
- It is the point to note that shared file is not the same as copy file . If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.

Advantages:

1.
   - We can share files.
   - Searching is easy due to different-different paths.

Disadvantages:

- We share the files via linking, in case of deleting it may create the problem,
- If the link is softlink then after deleting the file we left with a dangling pointer.
- In case of hardlink, to delete a file we have to delete all the reference associated with it.

2. **General graph directory structure -**

   In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.

   The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.

Advantages:

- It allows cycles.
- It is more flexible than other directories structure.

Disadvantages:

- It is more costly than others.
- It needs garbage collection.

10. Explain the various file directory structures. (NOV/DEC 2012) Directory and Disk Structure

There are five directory structures. They are

1. Single-level directory
2. Two-level directory
3. Tree-Structured directory
4. Acyclic Graph directory
5. General Graph directory

1. Single - Level Directory

- The simplest directory structure is the single- level directory.
- All files are contained in the same directory.

- **Disadvantage:**

➢ When the number of files increases or when the system has more than one user, since all files are in the same directory, they must have unique names.

2. Two - Level Directory

- In the two level directory structures, each user has her own user file directory (UFD).
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is

indexed by user name or account number, and each entry points to the UFD for that user.

- When a user refers to a particular file, only his own UFD is searched.
- Thus, different users may have files with the same name.
- Although the two - level directory structure solves the name-collision problem

Disadvantage:

➢ Users cannot create their own sub-directories.

3. Tree - Structured Directory

- A tree is the most common directory structure.
- The tree has a root directory. Every file in the system has a unique path name.
- A **path name** is the path from the root, through all the subdirectories to a specified file.
- A directory (or sub directory) contains a set of files or sub directories.
- A directory is simply another file. But it is treated in a special way.

All directories have the same internal format

- One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.
- Path names can be of two types: absolute path names or relative path names.
- 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.

4. Acyclic Graph Directory.

- An acyclic graph is a graph with no cycles.
- To implement shared files and subdirectories this directory structure is used.
- An acyclic - graph directory structure is more flexible than is a simple tree structure, but it is also more complex. In a system where sharing is implemented by symbolic link, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed.
- Another approach to deletion is to preserve the file until all references to it are deleted. To implement this

approach, we must have some mechanism for determining that the last reference to the file has been deleted.

11. (i) **Explain the different file access methods in detail. (MAY/JUNE 2014)**

(iii) Types of Access

- Complete protection is provided by prohibiting access.

• Free access is provided with no protection.

• Both approaches are too extreme for general use.

• What is needed is **controlled access**.

• Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

7. **Read:** Read from the file.
8. **Write:** Write or rewrite the file.
9. **Execute:** Load the file into memory and execute it.
10. **Append:** Write new information at the end of the file.
11. **Delete:** Delete the file and free its space for possible reuse.
12. **List:** List the name and attributes of the file.

(iv) Access Control

• Associate with each file and directory an access-control list (ACL) specifying the user name and the types of access allowed for each user.

• When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs and the user job is denied access to the file.

• This technique has two undesirable consequences:

□ Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.

□ The directory entry, previously of fixed size, now needs to be of variable size, resulting in more complicated space management.

• To condense the length of the access control list, many systems recognize three classifications of users in connection with ea

- ➢ **Owne**
- ➢ **Grou** roup, or work group.
- ➢ **Universe:** All other users in the system constitute the universe.

12.Describe the two level and acyclic graph schemes for defining the logical structure of a directory. (MAY/JUNE 2013)

Two - Level Directory

• In the two level directory structures, each user has her own user file directory (UFD).

• When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is

indexed by user name or account number, and each entry points to the UFD for that user.

• When a user refers to a particular file, only his own UFD is searched.

• Thus, different users may have files with the same name.

• Although the two - level directory structure solves the name-collision problem

Disadvantage:

➢ Users cannot create their own sub-directories.

Acyclic Graph Directory.

• An acyclic graph is a graph with no cycles.

• To implement shared files and subdirectories this directory structure is used.

