Scheme of Evaluation

Internal Assessment Test 1 – February 2025

Sub:	**Operating Systems**							**Sub Code:**	**MMC104**
Date:	07-02-25	Duration:	90 mins	Max Marks:	50	**Sem:**	I	**Branch:**	**MCA**

Q.NO	Description	Marks Distribution	Max Marks
1	**What are the functions of an OS?** • List out the functions of operating system and also explain it in brief.	**10**	**10**
2	**What is the user point of view and system point of view services of the operating system?** • Definition of User view and its types with neat diagram • Definition of System view and its types with neat diagram	**5** **5**	**10**
3	**Explain how the system calls works with neat diagram** • Define System Calls • Implementation of System calls with neat diagram • Explain about the types of System calls	**2** **5** **3**	**10**
4	**Explain with neat diagram of components of the computer system** • List out the components of the computer system and also explain it in brief.	**10**	**10**
5	**Explain Round robin and Priority scheduling with an example.** • Give explain about the Round robin with example • Give explain about the priority Scheduling with example •	**5** **5**	**10**
6	**Explain about the basic concepts of process scheduling and also common criteria used to evaluate scheduling algorithms? Explain each criterion.** • Explain Basic concepts of process scheduling • Explain about the Scheduling Criteria with detail notes	**5** **5**	**10**
7	**Explain any two classical problem of synchronization** • List out of classical problem of synchronization • Explain any two with detail notes	**2** **8**	**10**

8	**Calculate the average waiting time, turnaround time for i) FCFS(ii) SJF(Preemptive) and iii)Round Robin(tq=4ms) with the following set of process.** **Process** | **P1** | **P2** | **P3** | **P4** **Burst Time** | 8 | 4 | 9 | 5 **Arrival Time** | 0 | 1 | 2 | 3 • Solutions given by stepwise	**10**	**10**
9	**What is contiguous and non-contiguous memory allocation? Discuss its advantages and disadvantages.** • Explain about the contiguous memory allocation with pros and cons • Explain about the non-contiguous memory allocation with pros and cons	**5** **5**	**10**
10	**Explain the concept of swapping in memory management. How does it help in process execution?** • Definition of the Swapping • Explain the process of swapping with neat diagram	**2** **8**	**10**

Internal Assessment Test 1 – February 2025

Operating Systems							Sub Code:	MMC104
07/02/2025	Duration:	90 min’s	Max Marks:	50	Sem:	I	Branch:	MCA

PART I

1. **What are the functions of an OS?**

OPERATING SYSTEM FUNCTIONS

Process Management

 A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.

 Process needs resources to accomplish its task

 CPU, memory, I/O, files

 Initialization data

 Process termination requires reclaim of any reusable resources

 Single-threaded process has one **program counter** specifying location of next instruction to execute

 Process executes instructions sequentially, one at a time, until completion

 Multi-threaded process has one program counter per thread

 Typically system has many processes, some user, some operating system running concurrently on one or more CPUs

 Concurrency by multiplexing the CPUs among the processes / threads

Process Management Activities

 The operating system is responsible for the following activities in connection with process management:

 Creating and deleting both user and system processes

 Suspending and resuming processes

 Providing mechanisms for process synchronization

 Providing mechanisms for process communication

 Providing mechanisms for deadlock handling

Memory Management

 All data in memory before and after processing

 All instructions in memory in order to execute

 Memory management determines what is in memory when

 Optimizing CPU utilization and computer response to users

 Memory management activities

 Keeping track of which parts of memory are currently being used and by whom

 Deciding which processes (or parts thereof) and data to move into and out of memory

 Allocating and deallocating memory space as needed

Storage Management

 OS provides uniform, logical view of information storage

 abstracts physical properties to logical storage unit **- file**

 Each medium is controlled by device (i.e., disk drive, tape drive)

- Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
- Files usually organized into directories
- Access control on most systems to determine who can access what

OS activities include

- ・Creating and deleting files and directories ・Primitives to manipulate files and dirs

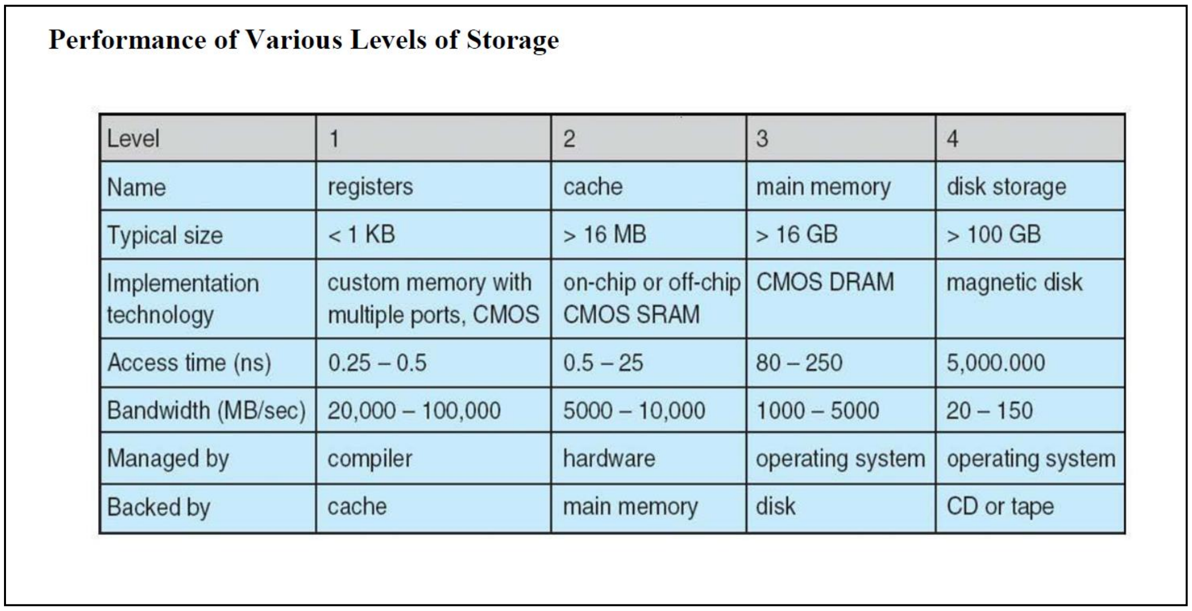
  ・Mapping files onto secondary storage ・Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a ―long‖ period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms

MASS STORAGE activities

- Free-space management
- Storage allocation
- Disk scheduling
- Some storage need not be fast
- Tertiary storage includes optical storage, magnetic tape
- Still must be managed
- Varies between WORM (write-once, read-many-times) and RW (read-write)

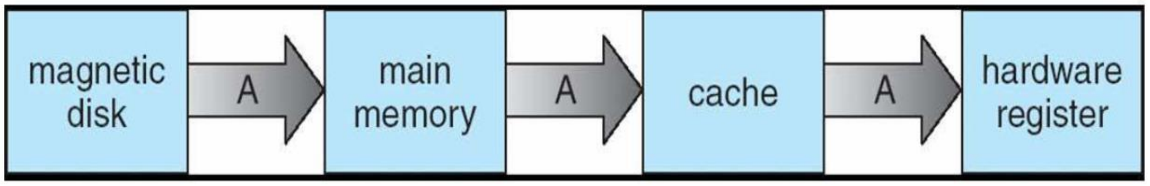


Performance of Various Levels of Storage

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Migration of Integer A from Disk to Register

• Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy.



• Distributed environment situation even more complex

• Several copies of a datum can exist

I/O Subsystem

• One purpose of OS is to hide peculiarities of hardware devices from the user

• I/O subsystem responsible for

• Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)

• General device-driver interface

• Drivers for specific hardware devices

Protection and Security

Protection – any mechanism for controlling access of processes or users to resources defined by the OS **Security** – defense of the system against internal and external attacks

• Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

• Systems generally first distinguish among users, to determine who can do what

• User identities (**user IDs**, security IDs) include name and associated number, one per user

• User ID then associated with all files, processes of that user to determine access control

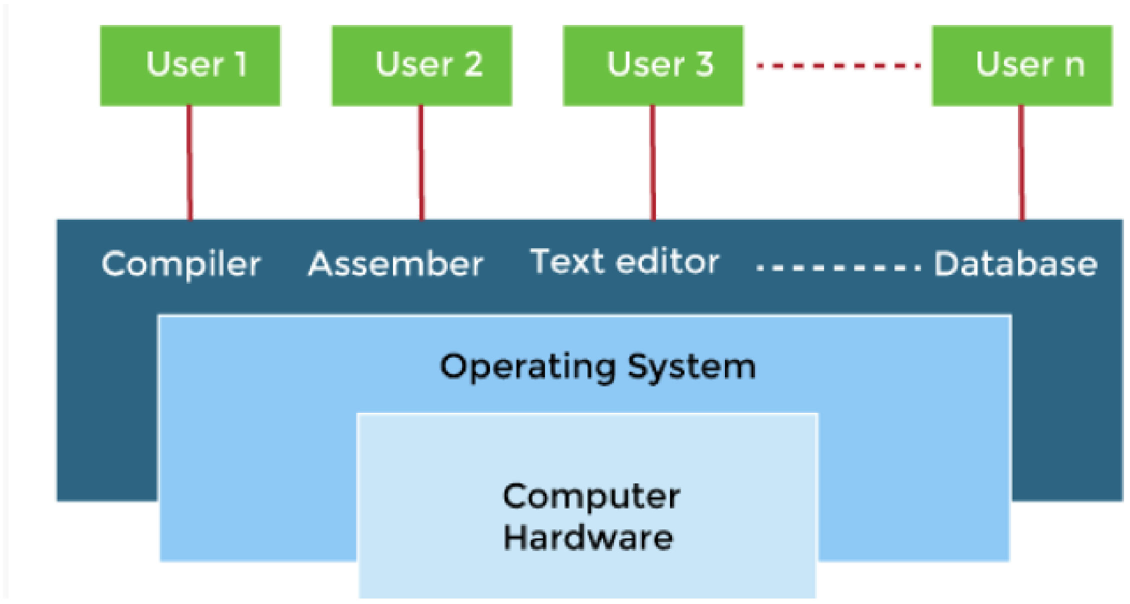
• Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file

• **Privilege escalation** allows user to change to effective ID with more rights

2. **What is the user point of view and system point of view services of the operating system?**

User View in OS

The user view depends on the system interface that the users use. The user view is all about how the user has to interact with the operating system with the help of various application programs. From the system point of view, we will see how the hardware interacts with the operating system to accomplish the various tasks.



Some systems are designed for a single user to monopolize the resources to maximize the user's task. In these cases, the OS is designed primarily for ease of use, with little emphasis on quality and none on resource utilization. The different types of user view experiences can be explained as follows:

1. Single User View:

Most computer users use a monitor, keyboard, mouse, printer, and other accessories to operate their computer system. These systems are much more designed for a single user experience and meet the needs of a single user, where the performance is not given focus as the multiple user systems. In some cases, the system is designed to maximize the output of a single user. As a result, more attention is laid on accessibility, and resource allocation is less important.

For example, if the user uses a personal computer, the operating system is largely designed to make the interaction easy. Attention is also paid to the system's performance, but there is no need for the operating system to worry about resource utilization. This is because the personal computer uses all the resources available, and there is no sharing.

2. Multiple User View:

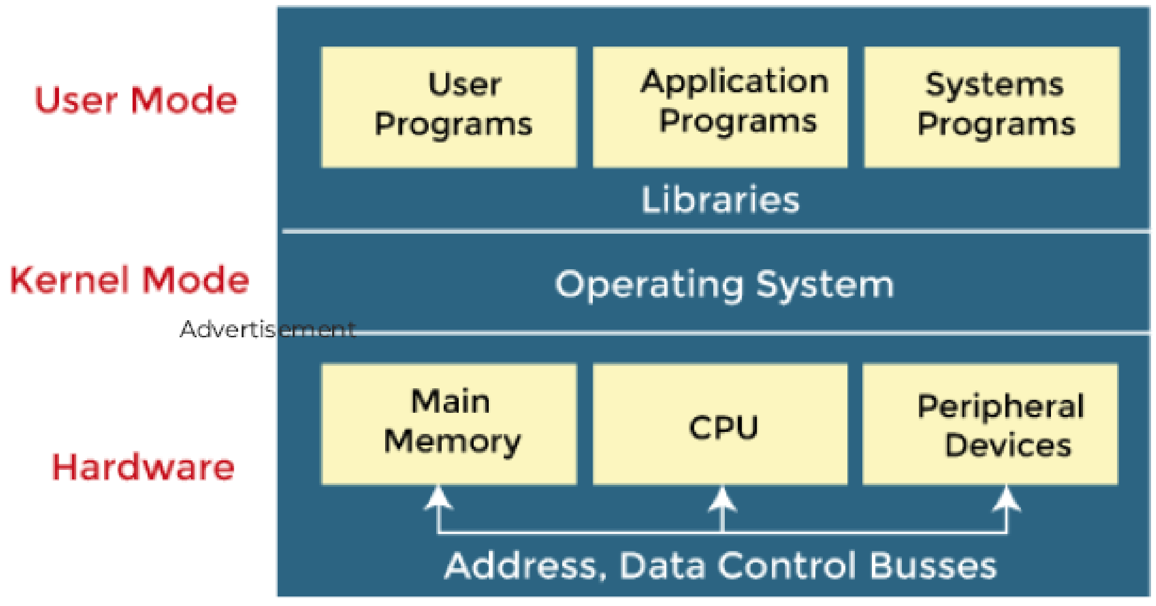
If the user is using a system connected to a mainframe or amini computer and many users on their computers trying to interact with their kernels over the mainframe to each other. The operating system is largely concerned with resource utilization. This is because there may be multiple terminals connected to the mainframe. The operating system makes sure that all the resources, such as CPU,memory, I/O devices, etc., are divided uniformly between them.

In such circumstances, memory allocation by the CPU must be done effectively to give agood user experience. The client-server architecture is another good example wheremany clients may interact through a remote server, and the same constraints of effectiveuse of server resources may arise.

3. Handled User View:
If the user uses a handheld computer such as a mobile, then theoperating system handles the device's usability, including a few remote operations. Thebattery level of the device is also taken into account. Smartphones interact via wirelessdevices to perform numerous operations, but they're not as efficient as a computerinterface, limiting their usefulness. However, their operating system is a great example ofcreating a device focused on the user's point of view.

4. Embedded System User View: Some systems, like embedded systems, lack a user point of view. The remote control used to turn onor off the tv is all part of an embedded system in which the electronic device communicates with another program where the user viewpoint is limited and allows the user to engage with the application.
Some devices contain very less or no user view because there is no interaction with the users, such as embedded systems. The remote you use for turning on or off the TV is part of an embedded system where the electronic device interacts with the other application. The user viewpoint is not much, but it allows users to interact with the application.

System View in OS



From the user point of view, we've discussed the numerous applications that requirevarying degrees of user participation. However, we are more concerned with how thehardware interacts with the operating system than the user from a system viewpoint.The different types of system views for the operating system can be explained as follows:
1. Resource Allocation:
The hardware contains several resources like registers, caches,CPU time, memory space, file storage space, RAM, ROM, CPUs, I/O interaction, etc. Theseare all resources that the operating

system needs when an application programdemands them. The operating system has to allocate these resources judiciously to theprocesses so that the computer system can run as smoothly as possible.
Only the operating system can allocate resources, and it has used several tactics and strategies to maximize its processing and memory space. The operating system uses a variety of strategies to get the most out of the hardware resources, including paging, virtual memory, caching, and so on. These are very important in the case of various user view points because inefficient resource allocation may affect the user viewpoint, causing the user system to lag or hang, reducing the user experience.

2. Control Program: The operating system can also work as a control program. It manages all the processes and I/O devices to work smoothly, and there are no errors. The user may request an action that can only be done with I/O devices; in this case, the operating system must also have proper communication, control, detect, and handle such devices. It makes sure that the I/O devices work correctly without creating problems.
3. Operating systems can also be viewed as making using hardware easier.
4. Computers were required to solve user problems easily. However, it is not easy to work directly with computer hardware. So, operating systems were developed to communicate with the hardware easily.

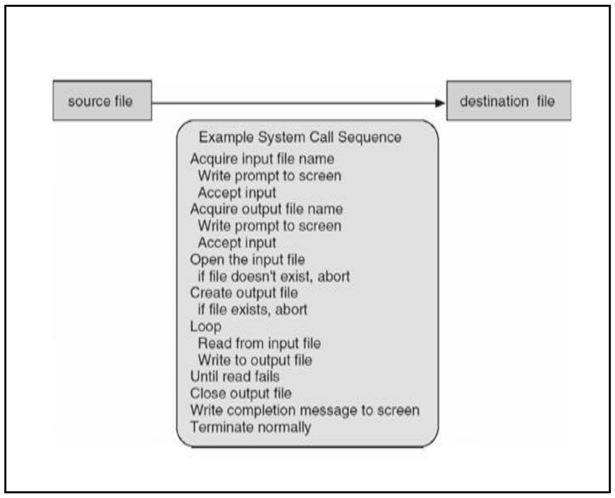
5. An operating system can also be considered as a program running at all times in the background of a computer system (known as the Kernel) and handling all the application programs. This is the definition of the operating system that is generally followed.

PART II

3. Explain how the system calls works with neat diagram

- 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 usenThree 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)

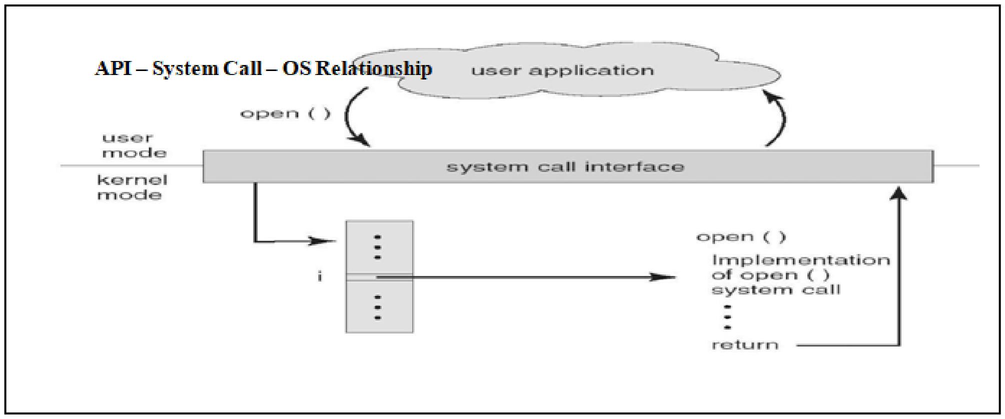
Example of System Calls



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

1.2.2 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 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

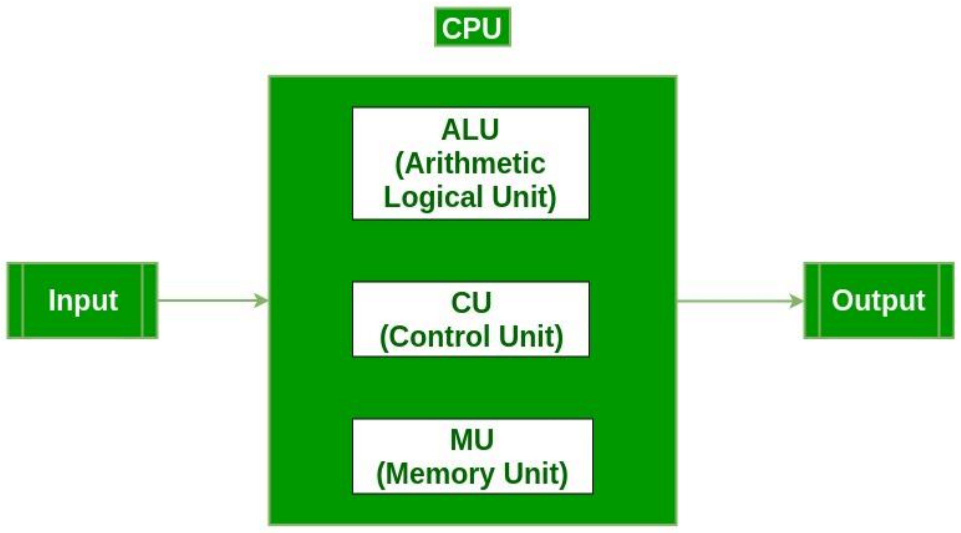
There are two common models of inter process communication: the message passing model and the shared-memory model. In the message-passing model, the communicating processes exchange messages with one another to transfer information. Messages can be exchanged between the processes either directly or indirectly 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.

4. Explain with neat diagram of components of the computer system

Computer: A computer is a combination of **hardware and software** resources which integrate together and provides various functionalities to the user. Hardware are the physical components of a computer like the processor, memory devices, monitor, keyboard etc. while software is the set of programs or instructions that are required by the hardware resources to function properly. There are a few basic components that aids the working-cycle of a computer i.e. the Input- Process- Output Cycle and these are called as the functional components of a computer. It needs certain input, processes that input and produces the desired output. The input unit takes the input, the central processing unit does the processing of data and the output unit produces the output. The memory unit holds the data and instructions during the processing.
Digital Computer: A digital computer can be defined as a programmable machine which reads the binary data passed as instructions, processes this binary data, and displays a calculated digital output. Therefore, Digital computers are those that work on the digital data.

Details of Functional Components of a Digital Computer



- **Input Unit :**The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner etc.
- **Central Processing Unit (CPU):** Once the information is entered into the computer by the input device, the processor processes it. The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. If required, data is fetched from memory or input device. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers
- **Arithmetic and Logic Unit (ALU) :** The ALU, as its name suggests performs mathematical calculations and takes logical decisions. Arithmetic calculations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.
- **Control Unit :** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units. It is also responsible for carrying out all the instructions stored in the program. It decodes the fetched instruction,

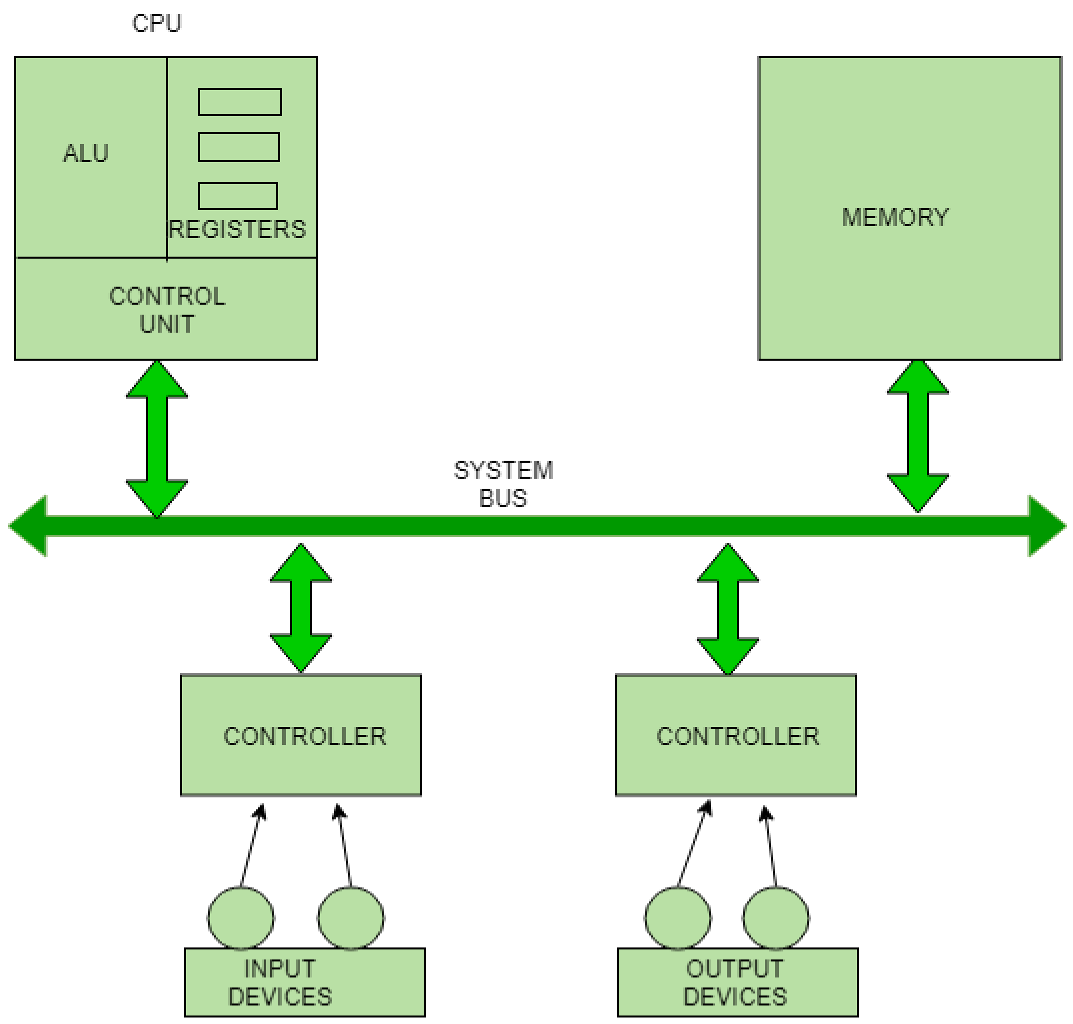
interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

- **Memory Registers :** A register is a temporary unit of memory in the CPU. These are used to store the data which is directly used by the processor. Registers can be of different sizes(16 bit, 32 bit, 64 bit and so on) and each register inside the CPU has a specific function like storing data, storing an instruction, storing address of a location in memory etc. The user registers can be used by an assembly language programmer for storing operands, intermediate results etc. Accumulator (ACC) is the main register in the ALU and contains one of the operands of an operation to be performed in the ALU.
- **Memory :** Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. when a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM). Read this for different types of RAMs
- **Output Unit :** The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, plotter etc.

Interconnection between Functional Components

A computer consists of input unit that takes input, a CPU that processes the input and an output unit that produces output. All these devices communicate with each other through a common bus. A bus is a transmission path, made of a set of conducting wires over which data or information in the form of electric signals, is passed from one component to another in a computer. The bus can be of three types – Address bus, Data bus and Control Bus.

Following figure shows the connection of various functional components:



The address bus carries the address location of the data or instruction. The data bus carries data from one component to another and the control bus carries the control signals. The system bus is the common communication path that carries signals to/from CPU, main memory and input/output devices. The input/output devices communicate with the system bus through the controller circuit which helps in managing various input/output devices attached to the computer.

PART III

5. Explain Round robin and Priority scheduling with an example.

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of First come First Serve CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

Characteristics:

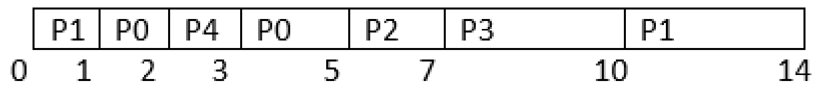
- It's simple, easy to use, and starvation-free as all processes get the balanced CPU allocation.
- One of the most widely used methods in CPU scheduling as a core.
- It is considered preemptive as the processes are given to the CPU for a very limited time.

Advantages:

- Round robin seems to be fair as every process gets an equal share of CPU.
- The newly created process is added to the end of the ready queue.

Example: Time Quantum = 1 ms

Process ID	**Arrival Time**	**Burst Time**
P0	1	3
P1	0	5
P2	3	2
P3	4	3
P4	2	1



Process ID	**Arrival Time**	**Burst Time**	**Completion Time**	**Turn Around Time**	**Waiting Time**
P0	1	3	5	4	1
P1	0	5	14	14	9
P2	3	2	7	4	2
P3	4	3	10	6	3
P4	2	1	3	1	0

Avg Turn Around Time = (4+14+4+6+1) / 5 = 5.8 ms
Avg Waiting Time = (1+9+2+3+0) / 5 = 3 ms

Priority Scheduling:
Preemptive Priority CPU Scheduling Algorithm is a pre-emptive method of CPU scheduling algorithm that works **based on the priority** of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there are more than one processor with equal value, then the most important CPU planning algorithm works on the basis of the FCFS **Characteristics:**

- Schedules tasks based on priority.
- When the higher priority work arrives while a task with less priority is executed, the higher priority work takes the place of the less priority one and
- Lower is the number assigned, higher is the priority level of a process.

Advantages:

- The average waiting time is less than FCFS
- Less complex

Disadvantages:

- One of the most common demerits of the Preemptive priority CPU scheduling algorithm is the Starvation Problem. This is the problem in which a process has to wait for a longer amount of time to get scheduled into the CPU. This condition is called the starvation problem.

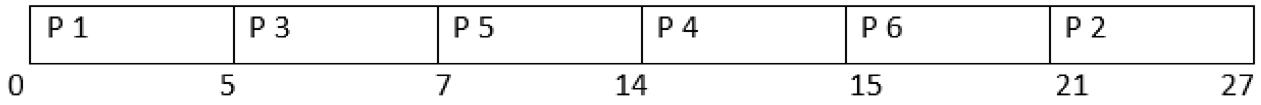
Example:

S. No	**Process ID**	**Arrival Time**	**Burst Time**	**Priority**
1	P 1	0	5	5
2	P 2	1	6	4
3	P 3	2	2	0
4	P 4	3	1	2
5	P 5	4	7	1
6	P 6	4	6	3

(5 has the least priority and 0 has the highest priority)

Solution:

Gantt Chart:



Process Id	**Arrival Time**	**Burst Time**	**Priority**	**Completion Time**	**Turn Around Time TAT=CT-AT**	**Waiting Time WT=TAT-BT**
P 1	0	5	5	5	5	0
P 2	1	6	4	27	26	20
P 3	2	2	0	7	5	3
P 4	3	1	2	15	12	11
P 5	4	7	1	14	10	3
P 6	4	6	3	21	17	11

Avg Waiting Time = (0 + 20 + 3 + 11 + 3 + 11) / 6 = 48 / 6 = 8 ms

Avg Turn Around Time = (5 + 26 + 5 + 11 + 10 + 17) / 6 = 74 / 6 = 12.33 ms

6. **Explain about the basic concepts of process scheduling and also common criteria used to evaluate scheduling algorithms? Explain each criterion.**

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Categories of Scheduling
There are two categories of scheduling:

Non-preemptive: Here the resource can’t be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.
Preemptive: Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

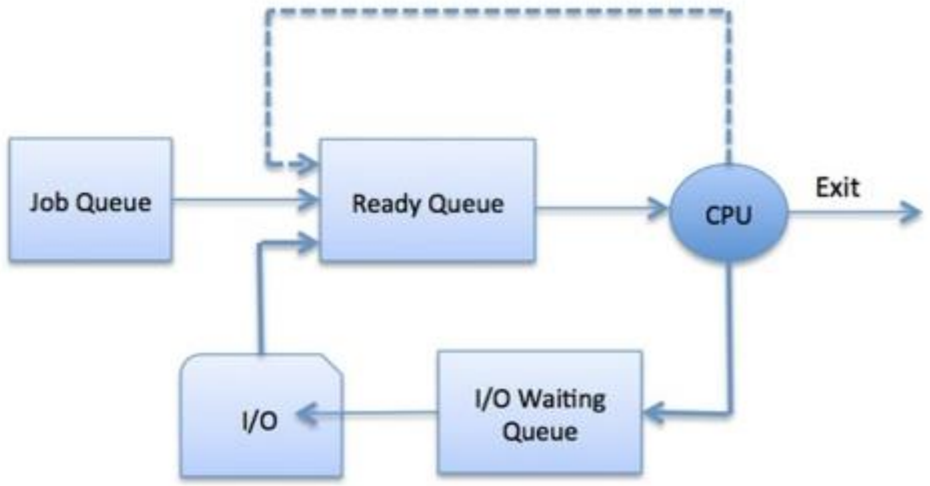
Process Scheduling Queues
The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

Job queue − This queue keeps all the processes in the system.

Ready queue − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

Device queues − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a job scheduler. A long-term scheduler determines which programs are admitted

to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling

the swapped out-processes. A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

THE SCHEDULING CRITERIA

CPU utilization:

The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.

Throughput:

The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.

Turn round Time:

For a particular process, the important conditions are how long it takes to perform that process. The time elapsed from the time of process delivery to the time of completion is known as the conversion time. Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.

Waiting Time:

The Scheduling algorithm does not affect the time required to complete the process once it has started performing. It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.

Response Time:

In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

PART IV

7. **Explain any two classical problem of synchronization**

Semaphore can be used in other synchronization problems besides Mutual Exclusion.

Below are some of the classical problem depicting flaws of process synchronaization in systems where cooperating processes are present.

We will discuss the following three problems:

1. Bounded Buffer (Producer-Consumer) Problem
2. Dining Philosophers Problem
3. The Readers Writers Problem

Bounded Buffer Problem

Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.

- This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.
- Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.
- In this Producers mainly produces a product and consumers consume the product, but both can use of one of the containers each time.
- The main complexity of this problem is that we must have to maintain the count for both empty and full containers that are available.

Producer-Consumer problem

A producer process produces information that is consumed by a consumer process. For example, a print program produces characters that are consumed by the printer driver.
A producer can produce one item while the consumer is consuming another item. The Producer and Consumer must be synchronized. The consumer does not try to consume an item, the consumer must wait until an item is produced.

Unbounded-Buffer

- no practical limit on the size of the buffer.
- Producer can produce any number of items.
- Consumer may have to wait

Bounded-Buffer

- assumes that there is a fixed buffer size.

Bounded-Buffer – Shared-Memory Solution:

Shared data

```
#define BUFFER_SIZE 10 Typedef struct
{
. . .
} item;
item buffer[BUFFER_SIZE]; int in = 0;

int out = 0;
```

Bounded-Buffer – Producer Process:

```
item next Produced; while (1)
{
while (((in + 1) % BUFFER_SIZE) == out); /* do nothing */ buffer[in] = nextProduced;
in = (in + 1) % BUFFER_SIZE;
}
```

Bounded-Buffer – Consumer Process:

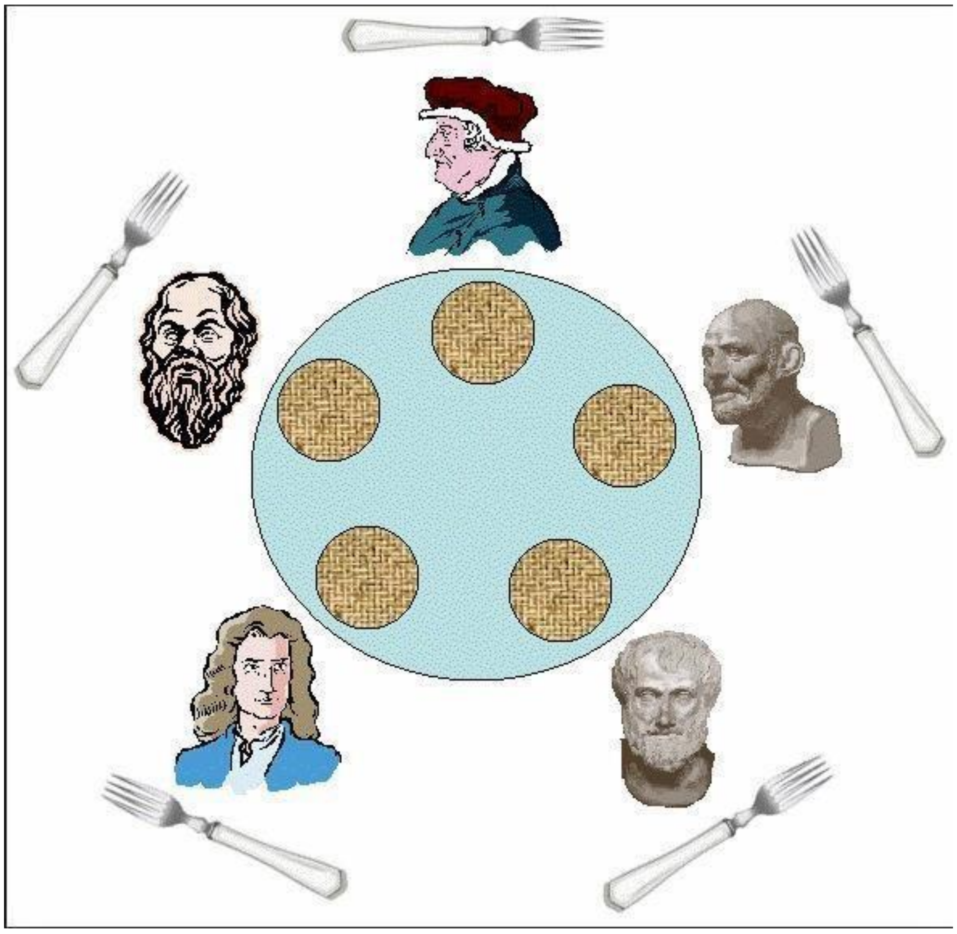
```
item next Consumed; while (1)
{
while (in == out); /* do nothing */ next Consumed = buffer[out];
out = (out + 1) % BUFFER_SIZE;

}
```

Dining Philosophers Problem

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. There are five philosophers sitting around a table, in which there are five chopsticks/forks kept beside them and a bowl of rice in the centre, 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.



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.

The Readers Writers Problem

A database is to be shared among several concurrent processes. some processes may want only to read the database, some may want to update the database.If two readers access the shared data simultaneously no problem .if a write, some other process access the database simultaneously problem arised. Writes have exclusive access to the shared database while writing to the database.This problem is known as readers- writes problem.

First readers-writers problem
No reader be kept waiting unless a writer has already obtained permission to use the shared resource.
Second readers-writes problem:
Once writer is ready,that writer performs its write as soon as possible.
A process wishing to modify the shared data must request the lock in write mode. multiple processes are permitted to concurrently acquire a reader-writer lock in read mode. A reader writer lock in read mode. but only one process may acquire the lock for writing as exclusive access is required for writers.
Semaphore mutex initialized to 1
- Semaphore wrt initialized to 1
- Integer read count initialized to 0

The structure of a writer process

```
do {
wait (wrt) ;
// writing is performed
signal (wrt) ;
} while (TRUE);
```

The structure of a reader process

```
do {
wait (mutex) ;
readcount ++ ;
if (readcount == 1)
wait (wrt) ;
signal (mutex)
// reading is performed wait (mutex) ;
readcount - - ;
if (readcount == 0)
signal (wrt) ;
signal (mutex) ;
} while (TRUE);
```

8. **Calculate the average waiting time, turnaround time for i) FCFS(ii) SJF(Preemptive) and iii)Round Robin(tq=4ms) with the following set of process**.

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

Solution:

Process ID	**Arrival Time**	**Burst Time**
P1	0	8
P2	1	4
P3	2	9
P4	3	5

1) First Come First Serve (FCFS) Scheduling

Step 1: Arrange processes in order of arrival time.
Processes already sorted by arrival time.

Step 2: Create Gantt Chart

- P1 starts at **0** ms and executes for **8** ms.
- P2 starts at **8** ms and executes for **4** ms.
- P3 starts at **12** ms and executes for **9** ms.
- P4 starts at **21** ms and executes for **5** ms.

P1	P2	P3	P4

0 8 12 21 26

Step 3: Calculate Waiting & Turnaround Times
Turnaround Time (TAT) = Completion Time - Arrival Time
Waiting Time (WT) = Turnaround Time - Burst Time

Process	**Arrival Time**	**Burst Time**	**Completion Time**	**Turnaround Time (TAT)**	**Waiting Time (WT)**
P1	0	8	8	8 - 0 = 8	8 - 8 = 0
P2	1	4	12	12 - 1 = 11	11 - 4 = 7
P3	2	9	21	21 - 2 = 19	19 - 9 = 10
P4	3	5	26	26 - 3 = 23	23 - 5 = 18

Step 4: Calculate Averages

- **AWT** = (0 + 7 + 10 + 18) / 4 = **8.75 ms**
- **ATAT** = (8 + 11 + 19 + 23) / 4 = **15.25 ms**

2. Shortest Job First (SJF - Preemptive)

Step 1: Sort Processes by Arrival Time & Execute Shortest Job First

We track time step by step:

Ready Queue:

P1	P2	P4	P1	P3

Gantt Chart

P1	P2	P4	P1	P3

0 1 5 10 17 26

Step 2: Calculate Waiting & Turnaround Times

Turnaround Time (TAT) = Completion Time - Arrival Time
Waiting Time (WT) = Turnaround Time - Burst Time

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	8	17	17 - 0 = 17	17 - 8 = 9
P2	1	4	5	5 - 1 = 4	4 - 4 = 0
P3	2	9	26	26 - 2 = 24	24 - 9 = 15
P4	3	5	10	10 - 3 = 7	7 - 5 = 2

Step 3: Calculate Averages

- **AWT** = (9 + 0 + 15 + 2) / 4 = **6.5 ms**
- **ATAT** = (17 + 4 + 24 + 7) / 4 = **13 ms**

3.Round Robin (Time Quantum = 4ms)

Step 1: Gantt Chart Execution Order

- **P1** executes for 4ms, then **P2** executes for 4ms (completes)
- **P3** executes for 4ms, then **P4** executes for 4ms
- **P1** executes remaining 4ms, then **P3** executes for 4ms
- **P3** executes for remaining 1ms

Ready Queue:

P1	P2	P3	P4	P1	P3	P4	P3

Gantt chart:

P1	P2	P3	P4	P1	P3	P4	P3

0 4 8 12 16 20 24 25 26

Step 2: Calculate Waiting & Turnaround Times

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	8	20	20 - 0 = 20	20 - 8 = 12
P2	1	4	8	8 - 1 = 7	7 - 4 = 3
P3	2	9	26	26 - 2 = 24	24 - 9 = 15
P4	3	5	25	25 - 3 = 22	22 - 5 = 17

Step 3: Calculate Averages

- **AWT** = (12 + 3 + 15 + 17) / 4 = **11.75 ms**
- **ATAT** = (20 + 7 + 24 + 22) / 4 = **18.25 ms**

PART V

9.What is contiguous and non-contiguous memory allocation? Discuss its advantages and disadvantages.

Memory is a huge collection of bytes, and memory allocation refers to allocating space to computer applications. There are mainly two types of memory allocation: contiguous and non-contiguous memory allocation. Contiguous memory allocation allows a single memory space to complete the tasks. On the other hand, non-contiguous memory allocation assigns the method to distinct memory sections at numerous memory locations.

Contiguous Memory Allocation

It is the type of *memory allocation method*. When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirements.

It is completed by partitioning the memory into fixed-sized partitions and assigning every partition to a single process. However, it will limit the degree of multiprogramming to the number of fixed partitions done in memory.

This allocation also leads to internal fragmentation. For example, suppose a fixed-sized memory block assigned to a process is slightly bigger than its demand. In that case, the remaining memory space in the block is referred to as internal fragmentation. When a process in a partition finishes, the partition becomes accessible for another process to run.

The OS preserves a table showing which memory partitions are free and occupied by processes in the variable partitioning scheme. Contiguous memory allocation speeds up process execution by decreasing address translation overheads.

Advantages and Disadvantages of Contiguous Memory Allocation

There are various advantages and disadvantages of contiguous memory allocation. Some of the advantages and disadvantages are as follows:

Advantages

1. It is simple to keep track of how many memory blocks are left, which determines how many more processes can be granted memory space.
2. The read performance of contiguous memory allocation is good because the complete file may be read from the disk in a single task.
3. The contiguous allocation is simple to set up and performs well.

Disadvantages

1. Fragmentation isn't a problem because every new file may be written to the end of the disk after the previous one.
2. When generating a new file, it must know its eventual size to select the appropriate hole size.
3. When the disk is filled up, it would be necessary to compress or reuse the spare space in the holes.

Non-Contiguous Memory Allocation

It allows a process to obtain multiple memory blocks in various locations in memory based on its requirements. The non-contiguous memory allocation also reduces memory wastage caused by ***internal*** and ***external*** fragmentation because it uses the memory holes created by internal and external fragmentation.

The two methods for making a process's physical address space non-contiguous are paging and segmentation. Non-contiguous memory allocation divides the process into blocks ***(pages or segments)*** that are allocated to different areas of memory space based on memory availability.

Non-contiguous memory allocation can decrease memory wastage, but it also raises address translation overheads. As the process portions are stored in separate locations in memory, the memory execution is slowed because time is consumed in address translation.

Advantages and Disadvantages of Non-Contiguous Memory Allocation

There are various advantages and disadvantages of non-contiguous memory allocation. Some of the advantages and disadvantages are as follows:

Advantages

1. It has the advantage of reducing memory waste, but it increases overhead because of the address translation.
2. It slows down the memory execution because time is consumed in address translation.

Disadvantages

1. The downside of this memory allocation is that the access is slow because you must reach the other nodes using pointers and traverse them.

10. Explain the concept of swapping in memory management. How does it help in process execution?

SWAPPING

When a process is executed it must have resided in memory. Swapping is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time.

The main part of swapping is transferred time and the total time is directly proportional to the amount of memory swapped.

Swapping is also known as roll-out, or roll because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.

Advantages

- If there is low main memory so some processes may has to wait for much long but by using swapping process do not have to wait long for execution on CPU.
- Using only single main memory, multiple process can be run by CPU using swap partition.
- The concept of virtual memory start from here and it utilize it in better way.
- This concept can be useful in priority based scheduling to optimize the swapping process.

Disadvantages

- If there is low main memory resource and user is executing too many processes and suddenly the power of system goes off there might be a scenario where data get erase of the processes which are took parts in swapping.
- Chances of number of page faults occur
- Low processing performance

Example:

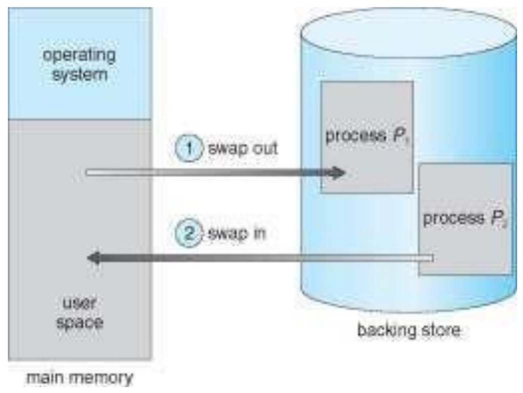
Suppose the user process's size is 2048KB and is a standard hard disk where swapping has a data transfer rate of 1Mbps. Calculate how long it will take to transfer from main memory to secondary memory.

User process size is 2048Kb

Data transfer rate is 1Mbps = 1024 kbps Time = process size / transfer rate

= 2048 / 1024 = 2 seconds or 2000 milliseconds

Now taking swap-in and swap-out time, the process will take 4000 ms or 4 seconds.



Swapping in OS is a memory management technique that temporarily swaps processes from main memory to secondary memory or vice versa which helps to increase the degree of multi-programming and increase main memory utilisation.
There are two steps in the process of Swapping in the operating system:

1. **Swap In:** Swap-in is the process of bringing a process from secondary storage/hard disc to main memory (RAM).
2. **Swap Out:** Swap-out takes the process out of the Main memory and puts it in the secondary storage.

Working of Swapping technique in Operating System

Swapping is a memory management technique employed by the operating system to efficiently utilize the limited physical memory (RAM) available in a computer system. When the system's RAM becomes overloaded due to running multiple processes simultaneously, some parts of a process or an entire process might be temporarily moved to secondary storage (usually the hard disk) to free up space for other processes.
The process of swapping involves the following steps:

Selection

The operating system selects a process that needs to be swapped out of the main memory. This selection is often based on priority, execution time, or other criteria.

Data Transfer

The data of the selected process, including its code and data, is transferred from the RAM to a designated area on the disk known as the swap space.

Process Suspension

The selected process is suspended, and its state, including register contents and program counter, is stored in memory.

Loading

If another process needs to be brought into memory for execution, the operating system selects a process from the disk's swap space, loads it into the RAM, and restores its saved state.

Resumption

The resumed process continues its execution from where it was suspended, typically unaware of being temporarily swapped out.

Swapping helps manage the memory demands of various processes efficiently, allowing more processes to run than the physical RAM can accommodate. However, swapping can introduce delays due to the time needed to move data between main memory and the disk. To minimize this performance impact, modern operating systems use sophisticated algorithms to decide which processes to swap and when, aiming to optimize overall system performance while ensuring smooth multitasking.