Paper Name with code: Operating Systems (PCCCS503)

Institute of Engineering & Management, Kolkata
University of Engineering & Management, Kolkata
University of Engineering & Management, Jaipur

Paper name: Operating System
Code: PCCCS503
Semester: 5
Contacts: 3L
Credits: 3

Detailed Syllabus

Pre-requisite: Basic knowledge of Data Structures and Computer Organization.

Module 1: Introduction (10L)

Generations & Concept of Operating Systems, Types of Operating Systems, OS Services, System Calls, Structure of an OS - Layered, Monolithic, Microkernel Operating Systems, Concept of Virtual Machine. Case study on UNIX and WINDOWS Operating System.

Processes: Definition, Process Relationship, Different states of a Process, Process State Transitions, Process Control Block (PCB), Context switching.

Thread: Definition, Various states, Benefits of threads, Types of threads, Concept of multithreads.

Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time; Scheduling algorithms: Pre-emptive and Non pre-emptive, FCFS, SJF, RR, Priority. Multiprocessor scheduling.

Module 2: Inter-Process Communication (10L)

Critical Section, Race Conditions, Mutual Exclusion, Hardware Solution, Strict Alternation, Peterson's Solution, The Producer Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Producer Consumer Problem, Dinning Philosopher Problem.

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

Module 3: Memory Management (10L)

Basic concept, Logical and Physical address map, Memory allocation:

Contiguous Memory allocation– Fixed and variable partition– Internal and External fragmentation and Compaction; Paging: Principle of operation –Page allocation Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory –Locality of reference, Page fault, Working Set, Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not Recently used (NRU) and Least Recently used (LRU).

Module 4: I/O Hardware, File and Disk Management (10L)

I/O Hardware: I/O devices, Device controllers, Direct memory access Principles of I/O Software: Goals of Interrupt handlers, Device drivers, Device independent I/O software

File Management: Concept of File, Access methods, File types, File operation, Directory structure, File System structure, Allocation methods (contiguous, linked, indexed), Free space management (bit vector, linked list, grouping), directory implementation (linear list, hash table), efficiency and performance.

Disk Management: Disk structure, Disk scheduling: FCFS, SSTF, SCAN, C SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks

COURSE OUTCOMES:

CO 1: Students will be able to understand the different services provided by Operating System and different scheduling algorithms at different level.

CO 2: Students will be able to learn synchronization techniques to avoid deadlock.

CO 3: Students will acquire a knowledge about different memory management techniques like paging, segmentation and demand paging etc.

CO 4: students will have a comprehensive understanding of I/O hardware and software principles, secondary-storage structures, file management, and disk management.

TEXT BOOK:

1. Operating System Concepts Essentials, 9th Edition by Abraham Silberschatz, Peter Galvin, Greg Gagne, Wiley Asia Student Edition.

2. Operating Systems: Internals and Design Principles, 5th Edition, William Stallings, Prentice Hall of India.

REFERENCE BOOKS:

1. Operating System Concepts, Ekta Walia, Khanna Publishing House (AICTE Recommended Textbook – 2018).
2. Operating System: A Design-oriented Approach, 1st Edition by Charles Crowley, Irwin Publishing.

ONLINE RESOURCES:

1. https://online.stanford.edu/courses/cs111-operating-systems-principles)
2. https://onlinecourses.nptel.ac.in/noc20_cs04/preview
3. https://www.coursera.org/specializations/codio-introduction-operating-systems
4. https://www.coursera.org/learn/akamai-operating-systems#modules

MODULE 1

Definition of Operating System: An operating system is system software that manages computer hardware and software resources and provides common services for computer programs.

Types of Operating Systems:

1. Batch Operating System –

This type of operating system do not interact with the computer directly. There is an operator which takes similar jobs having same requirement and group them into batches. It is the responsibility of operator to sort the jobs with similar needs.

2. Time-Sharing Operating Systems –

Each task has given some time to execute, so that all the tasks work smoothly. Each user gets time of CPU as they use single system. These systems are also known as Multitasking Systems. The task can be from single user or from different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to next task.

3. Distributed Operating System –

These types of operating system is a recent advancement in the world of computer technology and are being widely accepted all-over the world and, that too, with a great pace. Various autonomous interconnected computers communicate each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred as loosely coupled systems or distributed systems. These systems processors differ in sizes and functions. The major benefit of working with these types of operating system is that it is always possible that one user can access the files or software which are not actually present on his system but on some other system connected within this network i.e., remote access is enabled within the devices connected in that network.

4. Network Operating System –

These systems runs on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions. These type of operating systems allows shared access of files, printers, security, applications, and other networking functions over a

small private network. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc. and that's why these computers are popularly known as tightly coupled systems.

5. Real-Time Operating System –

These types of OSs serves the real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called response time.

Real-time systems are used when there are time requirements are very strict like missile systems, air traffic control systems, robots etc.

Two types of Real-Time Operating System which are as follows:

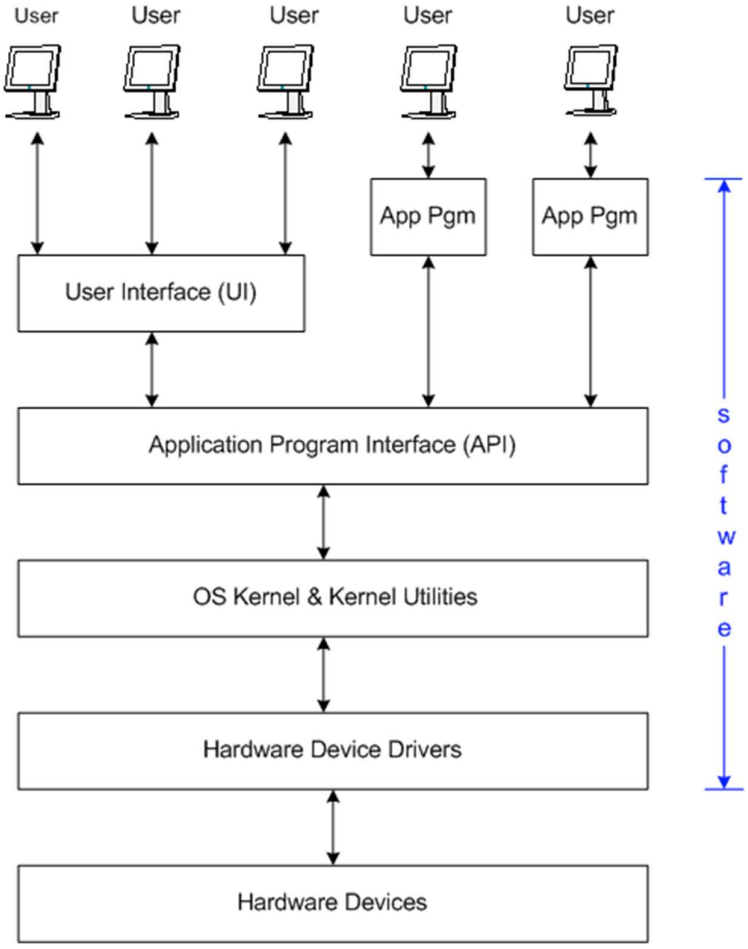
- **Hard Real-Time Systems:**

  These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident. Virtual memory is almost never found in these systems.

- **Soft Real-Time Systems:**

- **These OSs are for applications where for time-constraint is less strict.**

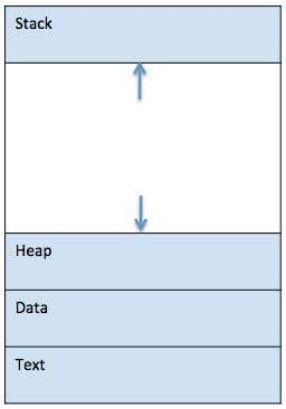
Operating Systems Components:


User
User
User
User
User
App Pgm
App Pgm
User Interface (UI)
Application Program Interface (API)
OS Kernel & Kernel Utilities
Hardware Device Drivers
Hardware Devices
software

Definition of Process:

- In computing, a process is the instance of a computer program that is being executed.
- A process is defined as an entity which represents the basic unit of work to be implemented in the system.

Component of a Process



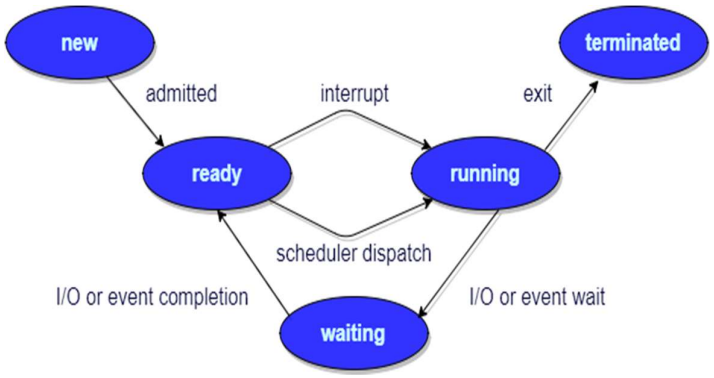
Stack
The process Stack contains the temporary data such as method/function parameters, return address and local variables.

Heap
This is dynamically allocated memory to a process during its run time.

Text
This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

Data
This section contains the global and static variables.

Process Life Cycle



Start
This is the initial state when a process is first started/created.

Ready
The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process.

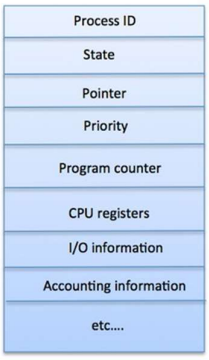
Running
Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Waiting
Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

Terminated or Exit
Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –



Process Scheduling:

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.

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 Queue- The processes which are blocked due to unavailability of an I/O device constitute this queue.

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- 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.

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.

Short-Term Scheduler- It is also called as CPU scheduler. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Context Switching

A context switch is the mechanism to store and restore the state or context of a process in Process Control block so that a process execution can be resumed from the same point at a later time.

First Come First Serve Scheduling

In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16

Waiting time for each process:

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Turn around time for each process:

Process	Turn around Time
P0	5 – 0=5
P1	8 – 1=7
P2	16 – 2=14
P3	22 – 3=19

Round Robin Scheduling

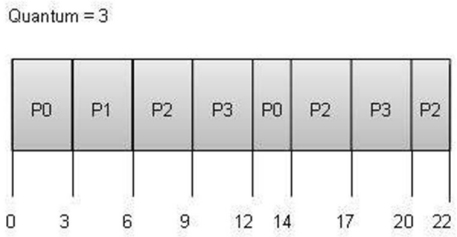
Round Robin is the preemptive process scheduling algorithm.

Each process is provided a fix time to execute, it is called a quantum.

Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

Context switching is used to save states of preempted processes.

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Shortest Job First Scheduling: Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJF is a non- preemptive algorithm.

Example

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

Gantt Chart

P1	P2	P4	P3	
0	**8**	**12**	**17**	**26**

PID	Waiting Time	Turnaround Time
P1	0 - 0 =0	8 – 0 =8
P2	8 - 1 =7	12 – 1=11
P3	17 - 2=15	26- 2 = 24
P4	12 - 3=9	17 – 3 =14

Avg. Waiting Time= (0+7+15+9)/4= 7.75ms

Avg. Turnaround Time= (8+11+24+14)/4 = 14.25ms

Throughput= 4jobs/26ms= 0.15385jobs/ms

Shortest Remaining Time First Scheduling: This Algorithm is the preemptive version of SJF scheduling. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Example

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

Gantt Chart

P1	P2	P4	P1	P3	
0	1	5	10	17	26

PID	Waiting Time	Turnaround Time
P1	0 + (10 – 1) =9	17 – 0 = 17
P2	1 - 1 = 0	5 – 1= 4
P3	17 - 2=15	26- 2 = 24
P4	5 – 3 = 2	10 – 3 = 7

Avg. Waiting Time= (9+0+15+2)/4= 6.5ms

Avg. Turnaround Time= (17+4+24+7)/4 = 13ms

Longest Remaining Time First Scheduling: This is a pre-emptive version of Longest Job First (LJF) scheduling algorithm. In this scheduling algorithm, we find the process with maximum remaining time and then process it. We check for the maximum remaining time after some interval of time to check if another process having more Burst Time arrived up to that time.

Priority Scheduling: Each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Priority Scheduling can be preemptive and non- preemptive.

Non- preemptive Priority Scheduling:

Example

PID	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

Gantt Chart

P1	P2	P4	P3	P5	
0	11	39	49	51	67

PID	Waiting Time	Turnaround Time
P1	0 – 0 = 0	11 – 0 = 11
P2	11 - 5 = 6	39 – 5= 34
P3	49 – 12 = 37	51- 12 = 39
P4	39 – 2 = 37	49 – 2 = 47
P5	51 – 9 = 42	67 – 9 = 58

Avg. Waiting Time= (0+6+37+37+42)/5= 24.4ms

Avg. Turnaround Time= (11+34+39+47+58)/5 = 37.8ms

Preemptive Priority Scheduling:

Example

PID	Arrival Time	Burst Time	Priority
P1	0	11	2

P2	**5**	**28**	**0**
P3	**12**	**2**	**3**
P4	**2**	**10**	**1**
P5	**9**	**16**	**4**

Gantt Chart

P1	P4	P2	P4	P1	P3	P5	
0	**2**	**5**	**33**	**40**	**49**	**51**	**67**

Thread: A thread is a path of execution within a process. A process can contain multiple threads.

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Process vs Thread?

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

Advantages of Thread over Process

1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

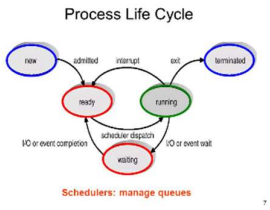
3. Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. Resource sharing: Resources like code, data, and files can be shared among all threads within a process.
Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.

5. Communication: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

6. Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

Types of Threads: There are two types of threads.
User Level Thread
Kernel Level Thread

Q.No.	Question	Blooms Level	CO	Marks
1	There are three jobs running in a multi-programming environment with the following requirements: Job1: Requires disk after every 2 min (device service time including wait and access= 2 min). Total processing time= 6 min. Job2: Requires printer after every 5 min (device service time including wait and access= 2 min). Total processing time= 7 min. Job3: Requires disk after every 3 min (device service time including wait and access= 2min). Total processing time= 5 min. Prepare a timing chart showing the CPU and I/O activities of the jobs. Compute the total time for execution using mono-programming and multi-programming and then compare the results.	4	1	10
2	What event handler would be executed in the following cases: a. The running process has finished its execution before completion of its time slice. b. The running process tries to access memory location that is not allowed to access. c. If there is failure in reading or writing an I/O device. d. A process is ready to execute but there is no space in the main memory. e. A periodic process is idle waiting for its next time slot to be executed.	4	1	10
3	Processes go through the following states in their lifetime.  Consider the following events and answer the questions that follow. Assume there are 5 processes, all either in the read or running states initially. Assume the processes are using a single processor. • At time 5: P1 executes a command to read from disk 3. • At time 15: P3's time slice ends. • At time 18: P4 executes a command to write to disk 3. • At time 20: P2 executes a command to read from disk 2. • At time 24: P3 executes a command to join with P5. • At time 33: An interrupt occurs indicating that P2's read is complete. • At time 36: An interrupt occurs indicating that P1's read is complete. • At time 38: P5 terminates. • At time 48: An interrupt occurs indicating that P4's write is complete. **For time 22, 37 and 47, identify which state each process is in. If it is waiting, indicate what it is waiting for.**	6	1	10
4	Consider the 3 processes, P1, P2 and P3 shown in the table. **Process Arrival time Time Units Required** P1 0 5	4	1	10

	P2 1 7 P3 3 4 What will be the completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2 time units)?			
5	Three processes A, B and C each execute a loop of 100 iterations. In each iteration of the loop, a process performs a single computation that requires tc CPU milliseconds and then initiates a single I/O operation that lasts for tio milliseconds. It is assumed that the computer where the processes execute has sufficient number of I/O devices and the OS of the computer assigns different I/O devices to each process. Also, the scheduling overhead of the OS is negligible. The processes have the following characteristics: **Process id tc tio** A 100ms 500 ms B 350 ms 500 ms C 200 ms 500 ms The processes A, B, and C are started at times 0, 5 and 10 milliseconds respectively, in a pure time sharing system (round robin scheduling) that uses a time slice of 50 milliseconds. What is the time in milliseconds at which process C would complete its first I/O operation?	6	1	10
6	Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.	5	1	10
7	Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?	6	1	10
8	Consider the following table of arrival time and burst time for three processes P0, P1 and P2. Process Arrival time Burst Time P0 0 ms 9 ms P1 1 ms 4 ms P2 2 ms 9 ms The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?	4	1	10
9	Assume every process requires 3 seconds of service time in a system with single processor. If new processes are arriving at the rate of 10 processes per minute, then estimate the fraction of time CPU is busy in system?	4	1	10
10	Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single- threaded solution on a single-processor system? Describe the actions taken by a thread library to context switch between user- level threads.	5	1	10

MODULE 2

On the basis of synchronization, processes are categorized as one of the following two types:

- **Independent Process** : Execution of one process does not affects the execution of other processes.
- **Cooperative Process** : Execution of one process affects the execution of other processes.

Process synchronization problem arises in the case of Cooperative process also because resources are shared in Cooperative processes.

Critical Section Problem

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

Any solution to the critical section problem must satisfy three requirements:

- Mutual Exclusion : If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- Progress : If no process is in the critical section, then no other process from outside can block it from entering the critical section.
- Bounded Waiting : A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

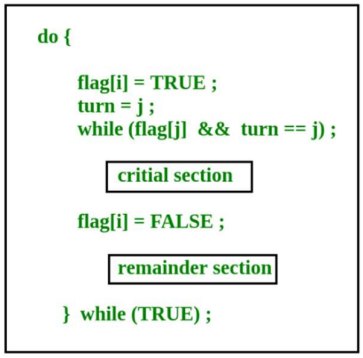
Peterson's Solution

Peterson's Solution is a classical software based solution to the critical section problem.

In Peterson's solution, we have two shared variables:

boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section

int turn : The process whose turn is to enter the critical section.



Semaphores

A Semaphore is an integer variable, which can be accessed only through two operations wait () and signal ().
There are two types of semaphores: Binary Semaphores and Counting Semaphores

Binary Semaphores: They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.

Counting Semaphores: They can have any value and are not restricted over a certain domain. They can be used to control access a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining

instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource

Reader Writer problem

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

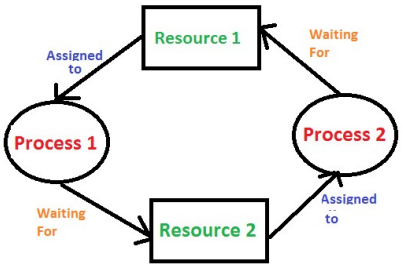
- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her.
- However if some person is reading the file, then others may read it at the same time.

Precisely in OS we call this situation as the readers-writers problem
Problem parameters:

- One set of data is shared among a number of processes
- Once a writer is ready, it performs its write. Only one writer may write at a time
- If a process is writing, no other process can read it
- If at least one reader is reading, no other process can write
- Readers may not write and only read

Deadlock

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)
Mutual Exclusion: One or more than one resource are non-sharable (Only one process

can use at a time)
Hold and Wait: A process is holding at least one resource and waiting for resources.
No Preemption: A resource cannot be taken from a process unless the process releases the resource.
Circular Wait: A set of processes are waiting for each other in circular form.

Methods for handling deadlock

There are three ways to handle deadlock
1) Deadlock prevention or avoidance: The idea is to not let the system into deadlock state.
2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

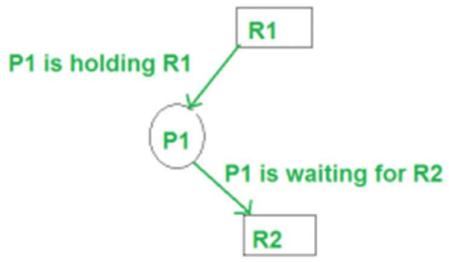
DEADLOCK PREVENTION:

Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tap drive and printer, are inherently non-shareable.

Eliminate Hold and wait

1. Allocate all required resources to the process before start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization. for example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remained blocked till it has completed its execution.
2. Process will make new request for resources after releasing the current set of resources. This solution may lead to starvation.



HOLD AND WAIT

Eliminate No Preemption
Preempt resources from process when resources required by other high priority process.

Eliminate Circular Wait
Each resource will be assigned with a numerical number. A process can request for the resources only in increasing order of numbering.
For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Deadlock Avoidance
Deadlock avoidance can be done with Banker's Algorithm.

Banker's Algorithm
Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it check for safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it don't allow the request made by the process.
Inputs to Banker's Algorithm
1. Max need of resources by each process.
2. Currently allocated resources by each process.
3. Max free available resources in the system.
Request will only be granted under below condition.
1. If request made by process is less than equal to max need to that process.
2. If request made by process is less than equal to freely available resource in the system.

<table>
<tr><th>Q.No.</th><th>Question</th><th>Blooms Level</th><th>CO</th><th>Marks</th></tr>
<tr><td>1</td><td>
<pre>
P1()
 {
  C=B-1;
  B=2*C;
 }
</pre>
<pre>
P2()
{
  D=2*B;
  B-D-1;
}
</pre>
Here, B is a shared variable with initial value 2. How many values B can have? And what are the values?
</td><td>4</td><td>2</td><td>10</td></tr>
<tr><td>2</td><td>A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e. wait) on a counting semaphore S and invokes the V operation (i.e. signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution?</td><td>5</td><td>2</td><td>10</td></tr>
<tr><td>3</td><td>Does presence of cycle in a resource allocation graph necessarily creates deadlock. Explain.</td><td>6</td><td>2</td><td>10</td></tr>
<tr><td>4</td><td>Prove that deadlock prevention mechanism actually to prevent deadlock.</td><td>5</td><td>2</td><td>10</td></tr>
<tr><td>5</td><td>Consider the following snapshot of a system. There are no outstanding unsatisfied requests for resources. Check whether the system is in deadlock.
<table>
<tr><td></td><td colspan="4">Current allocation</td><td colspan="4">Maximum demand</td><td colspan="4">Available</td></tr>
<tr><td>process</td><td>R1</td><td>R2</td><td>R3</td><td>R4</td><td>R1</td><td>R2</td><td>R3</td><td>R4</td><td>R1</td><td>R2</td><td>R3</td><td>R4</td></tr>
<tr><td>P1</td><td>0</td><td>0</td><td>1</td><td>2</td><td>0</td><td>0</td><td>1</td><td>2</td><td>1</td><td>5</td><td>2</td><td>0</td></tr>
<tr><td>P2</td><td>2</td><td>0</td><td>0</td><td>0</td><td>2</td><td>7</td><td>5</td><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>P3</td><td>0</td><td>0</td><td>3</td><td>4</td><td>6</td><td>6</td><td>5</td><td>6</td><td></td><td></td><td></td><td></td></tr>
<tr><td>P4</td><td>2</td><td>3</td><td>5</td><td>4</td><td>4</td><td>3</td><td>5</td><td>6</td><td></td><td></td><td></td><td></td></tr>
<tr><td>P5</td><td>0</td><td>3</td><td>3</td><td>2</td><td>0</td><td>6</td><td>5</td><td>2</td><td></td><td></td><td></td><td></td></tr>
</table>
</td><td>4</td><td>2</td><td>10</td></tr>
<tr><td>6</td><td>Consider method used by process P1 and P2 for accessing critical section. The initial values of shared Boolean variables S1 and S2 are randomly selected.
<pre>
P1()   P2()
{   {
 While (S1==S2);   While(S1!=S2);
 critical section   critical section
 S1=S2;   S1=not (S2);
</pre>
Which of the following is true? Give explanation.
a. Mutual exclusion + Progress
b. Mutual exclusion only
c. Progress only
d. None
</td><td>5</td><td>2</td><td>10</td></tr>
</table>

7	Consider method used by process P1 and P2 for accessing critical section.
	P1()
	{
	While (T)
	var P=T ;
	while(var Q==T);
	critical section
	var P=F;
	}
	P2()
	{
	While(T)
	var Q=T
	while(var P==T);
	critical section
	var Q=F;
	}
	Which of the following is true? Give explanation.
	a. No Mutual exclusion + No Deadlock
	b. Mutual exclusion only
	c. Deadlock only
	d. Mutual exclusion + Deadlock
		5	2	10

8	A counting semaphore S is initialized to 10. Then, 6 P(Wait) operations and 4 V(Signal) operations are performed on S. What is the final value of S? Show the working.	5	2	5
9	"If there is a cycle in the resource allocation graph, it may or may not be in deadlock state". Comment on this statement with a suitable example.	5	2	10

MODULE 3

Memory Management

In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is Required?

Allocate and de-allocate memory before and after process execution.

To keep track of used memory space by processes.

To minimize fragmentation issues.

To proper utilization of main memory.

To maintain data integrity while executing of process.

Logical and Physical Address Space

Logical Address Space: An address generated by the CPU is known as a "Logical Address". It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.

Physical Address Space: An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a "Physical Address". A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

Static and Dynamic Loading

Loading a process into the main memory is done by a loader. There are two different types of loading :

Static Loading: Static Loading is basically loading the entire program into a fixed address. It requires more memory space.

Dynamic Loading: The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that the unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

Static and Dynamic Linking

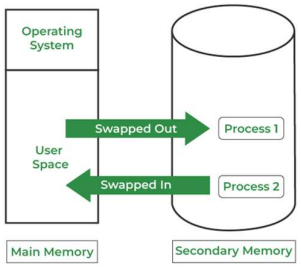
To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

Static Linking: In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.

Dynamic Linking: The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, "Stub" is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

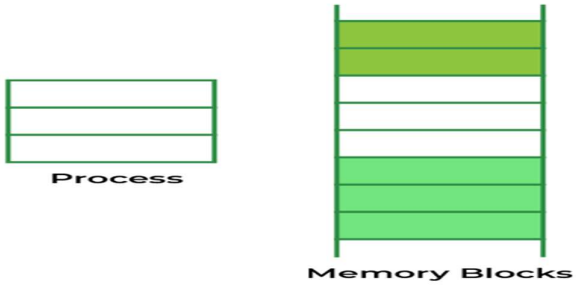
Swapping

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.



Contiguous Memory Allocation

The main memory should accommodate both the operating system and the different client processes. Therefore, the allocation of memory becomes an important task in the operating system. The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In adjacent memory allotment, each process is contained in a single contiguous segment of memory.



Memory Allocation

To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.

Multiple partition allocation: In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.

Fixed partition allocation: In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as a "Hole". When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement is fulfilled then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size n from a list of free holes. There are some solutions to this problem:

Fragmentation

Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process. To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problems. In the operating systems two types of fragmentation:

Internal fragmentation: Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is left over and creating an internal fragmentation problem.Example: Suppose there is a fixed partitioning used for memory allocation and the different sizes of blocks 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demands a block of memory. It gets a memory block of 3MB but 1MB block of memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

External fragmentation: In External Fragmentation, we have a free memory block, but we can not assign it to a process because blocks are not contiguous. Example: Suppose (consider the above example) three processes p1, p2, and p3 come with sizes 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating the process p1 process and the p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not assign it because free memory space is not contiguous. This is called external fragmentation.

Paging

Paging is a memory management scheme that eliminates the need for a contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non-contiguous.

Logical Address or Virtual Address (represented in bits): An address generated by the CPU.

Logical Address Space or Virtual Address Space (represented in words or bytes): The set of all logical addresses generated by a program.

Physical Address (represented in bits): An address actually available on a memory unit.

Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses.

Example:

If Logical Address = 31 bits, then Logical Address Space = 231 words = 2 G words (1 G = 230)

If Logical Address Space = 128 M words = 27 * 220 words, then Logical Address = log2 227 = 27 bits

If Physical Address = 22 bits, then Physical Address Space = 222 words = 4 M words (1 M = 220)

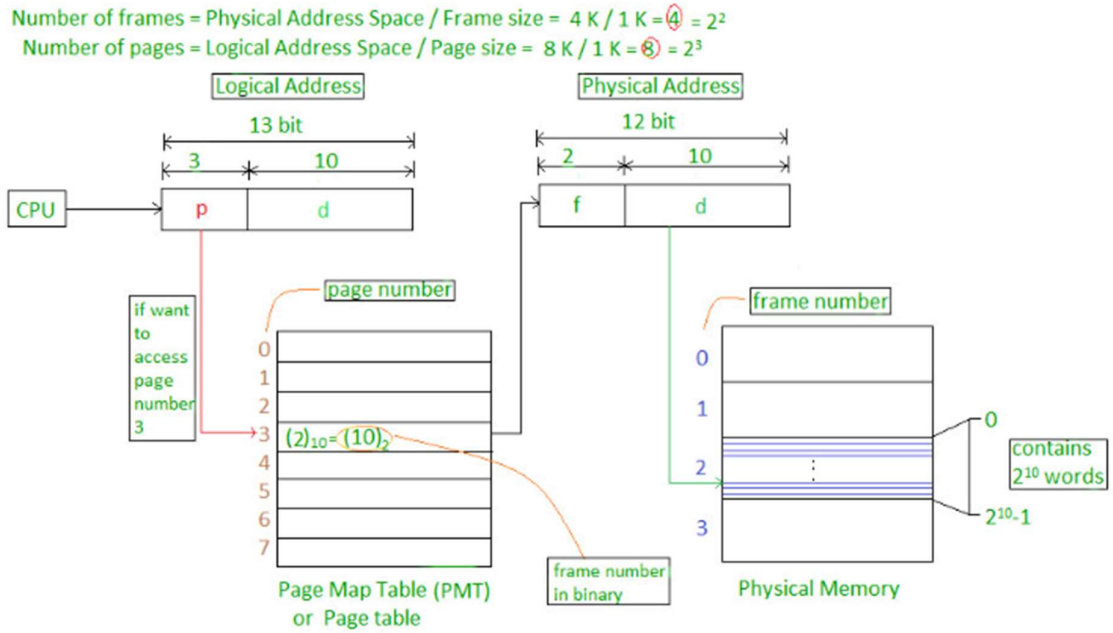
If Physical Address Space = 16 M words = 24 * 220 words, then Physical Address = log2 224 = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as the paging technique.

The Physical Address Space is conceptually divided into several fixed-size blocks, called frames.

The Logical Address Space is also split into fixed-size blocks, called pages.

Page Size = Frame Size



Paging

The address generated by the CPU is divided into:

Page Number(p): Number of bits required to represent the pages in Logical Address Space or Page number

Page Offset(d): Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into:

Frame Number(f): Number of bits required to represent the frame of Physical Address Space or Frame number frame

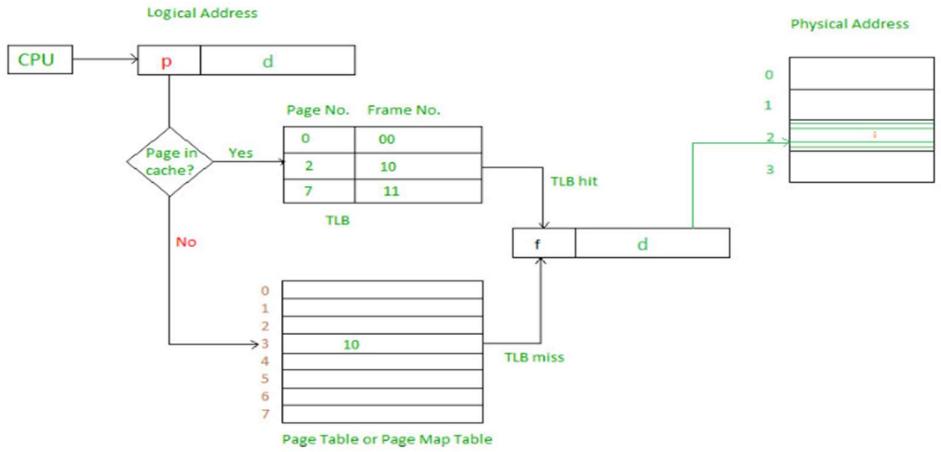
Frame Offset(d): Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

The hardware implementation of the page table can be done by using dedicated registers. But the usage of the register for the page table is satisfactory only if the page table is small. If the page table contains a large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look-up hardware cache.

The TLB is an associative, high-speed memory.

Each entry in TLB consists of two parts: a tag and a value.

When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.



Page Map Table

Q.No.	Question	Blooms Level	CO	Marks
1	"Consider the requests from processes in given order 300K, 25K, 125K, and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K. Which partition allocation schemes can satisfy the above requests? (NOTE : Here we assumed a partition can be allocated to a process even if some other process occupies a part of that partition.) Show the working neatly with diagram. Justify your answer."	6	3	5
2	Consider an imaginary disk with 51 cylinders. A request comes in to read a block on cylinder 11. While the seek to cylinder 11 is in progress, new requests come in for cylinders 1, 36, 16, 34, 9 and 12 in that order. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves, to satify all the pending requests, for each of the following disk scheduling algorithms: FCFS, SSTF, SCAN & LOOK?	5	3	10
3	i. Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with three frames and calculate number of page faults and page hits by using FIFO (First In First Out) Page replacement algorithm ii. Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with three frames and calculate number of page faults and page hits by using LRU Least Recently Used) Page replacement algorithm	5	3	10
4	Consider the 3 processes, P1, P2 and P3 shown in the table. **Process** **Arrival time** **Time Units Required** P1 0 5 P2 1 7 P3 3 4	5	3	10

	What will be the completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2 time units)?			
5	Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If TLB hit ratio is 80%, the effective memory access time is _____ msec.	4	3	5
6	Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.	4	3	5
7	Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?	5	3	10
8	Consider the following table of arrival time and burst time for three processes P0, P1 and P2. Process Arrival time Burst Time P0 0 ms 9 ms P1 1 ms 4 ms P2 2 ms 9 ms The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?	4	3	10
9	Assume every process requires 3 seconds of service time in a system with single processor. If new processes are arriving at the rate of 10 processes per minute, then estimate the fraction of time CPU is busy in system?	5		5
10	Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames. a. How many bits are there in the logical address? b. How many bits are there in the physical address?	4		10

MODULE 4

Disc Scheduling:It is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

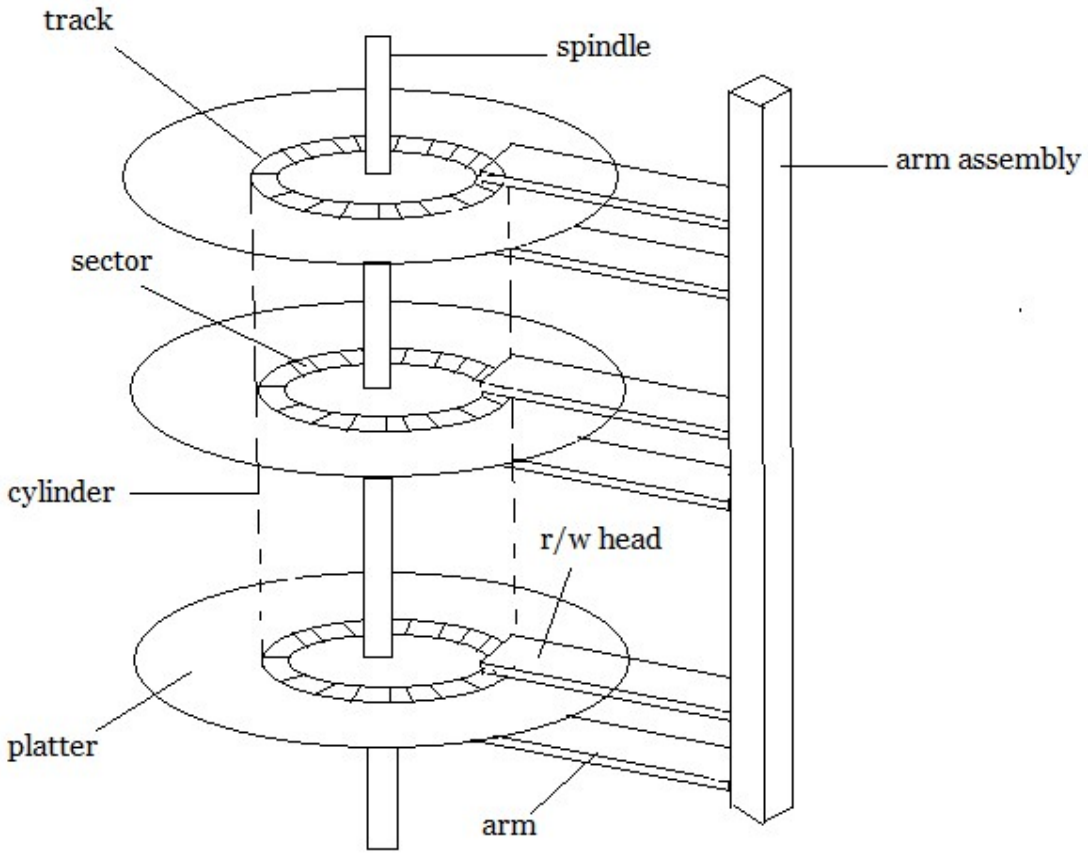
- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.
- Secondary storage is less expensive when compared to primary memory like RAMs.
- The speed of the secondary storage is also lesser than that of primary storage.
- Hence, the data which is less frequently accessed is kept in the secondary storage.
- A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.

Magnetic Disk Structure

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



Structure of a magnetic disk

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

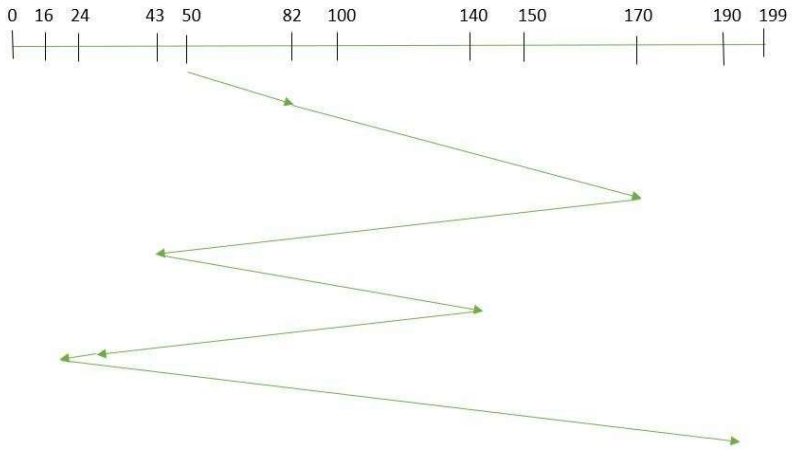
Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

Disk Scheduling Algorithms

There are several Disk Several Algorithms. We will discuss each one of them.

FCFS (First Come First Serve)

FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.



First Come First Serve

Example:

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is: 50

So, total overhead movement (total distance covered by the disk arm) =

(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

Advantages of FCFS

Here are some of the advantages of First Come First Serve.

- Every request gets a fair chance
- No indefinite postponement

Disadvantages of FCFS

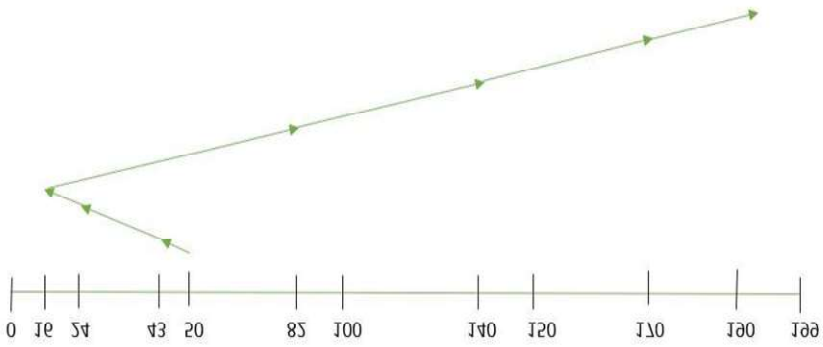
Here are some of the disadvantages of First Come First Serve.

- Does not try to optimize seek time
- May not provide the best possible service

SSTF (Shortest Seek Time First)

In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.

Example:



Shortest Seek Time First

Suppose the order of request is- (82,170,43,140,24,16,190)
And current position of Read/Write head is: 50

So,

total overhead movement (total distance covered by the disk arm) =

(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

Advantages of Shortest Seek Time First

Here are some of the advantages of Shortest Seek Time First.

- The average Response Time decreases
- Throughput increases

Disadvantages of Shortest Seek Time First

Here are some of the disadvantages of Shortest Seek Time First.

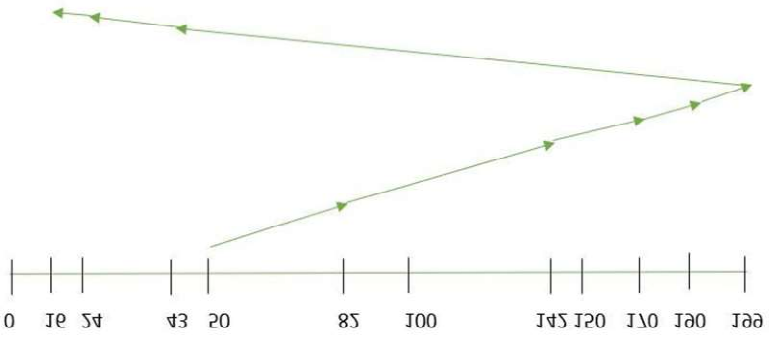
- Overhead to calculate seek time in advance

- Can cause Starvation for a request if it has a higher seek time as compared to incoming requests
- The high variance of response time as SSTF favors only some requests

SCAN

In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:



SCAN Algorithm

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as

= (199-50) + (199-16) = 332

Advantages of SCAN Algorithm

Here are some of the advantages of the SCAN Algorithm.

- High throughput
- Low variance of response time
- Average response time

Disadvantages of SCAN Algorithm

Here are some of the disadvantages of the SCAN Algorithm.

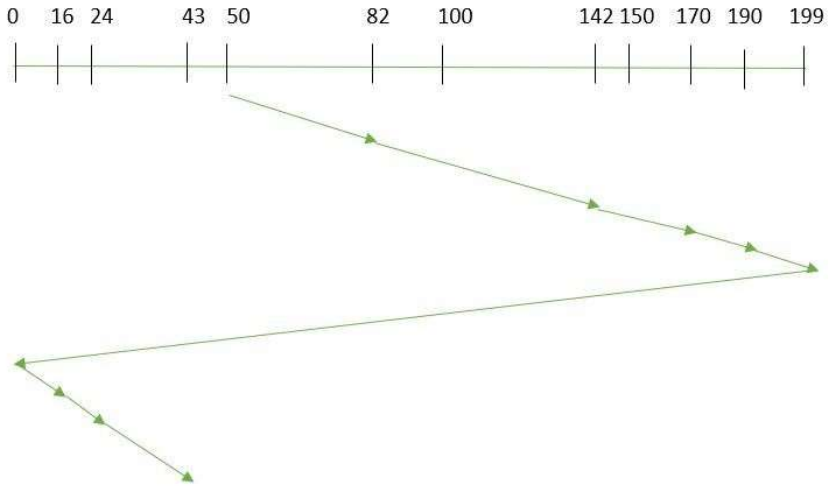
- Long waiting time for requests for locations just visited by disk arm

C-SCAN

In the SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in the *CSCAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).

Example:



Circular SCAN

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

=(199-50) + (199-0) + (43-0) = 391

Advantages of C-SCAN Algorithm

Here are some of the advantages of C-SCAN.

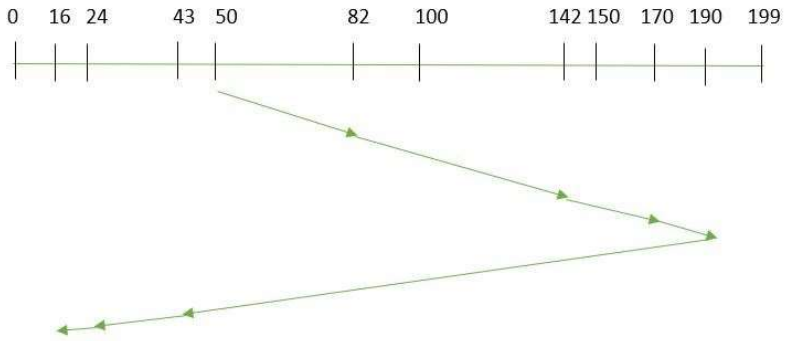
- Provides more uniform wait time compared to SCAN.

LOOK

LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be

serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:



LOOK Algorithm

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

So, the total overhead movement (total distance covered by the disk arm) is calculated as:

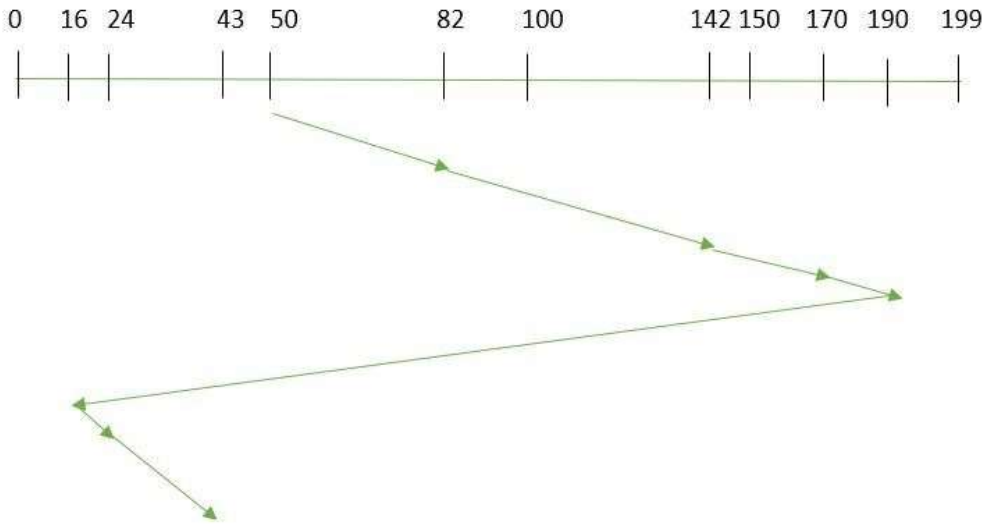
= (190-50) + (190-16) = 314

C-LOOK

As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value"**



C-LOOK

So, the total overhead movement (total distance covered by the disk arm) is calculated as

= (190-50) + (190-16) + (43-16) = 341

A computer file is defined as a medium used for saving and managing data in the computer system. The data stored in the computer system is completely in digital format, although there can be various types of files that help us to store the data.

What is a File System?

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device. Some common types of file systems include:

1. **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.
2. **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.
3. **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.
4. **HFS (Hierarchical File System):** A file system used by macOS.
5. **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices.

A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From the user's perspective, a file is the smallest allotment of logical secondary storage.

The name of the file is divided into two parts as shown below:

- name
- extension, separated by a period.

Issues Handled By File System

We've seen a variety of data structures where the file could be kept. The file system's job is to keep the files organized in the best way possible.
A free space is created on the hard drive whenever a file is deleted from it. To reallocate them to other files, many of these spaces may need to be recovered. Choosing where to store the files on the hard disc is the main issue with files one block may or may not be used to store a file. It may be kept in the disk's non-contiguous blocks. We must keep track of all the blocks where the files are partially located.

Files Attributes And Their Operations

Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append

Attributes	Types	Operations
Last Modified	Java	Truncate
protection	class	Delete
		Close
File type	**Usual extension**	**Function**
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats
Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video

File type	Usual extension	Function
		information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a ,so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing an ASCII or binary file.

File Directories

The collection of files is a file directory. The directory contains information about the files, including attributes, location, and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Below are information contained in a device directory.

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

The operation performed on the directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

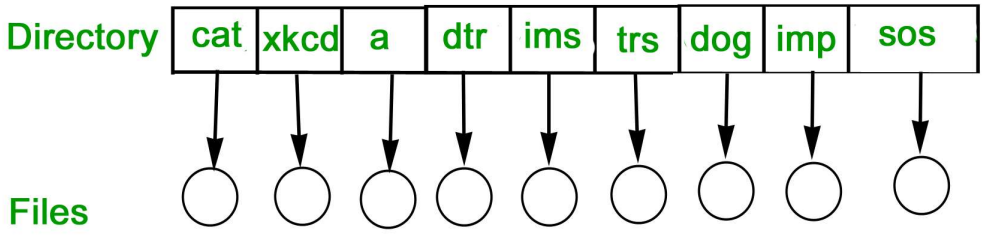
Advantages of Maintaining Directories

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

Single-Level Directory

In this, a single directory is maintained for all the users.

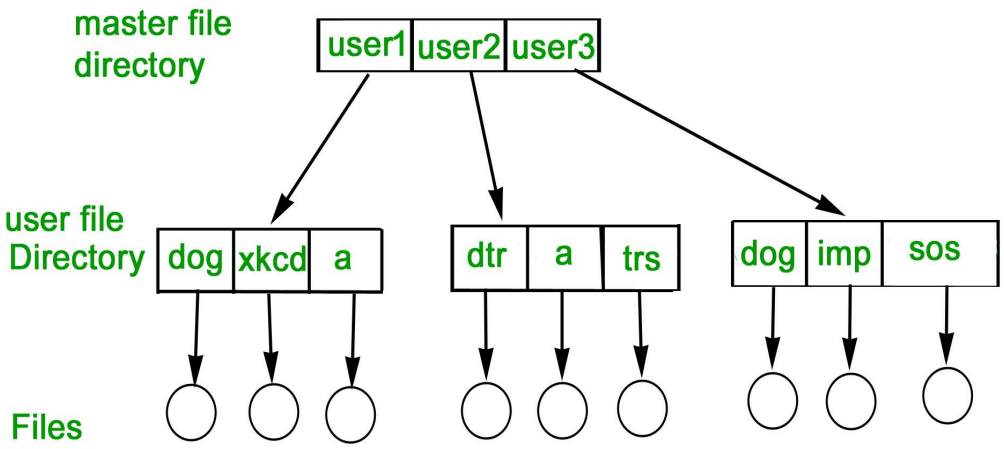
- **Naming problem:** Users cannot have the same name for two files.
- **Grouping problem:** Users cannot group files according to their needs.



Two-Level Directory

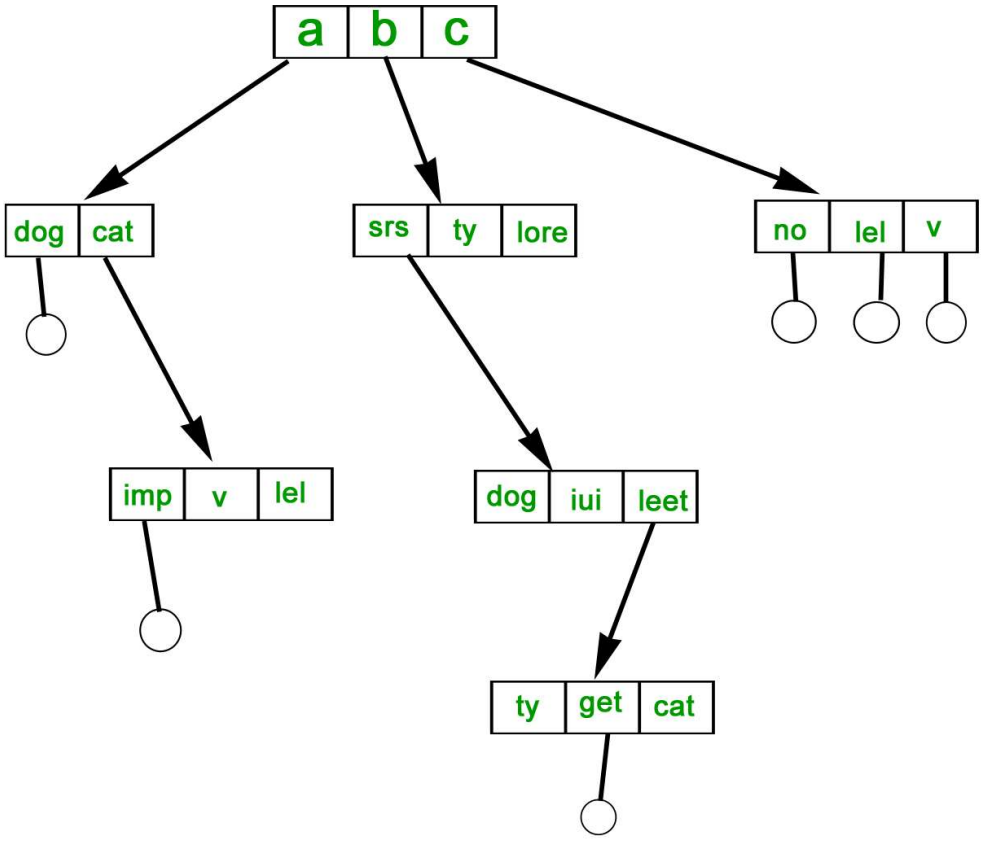
In this separate directories for each user is maintained.

- **Path name**: Due to two levels there is a path name for every file to locate that file.
- Now, we can have the same file name for different users.
- Searching is efficient in this method.



Tree-Structured Directory

The directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.



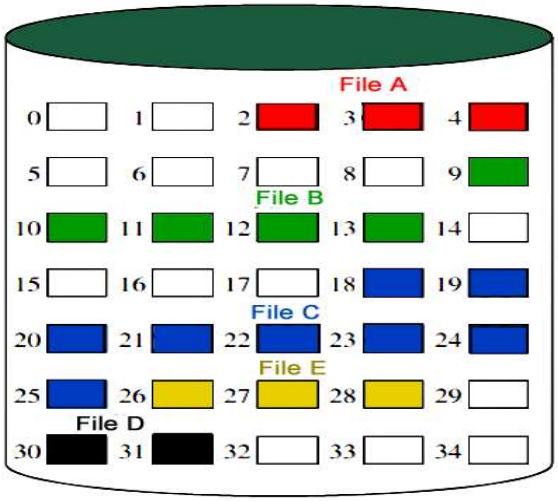
File Allocation Methods

There are several types of file allocation methods. These are mentioned below.

- Continuous Allocation
- Linked Allocation(Non-contiguous allocation)
- Indexed Allocation

Continuous Allocation

A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Disadvantages of Continuous Allocation

- External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. A compaction algorithm will be necessary to free up additional space on the disk.
- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

Linked Allocation(Non-Contiguous Allocation)

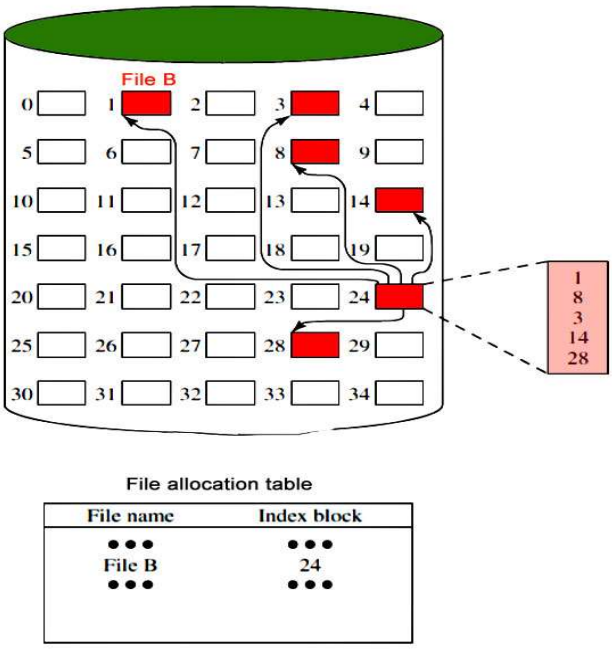
Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. An increase in file size is always possible if a free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of the file.

Disadvantage Linked Allocation(Non-contiguous allocation)

- Internal fragmentation exists in the last disk block of the file.
- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only the sequential access of files.

Indexed Allocation

It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. The allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.



File allocation table

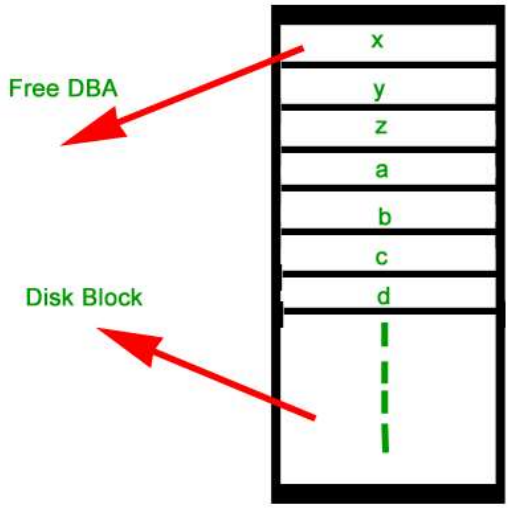
File name	Index block
•••	•••
File B	24
•••	•••

Disk Free Space Management

Just as the space that is allocated to files must be managed, so the space that is not currently allocated to any file must be managed. To perform any of the file allocation techniques, it is necessary to know what blocks on the disk are available. Thus we need a disk allocation

table in addition to a file allocation table. The following are the approaches used for free space management.

1. **Bit Tables**: This method uses a vector containing one bit for each block on the disk. Each entry for a 0 corresponds to a free block and each 1 corresponds to a block in use. For example 00011010111100110001 In this vector every bit corresponds to a particular block and 0 implies that that particular block is free and 1 implies that the block is already occupied. A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. Thus, a bit table works well with any of the file allocation methods. Another advantage is that it is as small as possible.
2. **Free Block List**: In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved block of the disk.



Advantages of File System

- **Organization:** A file system allows files to be organized into directories and subdirectories, making it easier to manage and locate files.
- **Data protection:** File systems often include features such as file and folder permissions, backup and restore, and error detection and correction, to protect data from loss or corruption.
- **Improved performance:** A well-designed file system can improve the performance of reading and writing data by organizing it efficiently on disk.

Disadvantages of File System

- **Compatibility issues:** Different file systems may not be compatible with each other, making it difficult to transfer data between different operating systems.
- **Disk space overhead:** File systems may use some disk space to store metadata and other overhead information, reducing the amount of space available for user data.
- **Vulnerability:** File systems can be vulnerable to data corruption, malware, and other security threats, which can compromise the stability and security of the system.

Q.No.	Question	Blooms Level	CO	Marks
1	Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing which number of requests?	5	4	10
2	Consider an imaginary disk with 51 cylinders. A request comes in to read a block on cylinder 11. While the seek to cylinder 11 is in progress, new requests come in for cylinders 1, 36, 16, 34, 9 and 12 in that order. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves, to satify all the pending requests, for each of the following disk scheduling algorithms: FCFS, SSTF, SCAN & LOOK?	5	4	10
3	Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. What will be the additional distance (in terms of number of tracks) that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution)	5	4	10
4	Evaluate the efficiency and effectiveness of different file organization methods (e.g., sequential, indexed, hashed) in managing large volumes of data.	5	4	10
5	Synthesize a comprehensive file management strategy for a multinational corporation with geographically dispersed offices, considering factors such as security, accessibility, and scalability.	6	4	10
6	Assess the trade-offs between different disk scheduling algorithms (e.g., FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK) in terms of throughput, response time, and fairness, and recommend the most suitable algorithm for specific system configurations.	5	4	10

Exercises

1.1 In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems.

a. What are two such problems?

b. Can we ensure the same degree of security in a time-shared machine as in a dedicated machine? Explain your answer.

1.2 The issue of resource utilization shows up in different forms in different types of operating systems. List what resources must be managed carefully in the following settings:

a. Mainframe or minicomputer systems

b. Workstations connected to servers

c. Handheld computers

1.3 Which of the functionalities listed below need to be supported by the operating system for the following two settings: (a) handheld devices and (b) real-time systems.

a. Batch programming

b. Virtual memory

c. Time sharing

1.4 Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?

1.5 Distinguish between the client–server and peer-to-peer models of distributed systems.

1.6 Consider a computing cluster consisting of two nodes running a database. Describe two ways in which the cluster software can manage access to the data on the disk. Discuss the benefits and disadvantages of each.

1.7 What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

1.8 Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.

a. How does the CPU interface with the device to coordinate the transfer?

b. How does the CPU know when the memory operations are complete?

c. The CPU is allowed to execute other programs while the DMA controller is transferring data. Does this process interfere with

the execution of the user programs? If so, describe what forms of interference are caused.

1.9 Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

1.10 Discuss, with examples, how the problem of maintaining coherence of cached data manifests itself in the following processing environments:

a. Single-processor systems

b. Multiprocessor systems

c. Distributed systems

1.11 What network configuration would best suit the following environments?

a. A dormitory floor

b. A university campus

c. A state

d. A nation

1.12 Define the essential properties of the following types of operating systems:

a. Batch

b. Interactive

c. Time sharing

d. Real time

e. Network

f. Parallel

g. Distributed

h. Clustered

i. Handheld

Bibliographical Notes

Brookshear [2003] provides an overview of computer science in general.

An overview of the Linux operating system is presented in Bovet and Cesati [2002]. Solomon and Russinovich [2000] give an overview of Microsoft Windows and considerable technical detail about the system internals and components. Mauro and McDougall [2001] cover the Solaris operating system. Mac OS X is presented at http://www.apple.com/macosx.

Coverage of peer-to-peer systems includes Parameswaran et al. [2001], Gong [2002], Ripeanu et al. [2002], Agre [2003], Balakrishnan et al. [2003], and

may come from files during batch-mode execution or directly from a terminal when in an interactive or time-shared mode. System programs are provided to satisfy many common user requests.

The types of requests vary according to level. The system-call level must provide the basic functions, such as process control and file and device manipulation. Higher-level requests, satisfied by the command interpreter or system programs, are translated into a sequence of system calls. System services can be classified into several categories: program control, status requests, and I/O requests. Program errors can be considered implicit requests for service.

Once the system services are defined, the structure of the operating system can be developed. Various tables are needed to record the information that defines the state of the computer system and the status of the system's jobs.

The design of a new operating system is a major task. It is important that the goals of the system be well defined before the design begins. The type of system desired is the foundation for choices among various algorithms and strategies that will be needed.

Since an operating system is large, modularity is important. Designing a system as a sequence of layers or using a microkernel is considered a good technique. The virtual-machine concept takes the layered approach and treats both the kernel of the operating system and the hardware as though they were hardware. Even other operating systems may be loaded on top of this virtual machine.

Throughout the entire operating-system design cycle, we must be careful to separate policy decisions from implementation details (mechanisms). This separation allows maximum flexibility if policy decisions are to be changed later.

Operating systems are now almost always written in a systems-implementation language or in a higher-level language. This feature improves their implementation, maintenance, and portability. To create an operating system for a particular machine configuration, we must perform system generation.

For a computer system to begin running, the CPU must initialize and start executing the bootstrap program in firmware. The bootstrap can execute the operating system directly if the operating system is also in the firmware, or it can complete a sequence in which it loads progressively smarter programs from firmware and disk until the operating system itself is loaded into memory and executed.

Exercises

2.1 The services and functions provided by an operating system can be divided into two main categories. Briefly describe the two categories and discuss how they differ.

2.2 List five services provided by an operating system that are designed to make it more convenient for users to use the computer system. In what cases it would be impossible for user-level programs to provide these services? Explain.

2.3 Describe how you could obtain a statistical profile of the amount of time spent by a program executing different sections of its code. Discuss the importance of obtaining such a statistical profile.

2.4 What are the five major activities of an operating system with regard to file management?

2.5 What is the purpose of the command interpreter? Why is it usually separate from the kernel? Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

2.6 What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

2.7 Why does Java provide the ability to call from a Java program native methods that are written in, say, C or C++? Provide an example of a situation in which a native method is useful.

2.8 It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.

2.9 In what ways is the modular kernel approach similar to the layered approach? In what ways does it differ from the layered approach?

2.10 What is the main advantage for an operating-system designer of using a virtual-machine architecture? What is the main advantage for a user?

2.11 What is the relationship between a guest operating system and a host operating system in a system like VMware? What factors need to be considered in choosing the host operating system?

2.12 The experimental Synthesis operating system has an assembler incorporated in the kernel. To optimize system-call performance, the kernel assembles routines within kernel space to minimize the path that the system call must take through the kernel. This approach is the antithesis of the layered approach, in which the path through the kernel is extended to make building the operating system easier. Discuss the pros and cons of the Synthesis approach to kernel design and system-performance optimization.

Project—Adding a System Call to the Linux Kernel

In this project, you will study the system call interface provided by the Linux operating system and how user programs communicate with the operating system kernel via this interface. Your task is to incorporate a new system call into the Linux kernel, thereby expanding the functionality of the operating system.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main()
{
pid_t pid;

  pid = fork();

  if (pid == 0) {/* child process */
    value += 15;
  }
  else if (pid > 0) {/* parent process */
    wait(NULL);
    printf("PARENT: value = %d",value); /* LINE A */
    exit(0);
  }
}
```

Figure 3.23 C program.

Exercises

3.1 Describe the differences among short-term, medium-term, and long-term scheduling.

3.2 Describe the actions taken by a kernel to context-switch between processes.

3.3 Using the program shown in Figure 3.23, explain what will be output at `Line A`.

3.4 What are the benefits and the disadvantages of each of the following? Consider both the system level and the programmer level.

a. Synchronous and asynchronous communication

b. Automatic and explicit buffering

c. Send by copy and send by reference

d. Fixed-sized and variable-sized messages

3.5 The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ... Formally, it can be expressed as:

$$fib_0 = 0$$
$$fib_1 = 1$$
$$fib_n = fib_{n-1} + fib_{n-2}$$

Write a C program using the `fork()` system call that that generates the Fibonacci sequence in the child process. The number of the sequence

Multilevel queue algorithms allow different algorithms to be used for different classes of processes. The most common model includes a foreground interactive queue that uses RR scheduling and a background batch queue that uses FCFS scheduling. Multilevel feedback queues allow processes to move from one queue to another.

Many contemporary computer systems support multiple processors and allow each processor to schedule itself independently. Typically, each processor maintains its own private queue of processes (or threads), all of which are available to run. Issues related to multiprocessor scheduling include processor affinity and load balancing.

Operating systems supporting threads at the kernel level must schedule threads—not processes—for execution. This is the case with Solaris and Windows XP. Both of these systems schedule threads using preemptive, priority-based scheduling algorithms, including support for real-time threads. The Linux process scheduler uses a priority-based algorithm with real-time support as well. The scheduling algorithms for these three operating systems typically favor interactive over batch and CPU-bound processes.

The wide variety of scheduling algorithms demands that we have methods to select among algorithms. Analytic methods use mathematical analysis to determine the performance of an algorithm. Simulation methods determine performance by imitating the scheduling algorithm on a "representative" sample of processes and computing the resulting performance. However, simulation can at best provide an approximation of actual system performance; the only reliable technique for evaluating a scheduling algorithm is to implement the algorithm on an actual system and monitor its performance in a "real-world" environment.

Exercises

5.1 Discuss how the following pairs of scheduling criteria conflict in certain settings.

a. CPU utilization and response time

b. Average turnaround time and maximum waiting time

c. I/O device utilization and CPU utilization

5.2 Consider the following set of processes, with the length of the CPU burst given in milliseconds:

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

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

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1).

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the algorithms in part a results in the minimum average waiting time (over all processes)?

5.3 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

5.4 Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served

b. Shortest job first

c. Round robin

d. Priority

5.5 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks. What is the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

5.6 Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

5.7 Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes:

a. FCFS

b. RR

c. Multilevel feedback queues

5.8 Using the Windows XP scheduling algorithm, what is the numeric priority of a thread for the following scenarios?

a. A thread in the REALTIME_PRIORITY_CLASS with a relative priority of HIGHEST

The operating system must provide the means to guard against timing errors. Several language constructs have been proposed to deal with these problems. Monitors provide the synchronization mechanism for sharing abstract data types. A condition variable provides a method by which a monitor procedure can block its execution until it is signaled to continue.

Operating systems also provide support for synchronization. For example, Solaris, Windows XP, and Linux provide mechanisms such as semaphores, mutexes, spinlocks, and condition variables to control access to shared data. The Pthreads API provides support for mutexes and condition variables.

A transaction is a program unit that must be executed atomically; that is, either all the operations associated with it are executed to completion, or none are performed. To ensure atomicity despite system failure, we can use a write-ahead log. All updates are recorded on the log, which is kept in stable storage. If a system crash occurs, the information in the log is used in restoring the state of the updated data items, which is accomplished by use of the undo and redo operations. To reduce the overhead in searching the log after a system failure has occurred, we can use a checkpoint scheme.

To ensure serializability when the execution of several transactions overlaps, we must use a concurrency-control scheme. Various concurrency-control schemes ensure serializability by delaying an operation or aborting the transaction that issued the operation. The most common ones are locking protocols and timestamp ordering schemes.

Exercises

6.1 The first known correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

```
boolean flag[2]; /* initially false */
int turn;
```

The structure of process P_i (i == 0 or 1) is shown in Figure 6.27; the other process is P_j (j == 1 or 0). Prove that the algorithm satisfies all three requirements for the critical-section problem.

6.2 Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems.

6.3 Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

6.4 Describe how the Swap() instruction can be used to provide mutual exclusion that satisfies the bounded-waiting requirement.

6.5 Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

a. Increase *Available* (new resources added).

b. Decrease *Available* (resource permanently removed from system).

c. Increase *Max* for one process (the process needs more resources than allowed; it may want more).

d. Decrease *Max* for one process (the process decides it does not need that many resources).

e. Increase the number of processes.

f. Decrease the number of processes.

7.5 Consider a system consisting of m resources of the same type being shared by n processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and m resources.

b. The sum of all maximum needs is less than $m + n$.

7.6 Consider the dining-philosophers problem where the chopsticks are placed at the center of the table and any two of them could be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

7.7 We can obtain the banker's algorithm for a single resource type from the general banker's algorithm simply by reducing the dimensionality of the various arrays by 1. Show through an example that the multiple-resource-type banker's scheme cannot be implemented by individual application of the single-resource-type scheme to each resource type.

7.8 Consider the following snapshot of a system:

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

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix *Need*?

b. Is the system in a safe state?

c. If a request from process P_1 arrives for (0,4,2,0), can the request be granted immediately?

Review Questions

5.1 List four design issues for which the concept of concurrency is relevant.
5.2 What are three contexts in which concurrency arises?
5.3 What is the basic requirement for the execution of concurrent processes?
5.4 List three degrees of awareness between processes and briefly define each.
5.5 What is the distinction between competing processes and cooperating processes?
5.6 List the three control problems associated with competing processes and briefly define each.
5.7 List the requirements for mutual exclusion.
5.8 What operations can be performed on a semaphore?

5.9 What is the difference between binary and general semaphores?
5.10 What is the difference between strong and weak semaphores?
5.11 What is a monitor?
5.12 What is the distinction between *blocking* and *nonblocking* with respect to messages?
5.13 What conditions are generally associated with the readers/writers problem?

Review Questions

6.1 Give examples of reusable and consumable resources.
6.2 What are the three conditions that must be present for deadlock to be possible?
6.3 What are the four conditions that create deadlock?
6.4 How can the hold-and-wait condition be prevented?
6.5 List two ways in which the no-preemption condition can be prevented.
6.6 How can the circular wait condition be prevented?
6.7 What is the difference among deadlock avoidance, detection, and prevention?

available			
r1	r2	r3	r4
2	1	0	0

process	current allocation				maximum demand				still needs			
	r1	r2	r3	r4	r1	r2	r3	r4	r1	r2	r3	r4
p1	0	0	1	2	0	0	1	2				
p2	2	0	0	0	2	7	5	0				
p3	0	0	3	4	6	6	5	6				
p4	2	3	5	4	4	3	5	6				
p5	0	3	3	2	0	6	5	2				

a. Compute what each process still might request and display in the columns labeled "still needs."
b. Is this system currently in a safe or unsafe state? Why?
c. Is this system currently deadlocked? Why or why not?
d. Which processes, if any, are or may become deadlocked?
e. If a request from p3 arrives for (0, 1, 0, 0), can that request be safely granted immediately? In what state (deadlocked, safe, unsafe) would immediately granting that whole request leave the system? Which processes, if any, are or may become deadlocked if this whole request is granted immediately?

6.6 Apply the deadlock detection algorithm to the following data and show the results.

$$\text{Available} = (2 \quad 1 \quad 0 \quad 0)$$

$$\text{Request} = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix} \qquad \text{Allocation} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$

Review Questions

7.1 What requirements is memory management intended to satisfy?

7.2 Why is the capability to relocate processes desirable?

7.3 Why is it not possible to enforce memory protection at compile time?

7.4 What are some reasons to allow two or more processes to all have access to a particular region of memory?

7.5 In a fixed-partitioning scheme, what are the advantages of using unequal-size partitions?

7.6 What is the difference between internal and external fragmentation?

7.7 What are the distinctions among logical, relative, and physical addresses?

7.8 What is the difference between a page and a frame?

7.9 What is the difference between a page and a segment?

Problems

7.1 In Section 2.3, we listed five objectives of memory management, and in Section 7.1, we listed five requirements. Argue that each list encompasses all of the concerns addressed in the other.

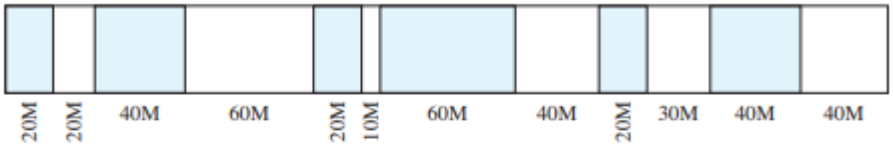
7.2 Consider a fixed partitioning scheme with equal-size partitions of 2^{16} bytes and a total main memory size of 2^{24} bytes. A process table is maintained that includes a pointer to a partition for each resident process. How many bits are required for the pointer?

7.3 Consider a dynamic partitioning scheme. Show that, on average, the memory contains half as many holes as segments.

7.4 To implement the various placement algorithms discussed for dynamic partitioning (Section 7.2), a list of the free blocks of memory must be kept. For each of the three methods discussed (best-fit, first-fit, next-fit), what is the average length of the search?

7.5 Another placement algorithm for dynamic partitioning is referred to as worst-fit. In this case, the largest free block of memory is used for bringing in a process. Discuss the pros and cons of this method compared to first-, next-, and best-fit. What is the average length of the search for worst-fit?

7.6 A dynamic partitioning scheme is being used, and the following is the memory configuration at a given point in time:



The shaded areas are allocated blocks; the white areas are free blocks. The next three memory requests are for 40M, 20M, and 10M. Indicate the starting address for each of the three blocks using the following placement algorithms:

a. First-fit
b. Best-fit
c. Next-fit. Assume the most recently added block is at the beginning of memory.
d. Worst-fit

7.7 A 1-Mbyte block of memory is allocated using the buddy system.

a. Show the results of the following sequence in a figure similar to Figure 7.6: Request 70; Request 35; Request 80; Return A; Request 60; Return B; Return D; Return C.
b. Show the binary tree representation following Return B.

7.8 Consider a buddy system in which a particular block under the current allocation has an address of 011011110000.

a. If the block is of size 4, what is the binary address of its buddy?
b. If the block is of size 16, what is the binary address of its buddy?

7.9 Let buddy $_k(x)$ = address of the buddy of the block of size 2^k whose address is x. Write a general expression for buddy $_k(x)$.

7.10 The Fibonacci sequence is defined as follows:

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n, \quad n \geq 0$$

a. Could this sequence be used to establish a buddy system?
b. What would be the advantage of this system over the binary buddy system described in this chapter?

7.11 During the course of execution of a program, the processor will increment the contents of the instruction register (program counter) by one word after each instruction fetch, but will alter the contents of that register if it encounters a branch or call instruction that causes execution to continue elsewhere in the program. Now consider Figure 7.8. There are two alternatives with respect to instruction addresses:

- Maintain a relative address in the instruction register and do the dynamic address translation using the instruction register as input. When a successful branch or call is encountered, the relative address generated by that branch or call is loaded into the instruction register.
- Maintain an absolute address in the instruction register. When a successful branch or call is encountered, dynamic address translation is employed, with the results stored in the instruction register.

Which approach is preferable?

7.12 Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of logical address space.

a. How many bits are in a logical address?
b. How many bytes in a frame?
c. How many bits in the physical address specify the frame?
d. How many entries in the page table?
e. How many bits in each page table entry? Assume each page table entry contains a valid/invalid bit.

page into each page frame, the time of last access to the page in each page frame, the virtual page number in each page frame, and the referenced (R) and modified (M) bits for each page frame are as shown (the times are in clock ticks from the process start at time 0 to the event — not the number of ticks since the event to the present).

Virtual page number	Page frame	Time loaded	Time referenced	R bit	M bit
2	0	60	161	0	1
1	1	130	160	1	0
0	2	26	162	1	0
3	3	20	163	1	1

A page fault to virtual page 4 has occurred at time 164. Which page frame will have its contents replaced for each of the following memory management policies? Explain why in each case.

a. FIFO (first-in-first-out)
b. LRU (least recently used)
c. Clock
d. Optimal (Use the following reference string.)
e. Given the aforementioned state of memory just before the page fault, consider the following virtual page reference string:

$$4, 0, 0, 0, 2, 4, 2, 1, 0, 3, 2$$

How many page faults would occur if the working set policy with LRU were used with a window size of 4 instead of a fixed allocation? Show clearly when each page fault would occur.

8.5 A process references five pages, A, B, C, D, and E, in the following order:

$$A; B; C; D; A; B; E; A; B; C; D; E$$

Assume that the replacement algorithm is first-in-first-out and find the number of page transfers during this sequence of references starting with an empty main memory with three page frames. Repeat for four page frames.

8.6 A process contains eight virtual pages on disk and is assigned a fixed allocation of four page frames in main memory. The following page trace occurs:

$$1, 0, 2, 2, 1, 7, 6, 7, 0, 1, 2, 0, 3, 0, 4, 5, 1, 5, 2, 4, 5, 6, 7, 6, 7, 2, 4, 2, 7, 3, 3, 2, 3$$

a. Show the successive pages residing in the four frames using the LRU replacement policy. Compute the hit ratio in main memory. Assume that the frames are initially empty.
b. Repeat part (a) for the FIFO replacement policy.
c. Compare the two hit ratios and comment on the effectiveness of using FIFO to approximate LRU with respect to this particular trace.

8.7 In the VAX, user page tables are located at virtual addresses in the system space. What is the advantage of having user page tables in virtual rather than main memory? What is the disadvantage?

Review Questions

12.1 What is the difference between a field and a record?

12.2 What is the difference between a file and a database?

12.3 What is a file management system?

12.4 What criteria are important in choosing a file organization?

12.5 List and briefly define five file organizations.

12.6 Why is the average search time to find a record in a file less for an indexed sequential file than for a sequential file?

12.7 What are typical operations that may be performed on a directory?

12.8 What is the relationship between a pathname and a working directory?

12.9 What are typical access rights that may be granted or denied to a particular user for a particular file?

12.10 List and briefly define three blocking methods.

12.11 List and briefly define three file allocation methods.

1. Three processes arrive at time zero with CPU bursts of 16, 20 and 10 milliseconds. If the scheduler has prior knowledge about the length of the CPU bursts, what will be the minimum achievable average waiting time for these three processes in a non-preemptive scheduler (rounded to nearest integer)?

2. An operating system needs to manage a set of tasks with mixed priority levels. There are three types of tasks:

a. **High-priority real-time tasks** that need immediate execution (e.g., hardware interrupts).
b. **Medium-priority CPU-bound tasks** that are computationally intensive but not urgent (e.g., data analysis).
c. **Low-priority I/O-bound tasks** that spend most of their time waiting for I/O operations (e.g., file transfers).

Process	**Priority**	**Burst Time (ms)**	**Arrival Time (ms)**
P1	High	15	0
P2	Low	20	5
P3	Medium	25	10
P4	High	5	12
P5	Low	30	20

Apply the **priority-based preemptive scheduling algorithm** to these tasks and determine the execution order. Calculate the average waiting time and turnaround time for the system.

3.

Process Id	Arrival time	Burst time	Priority
P1	0	8	3
P2	1	4	3
P3	2	5	4
P4	3	3	4
P5	4	1	5

If the CPU scheduling policy is priority preemptive, calculate the average waiting time and average turn around time. (Higher number represents higher priority)

4. If the CPU scheduling policy is SJF preemptive, in the following system, calculate the average waiting time and average turn around time.

Process Id	Arrival time	Burst time

P1	1	3
P2	2	5
P3	4	2
P4	0	4
P5	2	2

5. Consider the following table of arrival time and burst time for four processes P1, P2, P3 and P4.

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

   Calculate the average waiting time and average turn around time, if the system follows preemptive Longest Remaining Time First CPU scheduling algorithm.

6. A multilevel queue scheduling system is used in an OS with three different queues:

1. **System Queue**: Highest priority, time quantum of 10ms.
2. **Interactive Queue**: Medium priority, time quantum of 20ms.
3. **Background Queue**: Lowest priority, no specific time quantum (First-Come, First-Served).

The operating system needs to manage the following tasks:

Process	**Queue Type**	**Burst Time (ms)**	**Arrival Time (ms)**
P1	System	25	0
P2	Interactive	40	5
P3	Background	50	10
P4	System	10	15
P5	Interactive	20	20

Simulate the execution of these tasks using a **multilevel queue scheduling algorithm** with the given time quantum for each queue. Provide a Gantt chart of the execution process, and calculate the total CPU idle time.

1. Consider a system with the following processes and resources:

   - Processes: P1, P2, P3
   - Resources: R1 (2 instances), R2 (1 instance), R3 (1 instance)

The current resource allocation status is as follows:

- **P1** is holding 1 instance of **R1** and requesting 1 instance of **R2**.
- **P2** is holding 1 instance of **R2** and requesting 1 instance of **R3**.
- **P3** is holding 1 instance of **R3** and requesting 1 instance of **R1**.

Draw the Resource Allocation Graph (RAG) for this system. Based on the graph, determine if there is a deadlock. If deadlock exists, identify the processes involved.

2. A system with three processes (P1, P2, P3) and three resources (R1, R2, R3) is in a potential deadlock situation. The current resource allocation and request state is as follows:

   - **P1** holds **R1** and requests **R2**.
   - **P2** holds **R2** and requests **R3**.
   - **P3** holds **R3** and requests **R1**.

   (a) Draw the current Resource Allocation Graph (RAG) for this system.
   (b) Modify the graph by introducing one possible change to the allocation or request of resources such that the system avoids deadlock. Explain your reasoning behind the modification.

3. A system consists of four processes (P1, P2, P3, P4) and three resources (R1, R2, R3), each having only one instance. The following allocation and request table is given:

Process	**Currently Held Resources**	**Requested Resources**
P1	R1	R2
P2	R2	R3
P3	None	R3
P4	R3	R1

   The system has detected a deadlock involving some or all of these processes.

   (a) Analyze the system's state by identifying which processes are in a deadlock.
   (b) Propose a suitable **deadlock recovery strategy** (e.g., process termination or resource preemption).

4. A system has detected a deadlock involving three processes (P1, P2, P3) and two resources (R1, R2). The following information is available:

- **P1** holds **R1** and is requesting **R2**.
- **P2** holds **R2** and is requesting **R1**.
- **P3** is waiting for either **R1** or **R2** to be released.

The system administrator needs to terminate one process to resolve the deadlock.

(a) Analyze which process should be terminated to resolve the deadlock. Consider the resources each process holds and requests.
(b) Explain the reasoning behind your decision.

5. Consider a system with five processes (P1, P2, P3, P4, P5) and three resource types (R1, R2, R3). The following tables show the **allocation**, **maximum**, and **available** resources:

Process	**Allocation (R1, R2, R3)**	**Maximum (R1, R2, R3)**
P1	1, 0, 0	3, 2, 2
P2	2, 1, 1	5, 3, 2
P3	3, 1, 1	4, 2, 2
P4	0, 2, 1	3, 3, 2
P5	1, 1, 1	4, 3, 3

Available resources: **R1 = 1**, **R2 = 1**, **R3 = 2**

Apply the Banker's Algorithm to determine whether the system is in a **safe state**.

6. Define deadlock in the context of operating systems. Describe the four necessary conditions for a deadlock to occur, and explain how deadlock detection differs from deadlock prevention.

1. Consider a system with 1000 KB of total memory. The following processes arrive and request memory in the given order:

Process	Memory Requested (KB)
P1	200
P2	350
P3	100
P4	400

The system uses the **first-fit** allocation algorithm.
(a) Apply the first-fit algorithm to allocate memory for the processes.
(b) Calculate how much memory is left after all processes have been allocated, and identify any fragmentation.

2. A system has the following memory partitions available:

Partition	Size (KB)
P1	300
P2	500
P3	200
P4	600

Four processes arrive, each requesting memory:

Process	Memory Requested (KB)
P1	350
P2	200
P3	400
P4	100

(a) Use the **best-fit** algorithm to allocate memory to the processes.
(b) After all processes are allocated, calculate the amount of external fragmentation. How does the best-fit algorithm affect fragmentation compared to other allocation strategies?

3. Consider a system that uses paging for memory management. The page size is **4 KB**. A process has the following logical address:

- **Logical address**: 15,342

The system has the following page table:

Page Number	Frame Number
0	3
1	7
2	1
3	4

(a) Calculate the **page number** and **offset** from the given logical address.
(b) Use the page table to determine the corresponding **physical address**.

4. In a system using segmentation, the segment table for a process is as follows:

Segment Number	Base Address	Limit
0	1000	500
1	4000	1000
2	7000	200

The process generates the following **logical address** for a memory access:

- **Segment Number = 1**, **Offset = 600**

Translate the given logical address into a **physical address** using the segment table.

5. A process has been assigned a **relocation register** value of **5000** and a **limit register** value of **3000**. The process attempts to access the following logical addresses:

    1. **Logical address 1500**
    2. **Logical address 3200**

(a) For each logical address, determine the **physical address** by applying the relocation register.
(b) Check if each access is **valid** by comparing the logical address with the limit register. If the access is invalid, explain why.

6. Explain the function of a Translation Lookaside Buffer (TLB) in a paging system. How does it improve the performance of address translation? Discuss the implications of TLB size on hit rate and overall system performance.

1. A company is considering different RAID (Redundant Array of Independent Disks) levels for their data storage system, particularly RAID 0, RAID 1, RAID 5, and RAID 6.

   Compare the performance, data redundancy, and fault tolerance of each RAID level. Provide specific advantages and disadvantages for each configuration in the context of high availability and data recovery.

2. In a disk management system, fragmentation can lead to inefficient storage utilization and performance degradation.

   (a) Analyze the causes of fragmentation (both internal and external) in disk management systems.
   (b) Evaluate the impact of fragmentation on disk performance and suggest strategies to minimize fragmentation, including defragmentation techniques.

3. A hard disk drive has 4 platters, with each platter containing 1000 tracks. Each track has 50 sectors, and each sector can store 512 bytes of data.
   (a) Calculate the total storage capacity of the disk.
   (b) If a file requires 2048 bytes of storage, how many sectors and tracks will it occupy?

4. Describe the physical structure of a hard disk drive (HDD). Include details on components such as platters, tracks, sectors, and read/write heads. How do these components work together to store and retrieve data?

1. Explain the concept of **demand paging** in virtual memory systems. How does it differ from pre-paging? Discuss the benefits and drawbacks of demand paging in terms of memory utilization and system performance.

2. Define **thrashing** in the context of virtual memory management. Describe the conditions under which thrashing occurs and its impact on system performance.
   (a) Provide a scenario illustrating thrashing with specific reference to page fault rates.
   (b) Discuss strategies that can be implemented to reduce thrashing in a system.

3. Compare and contrast two common page replacement algorithms: **Least Recently Used (LRU)**, and **Optimal Page Replacement**.
   Consider a reference string of page requests: **[1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]** with 3 page frames. Calculate the number of page faults for each algorithm.

4. Discuss how the size of page frames affects the performance of a virtual memory system.
   Analyze the trade-offs involved in choosing larger versus smaller page sizes, particularly regarding page fault rates and internal fragmentation.