• An acyclic - graph directory structure is more flexible than is a simple tree structure, but it is also more complex. In a system where sharing is implemented by symbolic link, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed.

• Another approach to deletion is to preserve the file until all references to it are deleted. To implement this

approach, we must have some mechanism for determining that the last reference to the file has been deleted.

13.**Explain the Linked list and indexed file allocation methods with neat diagram. Mention their advantages and disadvantages. (MAY/JUNE 2013)(April/May 2019)**

Linked Allocation

• Linked allocation solves all problems of contiguous allocation.

• With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered anywhere on the disk.

• The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally bock 25.

• Each block contains a pointer to the next block. These pointers are not made available to the user.

• There is no external fragmentation with linked allocation, and any free block on the free space list can be used to satisfy a request.

• The size of a file does not need to the declared when that file is created. A file can continue to grow as long as free blocks are available consequently, it is never necessary to compacts disk space.

Disadvantages:

1.Used effectively only for sequential access files.

• To find the ith block of a file, we must start at the beginning of that file, and follow the pointers until we get to the ith block. Each aces to a pointer requires a disk read, and sometimes a disk seek consequently, it is inefficient to support a direct- access capability for linked allocation files.

2.Space required for the pointers

• If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.

• Solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate the clusters rather than blocks. For instance, the file system may define a clusters as 4 blocks, and operate on the disk in only cluster units.

3.Reliability

• Since the files are ure might result in picking up t ce list or into another file. Partial solution are to use doubly linked lists or to store the file names in a relative block number in each block; however, these schemes require even more over head for each file.

File Allocation Table(FAT)

• An important variation on the linked allocation method is the use of a file allocation table(FAT).

• This simple but efficient method of disk- space allocation is used by the MS-DOS and OS/2 operating systems.

• A section of disk at beginning of each partition is set aside to contain the table.

- The table has entry for each disk block, and is indexed by block number.
- The FAT is much as is a linked list.
- The directory entry contains the block number the first block of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until the last block which has a special end - of - file value as the table entry.
- Unused blocks are indicated by a 0 table value.
- Allocating a new block file is a simple matter of finding the first 0 - valued table entry, and replacing the previous end of file value with the address of the new block.
- The 0 is replaced with the end - of - file value, an illustrative example is the FAT structure for a file consisting of disk blocks 217,618, and 339.

Indexed Allocation

- Linked allocation solves the external - fragmentation and size- declaration problems of contiguous allocation.
- Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order.
- Indexed allocation solves this problem by bringing all the pointers together into one location: the **index block**.
- Each file has its own index block, which is an array of disk - block addresses.
- The ith entry in the index block points to the ith block of the file.
- The directory contains the address of the index block .
- To read the ith block, we use the pointer in the ith index - block entry to find and read the desired block this scheme is similar to the paging scheme .
- When the file is created, all pointers in the pointers in the index block are set to nil. when the ith block is first written, a block is obtained from the free space manager, and its address is put in the ith index - block entry.
- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space.

Disadvantages

1.Pointer Overhead

- Indexed allocation does suffer from wasted space. The pointer over head of the index block is generally greater than the pointer over head of linked allocation.

2. Size of Index block

If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue:

- **Linked Scheme:** An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we may link together several index blocks.
- **Multilevel index:** A variant of the linked representation is to use a first level index block to point to a set of second - level index blocks.
- **Combined scheme:**

o Another alternative, used in the UFS, is to keep the first, say, 15 pointers of the index block in the file's in node.

o The first 12 of these pointers point to direct blocks; that is for small (no more than 12 blocks) files do not need a separate index block The next pointer is the address of a single indirect block.

o The single indirect block is an index block, containing not data, but rather the addresses of blocks that do contain data.

o Then there is a double indirect block pointer, which contains the address of a block that contain pointers to the actual data blocks. The last pointer would contain pointers to the actual data blocks. o The last pointer would contain the address of a triple indirect block.

