		OR			
11	a)	Explain about bit vector and linked list in free space management.	L4	CO4	5 M
	b)	Explain the principles of protection with examples.	L2	CO1	5 M

PVP 23

Code: 23AM3503

III B.Tech - I Semester - Regular Examinations - NOVEMBER 2025

OPERATING SYSTEMS
(CSE - AIML)

Duration: 3 hours Max. Marks: 70

Note: 1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place.

BL – Blooms Level CO – Course Outcome

PART – A

		BL	CO
1. a)	What is multiprogramming?	L2	CO1
b)	How does the fork() system call work?	L2	CO1
c)	What is turnaround time?	L2	CO1
d)	What is interprocess communication?	L2	CO1
e)	Define monitor.	L2	CO1
f)	What is the critical section in process synchronization?	L2	CO1
g)	Define thrashing.	L2	CO1
h)	What is virtual memory?	L2	CO1
i)	What are the key components of a file system structure?	L2	CO1
j)	Define protection rings.	L2	CO1

PART – B

			BL	CO	Max. Marks
		UNIT-I			
2	a)	What are system programs? Give two examples.	L2	CO1	5 M
	b)	List and explain the major categories of system calls.	L2	CO1	5 M
		OR			
3	a)	Discuss the various computing environments in which operating systems work.	L2	CO1	5 M
	b)	Demonstrate the various types of user interfaces provided by operating systems.	L2	CO1	5 M
		UNIT-II			
4	a)	What are the main goals of CPU scheduling in operating systems?	L2	CO1	5 M
	b)	Illustrate multithreading models.	L3	CO2	5 M
		OR			
5	a)	Demonstrate the First-Come First-Served (FCFS) scheduling algorithm with an example.	L3	CO2	5 M
	b)	Explain about Process Concept and queuing diagram.	L2	CO1	5 M
		UNIT-III			
6	a)	Provide an example of implementing mutual exclusion using semaphores.	L3	CO2	5 M

	b)	List the four necessary conditions for deadlock occurrence.	L2	CO1	5 M
		OR			
7	a)	Demonstrate the Producer-Consumer Problem and how can it be solved using semaphores.	L3	CO3	5 M
	b)	Explain in detail about deadlock avoidance.	L4	CO4	5 M
		UNIT-IV			
8	a)	Demonstrate the First-In-First-Out (FIFO) page replacement algorithm with example.	L3	CO3	5 M
	b)	Illustrate the allocation of frames in virtual memory.	L3	CO3	5 M
		OR			
9	a)	Analyze the Least Recently Used (LRU) algorithm work with example.	L4	CO4	5 M
	b)	Analyze FCFS and SCAN disk scheduling algorithms for I/O requests on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. The disk head is initially at 53.	L4	CO4	5 M
		UNIT-V			
10	a)	Illustrate the contiguous and linked allocations with neat diagram.	L3	CO3	5 M
	b)	Compare and contrast sequential and linked access methods.	L4	CO4	5 M

23AM3503

PVP23

III B. Tech – I Semester-Regular Examinations-NOVEMBER 2025
OPERATING SYSTEMS
(CSE - AIML)

Duration: 3 Hours **Max. Marks: 70**

Note:

1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place

PART-A

10X2=20M

Q No	Question	Marks Awarded
1(a)	**What is multiprogramming?**	2M
	Definition -2m	
1(b)	**How does the fork() system call work?**	2M
	Working Procedure - 2M	
1(c)	**What is turnaround time?**	2M
	Definition -2M	
1(d)	**What is inter-process communication?**	2M
	Definition -2M	
1(e)	**Define monitor.**	2M
	Definition -2M	
1(f)	**What is the critical section in process synchronization?**	2M
	Definition -2M	
1(g)	**Define thrashing.**	2M
	Definition -2M	
1(h)	**What is virtual memory?**	2M
	Definition -2M	
1(i)	**What are the key components of a file system structure?**	2M
	Components of file system -2M	
1(j)	**Define protection rings.**	2M
	Definition -2M	

PART-B **5X10=50 M**

SNo	Question	Marks Awarded	Total Marks
	UNIT-1		
2(a)	**What are system programs? Give two examples.**		5M
	Definition of system programs	2M	
	Explain about any TWO examples	3M	
2(b)	**List and explain the major categories of system calls.**		
	System call definition	1M	5M
	List of System calls	1M	
	Explanation of any no. system calls	3M	
	OR		
3(a)	**Discuss the various computing environments in which operating systems work**		5M
	Explanation of Traditional computing, client server and peer to peer computing explanation	5M	
3(b)	**Demonstrate the various types of user interfaces provide by operating systems.**		5M
	CLI interface	2.5M	
	GUI Interface	2.5M	
	UNIT-II		
4(a)	**What are the main goals of CPU scheduling in operating systems?**		
	Scheduling definition	1M	5M
	Listing and explanation of Goals	4M	
4(b)	**Illustrate multithreading models**		5M
	Many to one, One to one and Many to many explanation	5M	
	OR		
5(a)	**Demonstrate the First-Come First-Served (FCFS) scheduling algorithm with an example.**		5M
	FCFS Explanation	1M	
	Own example Gantt chart and calculations	4M	
5(b)	**Explain about Process Concept and Queuing diagram.**		
	Process, Process states explanation	2M	5M
	Process Control Block	2M	
	Queuing Diagram explanation	1M	
	UNIT-III		
6(a)	**Provide an example of implementing mutual exclusion using semaphores.**		5M
	Deadlock Definition	1M	
	Listing and explanation of conditions	4M	
6(b)	**List the four necessary conditions for deadlock occurrence**		
	Deadlock Definition	1M	5M
	Listing and explanation of conditions	4M	
	OR		
7(a)	**Demonstrate the Producer -Consumer Problem and how can it be solved using semaphores.**		5M

	State the producer consumer problem and give solution process	3M	
	Explain code of mutual exclusion	2M	
7(b)	**Explain in detail about deadlock avoidance.**		5M
	Deadlock definition	1M	
	Safety and resource request algorithm ****NOTE: Example can be solved here= 4M***	4M	
	UNIT-IV		
8(a)	**Demonstrate the First-In-First-Out (FIFO) page replacement algorithm with example**		5M
	Writeup about page replacement algorithms	1M	
	FIFO algorithm solution with own example	4M	
8(b)	**Illustrate the allocation of frames in virtual memory.**		
	Definition	1M	5M
	Equal allocation	2M	
	Global vs Local allocation	2M	
	OR		
9(a)	**Analyze the Least Recently Used (LRU) algorithm works with example**		5M
	Writeup about page replacement algorithms	1M	
	LRU algorithm solution with own example	4M	
9(b)	**Analyse FCFS and SCAN disk scheduling algorithms for I/O requests on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. The disk head is initially at 53**		5M
	Write up about disk scheduling	1M	
	FCFS- graph and seek time calculation	2M	
	SCAN- graph and seek time calculation	2M	
	UNIT – V		
10(a)	**Illustrate the contiguous and linked allocations with neat diagram**		5M
	Contiguous allocation with diagram	2.5M	
	Linked allocations with diagram	2.5M	
10(b)	**Compare and contrast sequential and linked access methods**		5M
	Access methods comparison	5M	
	OR		
11(a)	**Explain about bit vector and linked list in free space management.**		5M
	Free space management definition	1M	
	Bit Vector explanation	2M	
	Linked list explanation	2M	
11(b)	**Explain the principles of protection with examples.**		5M
	Protection Definition	1M	
	Principles listing and explanation	4M	

PVP23

Code No: 23AM3503

III B. Tech – I Semester-Regular Examinations-NOVEMBER 2025
OPERATING SYSTEMS
(CSE - AIML)

Duration: 3 Hours **Max. Marks: 70**

Note:

1. This question paper contains two Parts A and B.
2. Part-A contains 10 short answer questions. Each Question carries 2 Marks.
3. Part-B contains 5 essay questions with an internal choice from each unit. Each Question carries 10 marks.
4. All parts of Question paper must be answered in one place

PART-A

1(a) What is multiprogramming? **(2M)**

ANS: Multiprogramming is the ability of an operating system to keep multiple programs in memory at the same time and execute them concurrently. When one program waits for I/O, the CPU is assigned to another, improving CPU utilization and throughput.

1(b) How does the fork() system call work? **(2M)**

ANS: The fork() call is used to create a new process. The fork() system call creates a new process by duplicating the calling process. The new process is called the child process and is an exact copy of the parent process except for the returned value. After fork(), both parent and child run separately. fork() returns a zero to the child process and returns the child's process ID to the parent, allowing both processes to run concurrently.

1(c) What is turnaround time? **(2M)**

ANS: The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting in the ready queue, executing on the CPU, and doing I/O

1(d) What is inter-process communication? **(2M)**

ANS: Cooperating processes require an inter-process communication (IPC) mechanism that will allow them to exchange data that is, send data to and receive data from each other. There are two fundamental models of inter-process communication: shared memory and message passing. In the shared-memory model, a region of memory that is shared by the cooperating processes is established. In the message-passing model communication takes place by means of messages exchanged between the cooperating processes

1(e) Define monitor? **(2M)**

ANS: A monitor is an abstract data type that provides a high-level form of process synchronization. A monitor uses condition variables that allow processes to wait for certain conditions to become true and to signal one another when conditions have been set to true. Only one process can be inside the monitor at a time, so it prevents conflicts automatically

1(f) What is the critical section in process synchronization? (2M)

ANS: A critical section is a segment of code where a process accesses shared resources. To prevent race conditions, only one process is allowed to execute in its critical section at a time, to maintain data consistency. Enforced by using synchronization tools like semaphores or mutexes etc.

1(g) Define thrashing. (2M)

ANS: Thrashing occurs when a system spends more time swapping pages in and out of memory rather than executing processes. It happens when there is insufficient memory, causing excessive paging and degrading system performance.

1(h) What is virtual memory? (2M)

ANS: Virtual memory is a memory management technique used by modern operating systems to create an illusion of a having a large and continuous memory space, even when the physical memory (RAM) is less. A part of the hard disk is used as the physical memory to store data and programs that are not currently in use

1(i) What are the key components of a file system structure? (2M)

ANS: The main components include the file system interface, file concept, directory structure, file system mounting, file sharing, and protection mechanisms. These components organize how files are stored, accessed, and managed efficiently.

1(j) Define protection rings. (2M)

ANS: In operating systems, protection rings are a hierarchical layering mechanism for security in systems by set of privilege levels used by the CPU. Where the innermost ring has the most privileges (kernel mode) and outer rings have less privilege (user mode). They enforce access controls and protect system integrity.

Ring 0 → Highest privilege (kernel mode)
Ring 3 → Lowest privilege (user mode)
Intermediate rings (Ring 1, Ring 2) are available but often unused in modern systems.

PART-B

UNIT – I

2(a) What are system programs? Give two examples. **(5M)**

ANS: System programs, also known as system utilities provide a convenient environment for program development and execution. While the kernel provides the basic functions like CPU scheduling, memory management, and device handling etc, the system programs extend these functions to make the computer more usable and user-friendly. They act as an interface between the user and the operating system, offering higher-level services such as file management, status information, programming language support, program loading and execution, and communication facilities.

They can be divided into:

- **File management**: These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.
    - **Ex:** cp, mv, rm etc
- **Status information:** Provide system status, performance statistics, logging, and monitoring and debugging information.
- **Programming language support:** Compilers, assemblers, debuggers, and interpreters for development of programming languages are often provided with the operating system.
    - **Ex:** GCC compiler, Python interpreter
- **Program loading and execution:** Once a program is assembled or compiled, it must be loaded into memory for execution. So, Loaders, linkers, and libraries that prepare programs for execution.
    - **Ex:** ld, dynamic link libraries (DLLs)
- **Communications:** These programs enable communication among processes, users, and computer systems. Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
    - **Ex:** FTP, SSH
- **Application programs (Utility):** Most operating systems are supplied with programs that are useful in solving common problems or performing common operations. Such application programs include, Web browsers, word processors, spreadsheets and database systems
    - **Ex:** Text editors, browsers, word processors

NOTE: STUDENTS CAN EXPLAIN ANY 2 FROM ABOVE

2(b) List and explain the major categories of system calls. **(5M)**

ANS: System call is the special function that is used by the process to request action from the operating system. It provides the interface between the process and the operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks may have to be written using assembly-language instructions. They cover many functions grouped broadly into categories as follows:

Process Control, File Manipulation, Device Manipulation, Information Maintenance, Communications, And Protection.

- **Process control:** System calls that create, execute, and terminate processes, ensuring proper synchronization and resource use
  - create process, terminate process
  - load, execute
  - get process attributes, set process attributes
  - wait event, signal event
  - allocate and free memory
- **File management:** System calls that handle file operations such as creating, opening, reading, writing, and deleting files
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- **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. Otherwise, the process will have to wait
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- **Information maintenance:** System calls that provide or set system and process information like time, process ID, or file attributes
  - get time or date, set time or date
  - get system data, set system data
  - get process, file, or device attributes
  - set process, file, or device attributes
- **Communications:** System calls that enable data exchange between processes, either through message passing or shared memory
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices
- **Protection:** System calls that controls the access of process and resources for preventing misuse
  - get file permissions
  - set file permissions

NOTE: STUDENTS CAN EXPLAIN ANY MAJOR FRM ABOVE

OR

3(a) Discuss the various computing environments in which operating systems work. (5M)

ANS: A computing environment refers to the overall setup in which a computer system operates, including the hardware, software, operating system, and network resources that together provide services to users and applications. It defines how users interact with the system and how resources are managed.

Operating systems are used in a variety of computing environments or evaluation of operating systems will be discussed here.

1. Traditional Computing
2. Client server Computing
3. Peer to peer Computing

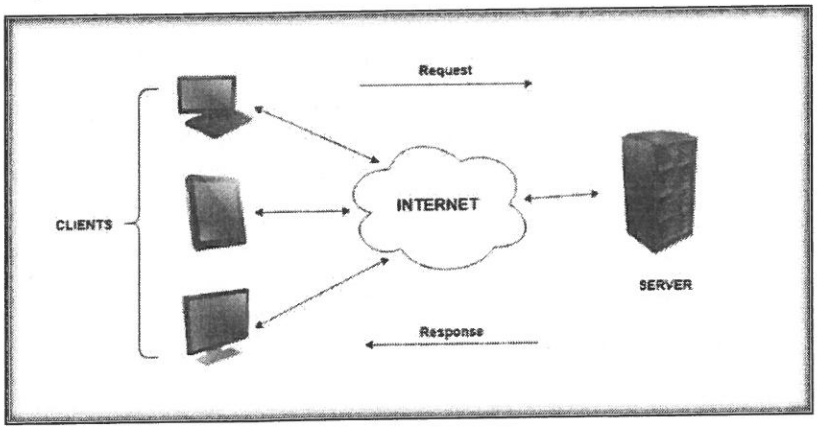
1. Traditional Computing:

- Computing has changed a lot over time. Earlier, office computers were linked to local servers and laptops were the only way to work remotely.
- Now, faster internet and web tools let people work from anywhere using online company portals. At home, slow modems have been replaced by fast networks that connect printers, servers, and even host websites, all protected by firewalls.
- Historically, computing resources were either batch-processing process in bulk or interactive responding to user input. Now, time-sharing systems emerged to let multiple users share resources

2. Client server Computing: A client–server system is a way computers talk to each other. One computer (the client) asks for something like a file or webpage. Another computer (the server) finds it and sends it back. Modern networks use a client–server model, where servers respond to requests from client devices (like computers or phones). There are two main kinds:

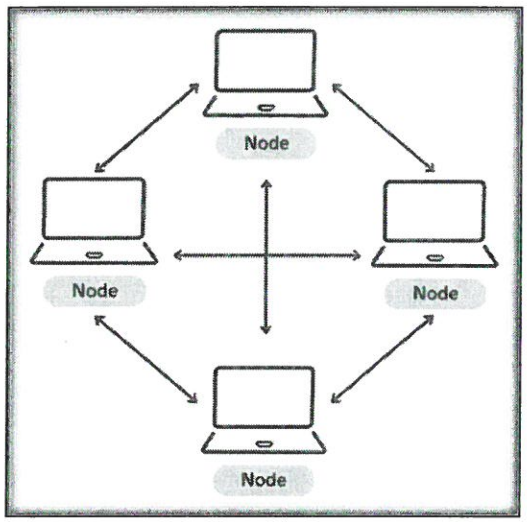
- **Compute Servers:** Do tasks for clients—like searching a database and sending back results.
- **File Servers:** Store and share files—like a web server sending webpages or videos to your browser.

This system helps organize tasks and resources so users can access what they need quickly and smoothly



3. Peer to Peer Computing:

- In a peer-to-peer (P2P) system, all computer called nodes are treated equally, without separating them into clients and servers.
- Any node can request or provide services depending on its role at the moment. This model helps avoid server bottlenecks common in client–server setups by distributing services across many nodes in the network.
- To participate, a node first joins the network and then either registers its services with a centralized lookup system or uses a discovery protocol that broadcasts requests to other nodes.
- Popular examples of P2P networks include Napster and Gnutella, which allowed users to share files directly. Napster used a central index to match users with files, while Gnutella relied on broadcasting to locate them. However, legal issues around copyright led to Napster's shutdown.
- Skype is another P2P example, using a hybrid method: a central server for login and decentralized peers for actual voice, video, and message communication. This approach allows flexible sharing and interaction among many users over a distributed network.
- Most satellite systems follow a centralized model, where ground stations control and coordinate satellite operations



3(b) Demonstrate the various types of user interfaces provide by operating systems. (5M)

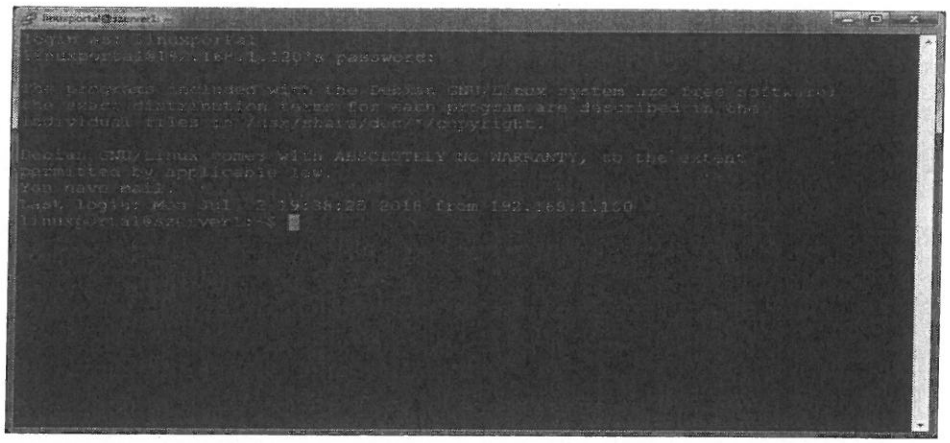
ANS: There are several ways for users to interface with the operating system. One allows users to directly enter commands to be performed by the operating system. The other allows users to interface with the operating system via a graphical user interface, or GUI.

i. Command Interpreters (CLI)

- CLI allows direct command entry.
- Most operating systems, including Linux, UNIX, and Windows, treat the command interpreter as a special program that is running when a process is initiated or when a user first logs on (on interactive systems). On systems with multiple command interpreters to

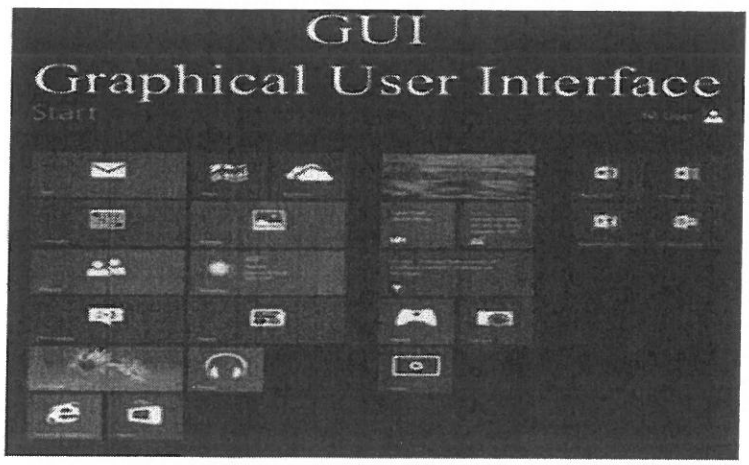
choose from, the interpreters are known as shells. For example, on UNIX and Linux systems, a user may choose among several different shells, including the Cshell, Bourne-Again shell, Korn shell, and others.

- The main function of the command interpreter is to get and execute the next user-specified command. Many of the commands given at this level manipulate files: create, delete, list, print, copy, execute, and so on.
- It have the disadvantage of command complexity, difficult to remember etc.



ii. Graphic User Interfaces (GUI)

- A second strategy for interfacing with the operating system is through a user-friendly graphical user interface, or GUI. Here, rather than entering commands directly via a command-line interface, users employ a mouse-based window and-menu system characterized by a desktop metaphor.
- The user moves the mouse to position its pointer on images, or icons, on the screen that represent programs, files, directories. Depending on the mouse pointer's location, clicking a button on the mouse can invoke a program.
- Many systems now include both CLI and GUI interfaces
  - Microsoft Windows is GUI with CLI "command" shell
  - Apple Mac OS X is "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)



UNIT-II

4(a)What are the main goals of CPU scheduling in operating systems? (5M)

ANS: CPU Scheduling is the process by which the operating system decides which process from the ready queue will be allocated the CPU for execution. Since only one process can run on the CPU at a time, ensuring efficient and fair resource utilization. It is a fundamental part of process management, aiming to The goals are:

1. Maximize CPU Utilization
2. Maximize Throughput
3. Minimize Turnaround Time
4. Minimize Waiting Time
5. Minimize Response Time
6. Ensure Fairness

All the goals are explained below:

1. Maximize CPU Utilization: CPU utilization refers to keeping the processor as busy as possible. The operating system schedules processes in such a way that the CPU rarely sits idle, ensuring efficient use of resources and improving overall system performance.

2. Maximize Throughput: Throughput is the number of processes completed in a given time period. A good scheduling algorithm increases throughput by reducing delays and ensuring that more jobs finish execution within less time, thereby improving system productivity.

$$\textbf{Throughput} = \frac{\text{Number of processes completed}}{\text{Total Time}}$$

3. Minimize Turnaround Time: Turnaround time is the total time taken from the submission of a process until its completion. Scheduling aims to minimize this duration so that users and applications receive results faster, which is especially important in batch processing systems.

Turnaround Time= Completion time- Arrival time

4. Minimize Waiting Time: Waiting time is the amount of time a process spends in the ready queue before getting CPU access. Efficient scheduling reduces waiting time, ensuring that processes are not left waiting unnecessarily and preventing starvation of lower-priority jobs

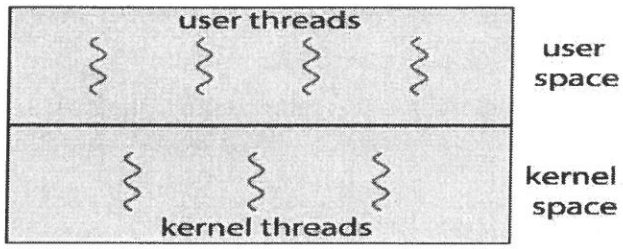
Waiting Time = Turnaround time -Burst time

5. Minimize Response Time: Response time is the interval between process submission and the first response or output. In interactive systems, minimizing response time is crucial because users expect quick feedback from the system, making the environment more responsive and user-friendly.

6. Ensure Fairness: Fairness means that all processes get a reasonable share of CPU time and no process is indefinitely delayed. Scheduling policies must prevent starvation and balance system performance with equitable distribution of resources among all processes.

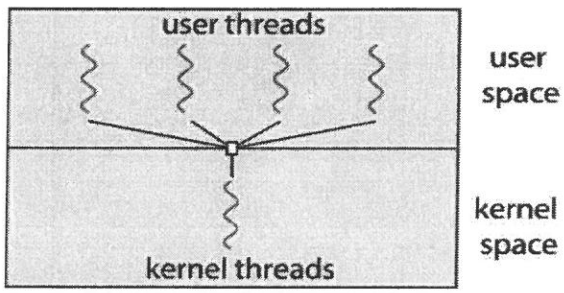
4(b) Illustrate multithreading models **(5M)**

ANS: User threads are managed by user-level thread libraries without the kernel support. Three primary thread libraries are POSIX Pthreads, Windows threads, Java Threads. Kernel threads are supported and managed directly by the operating system. Virtually all general-purpose operating systems including Windows, Linux, Mac OS X, and Solaris support kernel threads.



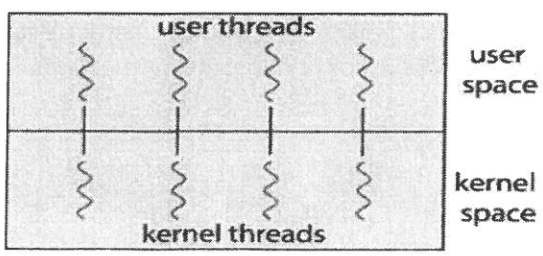
1.1. Many-to-one Model

- The many-to-one model maps many user-level threads to one kernel thread. Thread management is done by the thread library in user space, so it is efficient.
- The entire process will block if a thread makes a blocking system call.
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time. Very few systems continue to use the model because of its inability to take the advantage of multiple processing cores.
- **Examples**: Solaris Green Threads and GNU Portable Threads



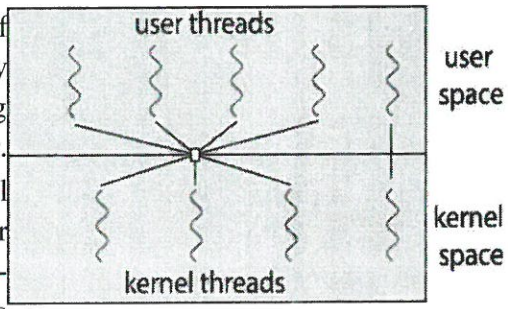
1.2. One-to-One Model

- The one-to-one model maps each user thread to a kernel thread.
- It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call. It also allows multiple threads to run in parallel on multiprocessors.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Number of threads per process sometimes restricted due to overhead. **Ex**: Windows and Linux



1.3. Many-to-Many Model

- Allows many user-level threads to be mapped to many kernel threads.
- The many-to-one model allows the developer to create as many user threads as he wishes, it does not result in true concurrency, because the kernel can schedule only one thread at a time.
- The one-to-one model allows greater concurrency, but the developer has to be careful not to create too many threads within an application.
- The many-to-many model suffers from neither of these shortcomings: developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
- One variation of the many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads but also allows a user-level thread to be bound to a kernel thread. This variation is sometimes referred to as the two-level model.



OR

5(a) Demonstrate the First-Come First-Served (FCFS) scheduling algorithm with an example. (5M)

ANS: The simplest CPU-scheduling algorithm is the first-come first-serve (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. On the negative side, the average waiting time under the FCFS is often quite long. Thus, FCFS is fair but may cause the convoy effect when long processes delay shorter ones.

For example, Steps:

1. Any relevant example can be taken
2. Draw the Gantt chart according to FCFS scheduling algorithm
3. Calculate the Turnaround Time, Waiting Time in a Tabular Form by using the below formulae.

Turnaround Time (TAT):

$$TAT = Completion\ Time - Arrival\ Time$$

Waiting Time (WT):

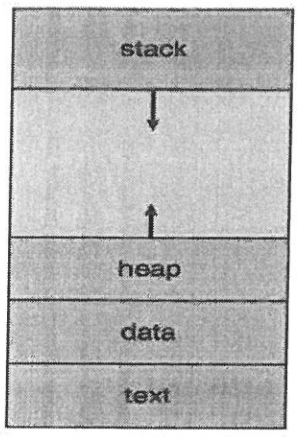
$$WT = TAT - Burst\ Time$$

4. Calculate the final Average Waiting time and Average Turnaround time.

5(b) Explain about Process Concept and Queuing diagram. **(5M)**

ANS: Process Concept: A process is the unit of work in a modern time-sharing system. An operating system executes a variety of programs that run as a process.

1.1. The Process



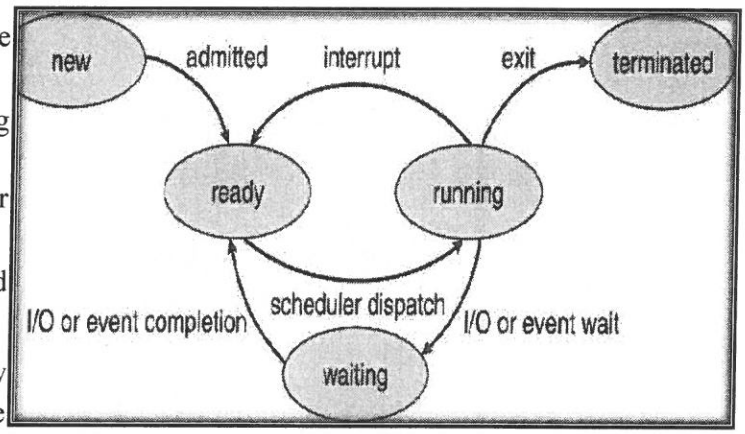
- **Process** – a program in execution; process execution must progress in sequential fashion. No parallel execution of instructions of a single process

The Memory Layout of a process:

- The program code, also called text section
- Stack containing temporary data
  - Function parameters, return addresses, local variables
- Data section containing global variables
- Heap containing memory dynamically allocated during run time
- **Program** is passive entity stored on disk (executable file): **process** is active. Program becomes process when an executable file is loaded into memory.

1.2. PROCESS STATES

As a process executes, it changes state. The current activity of the process defines its state.



- New: The process is being created
- Ready: The process is waiting to be assigned to a processor
- Running: Instructions are being executed
- Waiting: The process is waiting for some event to occur
- Terminated: The process has finished execution
- These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems.
- Only one process can be running on any processor core at any instant of time. Many processes may be ready and waiting, however. The state diagram corresponding to these states is presented in Figure above

1.3. PROCESS CONTROL BLOCK

Each process is represented in the operating system by a process control block (PCB) also called a task control block.

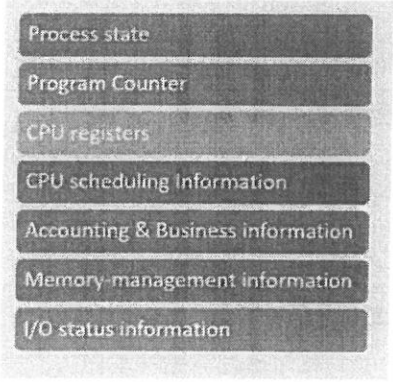


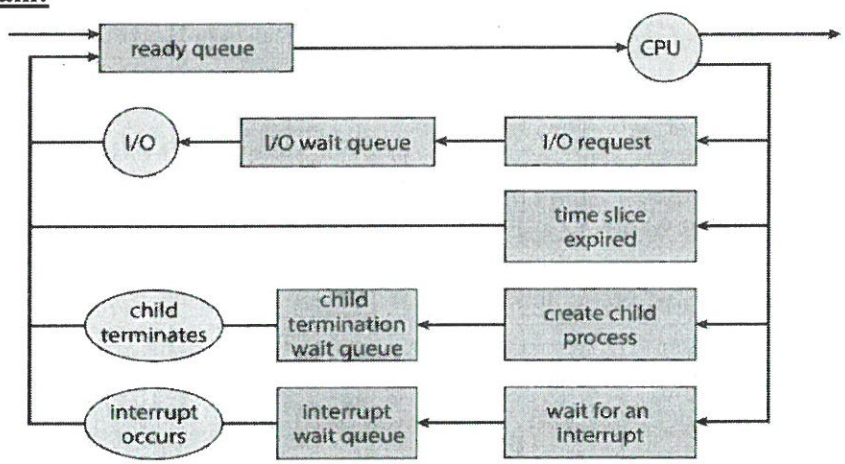
Fig 4 Process Control Block (PCB)

- Process state – the state many be new, running, waiting, halted, and so on.
- Program counter – address of next instruction to execute
- CPU registers – contents of all process, centric registers, they include accumulators, index registers, stack pointers, and

general-purpose registers.

- CPU scheduling information- set process priority, pointers to scheduling queues, and any other scheduling parameters.
- Memory-management information : memory allocated to the process: value of the base and limit registers and the page tables, or the segment tables.
- Accounting information – amount of CPU time used, time limits, account numbers, job or process numbers
- I/O status information – List of I/O devices allocated to process, list of open files and so on.
- In brief, the PCB simply serves as the repository for any information that may vary from process to process.
- PCB will be in OS. Without PCB, it can't execute any process

Queuing Diagram:



- A common representation for a discussion of process scheduling is a **queueing diagram**. Each rectangular box represents a queue.
- A new process is initially put in the ready queue. It waits there until it is selected for execution.Once the process is allocated the CPU and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O queue.
  - The process could create a new subprocess and wait for the subprocess's termination.
  - The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

UNIT-III

6(a) Provide an example of implementing mutual exclusion using semaphores. (5M)

ANS: Mutual exclusion ensures that only one process accesses a critical section at a time, preventing race conditions. Semaphores, a synchronization tool, can be used to achieve this. A binary semaphore initialized to 1 is used to control access. Before entering the critical section, a process calls the **wait (P)** operation on the semaphore which decrements its value. If the semaphore's value becomes less than or equal to 0, the process waits. When leaving the critical section, the process calls the **signal (V)** operation which increments the semaphore, allowing another waiting process to enter.

Code explaining Mutual exclusion:

```
semaphore mutex = 1; // initialized to 1
process() {
   wait(mutex);        // Entry section: acquire lock
   critical_section(); // Critical section: only one process here at a time
   signal(mutex);      // Exit section: release lock
   remainder_section(); // Non-critical section
}
```

Process explanation:

- The first process to execute **wait(mutex)** finds the semaphore's value at 1, decrements it to 0, and enters the critical section.
- If another process calls **wait(mutex)** while the value is 0, it will be blocked until the semaphore is signalled by the first process finishing its critical section.
- When the first process calls **signal(mutex),** the semaphore value increments back to 1, permitting the waiting process to enter the critical section.

Example with P1 and P2 Process: Suppose two processes P1 and P2 want to access a shared file.

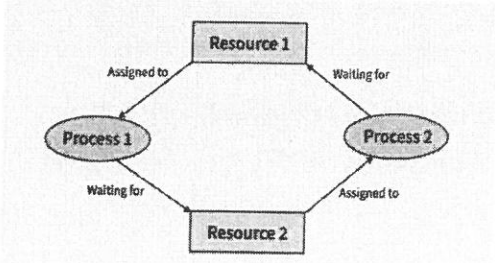
- P1 calls wait(mutex) → mutex becomes 0 → enters critical section.
- If P2 tries wait(mutex) while mutex = 0, it will be blocked until P1 executes signal(mutex).

This ensures only one process modifies shared resources at once, thereby preventing race conditions and ensuring mutual exclusion

6(b)List the four necessary conditions for deadlock occurrence **(5M)**

ANS: Deadlock is a situation in an operating system where a group of processes becomes permanently blocked because each process is holding a resource that another process needs, and none of them can proceed. This cycle of waiting causes the system to reach a standstill, with no process able to continue execution.

To better understand deadlock, consider a simple example: Process P1 holds Resource R1 and requests Resource R2, while Process P2 holds Resource R2 and requests Resource R1. Since neither can release their currently held resource until they get the other, both remain stuck, causing a deadlock.



A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. **Mutual exclusion**. At least one resource must be held in a non-sharable mode; that is, only one thread at a time can use the resource. If another thread requests that resource, the requesting thread must be delayed until the resource has been released.

2. **Hold and wait.** A thread must be holding at least one resource and waiting to acquire additional resources that are currently being held by other threads.

3. **No-preemption**. Resources cannot be preempted; that is, a resource can be released only voluntarily by the thread holding it, after that thread has completed its task.

4. **Circular wait**. A set{T0,T1,...,Tn} of waiting threads must exist such that T0 is waiting for a resource held by T1, T1 is waiting for a resource held by T2, ..., Tn−1 is waiting for a resource held by Tn, and Tn is waiting for a resource held by T0.

OR

7(a) Demonstrate the Producer -Consumer Problem and how can it be solved using semaphores.

(5M)

ANS: Producer puts information into the buffer, consumer takes it out. The problem arise when the producer wants to put a new item in the buffer, but it is already full. The solution is for the producer has to wait until the consumer has consumed atleast one buffer.

- Similarly, if the consumer wants to remove an item from the buffer and sees that the buffer is empty, it goes to sleep until the producer puts something in the buffer and wakes it up.
- Synchronization problems:
  1. We must guard against attempting to write data to the buffer when the buffer is full, i.e., the producer must wait for an 'empty space'.
  2. We must prevent the consumer from attempting to read data when the buffer is empty; i.e., the consumer must wait for 'data available'.
- To provide for each of these conditions, we require to employ three semaphores. The producer and consumer processes share the following data structure:

```
int n;

semaphore       mutex=1;
semaphore       empty=n;
semaphore full=0;
```

- We assume that the pool consists of **n** buffers, each capable of holding one item. The **mutex** semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.
- The empty and full semaphores count the number of empty and full buffers. The semaphore empty is initialized to **n**; the semaphore full is initialized to **0**.
- The code for the producer process is shown below:

```
do {

//produce an item in next_produced wait(empty) ;
```

```
        wait(mutex) ;

    // add next_produced to buffer
        signal(mutex) ; signal(full) ;

    }while (TRUE);
```

- The code for the consumer process is shown below:

```
    do {

        wait(full) ; wait
        (mutex) ;

        // remove an item from buffer to next_consumed signal(mutex) ;

        signal(empty) ;

        // consume the item in next_consumed

    }while(TRUE);
```

- We can interpret this code as the producer producing full buffers for the consumer or as the consumer producing empty buffers for the producer.

7(b) Explain in detail about deadlock avoidance. **(5M)**

ANS: Deadlock avoidance is a strategy where the operating system dynamically checks each resource request and grants it only if the system remains in a safe state.

The Banker's Algorithm is a classic example used for deadlock avoidance when multiple instances of resources exist. This approach guarantees that deadlock will never occur by ensuring the system never enters unsafe state.

• Multiple instances of resources. Each thread must a priori claim maximum use.

• When a thread requests a resource, it may have to wait. When a thread gets all its resources it must return them in a finite amount of time

Safety Algorithm

We can now present the algorithm for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

1. Let Work and Finish be vectors of length m and n, respectively. Initialize

Work =Available and Finish[i]=false for i = 0, 1, ..., n − 1.

2. Find an index i such that both

a. Finish[i]==false

b. Need(i) ≤ Work

If no such i exists, go to step 4.

3. Work =Work +Allocation(i)

Finish[i]=true

Goto step 2.

4. If Finish[i]==true for all i, then the system is in a safe state

Resource-Request Algorithm

Requesti = request vector for process Ti. If Requesti[j] = k then process Ti wants k instances of resource type Rj

1. If Requesti <= Needi go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If Requesti <= Available, go to step 3. Otherwise Ti must wait, since resources are not available
3. Pretend to allocate requested resources to Ti by modifying the state as follows:

Available = Available – Requesti; Allocationi = Allocationi + Requesti; Needi = Needi – Requesti;

If safe => the resources are allocated to Ti

If unsafe => Ti must wait, and the old resource-allocation state is restored.

Taking an example and explaining the above algorithms can also be considered.

UNIT-IV

8(a) Demonstrate the First-In-First-Out (FIFO) page replacement algorithm with example (5M)

ANS: Page replacement algorithms in operating systems are techniques used to determine which memory page should be removed when a page fault occurs and the physical memory is already full. These algorithms play a crucial role in efficient memory management by ensuring that the most appropriate page is replaced to minimize faults and optimize performance. The major page replacement strategies include First-In-First-Out (FIFO), Least Recently Used (LRU), Optimal Page Replacement, and others, each with its own method of deciding which page to evict and its own advantages and limitation.

The simplest page-replacement algorithm is a first-in, first-out (**FIFO**) algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. Notice that it is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue. Drawback: may replace frequently used pages; suffers from Belady's Anomaly.

Students can explain **FIFO** algorithm taking their own example by following the below working procedure:

1. Initialize Frames

•Decide the number of page frames available in memory.

• Start with all frames empty.

2. Process Each Page Reference

• Read the next page from the reference string (sequence of page requests).

• If the page is already in memory → Page Hit (no replacement needed).

• If the page is not in memory → Page Fault occurs.

3.Handle Page Fault

• If there is free space in memory → load the new page into the next empty frame and add it to the rear of the queue.

• If memory is full → remove the page at the front of the queue (the oldest one) and insert the new page at the rear.

4.Repeat Until End of Reference String

- Continue this process for all page requests.
- Count the total number of page faults to evaluate performance.

8(b) Illustrate the allocation of frames in virtual memory. **(5M)**

ANS: Allocation of Frames

• In this we see how **to assign pages of a process logical memory to a fixed set of allocated frames.** We also need to decide how many frames to allocate to each process. It is also possible to select the victim from a frame currently allocated to another process.

• There are fixed amount of free memory with various processes at different time in a system. The question is how this fixed amount of free memory is allocated among the different processes.

• Let us consider the single process system. All free memory for user programs can initially be put on the free frame list. When the user starts executing his program, it will generate a sequence of page faults. The user program would get all free frames from the free frame list.

• When this list was exhausted and the more free frames are required, the page replacement algorithm can be used to select one of the in-used pages to be replaced with the next required page and so on. After the program was terminated, all used pages are put on the free frame list again.

Equal Allocation

• Each process needs a certain minimum number of pages

i. Pages for instructions.

ii. Pages for local data.

iii. Pages for global data.

• Allocation may be fixed. Here "m" frames are splits among the "p" number of processes

$$\text{Frame allocation} = \frac{\text{Number of free frame}}{\text{Number of process}} = \frac{m}{p}$$

• For example, if there are 180 frames and 6 processes, give each process 30 frames.

• High priority jobs have same number of page frames and low priority jobs. Degree of multiprogramming might vary in equal allocation.

• Problem with equal allocation is that, there may be the wastage of free frames. Some process may require less number of free frames than allocated one. To solve this problem, proportional allocation method is used.

- **Proportional allocation**: Processes that have more logical memory get more frames.
- **Priority allocation :** In priority allocation, higher priority processes get more frames. System uses a proportional allocation scheme using priorities rather than size.

• If process P_i generates a page fault, then select for replacement one of its frames or select for replacement a frame from a process with lower priority number.

Global Vs Local Allocation

• If a process needs frames, should pages from other processes be discarded or just pages from the process which wants frames? Local policy means giving each process a share of the physical memory and swapping pages in and out on a per process basis.

• **Global allocation:** One process can select a replacement frame from the set of all frames. Process may not be able to control its page fault rate.

• **Local allocation:** Each process can only select from its own set of allocated frames. Process slowed down even if other less used pages of memory are available.

• Local page replacement is more predictable; depends on no external factors. A process which uses global page replacement cannot predict the page fault rate; may execute in 0.5 seconds once and 10.3 on another run. Overall, global replacement results in greater system throughput.

OR

9(a) Analyze the Least Recently Used (LRU) algorithm works with example (5M)

ANS: Page replacement algorithms in operating systems are techniques used to determine which memory page should be removed when a page fault occurs and the physical memory is already full. These algorithms play a crucial role in efficient memory management by ensuring that the most appropriate page is replaced to minimize faults and optimize performance. The major page replacement strategies include First-In-First-Out (FIFO), Least Recently Used (LRU), Optimal Page Replacement, and others, each with its own method of deciding which page to evict and its own advantages and limitation.

The Least Recently Used (**LRU**) algorithm is a page replacement technique in operating systems that replaces the page which has not been used for the longest period of time. It is based on the principle of temporal locality, which assumes that pages used recently are more likely to be used again soon.

Students can take their own example and explain **LRU** algorithm by following Working Principle

- The system keeps track of the order of page usage.
- When a page fault occurs and memory is full, the page that was least recently accessed is removed.
- This requires maintaining usage information such as timestamps, counters, or stacks.

Steps

1. **Initialize frames** – start with empty memory frame
2. **Process each page reference** – check if the page is in memory.
    - If present → **Page Hit** (no replacement).
    - If absent → **Page Fault** occurs.
3. **Handle page fault** –
    - If free space exists → load the page.
    - If memory is full → replace the page that was least recently used.
4. **Update usage record** – after each reference, update the access history so the algorithm knows which page is least recently used
5. Continue this process for all page requests.
    - Count the total number of page faults to evaluate performance.

9(b) Analyze FCFS and SCAN disk scheduling algorithms for I/O requests on cylinders 98, 183, 37, 122, 14, 124, 65, 67 in that order. The disk head is initially at 53 (5M)

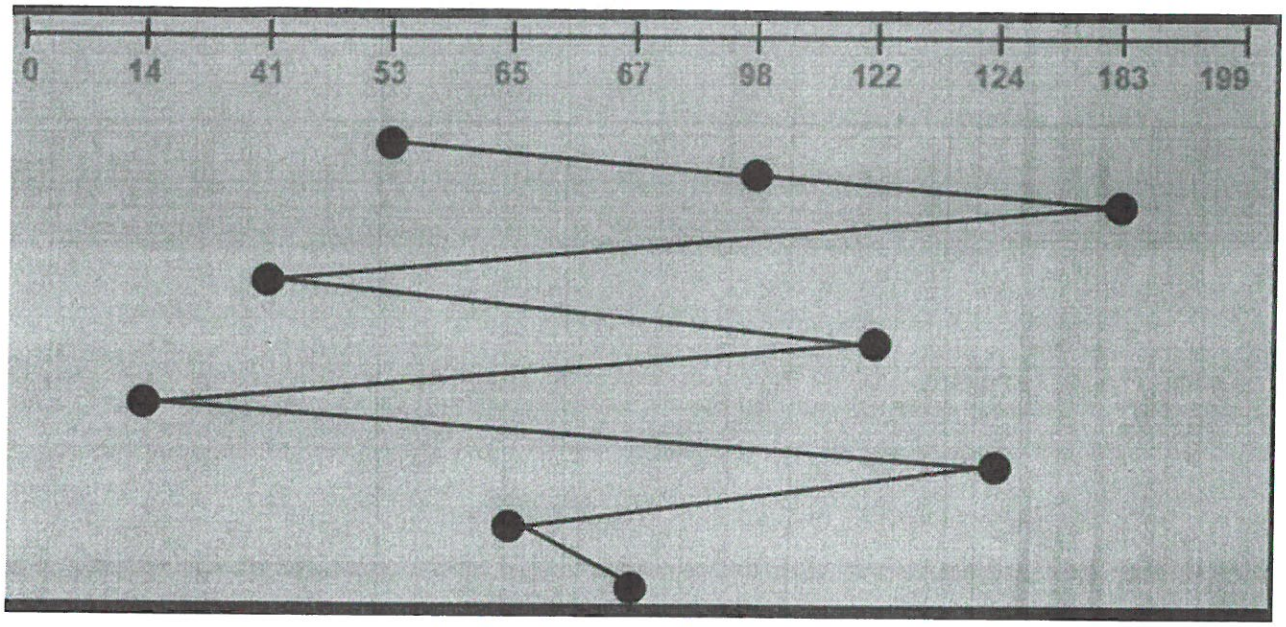
ANS: A Process makes the I/O requests to the operating system to access the disk. Disk Scheduling Algorithm manages those requests and decides the order of the disk access given to the requests. These algorithms help in minimizing the seek time by ordering the requests made by the processes. Seek Time is the time taken by the disk arm to locate the desired track.

They are various Disk Scheduling Algorithms:

1. **FCFS (First-Come, First-Served)**
2. **SSTF (Shortest Seek Time First)**
3. **SCAN (Elevator Algorithm)**
4. **C-SCAN (Circular SCAN)**
5. **LOOK**
6. **C-LOOK**

• **First Come First Serve (FCFS):** In this algorithm, the requests are served in the order they come in disk queue. Those who come first are served first. This is the simplest algorithm. It is simple, easy to understand and implement. It does not cause starvation to any request. It results in increased total seek time.

• The example is solved below : Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 37, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53.



Total head movements incurred while servicing these requests

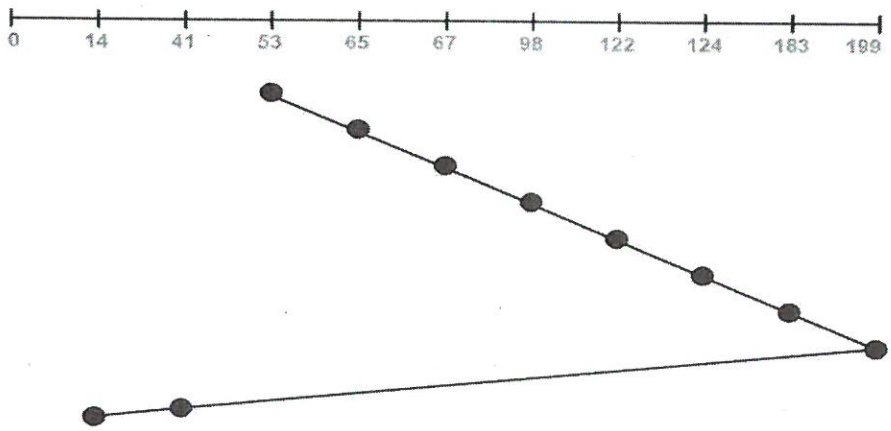
$= (98 - 53) + (183 - 98) + (183 - 37) + (122 - 37) + (122 - 14) + (124 - 14) + (124 - 65) + (67 - 65)$

$= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2$

Seek Time= 640.

SCAN Disk Scheduling Algorithm

- As the name suggests, this algorithm scans all the cylinders of the disk back and forth.
- Head starts from one end of the disk and move towards the other end servicing all the requests in between.
- After reaching the other end, head reverses its direction and move towards the starting end servicing all the requests in between.
- The same process repeats for all the cylinders.
- SCAN Algorithm is also called as **Elevator Algorithm.** This is because its working resembles the working of an elevator.



Total head movements incurred while servicing these requests

= (199 – 53) + (199 – 14)

= 146 + 185

Seek Time = 331

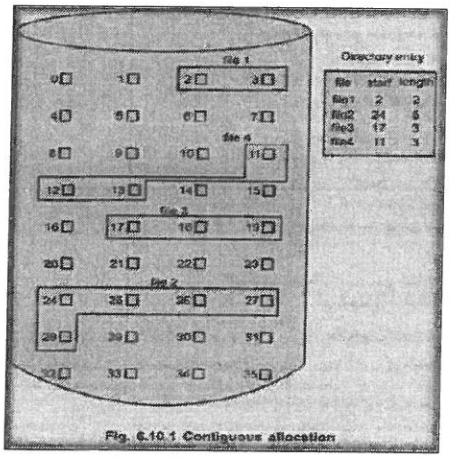
UNIT – V

10(a) Illustrate the contiguous and linked allocations with neat diagram **(5M)**

ANS: Many files are stored on the same device. The main problem is how to allocate space to these files so that storage space is utilized effectively and files can be accessed quickly. Three major methods of allocating secondary storage space are in wide use: **Contiguous, Linked, And Indexed**.

1) Contiguous Allocation

- When user creates a file, system allocates a set of contiguous blocks on disk. This is a pre-allocation method that uses portion of variable size. Contiguous allocation is simple method to implement. It only searches free list of correct number of consecutive blocks and marks them as used block.
- Disk address is a linear ordering on the disk. Because of linear ordering, accessing block b + 1 after block b normally requires no head movement. Here head movement is only one track. Contiguous allocation of a file is defined by the disk address and the length of the on first block.
- If the file size is "n" blocks and starting location is "L", then it occupies blocks L, L+1, L+2, L+3, ..., L+(n-1). The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.



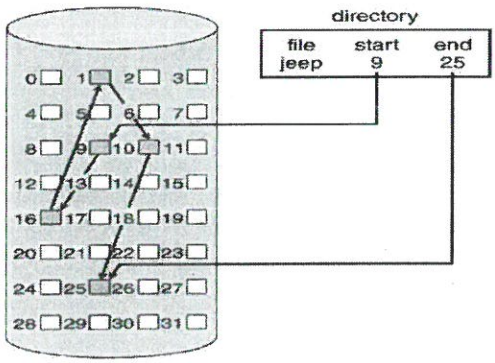
- Sequential and random access can be supported by contiguous allocation. It is easy to retrieve single block. To improve the I/O performance of sequential processing, multiple blocks can be brought in one at a time.
- Contiguous allocation also suffers from external fragmentation. Small free disk spaces are created after allocation free block and deleting files. External fragmentation means there will require free space available but that is not contiguous. To solve the problem of external fragmentation, compaction method is used.

2) Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- Each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

For example, a file of five blocks might start at block 9 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. Thus, if each block is 512 bytes in size, and a disk address requires 4 bytes, then the user sees blocks of 508 bytes.



- To create a new file, we simply create a new entry in the directory. With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to null to signify an empty file. The size field is also set to 0.
- A write to the file causes the free-space management system to find a free block, and this new block is written to and is linked 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.
- Linked allocation does have disadvantages
  - It can be used efficiently only for sequential-access files.
  - The space required for the pointers. Each file requires slightly more space than it would be.
  - Reliability: The files are linked together by pointers scattered all over the disk, and consider what would happen if a pointer were lost or damaged.
- An important variation on linked allocation is the use of a **file-allocation table (FAT)**. The table has one entry for each disk block and is indexed by block number. The FAT is used in much the same way as linked list.

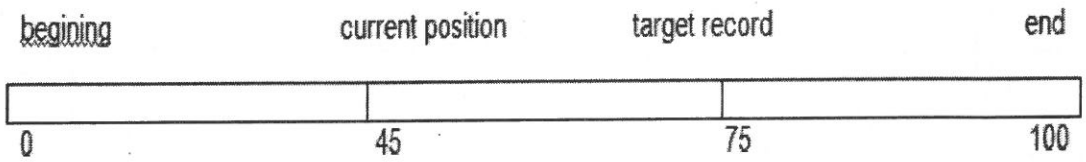
10(b) Compare and contrast sequential and linked access methods **(5M)**

ANS: Information and data are kept in files. Files are stored on the secondary storage device. When this information is to be used, it has to be accessed and brought into primary main memory. Information in files could be accessed in many ways. It is usually dependent on an application. File organization checks how efficiently the input-output storage medium used. File access method defines the way processes read and write files. Different access methods are available. Some operating systems provide only one access method and other systems provide many access methods. Sequential access, Direct /Random access, other access methods.

• Access method means accesses to files that use a particular file organization are implemented by an input output control system.

Sequential Access:

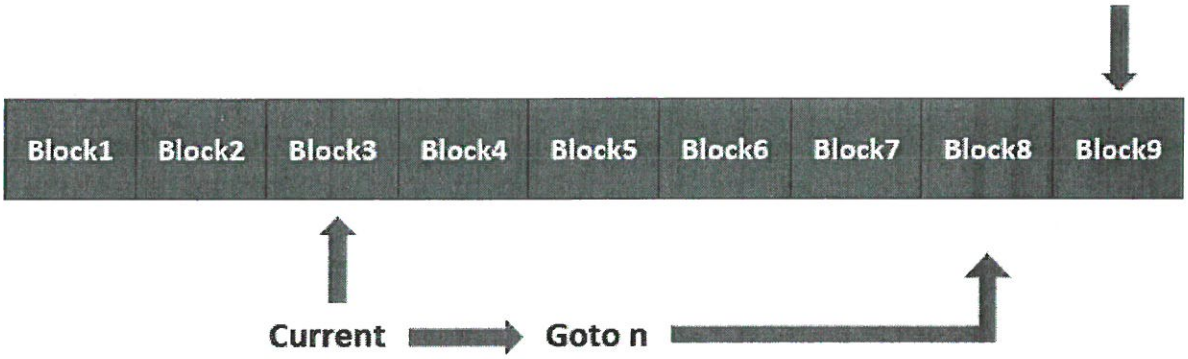
- Information in the file is processed in order, one record after the other. This mode of access is by far the most common for editors and compilers usually access files in this fashion.
- Read and writes make up the bulk of the operations on a file. A read operation read_next() reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation write_next() appends to the end of the file and advances to the end of the newly written material.
- Such a file can be reset to the beginning, and on some systems, a program may be able to skip forward or backward n records for some integer n perhaps only for n=1.
- Example: 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



2. **Direct Access (Random Access):**

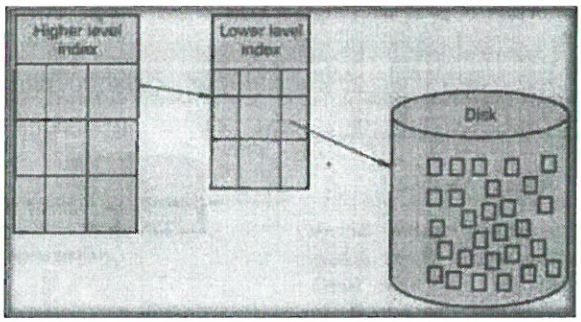
- Data can be accessed directly at any position within the file using physical addresses or calculated positions without reading preceding records. Another method is direct access (or relative access).
- Here, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order. The direct-access method is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records. wemayreadblock14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file. Direct-access files are of great use for

immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information



3. **Other Access methods:**

- Combines sequential and direct access by maintaining a separate index structure that maps logical keys or record identifiers to their physical addresses.
- Provides fast access for both sequential and random retrieval by first accessing the index and then the data.
- Suitable for large files requiring frequent searches and updates, like file systems or complex databases.



- Requires additional storage for the index and extra overhead to maintain the index during insertions, deletions, or modifications.

Feature	Sequential Access	Direct Access	Other Access
Access Order	Linear, one record after another	Random, any record via physical position	Uses index for fast lookup, supports both
Implementation	Simple	More complex	Most complex, requires index maintenance
Efficiency	Efficient for full file scans	Efficient for specific record retrieval	Efficient for both random and sequential
Use Case	Batch processing, logs	Databases, real-time systems	Large databases, file systems
Storage Overhead	Minimal	Moderate	Additional storage for indexes
Update Handling	Costly, requires shifting data	Flexible, easy updates	Extra overhead due to index updating

NOTE: STUDENTS CAN WRITE ANY TWO FROM THESE IN TABULAR OR THEORY FORM.

OR

11(a) Explain about bit vector and linked list in free space management. **(5M)**

ANS: The process of looking after and managing the free blocks of the disk is called free space management. The need of free space management is as limited memory space on the disk and necessary to reuse the space released from deleted files for the allocation of the new file. The free space list can be implemented mainly as:

- Bitmap
- Linked list

1. Bitmap or Bit vector:

The free-space list is implemented as a bit map or bit vector. A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values, 0 or1.

- 0 indicates that the block is allocated and
- 1 indicates a free block

Consider a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18,25,26, and 27 are free and the rest of the blocks are allocated. The free-space bit map would be

00111100111110001100000011100000000..

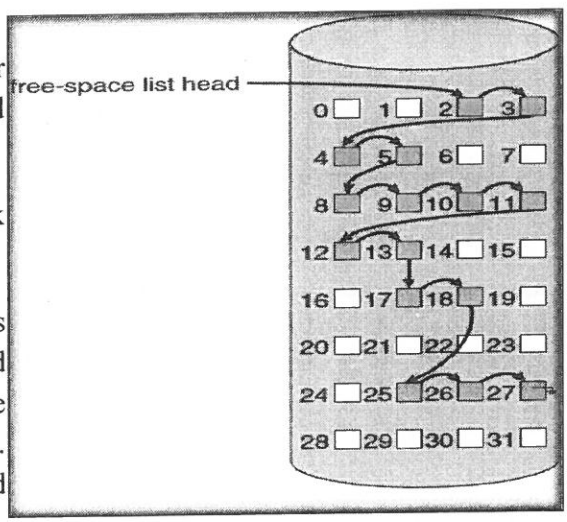
• The calculation of the block number is

(number of bits per word) x (number of 0-valued words) + offset of first 1 bit.

•Bit map requires extra space

Example: block size = 4KB = 212 bytes disk size = 240 bytes (1 terabyte) n = 240/212 = 228 bits (or 32MB) if clusters of 4 blocks -> 8MB of memory

2. Linked List



• Links together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.

• This first block contains a pointer to the next free disk block, and so on.

• For example, consider a disk where blocks 2,3,4,5,8,9,10,11,12,13,17,18,25,26, and 27 are free and the rest of the blocks are allocated. In this situation, we would keep a pointer to block 2 as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5 and so on.

• This scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time. Whenever a file is to be allocated a free block, the operating system can simply allocate the first block in free space list and move the head pointer to the next free block in the list is advantage. Limitations are searching the free space list will be very time consuming; each block will have to be read from the disk, which is read very slowly as compared to the main memory and not Efficient for faster access.

11(b) Explain the principles of protection with examples. **(5M)**

ANS: Protection and security are implemented to prevent interface with the use of files. Both logical and physical. A guiding principle simplifies design decisions and keeps the system consistent and easy to understand. In operating system design, such principles ensure **uniformity, clarity, and security**. When designing secure operating systems, guiding principles help simplify decisions and maintain consistency.

- One of the most important and time-tested principles is the principle of least privilege, which dictates that programs, users, and systems should be granted only the minimum privileges necessary to perform their tasks. For example, in UNIX it is a long-standing tenet that users should not run as root, because:
- Root has unrestricted power, so even a small human error (like deleting a critical file) can have grave consequences.
- Malicious attacks such as viruses triggered by accidental clicks or buffer overflow/code-injection attacks against root-privileged processes can cause catastrophic damage.
- By enforcing least privilege, malicious code is less likely to obtain root privileges, and permissions can act like an immune system to block or limit harmful operations.
- A related principle is compartmentalization, which protects each system component with specific permissions and access restrictions. This ensures that:
- If one component is compromised, another line of defense prevents the attacker from spreading further.
- Compartmentalization can be implemented through mechanisms like network demilitarized zones (DMZs) or virtualization.
- It also supports the creation of audit trails, which record divergences from allowed access. These logs can:
    - Provide early warnings of attacks.
    - Reveal which attack vectors were used.
    - Help assess the extent of damage.
- Finally, no single principle is sufficient on its own.