CBCS SCHEME

USN | | | | | | | | | | |

MMC104

# First Semester MCA Degree Examination, Dec.2024/Jan.2025
## Operating System

Time: 3 hrs. Max. Marks: 100

***Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.***
***2. M : Marks , L: Bloom's level , C: Course outcomes.***

| | | Module – 1 | M | L | C |
|---|---|---|---|---|---|
| Q.1 | a. | What is the significance of Operating System? Illustrate various services provided by the Operating System. | 10 | L2 | CO1 |
| | b. | What is the purpose of system calls? Describe different types of system calls used in Operating system with examples. | 10 | L2 | CO1 |
| | | **OR** | | | |
| Q.2 | a. | Illustrate the following operating system architectures with a neat diagram:<br>(i) Microkernel<br>(ii) Layered | 10 | L2 | CO1 |
| | b. | Illustrate with a neat diagram various states of process. Also discuss the significance of process control block (PCB). | 10 | L2 | CO1 |
| | | **Module – 2** | | | |
| Q.3 | a. | "CPU scheduling ensures proper execution of processes". Justify. Illustrate different scheduling criteria used by CPU scheduling algorithms. | 10 | L2 | CO1 |
| | b. | Discuss how dining philosophers problem is solved using semaphores. | 10 | L3 | CO1 |
| | | **OR** | | | |
| Q.4 | a. | What do you mean by Critical Section Problem? Explain the solution to the critical-section problem using mutex locks. | 10 | L2 | CO1 |
| | b. | Consider the set of processes with Arrival Time, CPU burst time (in milliseconds) and priority as shown below. (Lower number represents higher priority).<br>Draw the Gantt Chart and calculate the Average waiting time and Average Turnaround time using<br>1) SJF Scheduling (Non Pre-emptive)<br>2) Priority Scheduling (Non Pre-emptive)<br><br>(Note: Consider Arrival Time for both algorithms.) | 10 | L3 | CO1 |

| Process | Arrival Time | Burst Time | Priority |
|---|---|---|---|
| P1 | 0 | 10 | 3 |
| P2 | 1 | 1 | 1 |
| P3 | 2 | 2 | 4 |
| P4 | 3 | 1 | 5 |
| P5 | 4 | 5 | 2 |





| | | Module – 3 | M | L | C |
|---|---|---|---|---|---|
| Q.5 | a. | Illustrate deadlocks with their necessary conditions. | 10 | L2 | CO2 |
| | b. | Describe the working principles of Banker's algorithm for the following snapshot and find either the system is in safe state or not. | 10 | L2 | CO2 |
| | | **OR** | | | |
| Q.6 | a. | Discuss deadlock detection with a neat diagram. | 10 | L2 | CO2 |
| | b. | Explain different methods used for recovering from a deadlock in an operating system. | 10 | L2 | CO2 |
| | | **Module – 4** | | | |
| Q.7 | a. | Describe in detail the concept of Paging with a neat diagram. | 10 | L3 | CO3 |
| | b. | Differentiate between internal and external fragmentation. | 10 | L2 | CO3 |
| | | **OR** | | | |
| Q.8 | a. | Consider the page reference string: 1,0,7,1,0,2,1,2,3,0,3,2,4,0,3,6,2,1 for a memory with three frames. Determine the number of page faults using the FIFO, Optimal, and LRU replacement algorithms. Which algorithm is most efficient? | 10 | L3 | CO3 |
| | b. | Interpret the concepts of demand paging with neat diagram. | 10 | L2 | CO3 |
| | | **Module – 5** | | | |
| Q.9 | a. | Illustrate the following access methods.<br>i) Sequential access<br>ii) Direct access | 08 | L2 | CO3 |
| | b. | Illustrate in detail the various operations performed on a file. | 08 | L2 | CO3 |
| | c. | Explain the following:<br>i) Bit vector ii) Linked list | 04 | L2 | CO3 |
| | | **OR** | | | |
| Q.10 | a. | Illustrate various levels of directory structures. | 10 | L2 | CO3 |
| | b. | List the different file allocation methods and explain any two methods in detail. | 10 | L2 | CO3 |

| | Allocation | | | | Max | | | | Available | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | A | B | C | D | A | B | C | D |
| $P_0$ | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 5 | 2 | 0 |
| $P_1$ | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 | | | | |
| $P_2$ | 1 | 3 | 5 | 4 | 2 | 3 | 5 | 6 | | | | |
| $P_3$ | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 | | | | |
| $P_4$ | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 | | | | |



* * * * *

**FIRST SEMESTER MCA DEGREE EXAMINATION, DEC 2024/JAN 2025**

**MMC104-OPERATING SYSTEM**

**Module-1**

**Q1 a. What is the significance of operating system? Illustrate various services provided by the operating system.**

An Operating System (OS) is a vital component of any computer system. It acts as an intermediary between the user and the computer hardware and is responsible for the management of hardware resources. The OS enables efficient execution of application programs, facilitates user interaction, and ensures system security and stability.

An operating system is software that acts as an intermediary between the user and computer hardware. It is a program with the help of which we are able to run various applications. It is the one program that is running all the time. Every computer must have an operating system to smoothly execute other programs.

The OS coordinates the use of the hardware and application programs for various users. It provides a platform for other application programs to work. The operating system is a set of special programs that run on a computer system that allows it to work properly. It controls input-output devices, execution of programs, managing files, etc.

**Services of Operating System**

- Program execution
- Input Output Operations
- Communication between Process
- File Management
- Memory Management
- Process Management
- Security and Privacy
- Resource Management
- User Interface
- Networking
- Error handling
- Time Management

**Program Execution**

It is the Operating System that manages how a program is going to be executed. It loads the program into the memory after which it is executed. The order in which they are executed depends on the CPU Scheduling Algorithms. A few are FCFS, SJF, etc. When the program is in execution, the Operating System also handles deadlock i.e. no two processes come for execution at the same time. The Operating System is responsible for the smooth execution of both user and system programs. The Operating System utilizes various resources available for the efficient running of all types of functionalities.

**Input Output Operations**

Operating System manages the input-output operations and establishes communication between the user and device drivers. Device drivers are software that is associated with hardware that is being managed by the OS so that the sync between the devices works properly. It also provides access to input-output devices to a program when needed.

**Communication Between Processes**

The Operating system manages the communication between processes. Communication between processes includes data transfer among them. If the processes are not on the same computer but connected through a computer network, then also their communication is managed by the Operating System itself.

**File Management**

The operating system helps in managing files also. If a program needs access to a file, it is the operating system that grants access. These permissions include read-only, read-write, etc. It also provides a platform for the user to create, and delete files. The Operating System is responsible for making decisions regarding the storage of all types of data or files, i.e, floppy disk/hard disk/pen drive, etc. The Operating System decides how the data should be manipulated and stored.

**Memory Management**

Let's understand memory management by OS in simple way. Imagine a cricket team with limited number of player . The team manager (OS) decide whether the upcoming player will be in playing 11 ,playing 15 or will not be included in team , based on his performance . In the same way, OS first check whether the upcoming program fulfil all requirement to get memory space or not ,if all things good, it checks how much memory space will be sufficient for program and then load the program into memory at certain location. And thus , it prevents program from using unnecessary memory.

**Process Management**

Let's understand the process management in unique way. Imagine, our kitchen stove as the (CPU) where all cooking(execution) is really happen and chef as the (OS) who

uses kitchen-stove(CPU) to cook different dishes(program). The chef(OS) has to cook different dishes(programs) so he ensure that any particular dish(program) does not take long time(unnecessary time) and all dishes(programs) gets a chance to cooked(execution) .The chef(OS) basically scheduled time for all dishes(programs) to run kitchen(all the system) smoothly and thus cooked(execute) all the different dishes(programs) efficiently.

**Security and Privacy**

- **Security :** OS keep our computer safe from an unauthorized user by adding security layer to it. Basically, Security is nothing but just a layer of protection which protect computer from bad guys like viruses and hackers. OS provide us defenses like firewalls and anti-virus software and ensure good safety of computer and personal information.
- **Privacy :** OS give us facility to keep our essential information hidden like having a lock on our door, where only you can enter and other are not allowed . Basically , it respect our secrets and provide us facility to keep it safe.

**Resource Management**

System resources are shared between various processes. It is the Operating system that manages resource sharing. It also manages the CPU time among processes using CPU Scheduling Algorithms. It also helps in the memory management of the system. It also controls input-output devices. The OS also ensures the proper use of all the resources available by deciding which resource to be used by whom.

**User Interface**

User interface is essential and all operating systems provide it. Users either interacts with the operating system through the command-line interface or graphical user interface or GUI. The command interpreter executes the next user-specified command.

A GUI offers the user a mouse-based window and menu system as an interface.

**Networking**

This service enables communication between devices on a network, such as connecting to the internet, sending and receiving data packets, and managing network connections.