UNIT-V

PART-B

1. Explain in detail about the concepts of Linux system.

The Linux System

- An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs. The Linux open source operating system, or Linux OS, is a freely distributable, cross-platform operating system based on UNIX.
- The Linux consist of a kernel and some system programs. There are also some application programs for doing work. The kernel is the heart of the operating system which provides a set of tools that are used by system calls.
- The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by *Linus Torvalds.*
- A Linux-based system is a modular Unix-like operating system. It derives much of its basic design from principles established in UNIX. Such a system uses a monolit

hic kernel which handles process control,
networking, and peripheral and file system access.

Important features of Linux Operating System

Portable - Portability means software can work on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.

Open Source - Linux source code is freely available and it is community based development project.

Multi-User & **Multiprogramming** - Linux is a multiuser system where multiple users can access system resources like memory/ ram/ application programs at same time. Linux is a multiprogramming system means multiple applications can run at same time.

- Hierarchical File System - Linux provides a standard file structure in which system files/ user file are arranged.

- **Shell** - Linux provides a special interpreter program which can be used to execute commands of the operating system.
- **Security** - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** - Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It is consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

System Library - System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implements most of the functionalities of the operating system and do not requires kernel module's code access rights.

System Utility - System Utility programs are responsible to do specialized, individual level tasks

Installed components of a Linux system include the following:

- A **bootloader** is a program that loads the Linux kernel into the computer's main memory, by being executed by the computer when it is turned on and after the firmware initialization is performed.
- An **init** program is the first process launched by the Linux kernel, and is at the root of the process tree.
- **Software libraries**, which contain code that can be used by running processes. The most commonly used software library on Linux systems, the GNU C Library (glibc), C standard library and Widget toolkits.

User interface programs such as command shells or windowing environments. The user interface, also known as the shell, is either a command-line interface (CLI), a graphical user interface (GUI), or through controls attached the associated hardware.

Architecture

Linux System Architecture is consists of following layers

1. **Hardware la** tc).
2. **Kernel** - Core vides low level serv
3. **Shell** - An interface to kernel, hiding complexity of kernel's functions from users. Takes commands from user and executes kernel's functions.
4. **Utilities** - Utility programs giving user most of the functionalities of an operating systems.

2. Explain in detail about virtualization.

Virtualization

Virtualization refers to the act of creating a virtual (rather than actual) version of something, including a virtual computer hardware platform, operating system (OS), storage device, or computer network resources.

Hardware virtualization or *platform virtualization* refers to the creation of a virtual machine that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources. *Hardware virtualization* hides the physical characteristics of a computing platform from users,

For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine.

Hardware Virtualization

Benefits of Virtualization

1. Instead of deploying several physical servers for each service, only one server can be used. Virtualization let multiple OSs and applications to run on a server at a time. Consolidate hardware to get vastly higher productivity from fewer servers.
2. If the preferred operating system is deployed as an image, so we needed t.
4. **Increased uptime:** Most server virtualization platforms offer a number of advanced features that just aren't found on physical servers which increases servers' uptime. Some of features are live migration, storage migration, fault tolerance, high availability, and distributed resource scheduling.
5. **Reduce capital and operating costs:** Server consolidation can be done by running multiple virtual machines (VM) on a single physical server. Fewer servers means lower capital and operating costs.

Architecture - Virtualization

The heart of virtualization is the "virtual machine" (VM), a tightly isolated software container with an operating system and application inside. Because each virtual machine is completely separate and independent, many of them can run simultaneously on a single computer. A thin layer of software called a hypervisor decouples the virtual machines from the host and dynamically allocates computing resources to each virtual machine as needed.
This architecture redefines your computing equation and delivers:

Many applications on each server: As each virtual machine encapsulates an entire machine, many applications and operating systems can run on a single host at the same time.

Maximum server utilization, minimum server count: Every physical machine is used to its full capacity, allowing you to significantly reduce costs by deploying fewer servers overall.

- **Faster, easier application and resource provisioning:** As self-contained software files, virtual machines can be manipulated with copy-and-paste ease. Virtual machines can even be transferred from one physical server to another while running, via a process known as live migration. ·

3. Write about LINUX architecture and LINUX kernel with neat sketch. (Nov/Dec 2015)

The Linux System

- An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs. The Linux open source operating system, or Linux OS, is a freely distributable, cross-platform operating system based on UNIX. ·
- The Linux consist of a kernel and some system programs. There are also some application programs f es a set of tools tha

- The defini l first released on 5 October 1991 by *Linus Torvalds.*
- A Linux-based system is a modular Unix-like operating system. It derives much of its basic design from principles established in UNIX. Such a system uses a monolithic kernel which handles process control, networking, and peripheral and file system access.

Important features of Linux Operating System

Portable - Portability means software can work on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.

Open Source - Linux source code is freely available and it is community based development project.

Multi-User & **Multiprogramming** - Linux is a multiuser system where multiple users can access system resources like memory/ ram/ application programs at same time. Linux is a multiprogramming system means multiple applications can run at same time

- Hierarchical Fi les/ user file are ar

- **Shell** - Linux provides a special interpreter program which can be used to execute commands of the operating system.
- **Security** - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** - Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It is consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** - System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implements most of the functionalities of the operating system and do not requires kernel module's code access rights.

System Utility - System Utility programs are responsible to do specialized, individual level tasks

Installed components of a Linux system include the following:

- A **bootloader** is a program that loads the Linux kernel into the computer's main memory, by being exec n is performed.
- An **init** pro e process tree.

- **Software libraries**, which contain code that can be used by running processes. The most commonly used software library on Linux systems, the GNU C Library (glibc), C standard library and Widget toolkits. ·

User interface programs such as command shells or windowing environments. The user interface, also known as the shell, is either a command-line interface (CLI), a graphical user interface (GUI), or through controls attached the associated hardware.

Architecture

Linux System Architecture is consists of following layers

5. **Hardware layer** - Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
6. **Kernel** - Core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
7. **Shell** - An interface to kernel, hiding complexity of kernel's functions from users. Takes commands from user and executes kernel's functions.
8. **Utilities** - Utility programs giving user most of the functionalities of an operating systems.

4. Discuss the process and Memory Management in Linux .(April/May 2019)

PROCESS MAN

A Program does no ecution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPUs.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

Attributes of a process

The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process. Attributes which are stored in the PCB are described below.

1. Process ID

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

2. Program counter

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

3. Process State

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. We will discuss about them later in detail.

4. Priority

Every process has ocesses gets the CPU first.

5. General Purpose Registers

Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

6. List of open files

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

7. List of open devices

OS also maintain the list of all open devices which are used during the execution of the process.

Operations on the Process

1. Creation

Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for the execution.

2. Scheduling

Out of the many processes present in the ready queue, the Operating system chooses one process and start executing

ExecutionOnce the process is scheduled for the execution, the processor starts executing it. Process may come to the blocked or wait state during the execution then in that case the processor starts executing the other processes.

4. Deletion/killing

Once the purpose of the process gets over then the OS will kill the process. The Context of the process (PCB) will be deleted and the process gets terminated by the Operating system.

Process Schedulers

Operating system uses various schedulers for the process scheduling described below.

1. **Long term scheduler**
   Long term scheduler is also known as job scheduler. It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.
   Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.
   If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.
2. **Short term scheduler**
   Short term scheduler is also known as CPU scheduler. It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.
   A scheduling algorithm is used to select which job is going to be dispatched for the execution.
   The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.

This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

3. Medium term scheduler

Medium term sche ocesses
needs some IO tim ning to
waiting.

Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.

It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

Process Queues

The Operating system manages various types of queues for each of the process states. The PCB related to the process is also stored in the queue of the same state. If the Process is moved from one state to another state then its PCB is also unlinked from the corresponding queue and added to the other state queue in which the transition is made.

There are the following queues maintained by the Operating system.

1. Job Queue

In starting, all the processes get stored in the job queue. It is maintained in the secondary memory. The long term scheduler (Job scheduler) picks some of the jobs and put them in the primary memory.