**Error Handling**

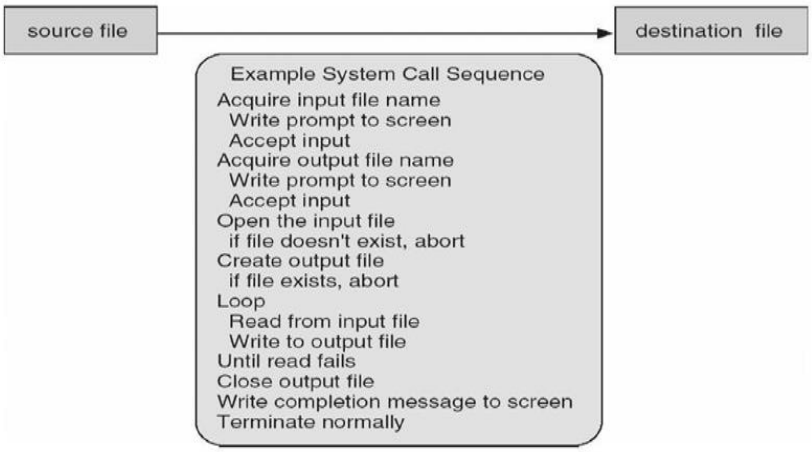
The Operating System also handles the error occurring in the CPU, in Input-Output devices, etc. It also ensures that an error does not occur frequently and fixes the errors. It also prevents the process from coming to a deadlock. It also looks for any type of error or bugs that can occur while any task. The well-secured OS sometimes also acts as a countermeasure for preventing any sort of breach of the Computer System from any external source and probably handling them.

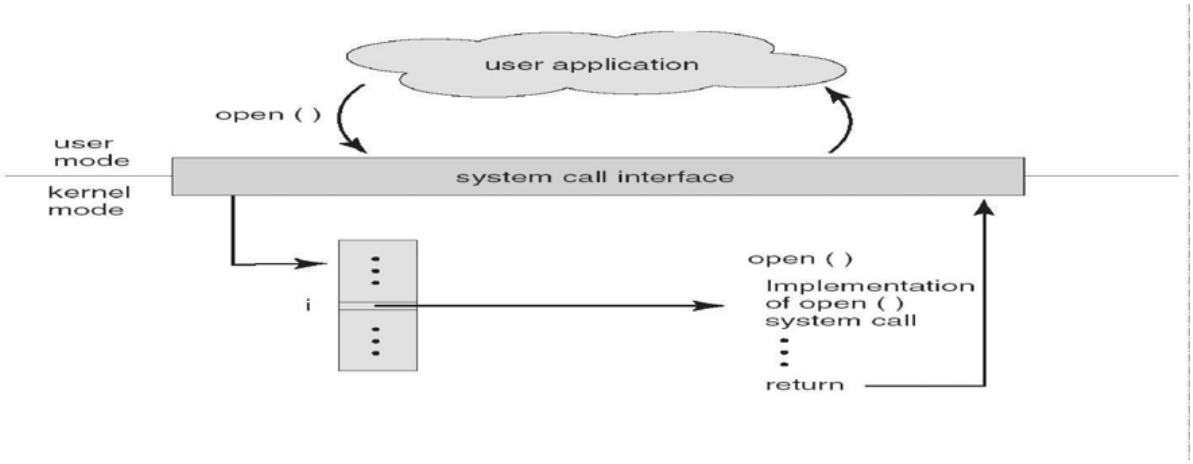
**Time Management**

Imagine traffic light as (OS), which indicates all the cars(programs) whether it should be stop(red)=>(simple queue), start(yellow)=>(ready queue),move(green)=>(under execution) and this light (control) changes after a certain interval of time at each side of the road(computer system) so that the cars(program) from all side of road move smoothly without traffic.

**Q1.b. What is the purpose of system calls? Describe different types of system calls used in operating system with example?**

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call usen
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)
- Why use APIs rather than system calls?(Note that the system-call names used throughout this text are generic)
  **Example of System Calls**

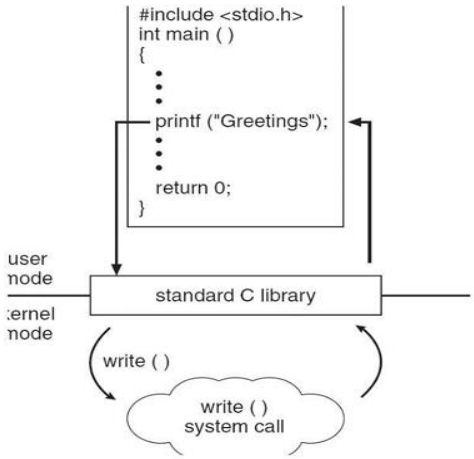





**System Call Implementation**

● Typically, a number associated with each system call
● System-call interface maintains a table indexed according to these numbers
● The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
● The caller need know nothing about how the system call is implemented
● Just needs to obey API and understand what OS will do as a result call
● Most details of OS interface hidden from programmer by API Managed by run-time support library (set of functions built into libraries included with compiler)

**API – System Call – OS Relationship**




System Call Parameter Passing

● Often, more information is required than simply identity of desired system call
● Exact type and amount of information vary according to OS and call
● Three general methods used to pass parameters to the OS

● Simplest: pass the parameters in registers In some cases, may be more parameters than registers
● Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register This approach taken by Linux and Solaris
● Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
● Block and stack methods do not limit the number or length of parameters being passed Parameter Passing

Types of System Calls

● Process control
● File management
● Device management
● Information maintenance
● Communications
● Protection

**Process Control**

A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated. The dump is written to disk and may be examined by a debugger—a system program designed to aid the programmer in finding and correcting bugs-—to determine the cause of the problem. Under either normal or abnormal circumstances, the operating system must transfer control to the invoking command interpreter. The command interpreter then reads the next command. In an interactive system, the command interpreter simply continues with the next command; it is assumed that the user will issue an appropriate command to respond to any error.

**File Management**

We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition (rewinding or skipping to the end of the file, for example). Finally, we need to close the file, indicating that we are no longer using it. We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system. In addition, for either files or directories, we need to be able to determine the values of various attributes and perhaps to reset them if necessary. File attributes include the file name, a file type, protection codes, accounting information, and so on.At least two system calls, get file attribute and set file attribute, are required for this function. Some operating systems provide many more calls, such as calls for file move and copy.

**Device Management**

A process may need several resources to execute—main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and

control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available. The various resources controlled by the operating system can be thought of as devices. Some of these devices are physical devices (for example, tapes), while others can be thought of as abstract or virtual devices (for example, files). If there are multiple users of the system, the system may require us to first request the device, to ensure exclusive use of it. After we are finished with the device, we release it. These functions are similar to the open and close system calls for files.

**Information Maintenance**

Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current t I m e and date . Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on. In addition, the operating system keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information (get process attributes and set process attributes) .

**Communication**

There are two common models of inter process communication: the message passing model and the shared-memory model. In the message-passing model, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly through a common mailbox. Before communication can take place, a connection must be opened. The name of the other communicator must be known, be it another process on the same system or a process on another computer connected by a communications network. Each computer in a network has a host name by which it is commonly known. A host also has a network identifier, such as an IP address. Similarly, each process has a process name, and this name is translated into an identifier by which the operating system can refer to the process. The get host id and get processid system calls do this translation. The identifiers are then passed to the general purpose open and close calls provided by the file system or to specific open connection and close connection system calls, depending on the system's model of communication.

In the shared-memory model, processes use shared memory creates and shared memory attaches system calls to create and gain access to regions of memory owned by other processes. Recall that, normally, the operating system tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction.

They can then exchange information by reading and writing data in the shared areas. The form of the data and the location are determined by the processes and are not under the operating system's control. The processes are also

responsible for ensuring that they are not writing to the same location simultaneously.

**Q2 a. Illustrate the following operating system architectures with a neat diagram :**

**i) Microkernel**

The kernel became large and difficult to manage. In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach. This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs. The result is a smaller kernel. Micro kernels provide minimal process and memory management, in addition to a communication facility. The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space. One benefit of the microkernel approach is ease of extending the operating system. All new services are added to user space and consequently do not require modification of the kernel. When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel. The resulting operating system is easier to port from one hardware design to another. The microkernel also provides more security and reliability, since most services are running as user rather than kernel processes. If a service fails, the rest of the operating system remains untouched.

**ii) Layered**

The operating system can then retain much greater control over the computer and over the applications that make use of that computer. Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems. Under the top down approach, the overall functionality and features are determined and are separated into components. Information hiding is also important, because it leaves programmers free to implement the low-level routines as they see fit, provided that the external interface of the routine stays unchanged and that the routine itself performs the advertised task.

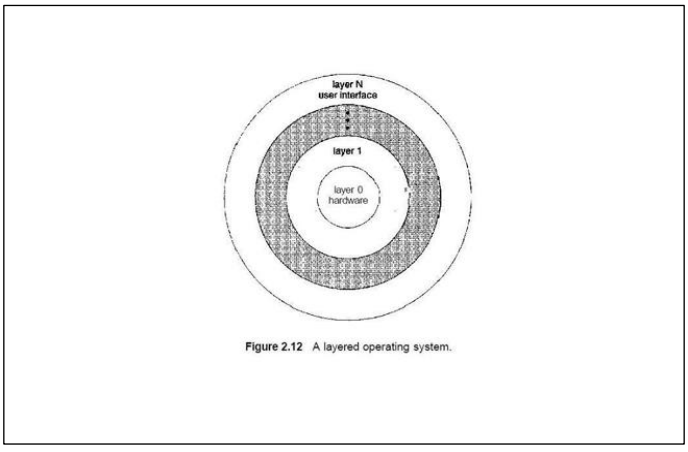


Figure 2.12 A layered operating system.

A system can be made modular in many ways. One method is the layered approach, in which the operating system is broken up into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data. A typical operating-system layer— say, layer M— consists of data structures and a set of routines that can be invoked by higher-level layers. Layer M, in turn, can invoke operations on lower-level layers.

The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers.

This Approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware (which is assumed correct) to implement its functions. Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Thus, the design and implementation of the system is simplified.

Each layer is implemented with only those operations provided by lower level layers. A layer does not need to know how these operations are implemented; it needs to know only what these operations do. Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

The major difficulty with the layered approach involves appropriately defining the various layers. The backing-store driver would normally be above the CPU scheduler, because the driver may need to wait for I/O and the CPU can be rescheduled during this time. A final problem with layered implementations is that they tend to be less efficient than other types. For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.

**Q2.b. Illustrate with a neat diagram various states of process. Also discuss the significance of process control block(PCB).**

**Process State** A Process has 5 states.

Each process may be in one of the following states –

**1. New** - The process is in the stage of being created.

**2. Ready** - The process has all the resources it needs to run. It is waiting to be assigned to the processor.

**3. Running** – Instructions are being executed.

**4. Waiting** - The process is waiting for some event to occur. For example, the process may be waiting for keyboard input, disk access request, inter process messages, a timer to go off, or a child process to finish.

**5. Terminated -** The process has completed its execution.

---

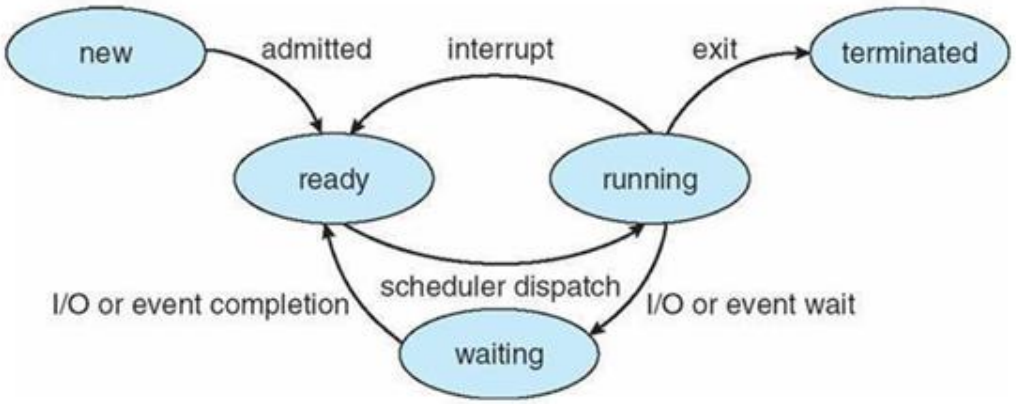



Figure: Diagram of process state

**Process Control Block**

For each process there is a Process Control Block (PCB), which stores the process specific information as shown below –

• **Process State** – The state of the process may be new, ready, running, waiting, and so on.

• **Program counter –** The counter indicates the address of the next instruction to be executed for this process.

• **CPU registers -** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

• **CPU scheduling information-** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

• **Memory-management information –** This includes information such as the value of the base and limit registers, the page tables, or the segment tables.

• **Accounting information –** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

• **I/O status information** – This information includes the list of I/O devices allocated to the process, a list of open files, and so on. The PCB simply serves as the repository for any information that may vary from process to process.

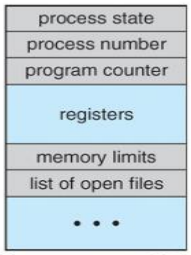



Figure: Process control block (PCB)

CPU Switch from Process to Process

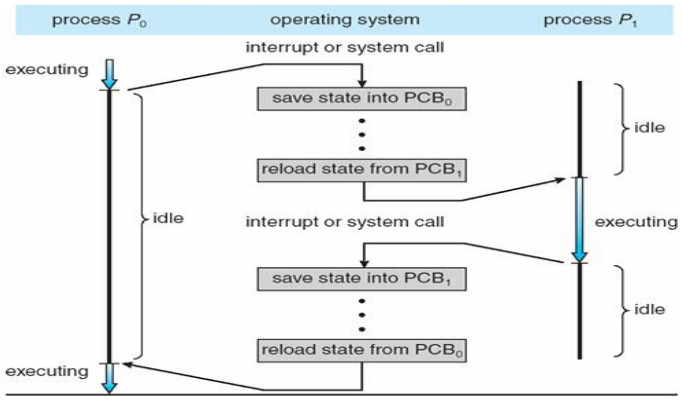



Figure: Diagram showing CPU switch from process to

## Module-2

**Q3 a. "CPU Scheduling ensures proper execution of process". Justify. Illustrate a different scheduling criteria used by CPU scheduling algorithms.**

CPU scheduling is a fundamental operating system function that determines which process will get the CPU next when multiple processes are ready to run. It ensures efficient and fair allocation of CPU time, thereby enabling concurrent process execution and maximizing CPU utilization.

Without CPU scheduling:

- Some processes may suffer from starvation.
- System throughput may degrade.
- Real-time constraints may be violated.
- Overall performance and responsiveness may drop.

Hence, CPU scheduling ensures proper execution by:

- Allowing multiple processes to share the CPU fairly.
- Providing timely CPU access to all processes.
- Supporting multitasking and multiprogramming.
- Improving responsiveness and minimizing delays.

**CPU Scheduling Criteria**

A CPU scheduling algorithm tries to maximize and minimize the following:

**Maximize**

**CPU utilization:**CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

**Throughput:** The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

**Minimize**

**Waiting time:** Waiting time is an amount that specific process needs to wait in the ready queue.

**Response time:** It is an amount to time in which the request was submitted until the first response is produced.

**Turnaround Time:** Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

**Interval Timer**

Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

Most of the multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

**Q3. b. Discuss how dining philosophers problem is solved using semaphores.**

Five philosophers are seated on 5 chairs across a table. Each philosopher has a plate full of noodles. Each philosopher needs a pair of forks to eat it. There are only 5 forks available all together. There is only one fork between any two plates of noodles. In order to eat, a philosopher lifts two forks, one to his left and the other to his right. if he is successful in obtaining two forks, he starts eating after some time, he stops eating and keeps both the forks down.

What if all the 5 philosophers decide to eat at the same time ?

All the 5 philosophers would attempt to pick up two forks at the same time. So,none of them succeed. One simple solution is to represent each fork with a semaphore.a philosopher tries to grab a fork by executing wait() operation on that semaphore.he releases his forks by executing the signal() operation.This solution guarantees that no two neighbours are eating

simultaneously. Suppose all 5 philosophers become hungry simultaneously and each grabs his left fork,he will be delayed forever.

The structure of Philosopher i:

```
do{
wait ( chopstick[i] );
wait ( chopStick[ (i + 1) % 5] );
// eat
signal ( chopstick[i] );
signal (chopstick[ (i + 1) % 5] );
// think
} while (TRUE);
```

**Several remedies:**

1) Allow at most 4 philosophers to be sitting simultaneously at the table.

2) Allow a philosopher to pickup his fork only if both forks are available.

3) An odd philosopher picks up first his left fork and then right fork. an even philosopher picks up his right fork and then his left fork.

**Q4. a. What do you mean by critical section problem? Explain the solution to the critical section problem using mutex locks.**

**Critical Section Problem:**

In a multiprogramming environment, several processes may need to access shared resources such as variables, files, or databases. The part of the program where the shared resource is accessed is called the critical section.

Critical Section Problem refers to the challenge of designing a protocol to ensure that:

Only one process can be in its critical section at a time.

No process is unnecessarily delayed.

Processes are not starved.

Requirements for a Solution (To ensure proper synchronization):

Mutual Exclusion: Only one process can enter the critical section at a time.

Progress: If no process is in the critical section, any process requesting entry should be allowed to enter without delay.

Bounded Waiting: A limit must exist on how many times other processes can enter their critical sections before a process waiting to enter is allowed access.

Solution Using Mutex Locks:

A mutex lock (short for mutual exclusion lock) is a synchronization primitive used to enforce mutual exclusion in a concurrent system.

Working:

A mutex is a binary lock (i.e., it has two states: locked or unlocked).

When a process wants to enter its critical section, it must acquire (lock) the mutex.

When the process exits the critical section, it releases (unlocks) the mutex.

If another process tries to enter the critical section while the mutex is locked, it must wait until the mutex is released.

Pseudocode Example:

```
mutex lock = 1;  // 1 means unlocked, 0 means locked
Process P1:
while (true) {
   wait(lock);         // acquire lock
   // critical section
   signal(lock);       // release lock
   // remainder section
}
Process P2:
while (true) {
   wait(lock);         // acquire lock
   // critical section
   signal(lock);       // release lock
   // remainder section
}
```

wait(lock) – Locks the mutex (if already locked, the process waits).

signal(lock) – Unlocks the mutex so that other processes can acquire it.

**Q4 b. Consider the set of processes with Arrival Time,CPU burst time (in milliseconds) and priority as shown below.(Lower number represents higher priority)**

| Process | Arrival Time | Burst Time | Priority |
|---|---|---|---|
| P1 | 0 | 10 | 3 |
| P2 | 1 | 1 | 1 |
| P3 | 2 | 2 | 4 |
| P4 | 3 | 1 | 5 |
| P5 | 4 | 5 | 2 |

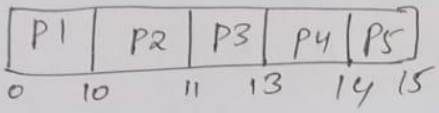
**Draw the Gantt chart and calculate the average waiting time and average turnaround time using i) SJF Scheduling( Non Preemptive) 2) Priority Scheduling( Non-Preemptive)**

Q.4)
b)

1) SJF scheduling (Non-Preemptive)

| Process | AT | BT | TAT | WT | CT |
|---|---|---|---|---|---|
| P1 | 0 | 10 | 10 | 0 | 10 |
| P2 | 1 | 1 | 10 | 9 | 11 |
| P3 | 2 | 2 | 11 | 9 | 13 |
| P4 | 3 | 1 | 11 | 10 | 14 |
| P5 | 4 | 5 | 15 | 10 | 15 |

Gantt Chart




TAT = CT - AT

P1 ⇒ 10 - 0 = 10
P2 ⇒ 11 - 1 = 10
P3 ⇒ 13 - 2 = 11
P4 ⇒ 14 - 3 = 11
P5 ⇒ 19 - 4 = 15

ATAT = $\frac{57}{5}$ = 11.4

WT = TAT - BT

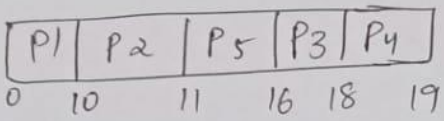
P1 ⇒ 10 - 10 = 0
P2 ⇒ 10 - 1 = 9
P3 ⇒ 10 - 2 = 9
P4 ⇒ 11 - 1 = 10
P5 ⇒ 15 - 5 = 10

AWT = $\frac{38}{5}$ = 7.6

2) Priority Scheduling (Non-Pre emptive)

| Process | AT | BT | CT | TAT | WT | Priority |
|---|---|---|---|---|---|---|
| P1 | 0 | 10 | 10 | 10 | 0 | 3 |
| P2 | 1 | 1 | 11 | 10 | 9 | 1 |
| P3 | 2 | 2 | 18 | 12 | 7 | 4 |
| P4 | 3 | 1 | 19 | 16 | 14 | 5 |
| P5 | 4 | 5 | 16 | 16 | 15 | 2 |

Gantt Chart




TAT = CT - AT

P1 ⇒ 10 - 0 = 10
P2 ⇒ 11 - 0 = 10
P3 ⇒ 16 - 4 = 12
P4 ⇒ 18 - 2 = 16
P5 ⇒ 19 - 3 = 16

ATAT = $\frac{64}{5}$ = 12.8

WT = TAT - BT

P1 ⇒ 10 - 10 = 0
P2 ⇒ 10 - 1 = 9
P3 ⇒ 12 - 5 = 7
P4 ⇒ 16 - 2 = 14
P5 ⇒ 16 - 1 = 15

AWT = $\frac{45}{5}$ = 9.

## Module-3

### Q5 a. Illustrate deadlocks with their necessary conditions

Conditions For Deadlock-

There are following 4 necessary conditions for the occurrence of deadlock-

1. Mutual Exclusion

2. Hold and Wait

3. No preemption

4. Circular wait

**1.Mutual Exclusion-**

By this condition,

- There must exist at least one resource in the system which can be used by only one process at a time.
- If there exists no such resource, then deadlock will never occur.
- Printer is an example of a resource that can be used by only one process at a time.

**2.Hold and Wait**-

By this condition,

- There must exist a process which holds some resource and waits for another resource held by some other process.

**3. No Preemption**

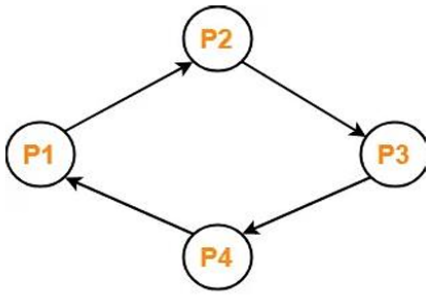
By this condition,

- Once the resource has been allocated to the process, it cannot be preempted.
- It means resource cannot be snatched forcefully from one process and given to the other process.
- The process must release the resource voluntarily by itself.

**4. Circular Wait**

By this condition,

- All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.




Circular Wait

Here,

- Process P1 waits for a resource held by process P2.
- Process P2 waits for a resource held by process P3.
- Process P3 waits for a resource held by process P4.
- Process P4 waits for a resource held by process P1.

**Q5.b. Describe the working principles of Banker's algorithm for the following snapshot and find either the systems is in safe or not.**

| | Allocation | | | | Max | | | | Available | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | A | B | C | D | A | B | C | D |
| $P_0$ | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 5 | 2 | 0 |
| $P_1$ | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 | | | | |
| $P_2$ | 1 | 3 | 5 | 4 | 2 | 3 | 5 | 6 | | | | |
| $P_3$ | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 | | | | |
| $P_4$ | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 | | | | |

Q.5)
b)

Step 1 : Calculate Need Matrix (Need = Max - Allocation)

| Process | Need | | | |
|---|---|---|---|---|
| | A | B | C | D |
| P0 | 0 | 0 | 0 | 0 |
| P1 | 0 | 7 | 5 | 0 |
| P2 | 1 | 0 | 0 | 2 |
| P3 | 0 | 0 | 2 | 0 |
| P4 | 0 | 6 | 4 | 2 |

Step 2 : Check safe sequence

✡ ITERATION 1 :

i) P0 : Need (0 0 0 0) & Available (5 2 0 0)
- Need ≤ Available ⇒ yes.
- New available = Allocation of P0 + Available.
- = (0 0 1 2) + (5 2 0 0)
- = (5 2 1 2)

ii) P1 : Need (0 7 5 0) & available (5 2 1 2)
- Need ≰ Available → No.
- Resources are not allocated to P1

iii) P2 : Need (1 0 0 2) & Available (5 2 1 2)

- Need ≤ Available ⇒ yes
- Resources are allocated P2
- P2 then releases it's resources
- New available = Available + Allocation of P2

= (5 2 1 2) + (1 3 5 4)

= (6 5 6 6)

iv) P3 : Need (0 0 2 0) & Available (6 5 6 6)

- Need ≤ Available ⇒ yes
- Resources are allocated to P3
- P3 then releases its resources
- New available = Available + Allocation of P3

= (6 5 6 6) + (0 6 3 2)

= (6 11 9 8)

v) P4 : Need (0 6 4 2) & Available (6 11 9 8)

- Need ≤ Available ⇒ yes
- Resources are allocated to P4
- P4 then releases it's resources.
- New available = Available + allocation of P4

= (6 11 9 8) + (0 0 1 4)

= (6 11 10 12)

Since $P_1$ has not got resources yet, another iteration is required.

## ITERATION 2

i) $P_1$ : Need (0 7 5 0) & Available (6 11 10 12)

- Need ≤ Available ⇒ yes
- $P_1$ executes
- Then releases it's resources.
- New available = Available + Allocation of $P_1$

$= (6\ 11\ 10\ 12) + (1\ 0\ 0\ 0)$

$= (7\ 11\ 10\ 12)$

Safe sequence: $P_0 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$

System is in a safe state.

**Q6. a. Discuss deadlock detection with a neat diagram.**

Deadlock Detection

• If a system does not use either deadlock-prevention or deadlock- avoidance algorithm then a deadlock may occur.

• In this environment, the system must provide

1) An algorithm to examine the system-state to determine whether a deadlock has occurred.

2) An algorithm to recover from the deadlock.

Single Instance of Each Resource Type

• If all there sources have only a single instance, then deadlock detection-algorithm can be defined using a wait-for-graph.

• The wait-for-graph is applicable to only a single instance of a resource type.

• Await-for-graph(WAG)is a variation of the resource-allocation-graph.

• Thewait-for-graphcanbeobtainedfromtheresource-allocation-graphby

→removing the resource nodes and

→collapsing the appropriate edges.

• An edge from Pi to Pj implies that process

Pi is waiting for process Pj to release are source that Pi needs.

• AnedgePi→Pjexistsifandonlyifthecorrespondinggraphcontainstwoedges

1) Pi→Rq and 2) Rq→Pj.

For example: Consider resource-allocation graph shown in below figure

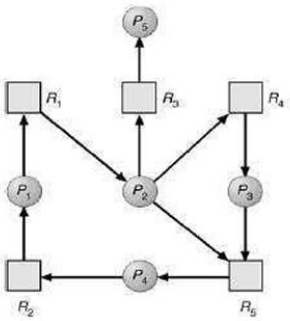
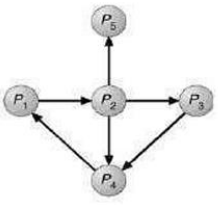



Figure3.6Resource-allocation-graph




for-graph. Figure3.7Correspondingwait-

A deadlock exists in the system if and only if the wait-for-graph contains a cycle.

• To detect deadlocks , the system needs to →maintain the wait-for-graph and →periodically execute an algorithm that searches for a cycle in the graph.

Several Instances of a Resource Type

• Thewait-for-graphisapplicabletoonlyasingleinstanceofaresourcetype.

• Problem:

However, the wait-for- graph is not applicable to a multiple instance of a resource type.

**• Solution:**

The following detection-algorithm can be used for a multiple instance of a resource type

• Assumptions:

Let 'n' be the number of processes in the system

Let 'm' be the number of resources types. • Following datastructures are used to implement this algorithm.

**1) Available[m]**

¬ This vector indicates the no. of available resources of each type.

¬ If Available[j]=k, then k instances of resource type Rj is available.

**2) Allocation[n][m]**

¬ Thismatrixindicatesno.ofresourcescurrentlyallocatedtoeachprocess.

¬ If Allocation [i,j]= k, then Pi is currently allocated k instances of Rj.

3) Request[n][m]

¬ This matrix indicates the current request of each process.

¬ If Request[i,j]=k, then process Pi is requesting k more instances of resource type Rj.

```
Step 1:
        LetWorkandFinishbevectorsoflengthmandnrespectively.
                a) InitializeWork=Available
                b)      Fori=0,1,2 .....n
                        ifAllocation(i)!
                        =0 then
                                Finish[i]=false;
                        else
                                Finish[i]=true;
Step 2:
        Findanindex(i)suchthatboth
                a) Finish[i]=false
                b) Request(i)<=Work.
                Ifnosuchiexist,gotostep4.
Step 3:
                Set:
                    Work=Work+Allocation(i)
                    Finish[i] = true
        Gotostep 2.
Step 4:
        IfFinish[i]=falseforsomeiwhere0<i<n,thenthesystemisin adeadlockstate.
```

**Detection-Algorithm Usage**

The detection-algorithm must be executed based on following factors:

1) The frequency of occurrence of a deadlock.

2)The no .of processes affected by the deadlock.

• If deadlocks occur frequently, then the detection-algorithm should be executed frequently.

• Resources allocated to deadlocked-processes will be idle until the deadlock is broken.

• Problem: Deadlock occurs only when some processes make a request that cannot be granted immediately.

• Solution1:

¬ The deadlock-algorithm must be executed whenever a request for allocation cannot be granted immediately.

¬ In this case, we can identify →set of deadlocked-processes and → specific process causing the deadlock.

• Solution2: ¬ The deadlock-algorithm must be executed in periodic intervals.

¬ For example: → once in an hour →whenever CPU utilization drops below certain threshold

**Q6. b. Explain different methods used for recovering from a deadlock in an operating system.**

**Deadlock Recovery in Operating Systems**

When a deadlock is detected in a system, the operating system must take action to recover and allow processes to continue execution. The goal is to break the circular wait and restore normal operations.

There are three primary methods used for deadlock recovery:

**1. Process Termination**

This method involves terminating one or more processes involved in the deadlock to break the cycle.

**(a) Abort All Deadlocked Processes**

- All processes involved in the deadlock are terminated.
- Advantage: Quick and simple.
- Disadvantage: Results in significant loss of work and resources.

**(b) Abort One Process at a Time**

- One process is terminated at a time until the deadlock is resolved.
- Selection Criteria:
    - Priority of the process.
    - Process completion time.
    - Resources held by the process.
    - Number of times the process has been aborted.
- Advantage: Less severe than aborting all.
- Disadvantage: Overhead of multiple aborts and restarts.

**2. Resource Preemption**

In this method, resources are forcibly taken from some processes and allocated to others to resolve the deadlock.

- Steps Involved:
    - Select a victim process to take resource(s) from.
    - Save the state of the victim for potential rollback.

    - Preempt the resource and assign it to a waiting process.
    - Restart the victim process later if needed.
- Considerations:
    - Which resource and which process to choose.
    - Cost of rollback (especially for long-running processes).
    - Fairness to avoid starvation.

### 3. Process Rollback

This involves rolling back one or more processes to a previous safe state, before they acquired the resources leading to deadlock.

- Checkpointing: The OS periodically saves the state of processes.
- Rollback: Affected processes are reverted to their last checkpoint.
- Retry Execution: After rollback, processes can retry their execution.
- Advantages:
    - Less destructive than termination.
    - Allows progress without starting from scratch.
- Disadvantages:
    - Requires checkpointing mechanisms.
    - May lead to repeated rollbacks and inefficiency.

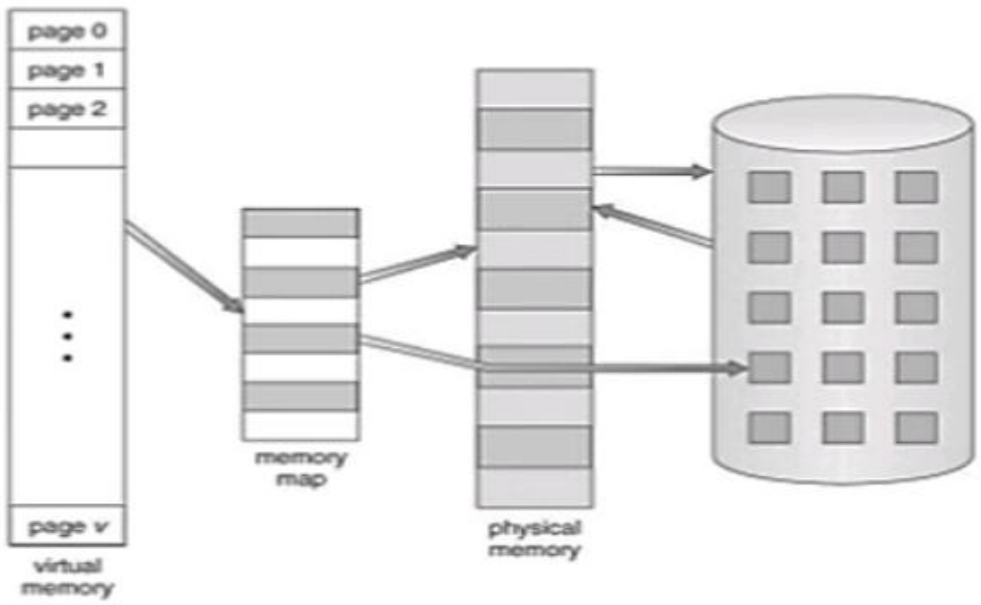
## Module-4

### Q7. a. Describe in detail the concept of paging with a neat diagram.

- Paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory.
- In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages.
- Paging allows the physical address space of the process to be non contiguous
- Paging is to deal with external fragmentation problem. This is to allow the logical address space of a process to be non-contiguous, which makes the process to be allocated physical memory.
- Logical address space of a process can be non-contiguous; process is allocated physical memory.

**Whenever the latter is available.**

- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called pages.
- Keep track of all free frames.
- To run a program of size n pages, need to find n free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation-allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- The logical memory of the process is contiguous, but pages need not be allocated contiguously in memory.
- By dividing memory into fixed size pages, we can eliminate external fragmentation.
- Paging does not eliminate internal fragmentation




Address generated by CPU is divided into:

1. **Page number (p)** – used as an index into a page table which contains base address of each page in physical memory.

2. **Page offset (d)** – combined with base address to define the physical memory address that is sent to the memory unit.

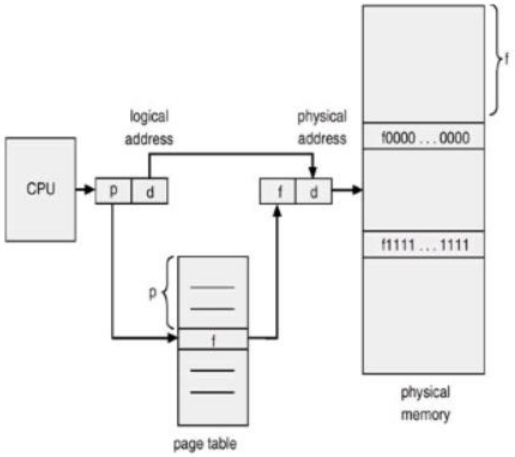



Fig. Address translation scheme

**Implementation of page table:**

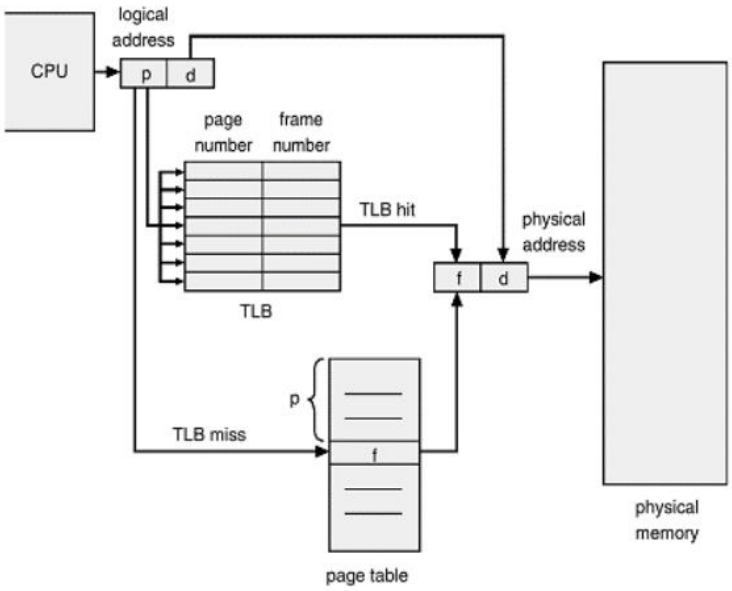
- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.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

Address translation (A', A'')

- If A' in associative register, get frame number out.
- Otherwise get frame number from page table in memory.
    - Hit ratio - percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
    - Effective Access Time (EAT)
    - associative lookup = e time units
    - memory cycle time = m time units
    - hit ratio = a

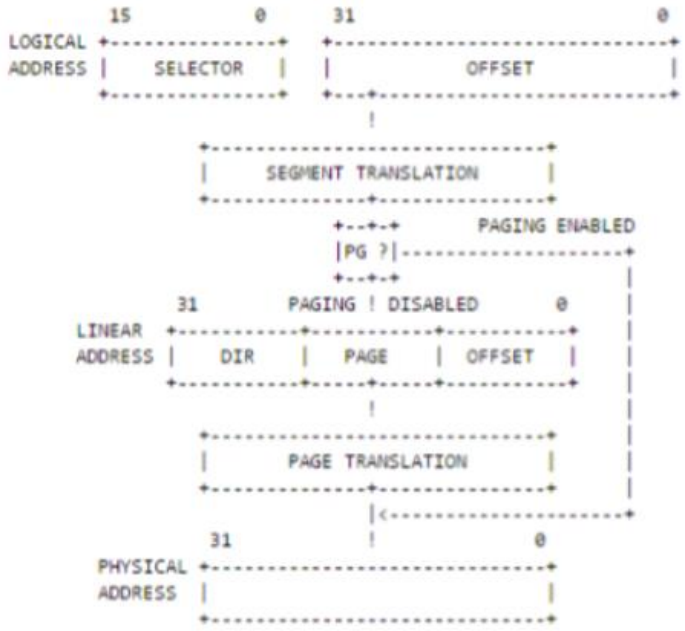
EAT = (m + e) a+ (2m + e) (1 - a) = 2m + e - ma

- Memory protection implemented by associating protection bits with each frame.
- Valid-invalid bit attached to each entry in the page table:
    - ``valid'' indicates that the associated page is in the process' logical address space, and is thus a legal page.
    - ``invalid'' indicates that the page is not in the process' logical address space.
- Write bit attached to each entry in the page table.
    - Pages which have not been written may be shared between processes
    - Do not need to be swapped - can be reloaded.

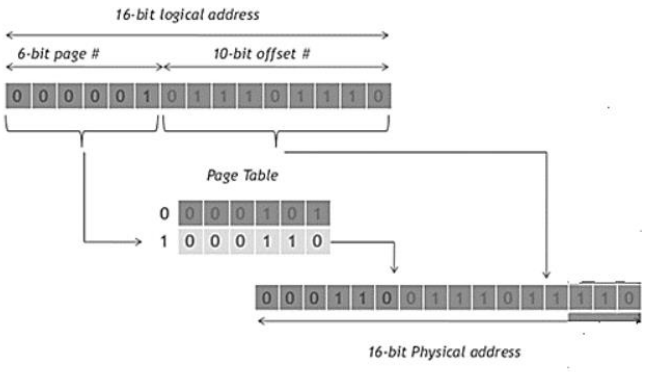



Conversion of logical address to physical address

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
    - Logical address - generated by the CPU; also referred to as virtual address.
    - Physical address - address seen by the memory unit.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.



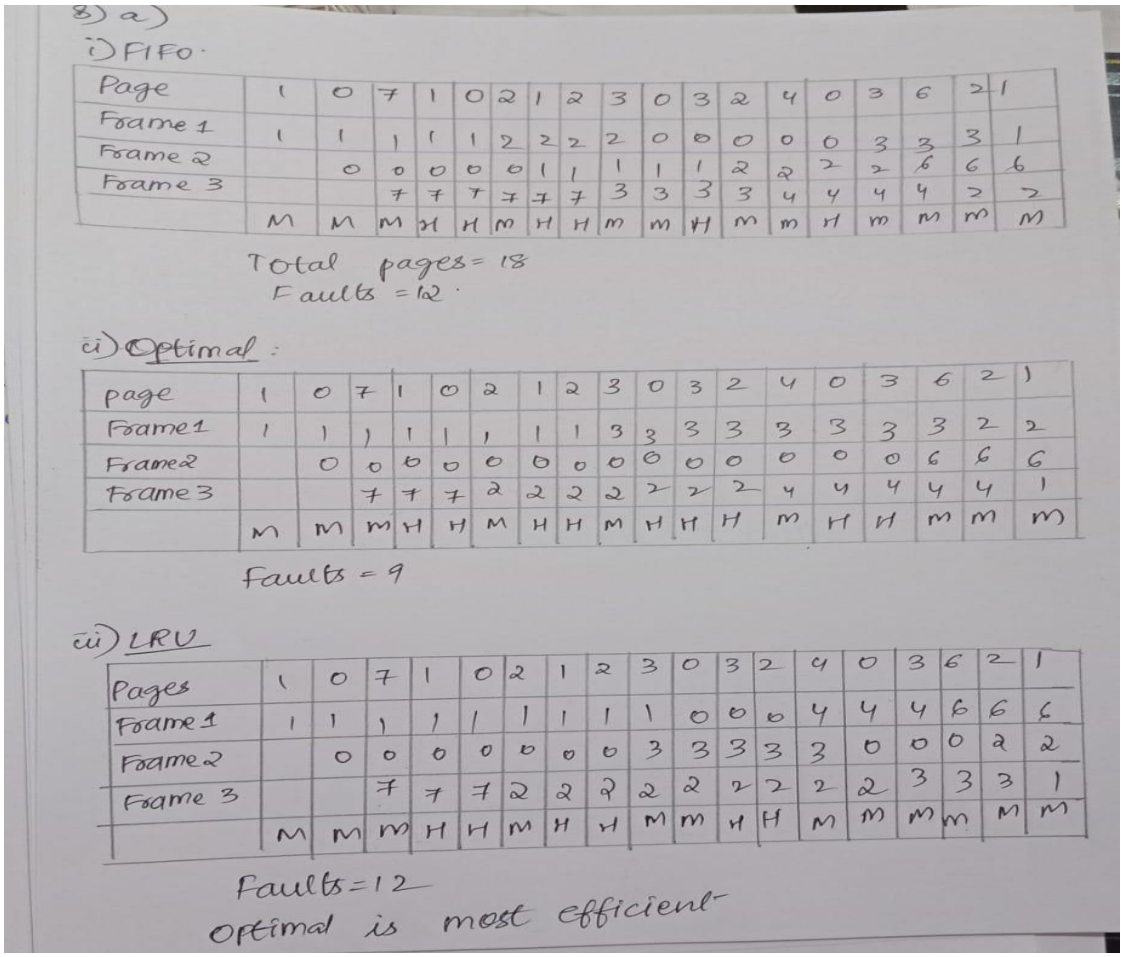

- There are three steps to translate logical address to physical address
- **Step1:** Find the index field from the segment selector and use the index field to locate the segment descriptor for the segment in global descriptor table (GDT)
- **Step 2:** Test the access and limit the field of descriptor to make sure that the segment is accessible and the offset is within the limit of the segment
- **Step3:** The base address of the segment will be obtained from the segment descriptor. Then the base address of the segment will be added to the offset to determine a linear address




## Q7. b. Differentaite between internal and external fragmentation

| Internal fragmentation | External fragmentation |
| --- | --- |
| In internal fragmentation fixed-sized memory, blocks square measure appointed to process. | In external fragmentation, variable-sized memory blocks square measure appointed to the method. |
| Internal fragmentation happens when the method or process is smaller than the memory. | External fragmentation happens when the method or process is removed. |
| The solution of internal fragmentation is the best-fit block. | The solution to external fragmentation is compaction and paging. |
| Internal fragmentation occurs when memory is divided into fixed-sized partitions. | External fragmentation occurs when memory is divided into variable size partitions based on the size of processes. |
| The difference between memory allocated and required space or memory is called Internal fragmentation. | The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation. |
| Internal fragmentation occurs with paging and fixed partitioning. | External fragmentation occurs with segmentation and dynamic partitioning. |
| It occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance. | It occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required. |
| It occurs in worst fit memory allocation method. | It occurs in best fit and first fit memory allocation method. |

**Q8 a. Consider the page reference string:**
**1,0,7,1,0,2,1,2,3,0,3,2,4,0,3,6,2,1 for a memory with three frames. Determine the number of page faults using the FIFO, Optimal and LRU replacement algorithms. Which algorithm is most efficient?**



8) a)

i) FIFO:

| Page | 1 | 0 | 7 | 1 | 0 | 2 | 1 | 2 | 3 | 0 | 3 | 2 | 4 | 0 | 3 | 6 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 1 |
| Frame 2 | | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 6 | 6 | 6 |
| Frame 3 | | | 7 | 7 | 7 | 7 | 7 | 7 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 2 | 2 |
| | M | M | M | H | H | M | H | H | M | M | H | M | M | H | M | M | M | M |

Total pages = 18
Faults = 12

ii) Optimal:

| page | 1 | 0 | 7 | 1 | 0 | 2 | 1 | 2 | 3 | 0 | 3 | 2 | 4 | 0 | 3 | 6 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 |
| Frame2 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 6 | 6 |
| Frame3 | | | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 1 |
| | M | M | M | H | H | M | H | H | M | H | H | H | M | H | H | M | M | M |

Faults = 9

iii) LRU

| Pages | 1 | 0 | 7 | 1 | 0 | 2 | 1 | 2 | 3 | 0 | 3 | 2 | 4 | 0 | 3 | 6 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Frame 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 4 | 4 | 4 | 6 | 6 | 6 |
| Frame 2 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 2 | 2 |
| Frame 3 | | | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 1 |
| | M | M | M | H | H | M | H | H | M | M | H | H | M | M | M | M | M | M |

Faults = 12
Optimal is most efficient

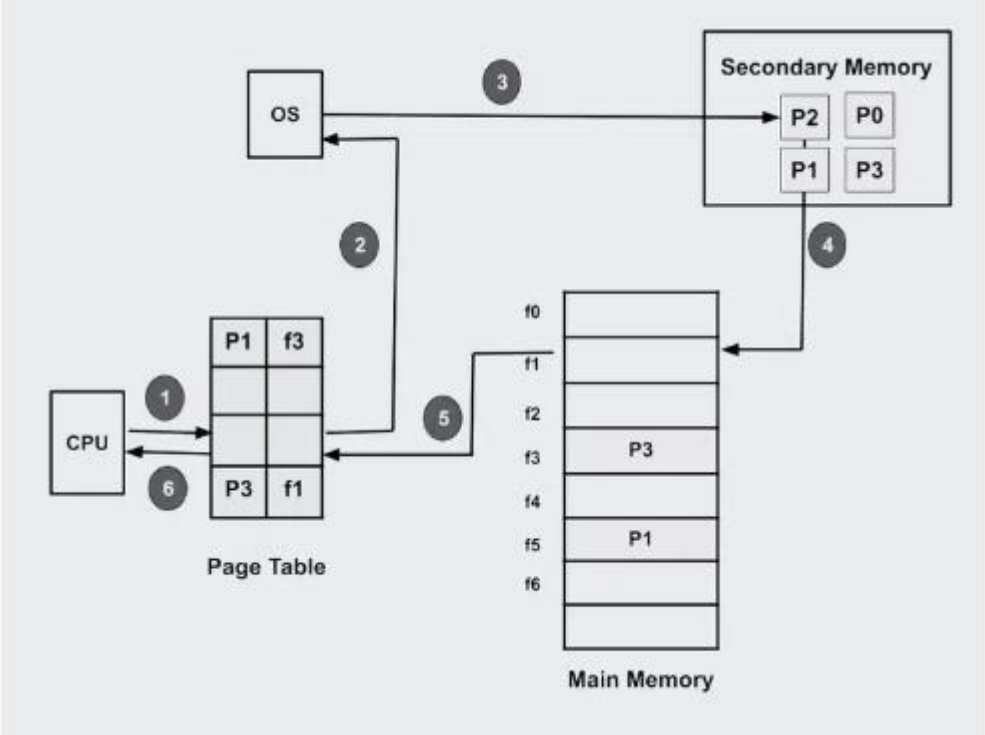
**Q8. b. Interpret the concepts of demand paging with neat diagram.**

Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU. In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start. A page fault occurred when the program needed to access a page that is not currently in memory.

The operating system then loads the required pages from the disk into memory and updates the page tables accordingly. This process is transparent to the running program and it continues to run as if the page had always been in memory.

**Working Process of Demand Paging**

Let us understand this with the help of an example. Suppose we want to run a process P which have four pages P0, P1, P2, and P3. Currently, in the page table, we have pages P1 and P3.




The operating system's demand paging mechanism follows a few steps in its operation.

- **Program Execution:** Upon launching a program, the operating system allocates a certain amount of memory to the program and establishes a process for it.
- **Creating Page Tables:** To keep track of which program pages are currently in memory and which are on disk, the operating system makes page tables for each process.
- **Handling Page Fault:** When a program tries to access a page that isn't in memory at the moment, a page fault happens. In order to determine whether

the necessary page is on disk, the operating system pauses the application and consults the page tables.

- **Page Fetch:** The operating system loads the necessary page into memory by retrieving it from the disk if it is there.
- The page's new location in memory is then reflected in the page table.
- **Resuming The Program:** The operating system picks up where it left off when the necessary pages are loaded into memory.
- **Page Replacement:** If there is not enough free memory to hold all the pages a program needs, the operating system may need to replace one or more pages currently in memory with pages currently in memory. on the disk. The page replacement algorithm used by the operating system determines which pages are selected for replacement.
- **Page Cleanup:** When a process terminates, the operating system frees the memory allocated to the process and cleans up the corresponding entries in the page tables.

## Module-5

**Q9. a. Illustrate the following access methods.**

### i) Sequential access

It is the simplest access method. Information in the file is processed in order i.e. one record after another. A process can read all the data in a file in order starting from beginning but can't skip & read arbitrarily from any location. Sequential files can be rewound. It is convenient when storage medium was magnetic tape rather than disk.

Eg : A file consisting of 100 records, the current position of read/write head is 45th record, suppose we want to read the 75th record then, it access sequentially from 45, 46, 47 ..... 74, 75. So the read/write head traverse all the records between 45 to 75.

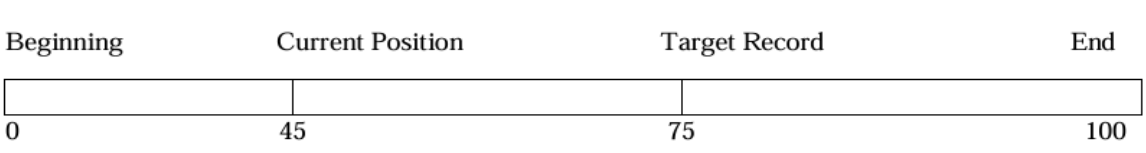



Fig 1: sequential access file

### ii) Direct access

Direct Access: A file is made up of fixed length-logical records that allow programs to read & write records rapidly in no particular order. This method can be used when disk are used for storing files. This method is used in many applications e.g. database systems. If an airline customer wants to reserve a

seat on a particular flight, the reservation program must be able to access the record for that flight directly without reading the records before it. In a direct access file, there is no restriction in the order of reading or writing. For example, we can read block 14, then read block 50 & then write block 7 etc. Direct access files are very useful for immediate access to large amount of information.

**Q9. b. Illustrate in detail the various operations performed on a file**

**File Operations:** Any file system provides not only a means to store data organized as files, but a collection of functions that can be performed on files. Typical operations include the following:

**Creating files:** Two steps are necessary to create a file. First, space must be found for the file in the file system. Secondly, an entry must be made in the directory for the new file.

**Reading a file:** Data & read from the file at the current position. The system must keep a read pointer to know the location in the file from where the next read is to take place. Once the read has been taken place, the read pointer is updated.

**Writing a file:** Data are written to the file at the current position. The system must keep a write pointer to know the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

**Repositioning within a file:** The directory is searched for the appropriate entry & the current file position is set to a given value. After repositioning data can be read from or written into that position. Repositioning within a file does not need to involve any actual I/O. this file operation is also known as a file seek.

**Deleting a file:** To delete a file, we search the directory for the required file. After deletion, the space is released so that it can be reused by other files.

**Truncating a file:** the user may want to erase the contents of a file but keep its attribute. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged- except for file length- but lets the file be reset to length zero and its file space released.

**Q9. c. Explain the following i) Bit vector ii) Linked list**

**a) Bit Vector**

☐ Fastalgorithmsexistforquicklyfindingcontiguousblocksofagivensize

☐ One simple approach is to use a bit vector, in which each bit represents a disk block, set to 1 if free or 0 if allocated.

For example, consider a disk where blocks 2,3,4,5,8,9,10,11,12,13,17and18 are free, and the rest of the blocks are allocated. The free-space bitmap would be

0011110011111100011

☐ Easy to implement and also very efficient in finding the first free block or 'n'

Consecutive free blocks on the disk.

The down side is that a 40GB disk requires over 5MB just to store the bitmap.

**b) LinkedList**

a. A linked list can also be used to keep track of all free blocks.

b. Traversing the list and/or finding a contiguous block of a given size are not

easy, but fortunately are not frequently needed operations. Generally the

system just adds and removes single blocks from the beginning of the list.

c. The FAT table keeps track of the free list as just one more linked list on the table.

**Q10. a. Illustrate various levels of directory structures**

The most common schemes for defining the logical structure of a directory are described below

1. Single-level Directory

2. Two-Level Directory

3. Tree-Structured Directories

4. Acyclic-Graph Directories

5. General Graph Directory

**Single-level Directory:**

it is the simplest directory structure. All files are contained in the same directory which is easy to support and understand

**Advantages:**

¬ Since it is a single directory, so its implementation is very easy.

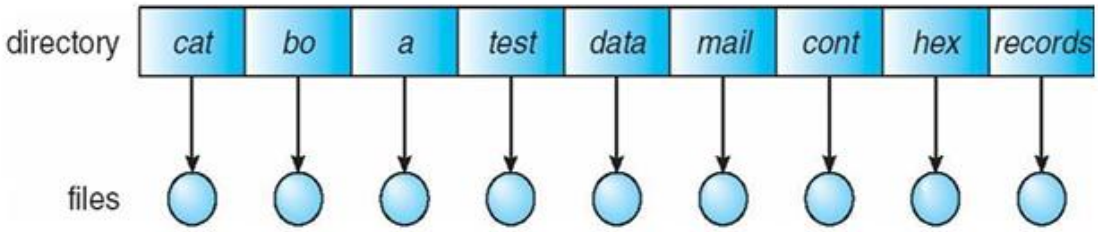
¬ If the files are smaller in size, searching will become faster.

¬ The operations like file creation, searching, deletion, updating is very easy in such a directory structure.

**Disadvantages:**

¬ There may chance of name collision because two files can not have the same name.
¬ Searching will become time taking if the directory is large.

¬ The same type of files cannot be grouped together.




**Two-level Directory:**

The disadvantage of single level directory is the confusion of files names between different users. The solution for this problem is to create a directory for each user as shown in figure.

In the two-level directory structure, each user has his own user files directory (UFD). Each user has similar structure but lists only the files of a single user. When user login, the system's master file directory (MFD) is searched. The master file directory is indexed by user name or account number and each entry points to the user directory for that user.

When users refer to a particular file, only their own user file directory is searched. Thus different users may have files with the same name, as long as all file names within each user file directory are unique.

**Advantages:**

¬ We can give full path like /User-name/directory-name.

¬ Different users can have same directory as well as file name.

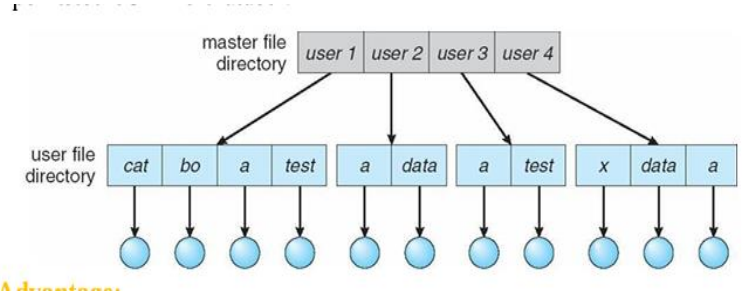
¬ Searching of files become more easy due to path name and user-grouping.

**Disadvantages:**

¬ A user is not allowed to share files with other users.

¬ Searching will become time taking if the directory is large.

¬ Two files of the same type cannot be grouped together in the same user.




**Tree- Structured Directory:**

The tree-structure allows user to create their own sub-

directories and organize their files accordingly. The tree has a root directory. Every file in the systems has a unique path name. A path is the path from the root through all the sub- directories to a specified file. A directory contains a set of files and or sub-directories.

Advantages:

¬ User can access other user's files by specifying the path name.

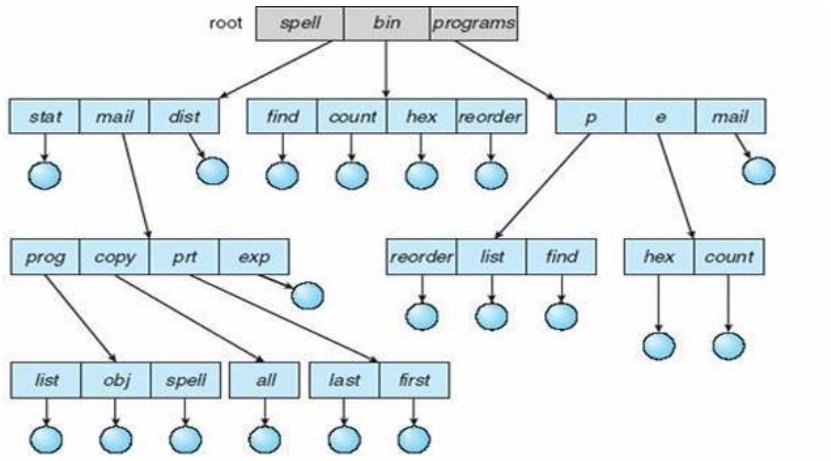
¬ User can create his own sub-directories.

¬ 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.

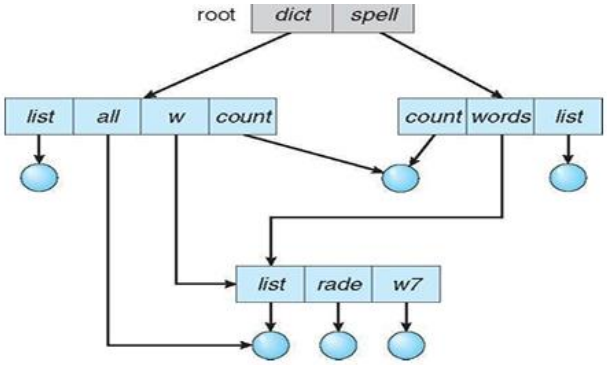
¬ We cannot share files.

¬ It is inefficient, because accessing a file may go under multiple directories.



**Acyclic Graph Directories:** A shared directory or file will exist in the file system in two or more places at once. A shared directory or file is not the same as two copies of the file. With a shared file there is only one actual field and any changes made by one person would be immediately visible to the other. An acyclic graph allows directories to have shared sub-directories and files. The same file or sub-directory may be in two or more process exists in th file system at a time. An acyclic graph directory structure is more flexible than a simple structure but it is also more complex.

Advantages: ¬ We can share files. ¬ Searching is easy due to different-different paths. ¬ Allow multiple directories to contain same file. Disadvantages: ¬ We share the files via linking; in case of deleting it may create the problem. ¬ Need to be cautions of dangling pointers when files are deleted.




**Q10. b. List the different file allocation methods and explain any two methods in detail.**

**ALLOCATION METHODS**

Allocation methods address the problem of allocating space to files so that disk space is utilized effectively and files can be accessed quickly.

Three methods exist for allocating disk space

- Contiguous allocation
- Linked allocation
- Indexed allocation

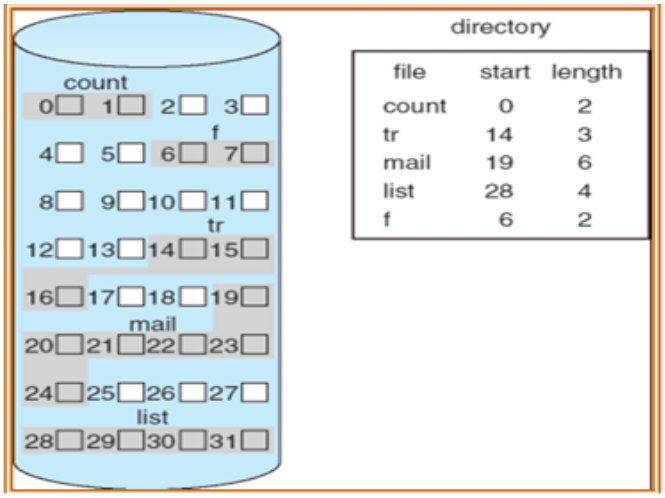
Contiguous allocation :

- Requires that each file occupy a set of contiguous blocks on the disk

• Accessing a file is easy – only need the starting location (block #) and length (number of blocks)

• 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.

• Accessing a file that has been allocated contiguous ly is easy. For sequential access, the file system remembers the disk address of the last block referenced and when necessary, reads the next block. For direct access to block i of a file that starts at block b, we can immediately access block b + i. Thus, both sequential and direct access can be supported by contiguous allocation.




Disadvantages:

1. Finding space for a new file is difficult. The system chosen to manage free space determines how this task is accomplished. Any management system can be used, but some are slower than others.

2. Satisfying a request of size n from a list of free holes is a problem. First fit and best fit are the most common strategies used to select a free hole from the set of available holes.

3. The above algorithms suffer from the problem of external fragmentation.

♣ As files are allocated and deleted, the free disk space is broken into pieces.

♣ External fragmentation exists whenever free 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, none of which is large enough to store the data.

♣ Depending on the total amount of disk storage and the average file size, external fragmentation maybe a minor or a major problem.

**Linked Allocation :**

• Solves the problems of contiguous allocation

• Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk

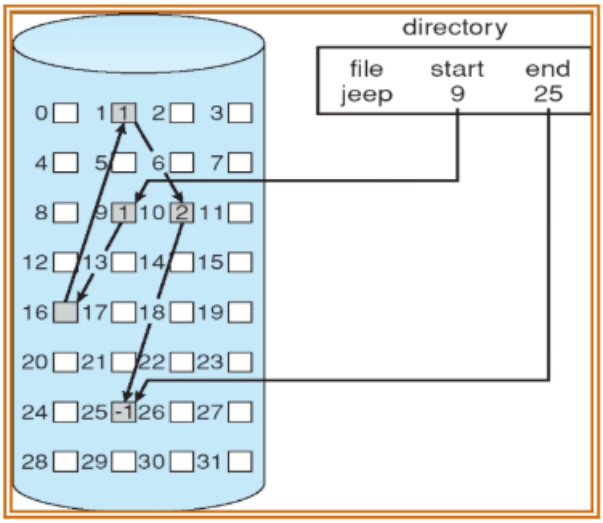
• The directory contains a pointer to the first and last blocks of a file

• Creating a new file requires only creation of a new entry in the directory

• Writing to a file causes the free-space management system to find a free block

¬ This new block is written to and is linked to the end of the file

¬Reading from a file requires only reading blocks by following the pointers from block to block.




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 and continue at block 16,

Then block 1, then block 10, and finally block 25. Each block contains a pointer to the next block. These pointers are not made available to the user. A disk address (the pointer) requires4 bytes in the disk.

☐ To create a new file, we simply create an even try the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. The size field is also set too.

☐ A write to the file causes the free-space management system to filed a free block, and this new block is written to an dislinked to the end of the file.

☐ To read a file, we simply read blocks by following the pointers from block to block. 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 need not be 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 compact diskspace.

☐ Disadvantages:

1. The major problem is that it can be used effectively only for sequential-

Access files. To filed the i th block of a file, we must start at the beginning of that

file and follow the pointers until we get to the ith block.

2. Space required for the pointers. Solution is clusters. Collect blocks into

multiples and allocate clusters rather than blocks

3. Reliability - the files are linked together by pointers scattered all over the

disk and if a pointer were lost or damaged then all the links are lost.