2. Ready Queue

3. **Waiting Queue**
   When the process needs some IO operation in order to complete its execution, OS changes the state of the process from running to waiting. The context (PCB) associated with the process gets stored on the waiting queue which will be used by the Processor when the process finishes the IO.

MEMORY MANAGEMENT

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space.**

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.

- The user program deals with virtual addresses; it never sees the real physical addresses

Static vs Dynamic Loading

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

Static vs Dynamic Linking

As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and

linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.

Memory Allocation

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

FragmentationAs processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of

S.N.	Fragmentation & Description
	External fragmentation Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
	Internal fragmentation Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

7.Explain the architecture of iOS. Discuss the media and service layers clearly.(April/May2019)

Lower layers gives the basic services which all application relies on and higher level layer gives sophisticated graphics and interface related services.

Apple provides most of its system interfaces in special packages called frameworks. A framework is a directory that holds a dynamic shared library that is .a files, related resources like as header files, images, and helper apps required to support that library. Every layer have a set of Framework which the developer use to construct the applications.

1. Core OS Layer:

The Core OS layer holds the low level features that most other technologies are built upon.

- Core Bluetooth Framework.
- Accelerate Framework.
- External Accessory

Framework.

- Security Services framework.
- Local Authentication

framework.

64-Bit support from IOS7 supports the 64 bit app development and enables the application to run

faster.

2. Core Services Layer

some of the Important Frameworks available in the core services layers are detailed:

- **Address book framework** - Gives programmatic access to a contacts database of user.
- **Cloud Kit framework** - Gives a medium for moving data between your app and iCloud.
- **Core data Fra**

Controller app.

· **Core Foundation framework -** Interfaces that gives fundamental data management and service features for iOS apps.

· **Core Location framework -** Gives location and heading information to apps.

· **Core Motion Framework -** Access all motion based data available on a device. Using this core motion framework Accelerometer based information can be accessed.

· **Foundation Framework -** Objective C covering too many of the features found in the Core Foundation framework

· **Healthkit framework -** New framework for handling health-related information of user

· **Homekit framework -** New framework for talking with and controlling connected devices in a user's home.

· **Social framework -** Simple interface for accessing the user's social media accounts.

· **StoreKit framework -** Gives support for the buying of content and services from inside your iOS apps, a feature known asIn-App Purchase.

3. Media Layer: Graphics, Audio and Video technology is enabled using the Media Layer. **Graphics Framework**:

· **UIKit Graphics -** It describes high level support for designing images and also used for animating the content of your views.

· **Core Graphics framework -** It is the native drawing engine for iOS apps and gives support for custom 2D vector and image based rendering.

· **Core Animation -** It is an initial technology that optimizes the animation experience of your apps.

· **Core Images -** gives advanced support for controlling video and motionless images in a nondestructive way

· **OpenGl ES and GLKit -** manages advanced 2D and 3D rendering by hardware accelerated interfaces

Audio Framework:

· **Media Player Framework -** It is a high level framework which gives simple use to a user's iTunes library and support for playing playlists.

· **AV Foundation -** It is an Objective C interface for handling the recording and playback of
audio and video.

· **OpenAL -** is an industry standard technology for providing audio.

Video Framework

· **AV Kit -** framework gives a collection of easy to use interfaces for presenting

video. · **AV Foundation -** gives advanced video playback and recording capability.

· **Core Media -** framework describes the low level interfaces and data types for operating
media.

Cocoa Touch Layer

· **EventKit framework -** gives view controllers for showing the standard system interfaces
for seeing and altering calendar related events

· **GameKit Framework -** implements support for Game Center which allows users share
their game related information online

· **iAd Framework -** allows you deliver banner-based advertisements from your app.

· **MapKit Framework -** gives a scrollable map that you can include into your user interface
of app.

· **PushKitFramework -** provides registration support for VoIP apps.

· **Twitter Framework -** supports a UI for generating tweets and support for creating URLs to
access the Twitter service.

· **UIKit Framework -** gives vital infrastructure for applying graphical, event-driven apps in
iOS. Some of the Important functions of UI Kit framework: