STEP TOWARDS SUCCESS

AKASH'S

Guru Gobind Singh Indra Prastha University Series

SOLVED PAPERS

[PREVIOUS YEARS SOLVED QUESTION PAPERS]

[MCA]

FIRST SEMESTER(MCA-105)
Operating System with Linux

Rs. 81.00/-

AKASH BOOKS
NEW DELHI

SYLLABUS
OPERATING SYSTEMS WITH LINUX
(MCA-105)

Applicable from the Academic Session 2021-22 Onwards

UNIT I

Operating System: Concept, Components of Operating System, Operating System Operations, Protection and Security. Computing Environment. **Abstract View of OS:** User view, System View, Operating System Services, **System Calls:** Concept, types of System Calls.

Computer System Architecture: Single-Processor Systems, Multiprocessor Systems.

Types of Operating systems: Batch Operating System, Multi-Programmed Operating System, Time-Shared Operating System, Real Time Operating System, Distributed Operating Systems.

Process Management: Process Concept, Operation on Processes, Cooperating Processes, Inter-Process Communication, Threads.

Linux Operating System: Introduction to Linux OS, Basic Commands of Linux OS.

[No. of Hrs. 11]

UNIT II

Process Synchronization: Introduction, The Critical-Section Problem with solution, Bakery Algorithm, Synchronization hardware, Semaphores, Semaphores Implementation, Classical Problems of Synchronization with algorithms, Critical Regions, Monitors.

CPU Scheduling: Basic Concepts, Scheduling Criteria, Scheduling algorithms, Multilevel Queue Scheduling, Multilevel Feedback Queue Scheduling.

Linux Operating System: Process Management Commands and System Calls.

[No. of Hrs. 11]

UNIT III

Deadlock: System Models, Deadlock Characterization, Resource Allocation Graph, Deadlock Prevention, Avoidance, Detection and Recovery, Banker's algorithm.

Memory Management: Main Memory: Contiguous Memory Allocation, Fragmentation, Paging, And Segmentation. **Virtual Memory:** Demand Paging, Page Replacement, Page replacement algorithm, Allocation of frames, Thrashing.

Linux Operating System: Memory Management Commands and System Calls.

[No. of Hrs. 12]

UNIT IV

File, Devices and Secondary Storage Management: File-System Interface: Concepts, Access Methods, Directory and Disk Structure. File-System Structure, File-System Implementation, Directory Implementation, Allocation Methods, Free-Space Management.

Devices: Types of devices, Channels and Control Unit, Multiple Paths, Block Multiplexing.

Secondary Storage: Mass-Storage Structure, Disk Structure, Disk Scheduling Algorithms, Disk Management, RAID structure of disk.

Linux Operating System: File Management Commands and System Calls.

[No. of Hrs. 10]

SYLLABUS (2016-17)

OPERATING SYSTEMS
[MCA-106]

UNIT-I

Operating System: Introduction, Role, Types of OS; Batch Systems, multi programming, time-sharing parallel, distributed and real-time systems, Operating system structure, Operating system components and services, System calls.

Process: Process Concept Process Scheduling, Operation on Process, Cooperating Processes, threads.

CPU Scheduling: Basic Concepts, Scheduling Criteria, Scheduling Algorithms, Multiple-Processor Scheduling, Real-Time Scheduling, Algorithm Evaluation.

[No. of Hrs.: 10]

UNIT-II

Interprocess Communication and Synchronization: Background, The Critical-Section Problem, Synchronization Hardware, Semaphores, Classical Problems of Synchronization, Critical Regions, Monitors, Message Passing.

Deadlocks: System Model, Deadlock Characterization, methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, Recovery from Deadlock, Combined Approach to Deadlock Handling.

Memory Management: Background, Logical vs. Physical Address space, swapping, Contiguous allocation, Paging, Segmentation, Segmentation with Paging.

Virtual Memory: Demand Paging, Page Replacement, Page-replacement Algorithms Performance of Demand Paging Allocation of Frames, Thrashing, Other Considerations, Demand Segmentation. **[No. of Hrs: 11]**

UNIT-III

Device Management: Techniques for Device Management, Dedicated Devices, Shared Devices, Virtual Devices; Devices Characteristics-Hardware Consideration, Channels and Control Units, Independent Device Operation, Buffering, Multiple paths, Block Multiplexing, Device Allocation Consideration

Secondary-Storage Structure: Disk Structure, Disk Management, Swap-Space Management, Disk Reliability. **[No. of Hrs.: 12]**

UNIT-IV

File-System Interface: File Concept, Access Methods, Directory Structure.

File-System Implementation: Introduction, File-System Structure, Basic File System, Allocation methods, Free-Space management, Directory Implementation.

Security: The Security problem, Goals of protection, Access matrix, Authentication, program threats, System threats, Intrusion detection, Cryptography.

Case Study: Linux Operating System and Windows XP. **[No. of Hrs.: 10]**

END TERM EXAMINATION [JUNE 2014]
SECOND SEMESTER [MCA]
OPERATING SYSTEM [MCA–106]

Time: 3 hours **M.M. : 60**

Note: *Attempt any five questions including Q.no.*1 *which is compulsory.*

Q.1 Explain in brief:- (3 × 4 = 12)

Q.1. (a) . Discuss the various types of Operating System.

Ans. Batch operating system: The users of batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers left their programs with the operator. The operator then sorts programs into batches with similar requirements.

Time-sharing operating systems: Time sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, objective is to maximize processor use, whereas in Time-Sharing Systems objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receives an immediate response. For example, in a transaction processing, processor execute each user program in a short burst or quantum of computation. That is if n users are present, each user can get time quantum. When the user submits the command, the response time is in few seconds at most.

Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Network operating System: Network Operating System runs on a server and and provides server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks. Examples of network operating systems are Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

Real Time operating System: Real time system is defines as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. Real time processing is always on line whereas on line system need not be real time. The time taken by the system to respond to an input and display of required updated information is termed as response time. So in this method response time is very less as compared to the online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail.For example Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-applicance controllers, Air traffic control system etc.

There are two types of real-time operating systems.

Hard real-time systems: Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems secondary storage is limited or missing with data stored in ROM. In these systems virtual memory is almost never found.

Soft real-time systems: Soft real time systems are less restrictive. Critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems.For example, Multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers etc.

Q.1. (b) What are the advantages in having different time quantum on different levels of multilevel queuing systems?

Ans. Processes which need more frequent servicing, for instance interactive processes such as editors, can be in a queue with a small time quantum. Processes with no need for frequent servicing can be in a queue with a larger time quantum, requiring fewer context switches to complete the processing, making more efficient use of the computer.

Q.1. (c) Differentiate between Pre-emptive and non pre-emptive scheduling?

Ans. Preemptive scheduling: The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.

Non-Preemptive scheduling: When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.

Q.2. Consider the following snapshot of the system:-

	Process Allocation	Max	Available
	ABCD	ABCD	ABCD
P0	0012	0012	1520
P1	1000	1750	
P2	1354	2356	
P3	0632	0652	
P4	0014	0656	

Answer the following questions using bankers algorithm:-

(a)What is the content of the matrix Need? Is the system is in a safe state? (6)

Ans. Content of the matrix need can be calculated as

Need = maximum resources – currently allocated resources

	ABCD(NEED)
P0	0000
P1	0750
P2	1002
P3	0020
P4	0642

To determine the safe state We need to iterate and apply the following algorithm.

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

Step I. Work is a working copy of the available resources, which will be modified during the analysis.

Step II. Finish is a vector of booleans indicating whether a particular process can finish. (or has finished so far in the analysis.)

Step III. Initialize Work to Available, and Finish to false for all elements.

In this case Work will be 1520

Step IV. Find an i such that both (A) Finish[i] == false, and (B) Need[i] < Work.

So P0 < Work added to sequence and Work updated to 1520

But P1 > Work skip this

P2 > Work skip this

P3 < Work added to sequence (P0,P3) and work updated to 1540

P2 < Work added to sequence (P0,P3,P2) and work updated to 2542

So no next safe sequence cannot be found further p1 and p4 resource need cannot be fulfilled.

So the system is not in the safe state.

This process has not finished, but could with the given available working set. If no such i exists, go to step 4.

Set Work = Work + Allocation[i], and set Finish[i] to true. This corresponds to process i finishing up and releasing its resources back into the work pool. Then loop back to step 2.

If finish[i] == true for all i, then the state is a safe state, because a safe sequence has been found.

Q.2. (b)If a request from Process P1 arrives for (0,4,2,0), can request be granted immediately? (6)

Ans. Work = 1,5,2,0

Yes it can be granted immediately because this will create a safe sequence (P1,P0,P2,P3,P4).

Q.3. Discuss Bankers Algorithm and its use to detect and resolve deadlock. Explain with example. (12)

Ans. The Banker's algorithm, sometimes referred to as the avoidance algorithm, is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes an "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue. When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time. For the Banker's algorithm to work, it needs to know three things:

- How much of each resource each process could possibly request[MAX]
- How much of each resource each process is currently holding[ALLOCATED]
- How much of each resource the system currently has available[AVAILABLE]

Resources may be allocated to a process only if it satisfies the following conditions:

1.request ≤ available, else process waits until resources are available.

Some of the resources that are tracked in real systems are memory, semaphores and interface access.

The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

•**Available:** A vector of length m indicates the number of available resources of each type. If Available [j] = k, there are k instances of resource type Rj available.

• **Max:** An n x m matrix defines the maximum demand of each process. If Max[i,j] = k, then Pi may request at most k instances of resource type Rj.

•**Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process Pi is currently allocated k instances of resource type Rj.

• **Need:** An n×m matrix indicates the remaining resource need of each process. If Need [i, j] = k, then Pi may need k more instances of resource type Rj to complete the task.

Note: Need[i, j] = Max[i, j] - Allocation[i,j].

Total system resources are:

A	B	C	D
6	5	7	6

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P0	1	2	2	1
P1	1	0	3	3
P2	1	2	1	0

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Need = maximum resources – currently allocated resources

Processes (possibly needed resources):

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

By identifying Safe and Unsafe States, we detect and resolve deadlock.

A state is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resource it only makes it easier on the system. A safe state is considered to be the decision maker if it is going to process ready queue. Safe State ensures the Security.

Given that assumption, the algorithm determines if a state is safe by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an unsafe state and We can show that the state given in the previous example is a safe state by showing that it is possible for each process to acquire its maximum resources and then terminate.

1.P1 acquires 2 A, 1 B and 1 D more resources, achieving its maximum

•[available resource: <3 1 1 2> – <2 1 0 1> = <1 0 1 1>]

•The system now still has 1 A, no B, 1 C and 1 D resource available

2.P1 terminates, returning 3 A, 3 B, 2 C and 2 D resources to the system

•[available resource: <1 0 1 1> + <3 3 2 2> = <4 3 3 3>]

•The system now has 4 A, 3 B, 3 C and 3 D resources available

3.P2 acquires 2 B and 1 D extra resources, then terminates, returning all its resources

•[available resource: <4 3 3 3> – <0 2 0 1> + <1 2 3 4> = <5 3 6 6>]

•The system now has 5 A, 3 B, 6 C and 6 D resources

4. P3 acquires 1 B and 4 C resources and terminates

•[available resource: <5 3 6 6> – <0 1 4 0> + <1 3 5 0> = <6 5 7 6>]

•The system now has all resources: 6 A, 5 B, 7 C and 6 D

5. Because all processes were able to terminate, this state is safe

For an example of an unsafe state, consider what would happen if process 2 was holding 1 more unit of resource B at the beginning.

Q.4. (a) Define Semaphores. How can it be used as general synchronization tools? What advantage a semaphore have as compared to a hardware test and set instruction? (6)

Ans. A semaphore is defined as a shared integer variable (with nonnegative values) that apart from initialization, is accessed only through two indivisible operations (also called atomic operations): wait and signal. These operations were originally termed p (for wait) and v (for signal).

The classical definitions of wait and signal are:

Resources may be allocated to a process only if it satisfies the following conditions:

1.request ≤ available, else process waits until resources are available.

Some of the resources that are tracked in real systems are memory, semaphores and interface access.

The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

•**Available:** A vector of length m indicates the number of available resources of each type. If Available [j] = k, there are k instances of resource type Rj available.

• **Max:** An n x m matrix defines the maximum demand of each process. If Max[i,j] = k, then Pi may request at most k instances of resource type Rj.

•**Allocation:** An n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process Pi is currently allocated k instances of resource type Rj.

• **Need:** An n×m matrix indicates the remaining resource need of each process. If Need [i, j] = k, then Pi may need k more instances of resource type Rj to complete the task.

Note: Need[i, j] = Max[i, j] - Allocation[i,j].

Total system resources are:

A	B	C	D
6	5	7	6

Available system resources are:

A	B	C	D
3	1	1	2

Processes (currently allocated resources):

	A	B	C	D
P0	1	2	2	1
P1	1	0	3	3
P2	1	2	1	0

Processes (maximum resources):

	A	B	C	D
P1	3	3	2	2
P2	1	2	3	4
P3	1	3	5	0

Need = maximum resources – currently allocated resources

Processes (possibly needed resources):

	A	B	C	D
P1	2	1	0	1
P2	0	2	0	1
P3	0	1	4	0

By identifying Safe and Unsafe States, we detect and resolve deadlock.

A state is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resource it only makes it easier on the system. A safe state is considered to be the decision maker if it is going to process ready queue. Safe State ensures the Security.

Given that assumption, the algorithm determines if a state is safe by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an unsafe state and We can show that the state given in the previous example is a safe state by showing that it is possible for each process to acquire its maximum resources and then terminate.

1.P1 acquires 2 A, 1 B and 1 D more resources, achieving its maximum

•[available resource: <3 1 1 2> – <2 1 0 1> = <1 0 1 1>]

•The system now still has 1 A, no B, 1 C and 1 D resource available

2.P1 terminates, returning 3 A, 3 B, 2 C and 2 D resources to the system

•[available resource: <1 0 1 1> + <3 3 2 2> = <4 3 3 3>]

•The system now has 4 A, 3 B, 3 C and 3 D resources available

3.P2 acquires 2 B and 1 D extra resources, then terminates, returning all its resources

•[available resource: <4 3 3 3> – <0 2 0 1> + <1 2 3 4> = <5 3 6 6>]

•The system now has 5 A, 3 B, 6 C and 6 D resources

4. P3 acquires 1 B and 4 C resources and terminates

•[available resource: <5 3 6 6> – <0 1 4 0> + <1 3 5 0> = <6 5 7 6>]

•The system now has all resources: 6 A, 5 B, 7 C and 6 D

5. Because all processes were able to terminate, this state is safe

For an example of an unsafe state, consider what would happen if process 2 was holding 1 more unit of resource B at the beginning.

Q.4. (a) Define Semaphores. How can it be used as general synchronization tools? What advantage a semaphore have as compared to a hardware test and set instruction? (6)

Ans. A semaphore is defined as a shared integer variable (with nonnegative values) that apart from initialization, is accessed only through two indivisible operations (also called atomic operations): wait and signal. These operations were originally termed p (for wait) and v (for signal).

The classical definitions of wait and signal are:

wait (S) : while S < 0 do no op;

S : = S – 1;

Signal (S) : S : = S + 1;

Usage of a general Synchronization Tool: We can use semaphore to deal with the n-process critical-section problem. The n processes share a semaphore, mutex (standing for mutual exclusion) initialized to 1. Each process Pi is organised as shown in Figure:

We can also use semaphores to solve various synchronization problem. For example, consider two concurrently running process P, with a statement S1, and P2 with a statement S2. Suppose that we require that S2 be executed only after S1 has completed. We can implement this scheme readily by letting P1 and P2 share a common semaphore synch, initialized to 0, and by inserting the statements S1;

S1; signal (synch); in process P1, and the statements wait (synch); S2; in Process P2. Because synch is initialized to 0. P2 will execute S2 only after P1 has involved signal (synch), which is after S1.

Repeat

Wait (mutex);

Signal (mutex);

Remainder section

Critical section

Untill False;

Mutual Exculstion Implementation with Semaphore

Advantage of semaphore over test and set instruction:

1. The main advantage of semaphore i.e., Test and set hardware are not easy to generalize to more complex problems. But to overcome this difficulty, we can use a synchronization tool, called a semaphore.

2. Another advantage is Test and set Hardware solution is not feasible in a multiprocessor environment. But by semaphore we will provide a solution for multiprocessor environment.

Q.4. (*b*) . Explain the four necessary conditions that must be in effect for a deadlock to exist? (6)

Ans. Four necessary conditions:

1. Mutual Exclusion Condition: The resources involved are non-shareable.

Explanation: At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and Wait Condition: Requesting process hold already, resources while waiting for requested resources.

Explanation: There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3. No-Preemptive Condition: Resources already allocated to a process cannot be preempted.

Explanation: Resources cannot be removed from the processes are used to completion or released voluntarily by the process holding it.

4. Circular Wait Condition: The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

Methods for Handling dead locks: The Banker's algorithm is a resource allocation and deadlock avoidance algorithm developed by EdsgerDijkstra that tests for safety by simulating the allocation of predetermined maximum possible amounts of all resources, and then makes a "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether alocation should be allowed to continue.

The algorithm was developed in the design process for the THE operating system and originally described (in Dutch) in EWD108. The name is by analogy with the way that bankers account for liquidity constraints.

Data Structures for the Banker's Algorithm: Available: Vector of length m. If available [j] = k, there are k instances of resources type Rj available.

Max: n x m matrix. If Max [i, j] = k, then process Pi may request at most k instances of resource type Rj.

Allocation: n x n matrix. If Allocation [i, j] = k then Pi is currently allocated k instances ofRj.

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

Need [i, j] = Max [i, j] – Allocation [i, j].

Q.5. A Computer has 1000K of main memory. The jobs arrive and finish in the following sequence Job1 requiring 200k arrives, Job2 requiring 350K arrives, Job3 requiring 300K arrives, Job1 finishes, Job4 requiring 120K arrives, Job5 requiring 150K arrives, Job 6 requiring 80K arrives.

Draw memory allocation table using best fit , first fit and next fit algorithm and find out which algorithm perform better for the sequence. (12)

Ans. JOB1 comes and acquires 200K out of 1000k. remaining 800K

BEST FIT

200*K* Allocated	800K

Worst FIT

200*K* Allocated	800K

First FIT

200*K* Allocated	800K

JOB2 comes and acquires 350 out of 800k remainig 450

BEST FIT

200*K* Allocated	350K Allocated	450K

Worst FIT

200*K* Allocated	350K Allocated	450K

Next Fit

200*K* Allocated	350K Allocated	450K

JOB3 comes and acquires 300 out of 450 and remaining 150 K

Best FIT

200K allocated	350K allocated	300k allocated	150K free

Worst Fit

200K allocated	350K allocated	300k allocated	150K free

NEXT FIT

200K allocated	350K allocated	300k allocated	150K free

JOB1 finishes and frees the memory

200K allocated	350K allocated	300k allocated	150K free

JOB4 comes and acquires 120K

200K free	350K allocated	300K allocated	120K allocated	130K free

Worst Fit

120K allocated	80K Free	350K allocated	300K allocated	150K free

Next Fit

120K allocated	80K Free	300K allocated	300 K allocated	150K free

JOB5 requires 150K memory

Worst Fit or Next Fit at Job 4 allocation cause Best FIT for JOB5

Best Fit

120K allocated	80K Free	300K allocated	300K allocated	150 K allocated

Best fit at JOB4 will become Worst FIT and Next FIT of JoB5.

150K allocated	50K free	350K allocated	300K allocated	120K allocated	30K free

For JOB4 worst Fit or Next Fit will be the best algorithm to fulfill BEST FIT for JOB.

JOB 6 requires 80K which can only be fulfilled by Best Fit algorithm at JOB5.

Best Fit

120K allocated	80K allocated	350K allocated	300K allocated	150K allocated

For JOB6: Best FIT is best algorithm

Q.6. (a) What is segmented memory technique? Discuss its various facts in detail? **(4)**

Ans. Segmentation: Meaningful Allocation: Segmentation is a technique for allocating memory in chunks of varying and meaningful sizes instead of one arbitrary page size. Since the segment sizes are selected by the compiler, internal fragmentation is reduced. Of course, since the segments are of different sizes, we have the same problems with external fragmentation and compaction that we discussed when we talked about variable sized partions for multiprogramming. For this reason, among others, we'll find

that although segmentation is supported by popular processors like those in the x86 family, paging is most often used by processors that support it, including those in the x86 family.

Meaningful Allocation: When programmers think of memory they see it as a collection of different objects, not as a linear array of bytes or pages. These objects, unlike pages vary in size. It is this view of memory they segmentation seeks to support.

A program might be broken down into segments for the code, stack and heap. In addition, separate segments might be created for statically allocated data structures such as heaps and arrays. Each of these segments is exactly the necessary size, so internal fragmentation is eliminated.

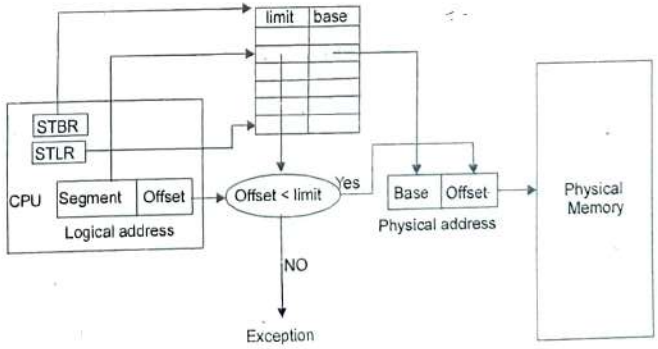
These segments are then numbered. An address in memory is specificed as a tuple consisting of the segment number and the offset within the segment: .

The Segment Table: Just as was the case for paging, we must provide a mechanism for mapping from this logical view of memory, to the linearly addressable physical memory. This mechanism is the segment table

Since the segments are defined by the compiler, the segment table can be fixed at link time. The table contains one entry for each segment. The entry contains the base and limit for the segment. The base is the physical address of the beginning of the segment in memory. The limit is the length of the segment, so that the last physical address within the segment is specified by the base plus the bound (base + bound).

Since each program enjoys its own virtual address space, each process has it's own segment table. The sement table of the current process is kept in memory and is referenced by two hardware registers, the segment-table base register (STBR) and the segment-table length register (STLR). On certain architectures, if the segment table is small enough, it can be stored entirely in registers within the CPU.

Using The Segment Table: When an address is specified as a tuple , the hardware translates it to a physical address using the segment table. The hardware checks the entry in the table specified by the offset number of find the base address of the segment. If the segment-number is out-of-bounds, the address is not valid. It then checks the limit to ensure that the offset is less than the limit. If the offset is not less than the limit, the address is invalid. If the address is valid, it is translated to physical address base+offset.



Protection and Sharing: Since segments represent logical units of a program, they are ideal for sharing. For example, it is logical for the code segment of several processes corresponding to the same program to be shared. Appropriate protection and security can be enforced by associating this information with the segment table.

When statically allocated arrays and other complex structures are associated with their own segment, bounds checking becomes almost free. It happens as a natural consequence of the lookup in the page table and requires no additional instructions.

Q.6. (b) Define the term overlay with an example? (4)

Ans. • The entire program and data of a process must be in the physical memory for the process to execute.

• The size of a process is limited to the size of physical memory.

• If a process is larger than the amount of memory, a technique called overlays can be used.

• Overlays is to keep in memory only those instructions and data that are needed at any given time.

• When other instructions are needed, they are loaded into space that was occupied previously by instructions that are no longer needed.

• Overlays are implemented by user, no special support needed from operating system, programming design of overlay structure is complex

Example: Consider a two-pass assembler. – Pass1 constructs a symbol table. – Pass2 generates machine-language code. – Assume the following:

	Size (K = 1024 bytes)
Pass 1	70k
Pass 2	80k
Symbol table	20k
Common routines	30k
Total size	200k

– To load everything at once, we need 200k of memory.

If only 150K is available, we cannot run our process.

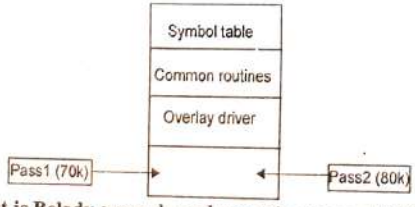
• Notice that Pass1 and Pass2 do not need to be in memory at same time. So, we define two overlays:- Overlay A: symbol table, common routines, and

Pass1.– Overlay B: symbol table, common routines, and pass 2.

• We add overlay driver 10k and start with overlay A in memory.

• When finish Pass1, we jump to overlay driver, which reads overlay B into memory overwriting overlay A and transfer control to Pass.

• Overlay A needs 130k and Overlay B needs 14ok



Q.6. (c) What is Belady anomaly and page size anomaly? Discuss with an example. (4)

Ans. Belady's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern. This phenomenon is commonly experienced when using the First in First Out (FIFO) page replacement algorithm.

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	3	2	4	4	4	1	0	0
		3	2	1	0	3	2	2	2	4	1	1
Oldest Page			3	2	1	0	3	3	3	2	4	4
Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	0	0	4	3	2	1	0	4
		3	2	1	1	1	0	4	3	2	1	0
			3	2	2	2	1	0	4	3	2	1
Oldest Page				3	3	3	2	1	0	4	3	2

An example of Belady's anomaly. Using three page frames, 9 page faults occur. Increasing to four page frames causes 10 page faults to occur. Page faults are in red.

Q.7. (a).Discuss various file allocation strategies to store the files in secondary storage devices. (6)

Ans. •There are three major methods of storing files on disks: contiguous, linked, and indexed.

1. Contiguous Allocation

•**Contiguous Allocation** requires that all blocks of a file be kept together contiguously.

• Performance is very fast, because reading successive blocks of the same file generally requires no movement of the disk heads, or at most one small step to the next adjacent cylinder.

• Storage allocation involves the same issues discussed earlier for the allocation of contiguous blocks of memory (first fit, best fit, fragmentation problems, etc.) The distinction is that the high time penalty required for moving the disk heads from spot to spot may now justify the benefits of keeping files contiguously when possible.

• (Even file systems that do not by default store files contiguously can benefit from certain utilities that compact the disk and make all files contiguous in the process.)

•Problems can arise when files grow, or if the exact size of a file is unknown at creation time:

→ Over-estimation of the file's final size increases external fragmentation and wastes disk space.

→ Under-estimation may require that a file be moved or a process aborted if the file grows beyond its originally allocated space.

→ If a file grows slowly over a long time period and the total final space must be allocated initially, then a lot of space becomes unusable before the file fills the space.

•A variation is to allocate file space in large contiguous chunks, called **extents.** When a file outgrows its original extent, then an additional one is allocated. (For example an extent may be the size of a complete track or even cylinder, aligned on an appropriate track or cylinder boundary.) The high-performance files system Veritas uses extents to optimize performance.

2. Linked Allocation

• Disk files can be stored as linked lists, with the expense of the storage space consumed by each link. (E.g. a block may be 508 bytes instead of 512.)

• Linked allocation involves no external fragmentation, does not require pre-known file sizes, and allows files to grow dynamically at any time.

• Unfortunately linked allocation is only efficient for sequential access files, as random access requires starting at the beginning of the list for each new location access.

• Allocating **clusters** of blocks reduces the space wasted by pointers, at the cost of internal fragmentation.

• Another big problem with linked allocation is reliability if a pointer is lost or damaged. Doubly linked lists provide some protection, at the cost of additional overhead and wasted space.

• The File Allocation Table, FAT, used by DOS is a variation of linked allocation, where all the links are stored in a separate table at the beginning of the disk. The benefit of this approach is that the FAT table can be cached in memory, greatly improving random access speeds.

3. Indexed Allocation

• Indexed Allocation combines all of the indexes for accessing each file into a common block (for that file), as opposed to spreading them all over the disk or storing them in a FAT table.

Some disk space is wasted (relative to linked lists or FAT tables) because an entire index block must be allocated for each file, regardless of how many data blocks the file contains. This leads to questions of how big the index block should be, and how it should be implemented. There are several approaches:

• **Linked Scheme** - An index block is one disk block, which can be read and written in a single disk operation. The first index block contains some header information, the first N block addresses, and if necessary a pointer to additional linked index blocks.

• **Multi-Level Index** - The first index block contains a set of pointers to secondary index blocks, which in turn contain pointers to the actual data blocks.

• **Combined Scheme** - This is the scheme used in UNIX inodes, in which the first 12 or so data block pointers are stored directly in the inode, and then singly, doubly, and triply indirect pointers provide access to more data blocks as needed. The advantage of this scheme is that for small files (which many are), the data blocks are readily accessible (up to 48K with 4K block sizes); files up to about 4144K (using 4K blocks) are accessible with only a single indirect block (which can be cached), and huge files are still accessible using a relatively small number of disk accesses (larger in theory than can be addressed by a 32-bit address, which is why some systems have moved to 64-bit file pointers.)

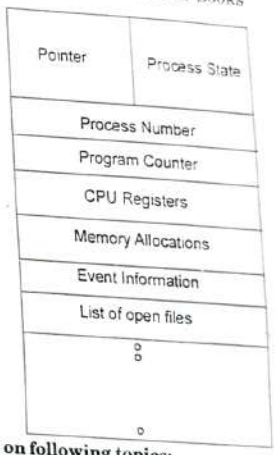
Q.7. (*b*) Explain the structure of process control block? **(6)**

Ans. Each process is represented in the operating system by a process control block (PCB) also called a task control block. PCB is the data structure used by the operating system. Operating system groups all information that needs about particular process.

PCB contains many pieces of information associated with a specific process which are described below.

S.N.	Information & Description
1.	Pointer: Pointer points to another process control block. Pointer is used for maintaining the scheduling list.
2.	Process State: Process state may be new, ready, running, waiting and so on.
3.	Program Counter: Program Counter indicates the address of the next instruction to be executed for this process.
4	CPU registers: CPU registers include general purpose register, stack pointers, index registers and accumulators etc. number of register and type of register totally depends upon the computer architecture.
5.	Memory management information: This information may include the value of base and limit registers, the page tables, or the segment tables depending on the memory system used by the operating system.This information is useful for deallocating the memory when the process terminates.
6.	Accounting information: This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc.

Process control block includes CPU scheduling, I/O resource management, file management information etc.. The PCB serves as the repository for any information which can vary from process to process. Loader/linker sets flags and registers when a process is created. If that process get suspended, the contents of the registers are saved on a stack and the pointer to the particular stack frame is stored in the PCB. By this technique, the hardware state can be restored so that the process can be scheduled to run again



Q.8. Write short notes on following topics: **(2 × 6 = 12)**

Q 8. (*a*). General Model of File system:

Ans. A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say "I have two filesystems" meaning one has two partitions on which one stores files, or that one is using the "extended filesystem", meaning the type of the filesystem.

The difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a filesystem.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are superblock, inode , data block, directory block , and indirection block. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

Q.8. (*b*). Dedicated, Shared and virtual devices.

Ans. Dedicated Devices: Are assigned to only one job at a time. – They serve that job for the entire time the job is active or until it releases them. – Some devices demand this kind of allocation scheme, because it would be awkward to let several users share them. • Example: tape drives, printers, and plotters.

Disadvantages

• They must be allocated to a single user for the duration of a job's execution, which can be quite inefficient, even though the device is not used 100% of the time.

Shared Devices: Can be assigned to several processes. – For example – a disk (DASD) can be shared by several processes at the same time by interleaving their requests;

• This interleaving must be carefully controlled by the Device Manager – All conflicts must be resolved based on predetermined policies.

• **Virtual Devices** – A combination of the first two types; – They're dedicated devices that have been transformed into shared devices.

• **Example: printer** – Converted into a shareable device through a spooling program that reroutes all print requests to a disk. – Only when all of a job's output is complete, and the printer is ready to print out the entire document, is the output sent to the printer for printing. – Because disks are shareable devices, this technique can convert one printer into several virtual printers, thus improving both its performance and use

Q.8. (c) Process scheduling algorithms:

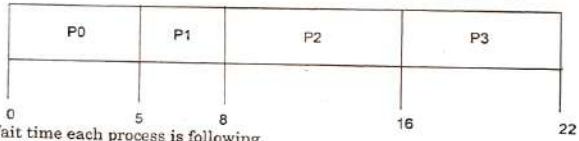
Ans. First Come First Serve (FCFS)

•Jobs are executed on first come, first serve basis.

• Easy to understand and implement.

•Poor in performance as average wait time is high.

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



Wait time each process is following

Process	Wait Time: Service Time-Arrival Time
Po	9 – 0 = 9
P1	6 – 1 = 5
P2	14 – 2 = 12
P3	0 – 0 = 0

Average wait Time: (9 + 5 + 12 + 0)/4 = 6.5

Round Robin Schedulin

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

• Once a process is executed for given time period. Process is preempted and other process executes for given time period.

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

Average wait Time: (0 + 4 + 6 + 13)/4 = 5.55

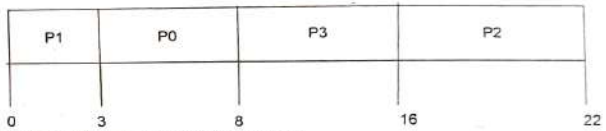
Shortest job First (SJF)

• Best approach to minimize waiting time.

• Imposible to implement

• Processer should know in advance how much time process will take.

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



Wait time of each process is following:

Process	Wait Time: Service Time – Arrival Time
P0	3 – 0 = 3
P1	0 – 0 = 0
P2	16 – 2 = 14
P3	8 – 3 = 5

Average Wait Time : (3 + 0 + 14 + 5)/4 = 5.50

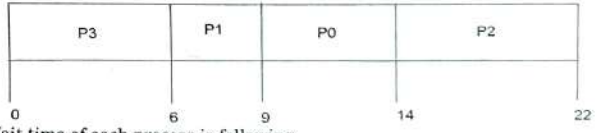
Priority Based Scheduling

• Each process is assigned a priority. Process with highest priority is to be executed first and so on.

• Processes with same priority are executed on first come first serve basis.

• Priority can be decided based on memory requirements, time requirements or any other resource requirement.

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



Wait time of each process is following

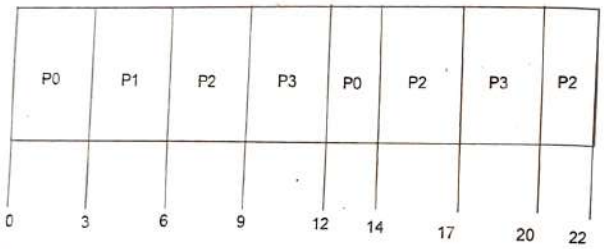
Process	Wait Time : Service Time – Arrival Time
P0	9 – 0 = 9
P1	6 – 1 = 5
P2	14 – 2 = 12
P3	0 – 0 = 0

Average Wait Time: (9 + 5 + 12 + 0)/4 = 6.5

Round Robin Scheduling

- Each process is provided a fix time to execute called quantum.
- Once a process is executed for given time period. Process is preempted and other process executes for given time period.
- Context switching is used to save states of preempted process.

Quantum = 3



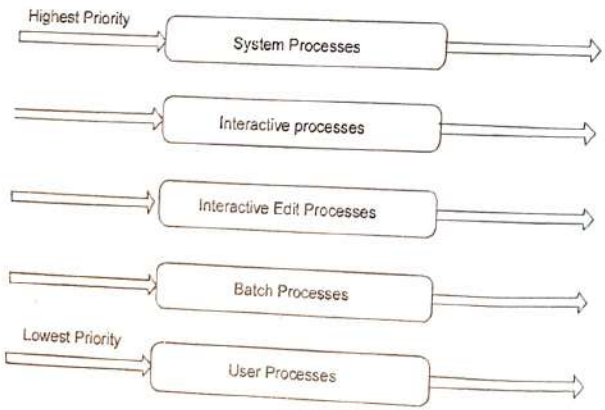
Wait time of each process is following:

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

Average Wait Time: (9+2+12+11)/4 = 8.5

Multi Queue Scheduling

- Multiple queues are maintained for processes.
- Each queue can have its own scheduling algorthms.
- Priorties are assigned to each queue.



END TERM EXAMINATION [JUNE 2015]
SECOND SEMESTER [MCA]
OPERATING SYSTEM [MCA–106]

Time: 3 hours M.M. : 60

Note: *Attempt any five questions including Q.no.1 which is compulsory.*

Q.1 Explain in brief: (2 × 10 = 0)

Q.1. (*a*) Define Semophare.

Ans. Semaphore: A semaphore is a protected variable whose value can be accessed and altered only by the operations P and V and initialization operation called 'Semaphoiinitislize'. Binary Semaphores can assume only the value 0 or the value 1 counting semaphores also called general semaphores can assume only nonnegative values.The P (or wait or sleep or down) operation on semaphores S, written as P(S) or wait (S), operates as follows:

P(S): IF S > 0

THEN S := S - 1

ELSE (wait on S)

The V (or signal or wakeup or up) operation on semaphore S, written as V(S) or signal (S), operates as follows:

V(S): IF (one or more process are waiting on S)

THEN (let one of these processes proceed)

ELSE S := S +1

Q.1. (*b*) Differentiate between types of real time systems.

Ans. Types of real time systems: Real time system is defines as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. Real time processing is always on line whereas on line system need not be real time. The time taken by the system to respond to an input and display of required updated information is termed as response time. So in this method response time is very less as compared to the online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. Real-time operating system has well-defined, fixed time constraints otherwise system will fail.For example Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-appliance controllers, Air traffic control system etc.

There are two types of real-time operating systems.

Hard real-time systems: Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems secondary storage is limited or missing with data stored in ROM. In these systems virtual memory is almost never found.

Soft real-time systems: Soft real time systems are less restrictive. Critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems.For example, Multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers etc.

Q.1. (c) List the responsiblilites of the file manager.

Ans. A file manager or file browser is a computer program that provides a user interface to manage files and folders. The most common operations performed on files or groups of files include creating, opening (e.g. viewing, playing, editing or printing), renaming, moving or copying, deleting and searching for files, as well as modifying file attributes, properties and file permissions. Folders and files may be displayed in a hierarchical tree based on their directory structure. Some file managers contain features inspired by web browsers, including forward and back navigational buttons.

Some file managers provide network connectivity via protocols, such as FTP, NFS, SMB or WebDAV. This is achieved by allowing the user to browse for a file server (connecting and accessing the server's file system like a local file system) or by providing its own full client implementations for file server protocols.

Responsibilities

• Four tasks

– Keep track of where each file is stored

– Implement a policy that will:

•Determine where and how files are stored

•Efficiently use available storage space

• Provide efficient file access

–Allocate each file when a user has been cleared for access to it, then record its use

– File de-allocation

• File returned to storage

•Communicate file availability to others waiting for it

• File Manager's Policy determines:

– File storage location

– System and user access files via device-independent commands

• Who will have access to which file. This depends on two factors:

• Flexibility of access to information (Factor 1)

– Shared files

– Providing distributed access

– Allowing users to browse public directories

• Subsequent protection (Factor 2)

–Protect files against system malfunctions

– Security checks

• Account numbers, passwords, lockwords

• File allocation

– Activate secondary storage device, load file into memory, update records

• File de-allocation

–Update file tables, rewrite file (if revised), notify waiting processes of file availability

Q.1. (d) Define external fragmentation and its occurrence?

Ans. External Fragmentation

External fragmentation: External fragmentation arises when free memory is separated into small blocks and is interspersed by allocated memory. It is a weakness

of certain storage allocation algorithms, when they fail to order memory used by programs efficiently. The result is that, although free storage is available, it is effectively unusable because it is divided into pieces that are too small individually to satisfy the demands of the application. The term "external" refers to the fact that the unusable storage is outside the allocated regions.

For example, consider a situation wherein a program allocates 3 continuous blocks of memory and then frees the middle block. The memory allocator can use this free block of memory for future allocations. However, it cannot use this block if the memory to be allocated is larger in size than this free block.

External fragmentation also occurs in file systems as many files of different sizes are created, change size, and are deleted. The effect is even worse if a file which is divided into many small pieces is deleted, because this leaves similarly small regions of free spaces.

0x0000	0x1000	0x2000	0x3000	0x4000	0x5000	Comments
						Start with all memory available for allocation.
A	B	C				Allocated three blocks A, B, and C, of size 0x1000.
A		C				Freed block B. Notice that the memory that B used cannot be included for an allocation larger than B's size.

Q.1. (e) What are the reasons for providing process cooperation.

Ans. Process Cooperation

Cooperation: Concurrent processes executing in the operating system allows for the processes to *cooperate* (both mutually or destructively) with other processes. Processes are cooperating if they can affect each other. The simplest example of how this can happen is where two processes are using the same file. One process may be writing to a file, while another process is reading from the file; so, what is being read may be affected by what is being written. Processes cooperate by sharing data. Cooperation is important for several reasons:

Cooperation between processes requires mechanisms that allow processes to *communicate* data between each other and *synchronize* their actions so they do not harmfully interfere with each other. The purpose of this note is to consider ways that processes can communicate data with each other, called Interprocess Communication (IPC). Another note will discuss process synchronization, and in particular, the most important means of synchronizing activity, the use of semaphores.

Q.1. (f) Define thrashing and its disadvantages.

Ans.Thrashing: In computer science, thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data in memory for data on disk, to the exclusion of most application-level processing.[1] This causes the performance of the computer to degrade or collapse. The situation may continue indefinitely until the underlying cause is addressed. The term is also used for various similar phenomena, particularly movement between other levels of the memory hierarchy, where a process progresses slowly because significant time is being spent acquiring resources.

In virtual memory systems, thrashing may be caused by programs or workloads that present insufficient locality of reference: if the working set of a program or a workload

cannot be effectively held within physical memory, then constant data swapping, *i.e.*, thrashing, may occur. The term was first used during the tape operating system days to describe the sound the tapes made when data was being rapidly written to and read. An example of this sort of situation occurred on the IBM System/370 series mainframe computer, in which a particular instruction could consist of an execute instruction (which crosses a page boundary) that points to a move instruction (which itself also crosses a page boundary), targeting a move of data from a source that crosses a page boundary, to a target of data that also crosses a page boundary. The total number of pages thus being used by this particular instruction is eight, and all eight pages must be present in memory at the same time. If the operating system allocates fewer than eight pages of actual memory, when it attempts to swap out some part of the instruction or data to bring in the remainder, the instruction will again page fault, and it will thrash on every attempt to restart the failing instruction.

To resolve thrashing due to excessive paging, a user can do any of the following:

- Increase the amount of RAM in the computer.
- Decrease the number of programs being run on the computer.
- Replace programs that are memory-heavy with equivalents that use less memory.
- Assign working priorities to programs, i.e. low, normal, high.
- Improve spatial locality by replacing loops like:

Q.1. (*g*) Define the role of channels and control units.

Ans. Role of Channel and control unit: I/O Channel — keeps up with I/O requests from CPU and pass them down the line to appropriate control unit. – Programmable units placed between CPU and control unit. – Synchronize fast speed of CPU with slow speed of the I/O device. – Make it possible to overlap I/O operations with processor operations so the CPU and I/O can process concurrently.

I/O control unit interprets signal sent by channel. – One signal for each function. • At start of I/O command, info passed from CPU to channel: – I/O command (READ, WRITE, REWIND, etc.) – Channel number – Address of physical record to be transferred (from or to secondary storage) – Starting address of a memory buffer from which or into which record is to be transferred

Q.1. (*h*) What is safe state and unsafe state?

Ans. Safe State: A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources there are enough for the next to fini

Unsafe state: A state that may allow deadlock

Q.1. (*i*) What is Overlay?

Ans. Overlay: In a general computing sense, overlaying means "the process of transferring a block of program code or other data into internal memory, replacing what is already stored".Overlaying is a programming method that allows programs to be larger than the computer's main memory. An embedded system would normally use overlays because of the limitation of physical memory, which is internal memory for a system-on-chip and the lack of virtual memory facilities.

Q.1. (*j*) Is it possible to have a deadlock involving only one process? State your answer.

Ans. It is not possible to have a deadlock involving only one single process. The **deadlock** involves a circular "hold-and-wait" condition between two or more **processes**, so **"one" process** cannot hold a resource, yet be waiting for another resource that it is holding.

Q.2. (*a*) Discuss the following Systems (10)

Ans. Multiprogramming Systems.

1.Multiprogramming is a rudimentary form of parallel processing in which several programs are run at the same time on a uniprocessor. Since there is only one processor, there can be no true simultaneous execution of different programs. Instead, the operating system executes part of one program, then part of another, and so on. To the user it appears that all programs are executing at the same time.

2.If the machine has the capability of causing an interrupt after a specified time interval, then the operating system will execute each program for a given length of time, regain control, and then execute another program for a given length of time, and so on. In the absence of this mechanism, the operating system has no choice but to begin to execute a program with the expectation, but not the certainty, that the program will eventually return control to the operating system.

3.If the machine has the capability of protecting memory, then a bug in one program is less likely to interfere with the execution of other programs. In a system without memory protection, one program can change the contents of storage assigned to other programs or even the storage assigned to the operating system. The resulting system crashes are not only disruptive, they may be very difficult to debug since it may not be obvious which of several programs is at fault.

Q.2. (*b*) Distributed systems.

Ans. Distributed systems: A distributed operating system is a software over a collection of independent, networked, communicating, and physically separate computational nodes.[1] Each individual node holds a specific software subset of the global aggregate operating system. Each subset is a composite of two distinct service provisioners. The first is a ubiquitous minimalkernel, or microkernel, that directly controls that node's hardware. Second is a higher-level collection of *system management components* that coordinate the node's individual and collaborative activities. These components abstract microkernel functions and support user applications. The microkernel and the management components collection work together. They support the system's goal of integrating multiple resources and processing functionality into an efficient and stable system. This seamless integration of individual nodes into a global system is referred to as *transparency*, or *single system image*; describing the illusion provided to users of the global system's appearance as a single computational entity.

Distributed Operating System is a model where distributed applications are running on multiplecomputers linked by communications. A distributed operating system is an extension of the network operating system that supports higher levels of communication and integration of the machines on the network.

This system looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs).

Q.2. (*c*) Time sharing system.

Ans. Time Sharing: Time sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the sametime. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

In computing, time-sharing is the sharing of a computing resource among many users by means of multiprogramming and multi-tasking at the same time.

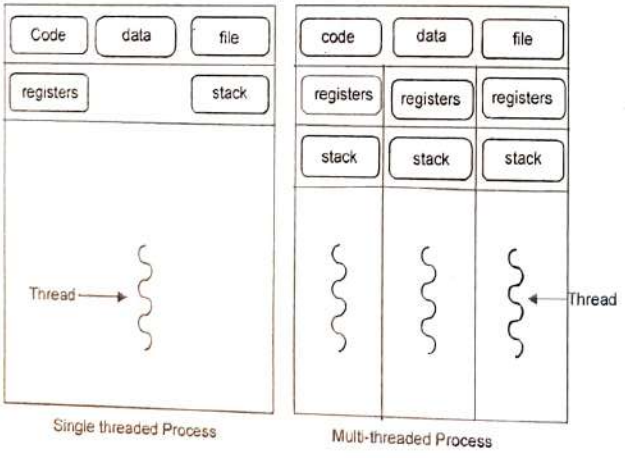
Its introduction in the 1960s by students and professors at Dartmouth College, and emergence as the prominent model of computing in the 1970s, represented a major technological shift in the history of computing.

By allowing a large number of users to interact concurrently with a single computer, time-sharing dramatically lowered the cost of providing computing capability, made it possible for individuals and organizations to use a computer without owning one, and promoted the interactive use of computers and the development of new interactive applications.

Q.3. What is a thread and what are the advantages of threads. Explain multithreading models in detail. **(10)**

Ans. A thread is a flow of execution through the process code, with its own program counter, system registers and stack. A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Following figure shows the working of the single and multithreaded processes.



Advantages of Thread

- Thread minimize context switching time.
- Use of threads provides concurrency within a process.

- Efficient communication.
- Economy- It is more economical to create and context switch threads.
- Utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread: Threads are implemented in following two ways

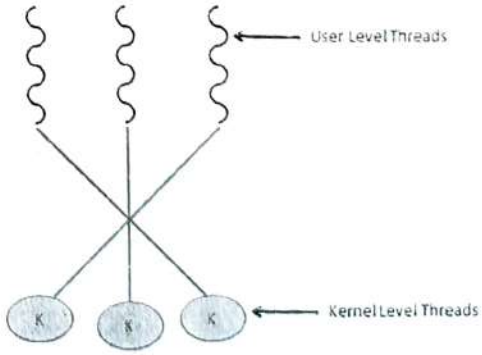
- User Level Threads — User managed threads
- Kernel Level Threads — Operating System managed threads acting on kernel, an operating system core.

Multithreading Models: Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

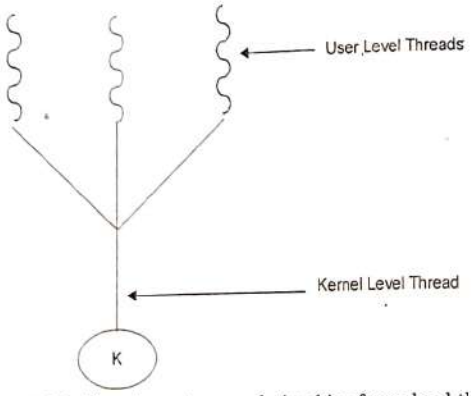
Many to Many Model: In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.

Following diagram shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.



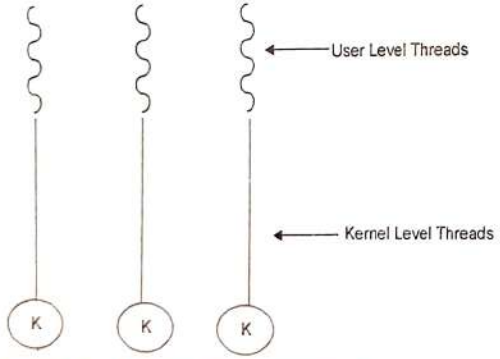
Many to One Model: Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



One to One Model: There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Q.4. Discuss in detail the File allocation methods. **(10)**

Ans. Contiguous Allocation: The contiguous allocation method requires each file to occupy a set of contiguous address on the disk. Disk addresses define a linear ordering on the disk. Notice that, with this ordering, accessing block b+1 after block b normally requires no head movement. When head movement is needed (from the last sector of one cylinder to the first sector of the next cylinder), it is only one track. Thus, the number of disk seeks required for accessing contiguous allocated files in minimal, as is seek time when a seek is finally needed. Contiguous allocation of a file is defined by the disk address and the length of the first block. If the file is n blocks long, and starts at location b, then it occupies blocks b, b + 1, b + 2, ..., b + n – 1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

The difficulty with contiguous allocation is finding space for a new file. If the file to be created is n blocks long, then the OS must search for n free contiguous blocks. First-fit, best-fit, and worst-fit strategies are the most common strategies used to select a free hole from the set of available holes. Simulations have shown that both first-fit and best-fit are better than worst-fit in terms of both time storage utilization. Neither first-fit nor best-fit is clearly best in terms of storage utilization, but first-fit is generally faster.

These algorithms also suffer from external fragmentation. As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists when enough total disk space exists to satisfy a request, but this space not contiguous; storage is fragmented into a large number of small holes.

Another problem with contiguous allocation is determining how much disk space is needed for a file. When the file is created, the total amount of space it will need must be known and allocated. How does the creator (program or person) know the size of the file to be created. In some cases, this determination may be fairly simple (e.g. copying an existing file), but in general the size of an output file may be difficult to estimate.

Linked Allocation: The problems in contiguous allocation can be traced directly to the requirement that the spaces be allocated contiguously and that the files that need these spaces are of different sizes. These requirements can be avoided by using linked allocation.

In linked allocation, each file is a linked list of disk blocks. The directory contains a pointer to the first and (optionally the last) block of the file. For example, a file of 5 blocks which starts at block 4, might continue at block 7, then block 16, block 10, and finally block 27. Each block contains a pointer to the next block and the last block contains a NIL pointer. The value -1 may be used for NIL to differentiate it from block 0.

With linked allocation, each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil (the end-of-list pointer value) to signify an empty file. A write to a file removes the first free block and writes to that block. This new block is then linked to the end of the file. To read a file, the pointers are just followed from block to block.

There is no external fragmentation with linked allocation. Any free block can be used to satisfy a request. Notice also that there is no need to declare the size of a file when that file is created. A file can continue to grow as long as there are free blocks.

Linked allocation, does have disadvantages, however. The major problem is that it is inefficient to support direct-access; it is effective only for sequential-access files. To find the ith block of a file, it must start at the beginning of that file and follow the pointers until the ith block is reached. Note that each access to a pointer requires a disk read.

Another severe problem is reliability. A bug in OS or disk hardware failure might result in pointers being lost and damaged. The effect of which could be picking up a wrong pointer and linking it to a free block or into another file.

Indexed Allocation: The indexed allocation method is the solution to the problem of both contiguous and linked allocation. This is done by bringing all the pointers together into one location called the index block. Of course, the index block will occupy some space and thus could be considered as an overhead of the method.

In indexed allocation, each file has its own index block, which is an array of disk sector of addresses. The ith entry in the index block points to the ith sector of the file. The directory contains the address of the index block of a file. To read the ith sector of the file, the pointer in the ith index block entry is read to find the desired sector.

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

Q.5. Discuss the various parameters /protocol for recovery deadlock. Why is deadlock avoidance more popular than deadlock prevention.? **(10)**

Ans. In concurrent programming, a deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

In a transactional database, a deadlock happens when two processes each within its own transaction updates two rows of information but in the opposite order. For example, process *A* updates row *1* then row *2* in the exact timeframe that process *B* updates row *2* then row *1*. Process *A* can't finish updating row *2* until process *B* is finished, but process *B* cannot finish updating row *1* until process *A* is finished. No matter how much time is allowed to pass, this situation will never resolve itself and because of this, database management systems will typically kill the transaction of the process that has done the least amount of work.

In an operating system, a deadlock is a situation which occurs when a process or thread enters a waiting state because a resourcerequested is being held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

Deadlock is a common problem in multiprocessing systems, parallel computing and distributed systems, where software and hardwarelocks are used to handle shared resources and implement process synchronization.

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

1.Mutual exclusion: at least one resource must be held in a non-shareable mode. Only one process can use the resource at any given instant of time.

2.Hold and wait or resource holding: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.

3.No preemption: a resource can be released only voluntarily by the process holding it.

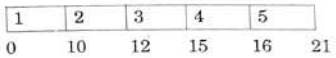
4.Circular wait: a process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, $P = \{P_1, P_2, ..., P_N\}$, such that P_1 is waiting for a resource held by P_2, P_2 is waiting for a resource held by P_3 and so on until $P_N$is waiting for a resource held by P_1.

Q.6. For the following information:

Job#	Arrival time	CPU Cycle
1	0	10
2	1	2
3	2	3
4	3	1
5	4	5

Find the turn around time for each job using FCFS, SJF and Round Robin (Time quantum = 2).

Ans. FCFS, Grant chart **(10)**

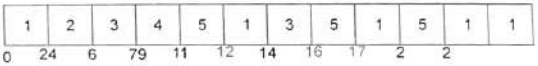


Total turn around time: 10+ 12+15+16+21 = 74

SJF

Total turn around time: 1 + 3 + 6 + 11 + 21 = 42

Round Robin: lets assume time quantum is 2



Total turn around time: 0+2+4+6+7+9+11+12+14 + 16 + 17 + 2 + 2 = 92

Q.7. N Processes share M resource units that can be reserved and released only one at time. The maximum need of each process does not exceed M and sum of all maximum is less than M+N. Show that deadlock cannot occur. **(10)**

Ans: Lets assume Max <m+n

Max >= 1 for all i

Proof : Need = Max – Allocation

If there exists a deadlock state then:

Allocation = M

To get Need + Allocation = Max <m+n

Need+m<m+n

This implies that there exists a process Pi such that Need = 0. Since Max >= 1 it follows that Pi has at least one resource that it can release. Hence the system cannot be in a deadlock state.

Q.8. Write short note on any two: **(5×2 = 10)**

Q.8.(*a*) Producer consumer problem:

Ans. The **producer–consumer problem** (also known as the **bounded-buffer problem**) is a classic example of a multi-process synchronization problem. The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate a piece of data, put it into the buffer and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer) one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer.

The solution for the producer is to either go to sleep or discard data if the buffer is full. The next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. In the same way, the consumer can go to sleep if it finds the buffer to be empty. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened. The problem can also be generalized to have multiple producers and consumers.

Semaphores solve the problem of lost wakeup calls. In the solution below we use two semaphores, fillCount and emptyCount, to solve the problem. fillCount is the number of items already in the buffer and available to be read, while emptyCount is the number of available spaces in the buffer where items could be written. fillCount is incremented and emptyCount decremented when a new item is put into the buffer. If the producer tries to decrement emptyCount when its value is zero, the producer is put to sleep. The next time an item is consumed, emptyCount is incremented and the producer wakes up. The consumer works analogously.

```
semaphorefillCount=0;// items produced
semaphoreemptyCount=BUFFER_SIZE;// remaining space
procedureproducer(){
while(true){
item=produceItem();
down(emptyCount);
putItemIntoBuffer(item);
up(fillCount);
}
}
procedureconsumer(){
while(true){
down(fillCount);
item=removeItemFromBuffer();
up(emptyCount);
consumeItem(item);
}
}
```

The solution above works fine when there is only one producer and consumer. With multiple producers sharing the same memory space for the item buffer, or multiple consumers sharing the same memory space, this solution contains a serious race condition that could result in two or more processes reading or writing into the same slot at the same time. To understand how this is possible, imagine how the procedure putItemIntoBuffer() can be implemented. It could contain two actions, one determining the next available slot and the other writing into it. If the procedure can be executed concurrently by multiple producers, then the following scenario is possible:

1.Two producers decrement emptyCount

2. One of the producers determines the next empty slot in the buffer

3. Second producer determines the next empty slot and gets the same result as the first producer

4.Both producers write into the same slot

To overcome this problem, we need a way to make sure that only one producer is executing putItemIntoBuffer() at a time. In other words, we need a way to execute acritical section with mutual exclusion. The solution for multiple producers and consumers is shown below.

```
semaphoremutex=1;
semaphorefillCount=0;
```



```
semaphoreemptyCount=BUFFER_SIZE;
procedureproducer(){
while(true){
item=produceItem();
down(emptyCount);
down(mutex);
putItemIntoBuffer(item);
up(mutex);
up(fillCount);
}
}
procedureconsumer(){
while(true){
down(fillCount);
down(mutex);
item=removeItemFromBuffer();
up(mutex);
up(emptyCount);
consumeItem(item);
}
}
```

Notice that the order in which different semaphores are incremented or decremented is essential: changing the order might result in a deadlock

Q.8. (b) Process Control block:

Ans. Process Control Block (PCB, also called Task Controlling Block, process table, Task Struct, or Switchframe) is a data structure in the operating system kernel containing the information needed to manage a particular process. The PCB is "the manifestation of a process in an operating system".

In modern sophisticated multitasking systems, the PCB stores many different items of data, all needed for correct and efficient process management. Though the details of these structures are obviously system-dependent, we can identify some very common parts, and classify them in three main categories:

- Process identification data
- Processor state data
- Process control data

Process identification data always include a unique identifier for the process (almost invariably an integer number) and, in a multiuser-multitasking system, data like the identifier of the parent process, user identifier, user group identifier, etc. The process id is particularly relevant, since it is often used to cross-reference the OS tables defined above, e.g. allowing to identify which process is using which I/O devices, or memory areas.

Processor state data are those pieces of information that define the status of a process when it is suspended, allowing the OS to restart it later and still execute correctly. This always includes the content of the CPU general-purpose registers, the CPU process status word, stack and frame pointers etc. During context switch, the running process is stopped and another process is given a chance to run. The kernel must stop the execution

of the running process, copy out the values in hardware registers to its PCB, and update the hardware registers with the values from the PCB of the new process.

Process control information is used by the OS to manage the process itself. This includes:

- The process scheduling state, e.g. in terms of "ready", "suspended", etc., and other scheduling information as well, like a priority value, the amount of time elapsed since the process gained control of the CPU or since it was suspended. Also, in case of a suspended process, event identification data must be recorded for the event the process is waiting for.
- Process structuring information: process's children id's, or the id's of other processes related to the current one in some functional way, which may be represented as a queue, a ring or other data structures.
- Interprocess communication information: various flags, signals and messages associated with the communication among independent processes may be stored in the PCB.
- Process privileges, in terms of allowed/disallowed access to system resources.
- Process state: State may enter into new, ready, running, waiting, dead depending on CPU scheduling.
- Processor No: a unique identification number for each process in the operating system.
- Program counter: a pointer to the address of the next instruction to be executed for this process.
- CPU registers: indicates various register set of CPU where process need to be stored for execution for running state.
- CPU scheduling information: indicates the information of a process with which it uses the CPU time through scheduling.
- Memory management information: includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- Accounting information: includes the amount of CPU used for process execution, time limits, execution ID etc.
- IO status information: includes a list of I/O devices allocated to the process.

Q.8. (c) Intrusion deletion.

Ans. An intrusion detection system (IDS) or intrusion removal system is a device or software application that monitors network or system activities for malicious activities or policy violations and produces electronic reports to a management station. IDS come in a variety of "flavors" and approach the goal of detecting suspicious traffic in different ways. There are network based (NIDS) and host based (HIDS) intrusion detection systems. NIDS is a network security system focusing on the attacks that come from the inside of the network (authorized users). Some systems may attempt to stop an intrusion attempt but this is neither required nor expected of a monitoring system. Intrusion detection and prevention systems (IDPS) are primarily focused on identifying possible incidents, logging information about them, and reporting attempts. In addition, organizations use IDPSes for other purposes, such as identifying problems with security policies, documenting existing threats and deterring individuals from violating security policies. IDPSes have become a necessary addition to the security infrastructure of nearly every organization.

END TERM EXAMINATION [MAY-JUNE 2016]
SECOND SEMESTER [MCA]
OPERATING SYSTEM [MCA–106]

Time : 3 hrs. **M.M. : 75**

Note: *Attempt any five questions from the following:*

Q.1. (a) Define the real time system. **(2×10=20)**

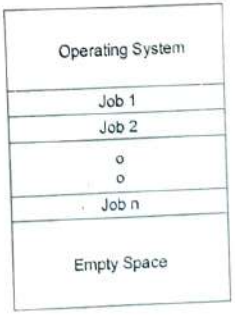
Ans. A real-time system is a computer system that requires not only that the computing results be "correct" but also that the results be produced within a specified deadline period. Results produced after the deadline has passed even if correct-may be of no real value. To illustrate, consider an autonomous robot that delivers mail in an office complex. If its vision-control system identifies a wall *after* the robot has walked into it, despite correctly identifying the wall, the system has not met its requirement. Contrast this timing requirement with the much less strict demands of other systems. In an interactive desktop computer system, it is desirable to provide a quick response time to the interactive user, but it is not mandatory to do so. Some systems -such as a batch-processing system-may have no timing requirements whatsoever.

Real-time systems executing on traditional computer hardware are used in a wide range of applications. In addition, many real-time systems are embedded in "specialized devices," such as ordinary home appliances (for example, microwave ovens and dishwashers), consumer digital devices (for exarnple, cameras and MP3 players), and communication devices.

Q.1. (b) Explain multiprogramming.

Ans. Sharing the processor, when two or more programs reside in memory at the same time, is referred as multiprogramming. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.



An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.

• Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensures that the CPU is never idle, unless there are no jobs to process.

Advantages

• High and efficient CPU utilization.

• User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

• CPU scheduling is required.

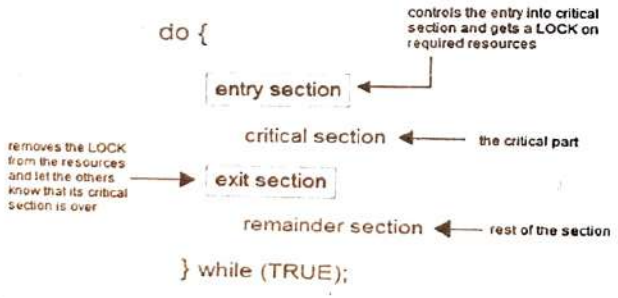
• To accommodate many jobs in memory, memory management is required.

Q.1. (c) What are the synchronization basic concepts.

Ans. Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes. Process Synchronization was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



Q.1. (d) Describe process scheduling.

Ans. The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Schedulers fell into one of the two general categories :

• **Non pre-emptive scheduling**. When the currently executing process gives up the CPU voluntarily.

• **Pre-emptive scheduling**. When the operating system decides to favour another process, pre-empting the currently executing process.

Process Scheduling Queues

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

The Operating System maintains the following important process scheduling queues "

• **Job queue**– This queue keeps all the processes in the system.

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

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

Q.1. (e) What is the security criteria for scheduling?

Ans. Scheduling Criteria:

There are several different criteria to consider when trying to select the "best" scheduling algorithm for a particular situation and environment, including:

• **CPU utilization** - Ideally the CPU would be busy 100% of the time, so as to waste 0 CPU cycles. On a real system CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

• **Throughput** - Number of processes completed per unit time. May range from 10 / second to 1 / hour depending on the specific processes.

• **Turnaround time** - Time required for a particular process to complete, from submission time to completion. (Wall clock time.)

• **Waiting time** - How much time processes spend in the ready queue waiting their turn to get on the CPU.

• (**Load average** - The average number of processes sitting in the ready queue waiting their turn to get into the CPU. Reported in 1-minute, 5-minute, and 15-minute averages by "uptime" and "who".)

• **Response time** - The time taken in an interactive program from the issuance of a command to the ***commence*** of a response to that command.

• In general one wants to optimize the average value of a criteria (Maximize CPU utilization and throughput, and minimize all the others.) However some times one wants to do something different, such as to minimize the maximum response time.

• Sometimes it is most desirable to minimize the ***variance*** of a criteria than the actual value. I.e. users are more accepting of a consistent predictable system than an inconsistent one, even if it is a little bit slower.

Q.1. (f) Write about contiguous allocation.

Ans. Contiguous Allocation requires that each file occupy a set of contiguous blocks on disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessil1.g the disk, accessing block $b + 1$ after block b normally requires no head movement. When head movement is needed (from the last sector of one cylil1.der to the first sector of the next cylinder), the head need only move from one track to the next. Thus, the number of disk seeks required for accessing contiguously allocated files is minimal, as is seek time when a seek is finally needed. The IBM VM/CMS operatil1.g system uses contiguous allocation because it provides such good performance. Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks $b, b + 1, b + 2, \ldots, b + n - 1$. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

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

Q.1. (g) Discuss the device management technique.

Ans. Dedicated Devices

Ans. A dedicated device is allocated to a job for the job's entire duration. Some devices lend themselves to this form of allocation. It is difficult, for example, to share a card reader, tape, or printer. If several users were to use the printer at the same time, would they cut up their appropriate output and paste it together? Unfortunately, dedicated assignment may be inefficient if the job does not fully and continually utilize the device. The other techniques, shared and virtual, are usually preferred whenever they are applicable.

Shared Devices

Some devices such, as disks, drums, and most other Direct Access Storage Devices (DASD) may be shared concurrently by several processes. Several processes can read from a single disk at essentially the same time. The management of a shared device can become quite complicated, particularly if utmost efficiency is desired. For example, if two processes simultaneously request a Read from Disk A, some mechanism must be employed to determine which request should be handled first. This may be done partially by software (the I/O scheduler and traffic controller) or entirely by hardware (as in some computers with very sophisticated channels and control units).

Policy for establishing which process request is to be satisfied first might be based on (1) a priority list or (2) the objective of achieving improved system out-put (for example, by choosing whichever request is nearest to the current position of the read heads of the disk).

Virtual Devices

Some devices that would normally have to be dedicated (e.g. card readers) may be converted into shared devices through techniques such as spooling link. For example. A SPOOLing program can read and copy all card input onto a disk at high speed. Later, when a process tries to read a card, the SPOOLing program intercepts the request and converts it to a read from the disk. Since a disk may be easily shared by several users, we have converted a dedicated device to a shared device, changing one card reader into many "virtual" card readers. This technique is equally applicable to a large number of peripheral devices, such as teletypes, printers, and most dedicated slow input/output devices.

Q.1. (h) What is disk reliability?

Ans. Although we often think good peljormance means high speed, another important aspect of performance is reliability. If we try to read some data and are unable to do so because of a drive or media failure, for all practical purposes the access time is infinitely long and the bandwidth is infinitely small. So it is important to understand the reliability of removable media. Removable n1.agnetic disks are somewhat less reliable than are fixed hard disks, because they are more likely to be exposed to harmful environmental conditions such as dust, large changes in temperature and humidity, and mechanical forces such as shock and bending. Optical disks are considered very reliable, because the layer that stores the bits is protected by a transparent plastic or glass layer. The reliability of magnetic tape varies widely, depending on the kind of drive. Some inexpensive drives wear out tapes after a few dozen uses; other drives are gentle enough to allow millions of reuses. By comparison with a magnetic-disk head,

the head in a magnetic-tape drive is a weak spot. A disk head flies above the media, but a tape head is in close contact with the tape. The scrubbing action of the tape can wear out the head after a few thousands or tens of thousands of hours.

Q.1. (i) What is shared devices?

Ans. Some devices such as disks, drams, and most other Direct Access Storage Devices (DASD) may be shared concurrently by several processes. Several processes can read from a single disk at essentially the same time. The management of a shared device can become quite complicated, particularly if utmost efficiency is desired. For example, if two processes simultaneously request a Read from Disk A, some mechanism must be employed to determine which request should be handled first. This may be done partially by software (the I/O scheduler and traffic controller) or entirely by hardware (as in some computers with very sophisticated channels and control units).

Policy for establishing which process request is to be satisfied first might be based on (1) a priority list or (2) the objective of achieving improved system out-put (for example, by choosing whichever request is nearest to the current position of the read heads of the disk).

Q.1. (j) Explain system interfaces.

Ans. A user interface, consisting of the set of dials, knobs, operating system commands, graphical display formats, and other devices provided by a computer or a program to allow the user to communicate and use the computer or program. A graphical user interface (GUI) provides its user a more or less "picture-oriented" way to interact with technology. A GUI is usually a more satisfying or user-friendly interface to a computer system.

A programming interface, consisting of the set of statements, functions, options, and other ways of expressing program instructions and data provided by a program or language for a programmer to use.

The physical and logical arrangement supporting the attachment of any device to a connector or to another device.

UNIT-I

Q.2. (a) What is the concept of process scheduling? Differentiate with multiple process scheduling. **(4)**

Ans. The act of determining which process in the ready state should be moved to the running state is known as Process Scheduling. The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Schedulers fell into one of the two general categories:

- **Non pre-emptive scheduling.** When the currently executing process gives up the CPU voluntarily.
- **Pre-emptive scheduling.** When the operating system decides to favour another process, pre-empting the currently executing process.

When multiple processors are available, then the scheduling gets more complicated, because now there is more than one CPU which must be kept busy and in effective use at all times.

Load sharing revolves around balancing the load between multiple processors.

Multi-processor systems may be heterogeneous, (different kinds of CPUs), or homogenous, (all the same kind of CPU). Even in the latter case there may be special

scheduling constraints, such as devices which are connected via a private bus to only one of the CPUs.

Q.2. (*b*) Describe the following: **(6)**

(*i*) Real Time Scheduling

(*ii*) Algorithm Evaluation

(*iii*) Threads

Ans. (*i*) In a real-time computing, its correctness depends not only on what results are derived, but also on when these results are derived. In general, such a system contains some real-time tasks, associated with certain degree of urgency. They typically respond to some outside events that happen in real time (sensors), thus have to keep up with those events and respond in a timely fashion. A real-time OS can be deterministic if it performs operations at fixed, predetermined times, or within predetermined time intervals. When multiple processes are competing for resources, no system can be fully deterministic. Whether an OS can deterministically satisfy competing requests depends on the speed with which it can handle interrupts, as well as whether the system has enough capacity to meet all the requests within a predetermined time frame.

Ans.*(ii)* Criteria are often defined in terms of CPU utilization, response time, or throughput. To select an algorithm, we must first define the relative importance of these elements. Our criteria may include several measures, such as:

• Maximizing CPU utilization under the constraint that the maximum response time is 1 second.

• Maximizing throughput such that turnaround time is (on average) linearly proportional to total execution time.

Once the selection criteria have been defined, we want to evaluate the algorithms under consideration. We next describe the various evaluation methods we can use.

Ans. (*iii*) Thread is an execution unit which consists of its own program counter, a stack, and a set of registers. Threads are also known as Lightweight processes. Threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.

As each thread has its own independent resource for process execution, multiple processes can be executed parallel by increasing number of threads.

There are two types of threads :

• User Threads

• Kernel Threads

User threads, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.

Kernel threads are supported within the kernel of the OS itself. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

Q.3. (*a*) Differentiate between time sharing systems and real time systems. **(4)**

Ans. Time Sharing System: It is a form of multiprogrammed Operating system which operates in an interactive mode with a quick response time. The user types a request to the computer through a keyboard. The computer processes it and a response (if any) is displayed on the user's terminal. A time sharing system allows the many users to simultaneously share the computer resources. Since each action or command in a time-shared system take a very small fraction of time, only a little CPU time is needed for each user. As the CPU switches rapidly from one user to another user, each user is given impression that he has his own computer, while it is actually one computer.

Most time sharing system use time-slice (round robin) scheduling of CPU. In this approach, Programs are executed with rotating priority that increases during waiting and drops after the service is granted. In Order to prevent a program from monopolizing the processor, a program executing longer than the system defined time-slice in interrupted by the operating system and placed at the end of the queue of waiting program.

Real- time Operating System

It is another form of operating system which are used in environments where a large number of events mostly external to computer systems, must be accepted and processed in a short time or within certain deadlines. Examples of such applications are flight control, real time simulations etc. Real time systems are also frequently used in military application.

Real-time Operating Systems work towards providing immediate processing and also responding to user's commands in a very short time. Such an operating system is more commonly used in chemical industry for process control and scientific processing like airplane control and space vehicle control operations. Success of a real time system does not depend only on the correctness of the result, but also on the timeliness of the result. A correct answer obtained after the expiration of time limit is as bad as a wrong answer. Some examples of real time operating systems are HP-RT and VT- works.

A primary objective of real-time system is to provide quick response times. User convenience and resource utilization are of secondary concern to real-time system. In the real-time system each process is assigned a certain level of priority. The processor is normally allocated to the highest priority process among those which are ready to execute. Higher priority process usually pre-empt execution of lower priority processes. This form of scheduling called, priority based pre- emptive scheduling, is used by a majority of real-time systems.

Q.3. (*b*) View about the segmentation and paging. **(3)**

Ans. Segmentation: The user's view of the memory is not the same as the actual physical memory. The user's view is mapped onto physical memory. This mapping allows differentiation between logical memory and physical memory.

Basic Method: Users prefer to view memory as a collection of variable sized segments with no necessary ordering among segments.

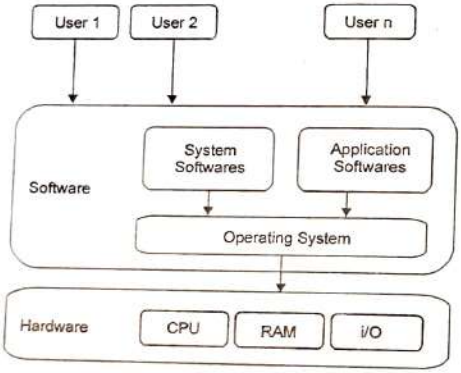
A program can be visualized as a main program with a set of methods, procedures or functions. It may also include various data structures: objects, arrays, stacks, variables etc. Each of these modules or data elements is referred to by name. Each of these segments is of variable length; length is intrinsically defined by the purpose of the segment in the program. Elements within a segment are identified by their offset from the beginning of the segment.

Paging: Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous. Paging avoids external fragmentation and the need for compaction. It also solves the considerable problem of fitting memory chunks of varying sizes onto the backing store; most memory management schemes used before the introduction of paging suffered from this problem. The problem arises because, when some code fragments or data residing in main memory need to be swapped out, space must be found on the backing store. The backing store has the same fragmentation problems discussed in connection with main memory, but access is much

slower, so compaction is impossible. Because of its advantages over earlier methods, paging in its various forms is used in most operating systems.

Q.3. (c) Explain operating system concepts. (3)

Ans. An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.



Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

UNIT-II

Q.4. (a) Describe the role and importance of critical regions. (5)

Ans. The key to preventing trouble involving shared storage is find some way to prohibit more than one process from reading and writing the shared data simultaneously. That part of the program where the shared memory is accessed is called the *Critical* Section. To avoid race conditions and flawed results, one must identify codes in Critical Sections in each thread. The characteristic properties of the code that form a Critical Section are

- Codes that reference one or more variables in a "read-update-write" fashion while any of those variables is possibly being altered by another thread.
- Codes that alter one or more variables that are possibly being referenced in "read-updata-write" fashion by another thread.

- Codes use a data structure while any part of it is possibly being altered by another thread.
- Codes alter any part of a data structure while it is possibly in use by another thread.

Here, the important point is that when one process is executing shared modifiable data in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is mutually exclusive in time.

Q.4. (b) What is deadlock avoidance? Explain the recovery process from deadlock? (5)

Ans. Deadlock avoidance: Most deadlock avoidance algorithms need every process to tell in advance the maximum number of resources of each type that it may need. Based on all these info we may decide if a process should wait for a resource or not and thus avoid chances for circular wait. If a system is already in a safe state, we can try to stay away from an unsafe state and avoid deadlock. Deadlocks cannot be avoided in an unsafe state. A system can be considered to be in safe state if it is not in a state of deadlock and can allocate resources upto the maximum available. A safe sequence of processes and allocation of resources ensures a safe state. Deadlock avoidance algorithms try not to allocate resources to a process if it will make the system in an unsafe state. Since resource allocation is not done right away in some cases, deadlock avoidance algorithms also suffer from low resource utilization problem.

A resource allocation graph is generally used to avoid deadlocks. If there are no cycles in the resource allocation graph, then there are no deadlocks. If there are cycles, there may be a deadlock. If there is only one instance of every resource, then a cycle implies a deadlock. Vertices of the resource allocation graph are resources and processes. The resource allocation graph has request edges and assignment edges. An edge from a process to resource is a request edge and an edge from a resource to process is an allocation edge. A calm edge denotes that a request may be made in future and is represented as a dashed line. Based on calm edges we can see if there is a chance for a cycle and then grant requests if the system will again be in a safe state.

Deadlock Recovery

Traditional operating system such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real time operating systems use Deadlock recovery.

Recovery method

1. Killing the process.

killing all the process involved in deadlock.
Killing process one by one. After killing each
process check for deadlock again keep repeating
process till system recover from deadlock.

2. Resource Preemption

Resources are preempted from the processes involved in deadlock, preempted resources are allocated to other processes, so that there is a possibility of recovering the system from deadlock. In this case system go into starvation.

Q.5. Write the short notes on the following: (5×2=10)

Q.5. (a) Message passing

Ans. Message passing suits diverse situations where exchange of information between processes plays a key role. One of its prominent uses is in the client server

paradigrm, where in a server process offers a service and other processes, called its clients, send messages to it to use its service. This paradigm is used widely a microkernel-based OS structures functionalities such as scheduling in the form of servers, a conventional OS offers services such as printing through servers, and, on the Internet, a variety of services arrc offered by Web servers. Another prominent use of message passing is in higher-level protocols for exchange of electronic mails and communication between tasks in parallel or distributed programs. Here message passing is used to exchange information, while other parts of the protocol are employed to ensure reliability.

The key issues in message passing are how the processes that send and receive messages identify each other, and how the kernel performs various actions related to delivery of messages, how it stores and delivers messages and whether it blocks a process that sends a message until its message is delivered. These features are operating system specific.

The semantics of message passing are as follows:. At a send call P_i, the kernel checks, whether process P_j is blocked on a *receive* call for receiving a message from process P_i. If so, it copies the message into msg_area and activates P_j. If process P_j has not already made a receive call the kernel arranges to deliver the message to it when P_j eventually makes a receive call. When process P_j receives the message, it interprets the message and takes an appropriate action.

Messages may be passed between processes that exist in the same computer or in different computers connected to a network. Also, the processes participating in message passing may decide on what a specific message means and. what actions the receiver process should perform on receiving it. Because of this flexibility, message passing is used in the following applications:

- Message passing is employed in the client -server paradigm, which is used to communicate between components of a microkernel-based operating system and user processes, to provide services such, as the print service such as the print service to processes within an OS, or to provide Web-based services to client processes located in other computers.
- Message passing is used as the backbone of higher-level protocols employed for communicating between computers or for providing the electronic mail facility.
- Message passing is used lo implement communication between tasks in a parallel or distributed program.

Q.5. (*b*) Critical Section Problem

Ans. Consider a system consisting of n processes {P0, P1 , ... , $P11$ _ I}. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time. The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit The remaining code is the remainder section. The general structure of a typical process Pi is shown in Figure. The entry section and exit section are enclosed in boxes to highlight these important segments of code.

do {

I *entry section* I

critical section

I *exit section* I

remainder section

} while (TRUE);

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual exclusion. If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded waiting. There exists a bound, or limit, 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.

Q.5. (*c*) Performance on demand paging

Ans. Demand paging can significantly affect the performance of a computer system. To see why, let's compute the effective access time for a demand-paged memory. For most computer systems, the memory-access time, denoted *ma*, ranges from 10 to 200 nanoseconds. As long as we have no page faults, the effective access time is equal to the memory access time. If, however a page fault occurs, we must first read the relevant page from disk and then access the desired word.

Let p be the probability of a page fault ($0 <= p <= 1$). We would expect p to be close to zero-that is, we would expect to have only a few page faults. The access is then

Effective access time= $(1 - p)$ x ma + p x page fault time.

To compute the effective access time, we must know how much time is needed to service a page fault. A page fault causes the following sequence to occur:

- Trap to the operating system.
- Save the user registers and process state.
- Determine that the interrupt was a page fault.
- Check that the page reference was legal and determine the location of the
- page on the disk
- Issue a read from the disk to a free frame:
  - Wait in a queue for this device until the read request is serviced.
  - Wait for the device seek and/ or latency time.
  - Begin the transfer of the page to a free frame.
- While waiting, allocate the CPU to some other user (CPU scheduling, optional).
- Receive an interrupt from the disk I/0 subsystem (I/0 completed).
- Save the registers and process state for the other user (if step 6 is executed).
- Determine that the interrupt was from the disk
- Correct the page table and other tables to show that the desired page is
- now in memory.
- Wait for the CPU to be allocated to this process again.
- Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

paradigrm, where in a server process offers a service and other processes, called its clients, send messages to it to use its service. This paradigm is used widely a microkernel-based OS structures functionalities such as scheduling in the form of servers, a conventional OS offers services such as printing through servers, and, on the Internet, a variety of services arrc offered by Web servers. Another prominent use of message passing is in higher-level protocols for exchange of electronic mails and communication between tasks in parallel or distributed programs. Here message passing is used to exchange information, while other parts of the protocol are employed to ensure reliability.

The key issues in message passing are how the processes that send and receive messages identify each other, and how the kernel performs various actions related to delivery of messages, how it stores and delivers messages and whether it blocks a process that sends a message until its message is delivered. These features are operating system specific.

The semantics of message passing are as follows:. At a send call P_i, the kernel checks, whether process P_j is blocked on a *receive* call for receiving a message from process P_i. If so, it copies the message into msg_area and activates P_j. If process P_j has not already made a receive call the kernel arranges to deliver the message to it when P_j eventually makes a receive call. When process P_j receives the message, it interprets the message and takes an appropriate action.

Messages may be passed between processes that exist in the same computer or in different computers connected to a network. Also, the processes participating in message passing may decide on what a specific message means and. what actions the receiver process should perform on receiving it. Because of this flexibility, message passing is used in the following applications:

- Message passing is employed in the client -server paradigm, which is used to communicate between components of a microkernel-based operating system and user processes, to provide services such, as the print service such as the print service to processes within an OS, or to provide Web-based services to client processes located in other computers.
- Message passing is used as the backbone of higher-level protocols employed for communicating between computers or for providing the electronic mail facility.
- Message passing is used lo implement communication between tasks in a parallel or distributed program.

Q.5. (b) Critical Section Problem

Ans. Consider a system consisting of n processes {P0, P1 , ... , P11 _ I}. Each process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two processes are executing in their critical sections at the same time. The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit The remaining code is the remainder section. The general structure of a typical process *Pi* is shown in Figure. The entry section and exit section are enclosed in boxes to highlight these important segments of code.

```
do {
I entry section I
critical section
```



```
I exit section I
remainder section
} while (TRUE);
```

A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual exclusion. If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded waiting. There exists a bound, or limit, 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.

Q.5. (c) Performance on demand paging

Ans. Demand paging can significantly affect the performance of a computer system. To see why, let's compute the effective access time for a demand-paged memory. For most computer systems, the memory-access time, denoted *ma*, ranges from 10 to 200 nanoseconds. As long as we have no page faults, the effective access time is equal to the memory access time. If, however a page fault occurs, we must first read the relevant page from disk and then access the desired word.

Let p be the probability of a page fault ($0 <= p <= 1$). We would expect p to be close to zero-that is, we would expect to have only a few page faults. The access is then

Effective access time= $(1 - p) \times ma + p \times$ page fault time.

To compute the effective access time, we must know how much time is needed to service a page fault. A page fault causes the following sequence to occur:

- Trap to the operating system.
- Save the user registers and process state.
- Determine that the interrupt was a page fault.
- Check that the page reference was legal and determine the location of the
- page on the disk
- Issue a read from the disk to a free frame:
  - Wait in a queue for this device until the read request is serviced.
  - Wait for the device seek and/ or latency time.
  - Begin the transfer of the page to a free frame.
- While waiting, allocate the CPU to some other user (CPU scheduling, optional).
- Receive an interrupt from the disk I/0 subsystem (I/0 completed).
- Save the registers and process state for the other user (if step 6 is executed).
- Determine that the interrupt was from the disk
- Correct the page table and other tables to show that the desired page is
- now in memory.
- Wait for the CPU to be allocated to this process again.
- Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

Q.5. (*d*) Allocation of frames

Ans. If we have 93 free frames and two processes, how many frames does each process get? The simplest case is the single-user system. Consider a single-user system with 128 KB of memory composed of pages 1 KB in size. This system has 128 frames. The operating system may take 35 KB, leaving 93 frames for the user process. Under pure demand paging, all 93 frames would initially be put on the free-frame list. When a user process started execution, it would generate a sequence of page faults. The first 93 page faults would all get free frames from the free-frame list. When the free-frame list was exhausted, a page-replacement algorithm would be used to select one of the 93 in-memory pages to be replaced with the 94th, and so on. When the process terminated, the 93 frames would once again be placed on the free-frame list. There are many variations on this simple strategy. We can require that the operating system allocate all its buffer and table space from the free-frame list. When this space is not in use by the operating system, it can be used to support user paging. We can try to keep three free frames reserved on the free-frame list at all times. Thus, when a page fault occurs, there is a free frame available to page into. While the page swap is taking place, a replacement can be selected, which is then written to the disk as the user process continues to execute. Other variants are also possible, but the basic strategy is clear: the user process is allocated any free frame.

Q.5. (*e*) Monitors

Although semaphores provide a convenient and effective mechanism for process synchronization, using them incorrectly can result in timing errors that are difficult to detect, since these errors happen only if some particular execution sequences take place and these sequences do not always occur. We have seen an example of such errors in the use of counters in our solution to the producer-consumer problem. In that example, the timing problem happened only rarely, and even then the counter value appeared to be reasonable-off by only 1. Nevertheless, the solution is obviously not an acceptable one. It is for this reason that semaphores were introduced in the first place.

Unfortunately, such timing errors can still occur when semaphores are used. To illustrate how, we review the semaphore solution to the critical-section problem. All processes share a semaphore variable mutex, which is initialized to 1. Each process must execute wait (mutex) before entering the critical section and signal (mutex) afterward. If this sequence is not observed, two processes may be in their critical sections simultaneously. Next, we examine the various difficulties that may result. Note that these difficulties will arise even if a *single* process is not well behaved. This situation may be caused by an honest programming error or an uncooperative programmer.

UNIT-III

Q.6. (*a*)What are the techniques for device management? How to share device? **(5)**

Ans. In a simple batch monitor *system* device allocation is very inefficient, in that only one job is run at a time and any devices, usually the majority of devices, not used by that particular job are idle. A multiprogramming system utilizes the sum of the devices required by all active jobs. An obvious objective of such a system is to maximize usage of all the peripheral devices and maintain an effective throughput. In a sense, the various concurrent jobs are allocated resources from the pool of peripheral devices. This allocation is typically of two forms: dedicated and shared access.

Certain peripherals, such as direct access devices (e.g., magnetic drums, disks, and data cells) can be used concurrently by more than one job. Information stored on a direct access device is directly accessible, much like the locations of memory; thus there is very little mutual interference when various jobs access different locations on the device,

There are two problems associated with the use of shared access devices that must be handled by the operating system. Although there is very little interference caused by multiple jobs sharing access to the device, there can be only one access occurring at a time (i.e., only one read or write per device can be serviced at once). Depending upon the device, an access may take from a small fraction of a second to several seconds. Thus if two or more jobs wish to access the same device simultaneously, the operating system must act like a traffic policeman. la fact, this function Is often called I/O traffic control. The effectiveness of the operating system in directing I/O traffic, which is usually of considerable volume in a large system, has a substantial effect on multiprogramming efficiency and throughput performance. Even dedicated devices contribute to the traffic control problem since the I/O channels I/O computers) have capacity limitations that may require scheduling if there is very high usage (e.g., it is usually not possible to handle the information transfer if every tape, disk, and drum is active simultaneously). The specific details of I/O traffic control will not be further discussed here since they are very dependent upon device characteristics and CPU peculiarities.

Q.6. (*b*) What is the deadlock characterization? Suggest two methods for deadlock handling. **(5)**

Ans. In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting. Before we discuss the various methods for dealing with the deadlock problem, we look more closely at features that characterize deadlocks.

Necessary Conditions

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

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

Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

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

Circular wait: A set { *P0* , Pl, ... , P11 } of waiting processes must exist such that *P*0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ... , Pn-1 is waiting for a resource held by *P,v* and P11 is waiting for a resource held by *P*0.

We emphasize that all four conditions must hold for a deadlock to occur. The circular-wait condition implies the hold-and-wait condition, so the four conditions are not completely independent.

Following are two methods for handling deadlock are:

Dead lock Preventing:

Dead lock Preventing provides a set of methods for ensuring that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining.

Deadlock Avoidance:

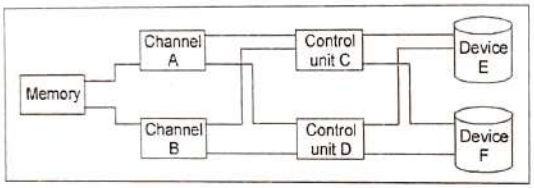
Deadlock Avoidance requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Q.7. Explain in brief the following: **(5×2.5=12.5)**

Q.7. (a) Multiple path

Ans. We could reduce the channel and control unit bottlenecks by buying more channels and control units. A more economical alternative, however, is to use the channels and control units in a more flexible manner and allow multiple paths to each device. Figure illustrates such an I/O configuration. In this example there are four different paths to Device E:

1. Channel A - Control Unit C - Device E
2. Channel A - Control Unit D - Device E
3. Channel B -Control Unit, C - Devices E
4. Channel B -Control Unit D - Devices E



If channel A and control unit C were both busy due to I/O for device F, a path could be found through channel B and control D to service device E. An actual I/O configuration may have eight control units per channel and eight devices per control unit rather than the sparse configuration. A completely flexible configuration would make all control units accessible to all channels and all devices accessible to all control units, thus providing possibly 256 alternate paths between memory and each device. For economic reasons, this extreme case, though feasible, is seldom found. In addition to providing flexibility, a multiple path I/O configuration is desirable for reliability. For example, if channel a malfunction and must be repaired, all the control units and devices can still be accessed through channel B. It is the responsibility of the device management routines to maintain status information on all paths in the I/O configuration and find an available' route to access the desired device.

Q.7. (b) Block multiplexing'

Ans. On conventional 370 selector channels, the techniques of independent device operation and buffering are applicable only to the last CCW of a channel program. The channel and control unit remain connected to the device for the duration of the channel program. A 370 block multiplexor channel can be servicing multiple channel programs at the same time (e.g., eight channel programs). When the channel encounters an I/O operation such as a buffered or independent device operation that does not need the channel for awhile, it automatically switches to another channel program that needs immediate servicing. In essence, a block multiplexor channel represents a hardware implementation of multiprogramming for channel programs.

This type of channel multiprogramming could be simulated on a selector channel by making all channel programs only one command long. The device management software routines could then reassign the channel as necessary. However, this approach is not very attractive unless the central processor is extremely fast, since the channel switching must be done very frequently and very quickly.

Q.7. (c) Semaphore

Ans. Semaphores are a pair composed of an integer variable that apart from initialization is accessed only through two standard atomic operations: wait and signal.

Wait: decrease the counter by one; if it gets negative, block the process and enter its id in the queue.

Signal: increase the semaphore by one; if it's still negative, unblock the first process of the queue, removing its id from the queue itself.

```
wait(S)
{
while(S<=0);
S - -;
}
signal(S)
{
S++;
}
```

The atomicity of the above operations is an essential requirement: mutual exclusion while accessing a semaphore must be strictly enforced by the operating system. This is often done by implementing the operations themselves as uninterruptible system calls. It's easy to see how a semaphore can be used to enforce mutual exclusion on a shared resource: a semaphore is assigned to the resource, it's shared among all processes that need to access the resource, and its counter is initialized to 1. A process then waits on the semaphore upon entering the critical section for the resource, and signals on leaving it. The first process will get access. If another process arrives while the former is still in the critical section, it'll be blocked, and so will further processes. In this situation the absolute value of the counter is equal to the number of processes waiting to enter the critical section. Every process leaving the critical section will let a waiting process use the resource by signaling the semaphore.

In order to fully understand semaphores, we'll discuss them briefly before engaging any system calls and operational theory.

The name semaphore is actually an old railroad term, referring to the crossroad "arms" that prevent cars from crossing the tracks at intersections. The same can be said about a simple semaphore set. If the semaphore is on (the arms are up), then a resource is available (cars may cross the tracks). However, if the semaphore is off (the arms are down), then resources are not available (the cars must wait).

Q.7. (d) Virtual Devices

Ans. Some devices that would normally have to be dedicated (e.g., card readers) may be converted into shared devices through techniques such as SPOOLing. For example, a SPOOLing program can read and copy all card input onto a disk at high speed. Later, when a process tries to read a card, the SPOOLing program intercepts the request and converts it to a read from the disk, Since a disk may be easily shared by several users, we have converted a dedicated devices to a shared device, changing one card reader into many "virtual" card readers. This technique is equally applicable to a large number of peripheral devices, such as teletypes, printers, and most dedicated slow input/output devices.

Q.7. (e) Disk Scheduling

Ans. One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. The access time has two major components. The seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector. The rotational latency is the additional time for the disk to rotate the

desired sector to the disk head. The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time and the bandwidth by managing the order in which disk I/O requests are serviced.

The Scheduling is used for Divide the Total Time of the CPU between the Number or Processes So that the Processes can execute Concurrently at a Single Time. following the Scheduling Techniques:

- First Come-First Serve (FCFS)
- Shortest Seek Time First (SSTF)
- Elevator (SCAN)
- Circular SCAN (C-SCAN)
- LOOK
- C-LOOK

UNIT-IV

Q.8. (*a*) Explain access control verification? Define the access methods. **(6.5)**

Ans. Users, and for each of those entities, access-control information is assigned. A similar function can be added to other aspects of a computer system. A good example of this is found in Solaris 10. Solaris 10 advances the protection available in the Sun Microsystems operating system by explicitly adding the principle of least privilege via role based access control. This facility revolves around privileges.

A privilege is the right to execute a system call or to use an option within that system call (such as opening a file with write access). Privileges can be assigned to processes, limiting them to exactly the access they need to perform their work. Privileges and programs can also be assigned to roles. Users are assigned roles or can take roles based on passwords to the roles. In this way a user can take a role that enables a privilege, allowing the user to run a program to accomplish a specific task.

There are several methods to access files

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Q.8. (*b*) Differentiate the logical file system with physical file system?

Ans. Logical File System

The fourth processing step is called the Logical file system (LFS). A typical call would be

CALL LFS (function, AFT entry, record number, location) such as

CALL LFS (READ,2,4,12000)

which is identical, in form, to the ACV. The Logical File system converts the request for a logical record into a request for a logical byte string. That is, a file is treated as a sequential byte string without any explicit record format by the Physical File System. In this simple case of fixed length records, the necessary conversion can be accomplished by using the record length information from the AFT entry. That is, the

Logical Byte Address=(Record Number – 1) × Record Length

and

Logical Byte Length=Record Length

The Logical Byte string Address and Length are used instead of the record number for the next step, the physical File System, which processes calls such as

CALL PFS (READ, 2, 1500, 500, 12000)

Note that by permanently assigning the BFD file a specific ID and a specific AFT entry number (such as AFT entry 1), the Basic File System can call upon the Logical File System to read and write entries in the Basic File Directory (It is necessary to have a special procedure for fetching BFD entry 1 into AFT entry 1 when the system is initially started or restarted.)

In Summary, the Logical File System is responsible for:

1 Conversion of record request to byte string request

2 Communication with Physical File system

Examples of more complex logical record organizations, such as variable-length records, are presented in later sections.

Physical File System

The fifth processing step is called the Physical File System (PFS). A typical call would be

CALL PFS (function, AFT entry, byte address, byte length, location)

such as

CALL PFS (READ,2,1500.500.12000)

The Physical file system uses the byte address plus the AFT entry to determine the physical block that contains the desired byte of the Device Strategy Module, and the byte string is extracted and moved to the user's buffer area.

The mapping from Logical Byte Address to Physical Block Number, for the simple contiguous organization

$$\text{Physical Block Number} = \frac{\text{Logical Byte Address}}{\text{physical block size}} + \text{address of first physical block}$$

For the request for the byte string starting at Logical Byte Address 1500, the Physical Block Number is:

$$\text{Physical Block Number} = \frac{1500}{1000} + 6 = 1 + 6 = 7$$

Q.9. Write in brief about the following: **(5×2.5=12.5)**

Q.9. (a). Free space management

Ans. Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. (Write-once optical disks only allow one write to any given sector, and thus such reuse is not physically possible.) To keep track of free disk space, the system maintains a free-space list. The free-space list records all *free* disk blocks-those not allocated to some file or directory. To create a file, we search the free-space list for the required amount of space and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list. The free-space list, despite its name, might not be implemented as a list, as we discuss next.

Free-space management system to filed a free block, and this new block is written to and is linked to the end of the file. To read a file, we simply read blocks by following the pointers from block to block. There is no external fragmentation with linked allocation, and any free block on the free-space list can be used to satisfy a request. The size of a file need not be declared when that file is created. A file can continue to grow as long as free blocks are available. Consequently, it is never necessary to compact disk space.

Linked allocation does have disadvantages, however. The major problem is that it can be used effectively only for sequential-access files. To find the ith block of a file, we must start at the beginning of that file and follow the pointers until we get to the ith block. Each access to a pointer requires a disk read, and some require a disk seek. Consequently, it is inefficient to support a direct-access capability for linked-allocation files.

Another disadvantage is the space required for the pointers. If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information. Each file requires slightly more space than it would otherwise.

The usual solution to this problem is called blocks into multiples, called and to allocate clusters rather than blocks. For instance, the file system may define a cluster as four blocks and operate on the disk only in cluster units. Pointers then use a much smaller percentage of the file's disk space. This method allows the logical-to-physical block mapping to remain simple.

Q.9. (b) Access matrix

Ans. Our model of protection can be viewed abstractly as a matrix, called an The rows of the access matrix represent domains, and the columns represent objects. Each entry in the matrix consists of a set of access rights. Because the column defines objects explicitly, we can omit the object name from the access right. The entry access(i,j) defines the set of operations that a process executing in domain *Oi* can invoke on object *OJ*. To illustrate these concepts, we consider the access matrix shown in Figure. There are four domains and four objects-three files (F1, F2, F3) and one laser printer. A process executing in domain 0 1 can read files F1 and *F3*. A process executing in domain 0 4 has the same privileges as one executing in domain 0 1; but in addition, it can also write onto files F1 and *F3*. Note that the laser printer can be accessed only by a process executing in domain 0 2. The access-matrix scheme provides us with the mechanism for specifying a variety of policies. The mechanism consists of implementing the access matrix and ensuring that the semantic properties we have outlined hold. More specifically, we must ensure that a process executing in domain n can access only those objects specified in row (and then only as allowed by the access-matrix entries. The access matrix can implement policy decisions concerning protection. The policy decisions

involve which rights should be included in the (i,j)th entry. We must also decide the domain in which each process executes. This last policy is usually decided by the operating system.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Q.9.(c) Cryptography

Ans. Cryptography is the branch of science dealing with encryption techniques. The original form of data is called the plaintext form and the transformed form is called the encrypted or ciphertext form. We use the following notation:

Pd Plaintext form of data *d*

Cd Ciphertext form of data *d*

where $Pd = d$. Encryption is performed by applying an encryption algorithm E with a specific encryption key k to data. Data is recovered by applying a decryption algorithm D with a key $k\_$. In the simplest formof encryption called *symmetric encryption*, decryption is performed by using the same key k. In advanced encryption techniques called *asymmetric encryption*, a different key $k\_$ is used to decrypt a ciphertext.

We represent encryption and decryption of data by using algorithms E and D with key k as application of functions Ek and Dk, respectively. Thus,

$Cd = Ek(d)$

$Pd = Dk(Cd)$

Obviously the functions Ek and Dk must satisfy the relation $Dk(Ek(d)) = d$, for all d Thus a process must be able to perform the transformation Dk in order to obtain the plaintext form of encrypted data.

Q.9. (d) Goals of protection

Ans. Protection was originally conceived as an adjunct to multiprogramming operating systems, so that untrustworthy users might safely share a common logical name space, such as a directory of files, or share a common physical name space, such as memory. Modern protection concepts have evolved to increase the reliability of any complex system that makes use of shared resources. We need to provide protection for several reasons. The most obvious is the need to prevent the mischievous, intentional violation of an access restriction by a user. Of more general importance, however, is the need to ensure that each program component active in a system uses system resources only in ways consistent with stated policies. This requirement is an absolute one for a reliable system.

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. Early detection of interface errors can often prevent contamination of a healthy subsystem by a malfunctioning subsystem. Also, an unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user. A protection-oriented system provides means to distinguish between

authorized and unauthorized usage. The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use. These policies can be established in a variety of ways. Some are fixed in the design of the system, while others are formulated by the management of a system. Still others are defined by the individual users to protect their own files and programs. A protection system must have the flexibility to enforce a variety of policies.

Q.9. (e) System Threats

Ans. System threats refer to misuse of system services and network connections to put user in trouble. System threats can be jsed to launch program threats on a complete network called as program attack. System threats create such an environment that operating system resources/ user files are misused. Following is Hie list of some well-known system threats

• worm -worm is a process which can cnoke down a system perrcrmance by using system resources to extreme levels. A Worm process generates its multiple copies where Bach copy uses system resources, prevents all other processes to get required resources. Worm processes can even shut down an entire network.

• **Port Scanning** - Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to'make an attack on the system.

• **Denial of Service** - Denial of service attacks normally prevents user to make legitimate use of die system. For example user may not be able to use internet if denial of service attacks browser's content settings.

END TERM EXAMINATION [MAY 2017]
SECOND SEMESTER [MCA]
OPERATING SYSTEMS [MCA–106]

Time : 3 hrs. **M.M. : 75**

Note: *Attempt all questions as directed. Internal choice is indicted.*

Q.1. Answer the following: (2.5)

(a) What is distributed O.S?

Ans. Distributed Operating System is a model where distributed applications are running on multiple computers linked by communications. A distributed operating system is an extension of the network operating system that supports higher levels of communication and integration of the machines on the network.

This system looks to its users like an ordinary centralized operating system but runs on multiple, independent central processing units (CPUs).

These systems are referred as *loosely coupled systems* where each processor has its own local memory and processors communicate with one another through various communication lines, such as high speed buses or telephone lines. By loosely coupled systems, we mean that such computers possess no hardware connections at the CPU - memory bus level, but are connected by external interfaces that run under the control of software.

The Distributed Os involves a collection of autonomous computer systems, capable of communicating and cooperating with each other through a LAN / WAN. A Distributed Os provides a virtual machine abstraction to its users and wide sharing of resources like as computational capacity, I/O and files etc.

Q.1. (b) What is multi processor scheduling. (2.5)

Ans. Multiprocessor scheduling

• When multiple processors are available, then the scheduling gets more complicated, because now there is more than one CPU which must be kept busy and in effective use at all times.

• ***Load sharing*** revolves around balancing the load between multiple processors.

• Multi-processor systems may be **heterogeneous,** (different kinds of CPUs), or **homogenous,** (all the same kind of CPU). Even in the latter case there may be special scheduling constraints, such as devices which are connected via a private bus to only one of the CPUs.

Approaches to Multiple-Processor Scheduling

• One approach to multi-processor scheduling is asymmetric multiprocessing, in which one processor is the master, controlling all activities and running all kernel code, while the other runs only user code. This approach is relatively simple, as there is no need to share critical system data.

• Another approach is symmetric multiprocessing, SMP, where each processor schedules its own jobs, either from a common ready queue or from separate ready queues for each processor.

• Virtually all modern OSes support SMP, including XP, Win 2000, Solaris, Linux, and Mac OSX.

Q.1. (c) Mention the main features of Linux Operating System. (2.5)

Ans. Following are some of the important features of Linux Operating System.

- **Portable** – Portability means softwares can works on different types of hardwares in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available and it is community based development project. Multiple teams works in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** – Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs etc.
- **Security** – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Other features includes Live CD/USB, Graphical user interface (X Window System), Application Support etc.

Q.1. (d) Mention any two differences between shared devices and virtual devices. (2.5)

Ans. Shared Devices

- Assigned to several processes. E.g., disk pack (or other direct access storage device) can be shared by several processes at same time by interleaving their requests.
- Interleaving must be carefully controlled by Device Manager.
- All conflicts must be resolved based on predetermined policies to decide which request will be handled first.

Virtual Devices

- Combination of dedicated devices that have been transformed into shared devices.

– E.g, printers are converted into sharable devices through a spooling program that reroutes all print requests to a disk.

– Output sent to printer for printing only when all of a job's output is complete and printer is ready to print out entire document.

– Because disks are sharable devices, this technique can convert one printer into several "virtual" printers, thus improving both its performance and use.

- They are dedicated devices that have been transformed into shared devices.

Q.1. (e) Name various file allocation methods. (2.5)

Ans. Contiguous Allocation Method:- The simplest allocation scheme is to store each file as a set of contiguous blocks on the disk. Thus on a disk with 1K blocks, a 50K file will be allocated 50 consecutive blocks. Contiguous allocation of a file is defined by the disk address and length of first block.

Linked Allocation Method:- This method solves the problem associated with contiguous allocation. In linked allocation each file is a linked list of disk blocks scattered

on the disk. The first word of each block is used as a pointer to the next one and the rest of block is used for data. The entry of directory contains a pointer to the first and last blocks of the block.

Indexed Allocation:- Linked allocation solves the external fragmentation and size declaration problems of contiguous allocation. But linked allocation cannot support efficient direct access as the pointer of the block are scattered. Index allocation solves this problem by bringing all the pointers together into one location called index block.

Q.1. (f) Compare program threats and system threats. (2.5)

Ans. Program Threats : Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as Program Threats.

Following is the list of some well-known program threats.

- **Trojan Horse**—Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.
- **Trap Door**—If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.
- **Logic Bomb**—Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.
- **Virus**—Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small code embedded in a program.

System Threats : System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack.

Following is the list of some well-known system threats.

- **Worm**—Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.
- **Port Scanning**—Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.
- **Denial of Service**—Denial of service attacks normally prevents user to make legitimate use of the system.

Q.1. (g) How does many-to-one thread model differ from one-to-one model? Explain. (2.5)

Ans. Many to One Thread Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

One to One Thread Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another

thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.

Q.1. (h) What is Multilevel Feedback Queue Scheduling. **(2.5)**

Ans. Multilevel feedback queue scheduling allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. **Multilevel Feedback Queue Scheduling (MLFQ)** keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.

The definition of a multilevel feedback queue scheduler makes it the most general CPU-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the **most general scheme**, it is also the **most complex**.

Q.1. (i) What is segmentation and paging. **(2.5)**

Ans. Segmentation is a Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a segment.

A table stores the information about all such segments and is called **Segment Table** which maps two dimensional Logical address into one dimensional Physical address.

Advantages of Segmentation:

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantage of Segmentation:

- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

Paging is a memory management scheme that eliminates the need for 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 memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Q.1. (j) What is the benefit of cryptography in O.S.? **(2.5)**

Ans. Cryptography provides for secure communication in the presence of malicious third-parties—known as adversaries. Encryption (a major component of cryptography) uses an algorithm and a key to transform an input (i.e., plaintext) into an encrypted output (i.e., ciphertext). A given algorithm will always transform the same plaintext into the same ciphertext if the same key is used.

Algorithms are considered secure if an attacker cannot determine any properties of the plaintext or key, given the ciphertext. An attacker should not be able to determine anything about a key given a large number of plaintext/ciphertext combinations which used the key.

Cryptography provides the four most basic services of information security

- **Confidentiality**– Encryption technique can guard the information and communication from unauthorized revelation and access of information.
- **Authentication**– The cryptographic techniques such as MAC and digital signatures can protect information against spoofing and forgeries.
- **Data Integrity**– The cryptographic hash functions are playing vital role in assuring the users about the data integrity.
- **Non-repudiation**– The digital signature provides the non-repudiation service to guard against the dispute that may arise due to denial of passing message by the sender.

UNIT–I

Q.2. (a) Define job queue, ready queue and device queue that are used in the process scheduling. **(5)**

Ans. The Operating System maintains the following important process scheduling queues

- **Job queue** – This queue keeps all the processes in the system.
- **Ready queue** – This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** – The processes which are blocked due to unavailability of an I/O device constitute this queue.

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

Q.2. (b) Explain various criteria considered in CPU scheduling algorithms. Explain the following methods: **(7.5)**

(i) Shortest Job First Scheduling

(ii) Round Robin Scheduling.

Ans. There are many different criterias to check when considering the **"best"** scheduling algorithm, they are:

CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (Ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process(Wall clock time).

Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

Response Time

Amount of time it takes from when a request was submitted until the first response is produced. Remember, it is the time till the first response and not the completion of process execution(final response).

In general CPU utilization and Throughput are maximized and other factors are reduced for proper optimization.

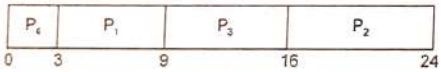
Scheduling Algorithms

(i) Shortest-Job-First Scheduling (SJF)

- The idea behind the SJF algorithm is to pick little job that needs to be done, get it out of the way first, and then pick the next smallest fastest job to do next. If two processes have the same CPU burst time then FCFS is used to break the tie. Technically this algorithm picks a process based on the next shortest **CPU burst**, not the overall process time.
- For example, consider the following four processes:

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

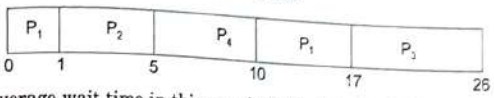
- Below is the Gantt chart of above CPU burst times, and the assumption that all jobs arrive at the same time.



- In the case above the average wait time is (0 + 3 + 9 + 16) / 4 = 7.0 ms, (as opposed to 10.25 ms for FCFS for the same processes.)
- SJF can be proven to be the fastest scheduling algorithm, but it suffers from one important problem: How do you know how long the next CPU burst is going to be?
- SJF can be either preemptive or non-preemptive. Preemption occurs when a new process arrives in the ready queue that has a predicted burst time shorter than the time remaining in the process whose burst is currently on the CPU. Preemptive SJF is sometimes referred to as shortest remaining time first scheduling.
- Consider the following four processes data.

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
p4	3	5

Given below is the Gantt chart of above data

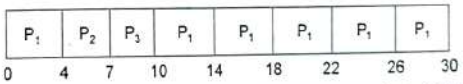


- The average wait time in this case is ((10 - 1) + 0 + (17 - 2) + (5 - 3)) / 4 = 26 / 4 = 6.5 ms. (As opposed to 7.75 ms for non-preemptive SJF or 8.75 for FCFS.)

(ii) Round Robin Scheduling

- Round robin scheduling is similar to FCFS scheduling, except that CPU bursts are assigned with limits called time quantum.
- When a process is given the CPU, a timer is set for whatever value has been set for a time quantum.
  - If the process finishes its burst before the time quantum timer expires, then it is swapped out of the CPU just like the normal FCFS algorithm.
  - If the timer goes off first, then the process is swapped out of the CPU and moved to the back end of the ready queue.
- The ready queue is maintained as a circular queue, so when all processes have had a turn, then the scheduler gives the first process another turn, and so on.
- RR scheduling can give the effect of all processors sharing the CPU equally, although the average wait time can be longer than with other scheduling algorithms. In the following example the average wait time is 5.66 ms, if time quantum is set to be 4ms.

Process	Burst Time
P1	24
P2	3
P3	3



- The performance of RR is sensitive to the time quantum selected. If the quantum is large enough, then RR reduces to the FCFS algorithm; If it is very small, then each process gets 1/nth of the processor time and share the CPU equally.

UNIT–II

Q.3. (a) Define semaphores. Explain the role of wait() and signal() function used in semaphores. **(6)**

Ans. Semaphore: A semaphore is a synchronization tool. A semaphore S is an integer variable that apart from initialization is accessed only through two standard atomic operations: wait and signal.

Wait and signal can be defined as

Wait (S): while S < 0 do no-op;

S = S –1;

Signal (S): S = S + 1;

The modification of S is done indivisibly. It can be used to deal with the n-process critical section problem.

This variable is used to solve critical section problem and to achieve process synchronization in the multi processing environment. The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphore can take non-negative integer values and Binary semaphore can take the value 0 & 1 only.

Suppose S is a semaphore whose private counter has been initialized to a non-negative integer.

- When Wait is executed by a thread, we have two possibilities:

o The counter of S is positive

In this case, the counter is decreased by 1 and the thread resumes its execution.

o The counter of S is zero

In this case, the thread is suspended and put into the private queue of S.

- When Signal is executed by a thread, we also have two possibilities:

o The queue of S has no waiting thread

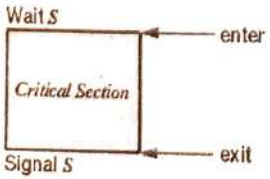
The counter of S is increased by one and the thread resumes its execution.

o The queue of S has waiting threads

In this case, the counter of S must be zero. One of the waiting threads will be allowed to leave the queue and resume its execution. The thread that executes Signal also continues.

The operations of Wait or Signal are atomic. This means once the activities of Wait start (i.e., testing and decreasing the value of the counter and putting the thread into the queue), they will continue to the end without any interruption. More precisely, even though there are many steps to carry out Wait and Signal, these steps are considered as a single non-interruptible instruction. Similarly, the same applies to Signal. Moreover, if more than one threads try to execute Wait (or Signal), only one of them will succeed.

Because Wait may cause a thread to block (i.e., when the counter is zero), it has a similar effect of the lock operation of a mutex lock. Similarly, a Signal may release a waiting threads, and is similar to the unlock operation. In fact, semaphores can be used as mutex locks. Consider a semaphore S with initial value 1. Then, Wait and Signal correspond to lock and unlock:



Q.3. (b) Mention the characteristics of a deadlocked system. Explain various deadlock recovery techniques. (6.5)

Ans. In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting. There are four conditions that must be present simultaneously for a deadlock to occur:

1. Mutual Exclusion

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process

2. Hold and Wait

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

3. No Preemption

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

4. Circular Wait

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing(or decreasing) order.

Deadlock Recovery

There are three basic approaches to recovery from deadlock:

- Inform the system operator, and allow him/her to take manual intervention.
- Terminate one or more processes involved in the deadlock
- Preempt resources.

Process Termination: Two basic approaches, both of which recover resources allocated to terminated processes:

- Terminate all processes involved in the deadlock. This definitely solves the deadlock, but at the expense of terminating more processes than would be absolutely necessary.
- Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.

Resource Preemption: When preempting resources to relieve deadlock, there are three important issues to be addressed:

- Selecting a victim - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
- Rollback - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (i.e. abort the process and make it start over.)
- Starvation - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

OR

Q.4. (a) Explain any two page replacement algorithms. Give an illustration. (7.5)

Ans. Page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with

newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

First In First Out (FIFO) –

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

For example-1, consider page reference string 1, 3, 0, 3, 5, 6 and 3 page slots.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> 3 Page Faults.

when 3 comes, it is already in memory so —> 0 Page Faults.

Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>1 Page Fault.

Finally 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>1 Page Fault.

Optimal Page replacement – In this algorithm, pages are replaced which are not used for the longest duration of time in the future.

Let us consider page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 and 4 page slots.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> 4 Page faults

0 is already there so —> 0 Page fault.

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>1 Page fault.

0 is already there so —> 0 Page fault..

4 will takes place of 1 —> 1 Page Fault.

Now for the further page reference string —> 0 Page fault because they are already available in the memory.

Q.4. (b) Explain the process of logical to physical address translation in segmentation with paging system. Give the respective block diagram. (5)

Ans. Addresses in a Segmented Paging System

√ The segment number indexes into the segment table which yields the base address of the page table for that segment.

√ Check the remainder of the address (page number and offset) against the limit of the segment.

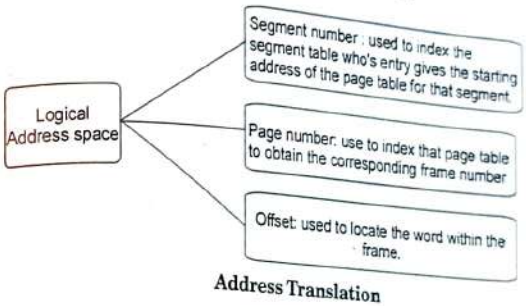
√ Use the page number to index the page table. The entry is the frame. (The rest of this is just like paging.)

√ Concat the frame and the offset to get the physical address.

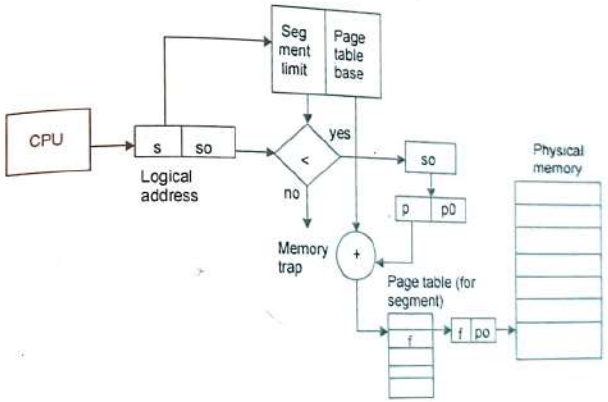
- Each process has:

one segment table:

several page table: one page table per segment



Address Translation



ADVANTAGES

- Reduces memory usage as opposed to pure paging
  - *Page table size limited by segment size*
  - *Segment table has only one entry per actual segment*
- Share individual pages by copying page table entries.
- Share whole segments by sharing segment table entries, which is the same as sharing the page table for that segment.
- Most advantages of paging still hold
  - *Simplifies memory allocation*
  - *Eliminates external fragmentation.*
- In general this system combines the efficiency in paging with the protection and sharing capabilities of the segmentation.

UNIT-III (6.5)

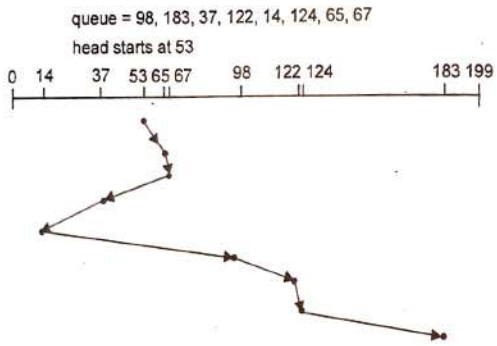
Q.5. (a) Explain the following disk scheduling algorithms:

(i) SSTF (ii) C-SCAN

Ans. (i) SSTF

- Selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests

- Illustration shows total head movement of 236 cylinders



Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

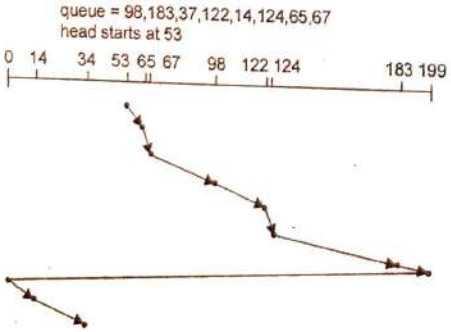
- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favours only some requests

Ans. (ii) C-SCAN

- Provides a more uniform wait time than SCAN
- The head moves from one end of the disk to the other, servicing requests as it goes

→ When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip

- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one



The disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

Advantages:

- Provides more uniform wait time compared to SCAN

Q.5. (b) (i) Explain the following features of devices management. (6)

(i) Device allocation methods.

Ans.

The devices can be managed and allocated by an OS in three possible ways:

(a) Dedicated

(b) Shared, and

(c) Virtua

Dedicated Devices

- Are assigned to only one job at a time.
- They serve that job for the entire time the job is active or until it releases them.
- Some devices demand this kind of allocation scheme, because it would be awkward to let several users share them.
  - o Example; tape drives, printers, and plotters
- Disadvantages
- They must be allocated to a single user for the duration of a job's execution, which can be quite inefficient, even though the device is not used 100% of the time.

Shared Devices

- Can be assigned to several processes.
- For example - a disk (DASD) can be shared by several processes at the same time by interleaving their requests;
  - o This interleaving must be carefully controlled by the Device Manager
  - o All conflicts must be resolved based on predetermined policies.

Virtual Devices

- A combination of the first two types;
- They're dedicated devices that have been transformed into shared devices.
  - o Example: printer
  - o Converted into a shareable device through a spooling program that reroutes all print requests to a disk.
  - o Only when all of a job's output is complete, and the printer is ready to print out the entire document, is the output sent to the printer for printing.
- Because disks are shareable devices, this technique can convert one printer into several virtual printers, thus improving both its performance and use.

(ii) Buffering and block multiplexing.

Ans. Buffering:

By providing a data buffer in the device or control unit, devices that would ordinarily require the channel for a long period can be made to perform independently of the channel. For example, a card reader may physically require about 60ms to read a card and transfer the data to memory through the channel. However, a buffered card reader always reads one card before it is needed and saves the 80 bytes in a buffer in the card reader or control unit. When the channel requests that a card be read, the contents of the 80-byte

buffer are transferred to memory at high speed (e.g., 100 μs) and the channel is released. The device then proceeds to read the next card and buffer it independently. Card readers, card punches, and printers are frequently buffered.

Block Multiplexing:

On conventional 370 selector channels, the techniques of independent device operation and buffering are applicable only to the last CCW of a channel program. The channel and control unit remain connected to the device for the duration of the channel program. A 370 block multiplexor channel can be servicing multiple channel programs at the same time (e.g., eight channel programs). When the channel encounters an I/O operation such as a buffered or independent device operation that does not need the channel for awhile, it automatically switches to another channel program that needs immediate servicing. In essence, a block multiplexor channel represents a hardware implementation of multiprogramming for channel programs.

This type of channel multiprogramming could be simulated on a selector channel by making all channel programs only one command long. The device management software routines could then reassign the channel as necessary. However, this approach is not very attractive unless the central processor is extremely fast, since the channel switching must be done very frequently and very quickly.

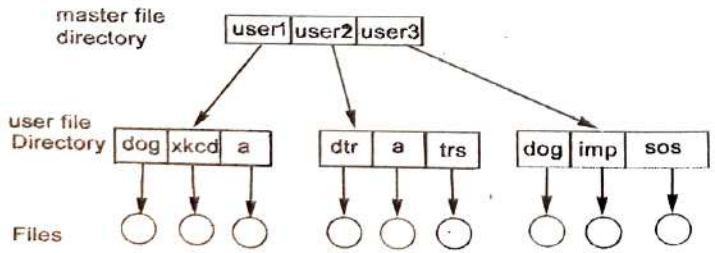
UNIT–IV

Q.6. Explain any two of the following.

(a) Two-Level Directory structure **(6.25)**

Ans. 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 same file name for different user.
- Searching is efficient in this method.



Features:

- One master file directory.
- Each user has their own user file directory.
- Each entry in the master file directory points to a user file directory.

Issues:

- Sharing - accessing other users files.
- System files.
- Grouping problem.

Q.6. (b) Intrusion Detection 2017–15

Ans. Intrusion detection, as its name suggests, strives to detect attempted or successful intrusions into computer systems and to initiate appropriate responses to the intrusions. Intrusion detection encompasses a wide array of techniques that vary on a number of *axes*. Some of these axes include: **(6.25)**

- The time that detection occurs: in real time (while it is occurring) or after the fact only.
- The types of inputs examined to detect intrusive activity. These could include user shell commands, process system calls, and network packet headers or contents. Some forms of intrusions might be detected only by correlating information from multiple such sources.
- The range of response capabilities. Simple forms of response include alerting an administrator of the potential intrusion or somehow halting the potentially intrusive activity, for example, killing a process engaged in apparently intrusive activity. In a sophisticated form of response, a system might transparently divert an intruder's activity to a trap—a false resource exposed to the attacker in order to monitor and gain information about the attack; to the attacker, the resource appears real.

These degrees of freedom in the design space for detecting intrusions in systems have yielded a wide range of solutions known as intrusion-detection systems (IDS).

Q.6. (c) Free space management in Linux. **(6.25)**

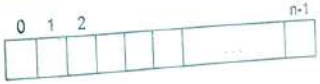
Ans. Free Space Management

- Disk space is limited.
- It is necessary to reuse the space freed when files are deleted for new files.
- *To* keep track of free disk space, a free-space fist is maintained by the system.
- The free-space list records alf disk blocks that are free.
- Free blocks are any that are not allocated to some file or directory.
- When a new file is created, we

1. Search the free-space list for the required amount of space
2. Allocate that space to the new file,

- *Any* allocated space is removed from the free-space list.
- When the file is deleted, its disk space is added to the free-space list.
- *A* free-space list is not always implemented as a list.

Bit vector or bit map (*n* blocks)



$$bit[I] = \begin{cases} 1 \Rightarrow \text{block[i] free} \\ 0 \Rightarrow \text{block[i] occupied} \end{cases}$$

Block number calculation

(number of bits per word)*
(number of 0-value word)+
offset of first 1 bit

CPUs have instructions to return offset within word of first "1" bit

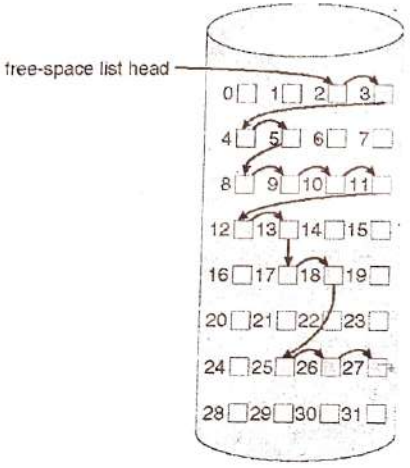
- Bit map requires extra space
- Easy to get contiguous files

Linked list (free list)

→ Cannot get contiguous space easily

→ No waste of space

→ No need to traverse the entire list (if # free blocks recorded)



- Grouping

→ Modify linked list to store address of next *n-1* free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one)

- Counting

→ Because space is frequently contiguously used and freed, with contiguous-allocation allocation, extents, or clustering

- Keep address of first free block and count of following free blocks
- Free space list then has entries containing addresses and counts.

END TERM EXAMINATION [MAY-JUNE 2018]
SECOND SEMESTER [MCA]
OPERATING SYSTEM [MCA–106]

Time : 3 hrs. **M.M. : 75**

Note: Attempt any five questions including Q. no. 1 which is compulsory.

Q. 1. (a) Write three differences between a process and a thread. What do you mean by user level and kernel level threads? Write any three difference among them. (5)

Process	Thread
1. System calls involved in process.	1. No system calls involved.
2. Context switching required.	2. No context switching required.
3. Processes are independent.	3. Threads exist as subsets of a process. They are dependent.

User Level Threads: The user-level threads are implemented by users and the kernel is not aware of the existence of these threads. It handles them as if they were single-threaded processes. User-level threads are small and much faster than kernel level threads. They are represented by a program counter(PC), stack, registers and a small process control block. Also, there is no kernel involvement in synchronization for user-level threads.

Kernel Level Threads: Kernel-level threads are handled by the operating system directly and the thread management is done by the kernel. The context information for the process as well as the process threads is all managed by the kernel. Because of this, kernel-level threads are slower than user-level threads.

User Level Thread	Kernel Level Thread
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris.

Q. 1. (b) Design a solution to a critical section problem string one semaphore and check it for mutual exclusion, progress and bounded waiting conditions. (5)

Ans. The solution to critical section problem which uses one semaphore and address the issues of Mutual Exclusion, progress and bounded waiting.

Peterson's Algorithm

- a simple algorithm that can be run by two processes to ensure mutual exclusion for one resource (say one variable or data structure)

- does not require any special hardware
- it uses busy waiting (a spinlock)

Peterson's Algorithm: Shared variables are created and initialized before either process starts. The shared variables flag[0] and flag[1] are initialized to FALSE because neither process is yet interested in the critical section. The shared variable turn is set to either 0 or 1 randomly (or it can always be set to say 0).

var flag: **array** [0..1] **of** boolean;
turn: 0..1;
flag[k] means that process[k] is interested in the critical section
flag[0] := FALSE;
flag[1] := FALSE;
turn := random(0..1)

After initialization, each process, which is called process *i* in the code (the other process is process *j*), runs the following code:

repeat
flag[i] := TRUE;
turn := j;
while (flag[j] **and** turn=j) **do** no-op;
CRITICAL SECTION
flag[i] := FALSE;
REMAINDER SECTION
until FALSE;
Information common to both processes:
turn = 0
flag[0] = FALSE
flag[1] = FALSE

Q. 1. (c) What do you mean by an inverted page table? (5)

Ans.

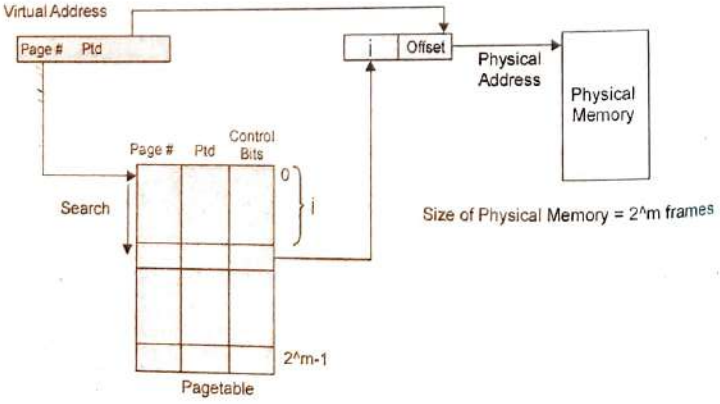


Fig. Inverted Page Table

Through inverted page table, the overhead of storing an individual page table for every process gets eliminated and only a fixed portion of memory is required to store the paging information of all the processes together. This technique is called as inverted paging as the indexing is done with respect to the frame number instead of the logical page number. Each entry in the page table contains the following fields.

- **Page number –** It specifies the page number range of the logical address.
- **Process id –** An inverted page table contains the address space information of all the processes in execution. Since two different processes can have similar set of virtual addresses, it becomes necessary in Inverted Page Table to store a process Id of each process to identify it's address space uniquely. This is done by using the combination of PId and Page Number. So this Process Id acts as an address space identifier and ensures that a virtual page for a particular process is mapped correctly to the corresponding physical frame.
- **Control bits –** These bits are used to store extra paging-related information. These include the valid bit, dirty bit, reference bits, protection and locking information bits.
- **Chained pointer –** It may be possible sometime that two or more processes share a part of main memory. In this case, two or more logical pages map to same Page Table Entry then a chaining pointer is used to map the details of these logical pages to the root page table.

The virtual address generated by the CPU contains the fields and each page table entry contains and the other relevant information required in paging related mechanism. When a memory reference takes place, this virtual address is matched by the memory-mapping unit and the Inverted Pagetable is searched to match and the corresponding frame number is obtained. If the match is found at the ith entry then the physical address of the process, is sent as the real address otherwise if no match is found then Segmentation Fault is generated.

Examples – The Inverted Pagetable and its variations are implemented in various systems like PowerPC, UltraSPARC and the IA-64 architecture. An implementation of Mach operating system on the RT-PC also uses this technique.

Q. 1. (d) Explain various file access methods. (5)

Ans. When file is used, information is read and accessed into computer memory and there are several ways to accesses these information of the file. Some system provide only one access method for files. Other system, such as those of IBM, support many access method, and choosing the right one for a particular application is a major design problem.

There are three ways to access a file into computer system: Sequential Access, Direct Access, Index sequential Method.

1. Sequential Access – It is the simplest access method. Information in the file *id* processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file. A read operation -*read next*- read the next position of the file and automatically advance a file pointer, which keep track I/O location. Similarly for the write *write next* append to the end of the file and advance to the newly written material.

Key points:

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file
- Such a method is reasonable for tape.

2. Direct Access – Another method is *direct access method* also known as *relati access method*. A filed-length logical record that allow program to read and write recor rapidly. in no particular order. The direct access is based on the disk model of a file sinc disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. A block number provided by the user to th operating system is normally a *relative block number*, the first relative block of the fil is 0 and then 1 and so on.

3. Index sequential method – It is the other method of accessing a file, which i built on the top of the direct access method these methods we construct an index for th file. The index, like an index in the back of a book, contains the pointer to the variou blocks. To find a record in the file, we first search the index and then by the help of pointe we access the file directly.

Key points:

- It is built on top of Sequential access.
- It control the pointer by using index.

Q. 1. (e) Explain how does a page size affect the performance of demand paging? (5)

Ans. • The number of frames is equal to the size of memory divided by the page-size. So and increase in page size means a decrease in the number of available frames.

• Having a fewer frames will increase the number of page faults because of the lower freedom in replacement choice.

• Large pages would also waste space by Internal Fragmentation.

• On the other hand, a larger page-size would draw in more memory per fault; so the number of fault may decrease if there is limited contention.

• Larger pages also reduce the number of TLB misses.

• Clearly, the smaller the page size, the lesser is the amount of internal fragmentation. To optimize the use of main memory, we would like to reduce internal fragmentation.

• On the other hand, smaller the page, the greater is the number of pages required per process which could mean that some portion of page tables of active processes must be in virtual memory, not in main memory. This eventually leads to double page fault for a single reference of memory.

If the number of page faults are more, then the rate of demand paging will increase leading to low performance.

Page Fault Rate

– $0 \le p \le 1.0$

– if $p = 0$,no page faults

– if $p = 1$, every reference is a fault

• Effective Access Time (EAT)

EAT = (1 – p) x memory access

+ p (page fault overhead

+ swap page out

+ swap page in

+ restart overhead)

Q. 2. (a) What do you mean by a process control block? Explain process state transition diagram with suitable transitions. (5)

Ans. Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.

It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.

Structure of the Process Control Block

The process control stores many data items that are needed for efficient process management. Some of these data items are explained with the help of the given diagram:

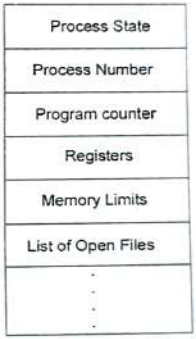


Fig. Process Control Block (PCB)

The following are the data items:

Process State

This specifies the process state i.e. new, ready, running, waiting or terminated.

Process Number

This shows the number of the particular process.

Program Counter

This contains the address of the next instruction that needs to be executed in the process.

Registers

This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.

List of Open Files

These are the different files that are associated with the process

CPU Scheduling Information

The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.

Memory Management Information

The memory management information includes the page tables or the segme tables depending on the memory system used. It also contains the value of the b registers, limit registers etc.

I/O Status Information

This information includes the list of I/O devices used by the process, the list of fil etc.

Accounting information

The time limits, account numbers, amount of CPU used, process numbers etc. a all a part of the PCB accounting information.

Location of the Process Control Block

The process control block is kept in a memory area that is protected from the norm user access. This is done because it contains important process information. Some the operating systems place the PCB at the beginning of the kernel stack for the proce as it is a safe location.

Process Transition Diagram:

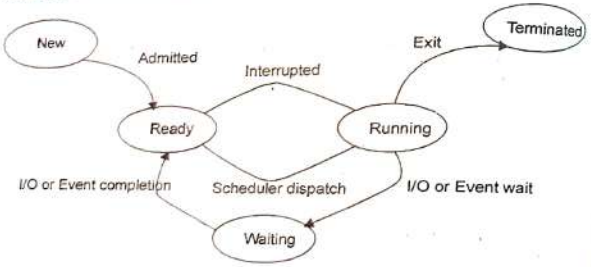


Fig. Process State Transition diagram.

Process can have one of the following five states at a time.

1. New state: A process that just has been created but has not yet been admitted t the pool of execution processes by the operating system. Every new operation which is requested to the system is known as the new born process.

2. Ready state: When the process is ready to execute but he is waiting for the CPU to execute then this is called as the ready state. After completion of the input and output the process will be on ready state means the process will wait for the processor to execute

3. Running state: The process that is currently being executed. When the proces is running under the CPU, or when the program is executed by the CPU, then this is called as the running process and when a process is running then this will also provid us some outputs on the screen.

4. Waiting or blocked state: A process that cannot execute until some event occurs or an I/O completion. When a process is waiting for some input and output operation then this is called as the waiting state and in this process is not under the execution instead the process is stored out of memory and when the user will provide the input and then this will again be on ready state.

5. Terminated state: After the completion of the process, the process will be automatically terminated by the CPU. So this is also called as the terminated state of the process. After executing the complete process the processor will also de allocate the memory which is allocated to the process. So this is called as the terminated process

Q. 2. (b) Given the following set of processes along with their data. Compute average waiting time for each process for preemptive SJF and Round Robin scheduling algorithms. (6.5)

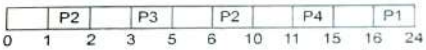
Processes	CPU bursts	Arrival time in ready queue
P1	8	0
P2	5	1
P3	2	2
P4	4	2

Assume context switching overhead is 1 time unit and time quantum used in Round Robin scheduling is 2 time units.

Ans.

Process	AT	BT	CT	TAT	WT
P1	0	8	24	24	16
P2	1	5	10	9	4
P3	2	2	5	3	1
P4	2	4	15	13	9

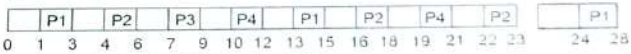
(1) Preemptive SJF (with unit context switch over head)



$$\text{A.W.T.} = \frac{16 + 4 + 1 + 9}{4}$$

$$= 7.5$$

(2) Round Robin (TQ = 2)



P	AT	BT	CT	TAT	WT
P1	0	8	28	28	20
P2	1	5	23	22	17
P3	2	2	9	7	5
P4	2	4	21	19	15

$$\text{A.W.T.} = \frac{15 + 5 + 17 + 20}{4} = 42$$

Q.3. (a) Explain with suitable example that rate monotonic and deadline monotonic algorithm of real time scheduling are identical if the periods of real time process are same to their corresponding relative deadlines. (6)

Ans. The Rate Monotonic scheduling algorithm is a simple rule that assigns priorities to different tasks according to their time period. That is task with smallest time period will have highest priority and a task with longest time period will have lowest priority for execution. As the time period of a task does not change so not its priority changes over time, therefore Rate monotonic is fixed priority algorithm. The priorities are decided before the start of execution and they does not change overtime.

Rate monotonic scheduling Algorithm works on the principle of preemption. Preemption occurs on a given processor when higher priority task blocked lower priority task from execution. This blocking occurs due to priority level of different tasks in a given task set. rate monotonic is a preemptive algorithm which means if a task with shorter period comes during execution it will gain a higher priority and can block or preemptive currently running tasks. In RM priorities are assigned according to time period. Priority of a task is inversely proportional to its timer period.

Task with lowest time period has highest priority and the task with highest period will have lowest priority. A given task is scheduled under rate monotonic scheduling Algorithm, if its satisfies the following equation:

$$\sum_{k=1}^{n} \frac{C_k}{T_k} \leq U_{RM} = n\left(2^{1/n} - 1\right)$$

where n is the number of tasks in a task set.

Deadline-monotonic priority assignment is a priority assignment policy used with fixed-priority pre-emptive scheduling.

With deadline-monotonic priority assignment, tasks are assigned priorities according to their deadlines. The task with the shortest deadline is assigned the highest priority. This priority assignment policy is optimal for a set of periodic or sporadic tasks which comply with the following restrictive system model:

1. All tasks have deadlines less than or equal to their minimum inter-arrival times (or periods).

2. All tasks have worst-case execution times (WCET) that are less than or equal to their deadlines.

3. All tasks are independent, and so do not block each other's execution (e.g., by accessing mutually exclusive shared resources).

4. No task voluntarily suspends itself.

5. There is some point in time, referred to as a critical instant, where all of the tasks become ready to execute simultaneously.

6. Scheduling overheads (switching from one task to another) are zero.

7. All tasks have zero release jitter (the time from the task arriving to it becoming ready to execute).

If restriction 7 is lifted, then "deadline minus jitter" monotonic priority assignment is optimal.

If restriction 1 is lifted, allowing deadlines greater than periods, then Audsley's optimal priority assignment algorithm may be used to find the optimal priority assignment.

Deadline monotonic priority assignment is not optimal for fixed priority non-pre-emptive scheduling.

A fixed priority assignment policy P is referred to as optimal if no task set exists which is schedulable using a different priority assignment policy which is not also schedulable using priority assignment policy P. Or in other words: Deadline-monotonic priority assignment (DMPA) policy is optimal if any process set, Q, that is schedulable by priority scheme W, is also schedulable by DMPA

Q. 3. (b) Write a solution to readers/writers problem using semaphore and justify its working. (6.5)

Ans. Readers writer problem is example of a classic synchronization problem. There are many variants of this problem, one of which is examined below.

The Problem Statement: There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are **reader** and **writer**. Any number of **readers** can read from the shared resource simultaneously, but only one **writer** can write to the shared resource. When a **writer** is writing data to the resource, no other process can access the resource. A **writer** cannot write to the resource if there are non zero number of readers accessing the resource at that time.

The Solution: From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

Here, we use one **mutex** m and a **semaphore** w. An integer variable read_count is used to maintain the number of readers currently accessing the resource. The variable read_count is initialized to 0. A value of 1 is given initially to m and w.

Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the read_count variable.

The code for the **writer** process looks like this:

```
while(TRUE)
{
wait(w);
/* perform the write operation */
signal(w);
}
```

And, the code for the **reader** process looks like this:

```
while(TRUE)
{
//acquire lock
wait(m);
read_count++;
if(read_count == 1)
wait(w);
//release lock
signal(m);
/* perform the reading operation */
// acquire lock
wait(m);
read_count– –;
if(read_count == 0)
signal(w);
// release lock
signal(m);
}
```

Here is the Code explained

• As seen above in the code for the writer, the writer just waits on the w semaphore until it gets a chance to write to the resource.

• After performing the write operation, it increments **w** so that the next writer can access the resource.

• On the other hand, in the code for the reader, the lock is acquired whenever the **read_count** is updated by a process.

• When a reader wants to access the resource, first it increments the **read_count** value, then accesses the resource and then decrements the **read_count** value.

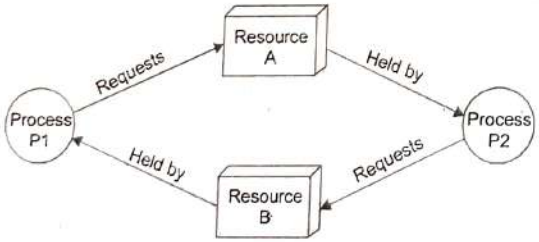
• The semaphore **w** is used by the first reader which enters the critical section and the last reader which exits the critical section.

• The reason for this is, when the first readers enters the critical section, the writer is blocked from the resource. Only new readers can access the resource now.

• Similarly, when the last reader exits the critical section, it signals the writer using the **w** semaphore because there are zero readers now and a writer can have the chance to access the resource.

Q. 4. (a) What do you mean by a circular wait condition in deadlock? Design a protocol which violates this condition and justify its worthliness. (6)

Ans. The circular wait condition states that there exists a chain of processes where each process is waiting for a resource that is being held by another resource. In this condition it does not necessarily mean that the process is holding onto a resource that the other is requesting, but rather that all processes are waiting for a resource that is being held by another process.



This condition can be avoided by using Resource Allocation Graph

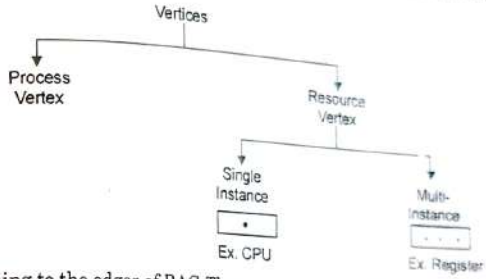
Resource allocation graph is explained as what is the state of the system in terms of **processes and resources**. Like how many resources are available, how many are allocated and what is the request of each process. Everything can be represented in terms of the diagram. One of the advantages of having a diagram is, sometimes it is possible to see a deadlock directly by using RAG, but then you might not be able to know that by looking at the table. But the tables are better if the system contains lots of process and Graph is better if the system contains less number of process and resource. We know that any graph contains vertices and edges. So RAG also contains vertices and edges. In RAG vertices are two type –

1. Process vertex – Every process will be represented as a process vertex. The process will be represented with a circle.

2. Resource vertex – Every resource will be represented as a resource vertex. It is also two type –

• **Single instance type resource –** It represents as a box, inside the box, there will be one dot.So the number of dots indicate how many instances are present of each resource type.

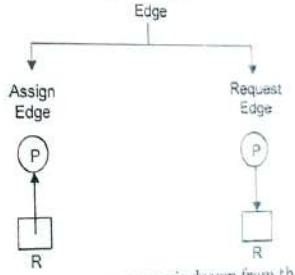
• **Multi-resource instance type resource –** It also represents as a box, inside the box, there will be many dots present.



Now coming to the edges of RAG.There are two types of edges in RAG –

1. Assign Edge – If you already assign a resource to a process then it is called Assign edge.

2. Request Edge – It means in future the process might want some resource to complete the execution, that is called request edge.



So, if a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

Example 1 (Single instances RAG) –

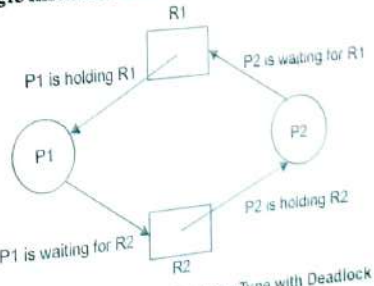


Fig. Single Instance Resource Type with Deadlock

If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.

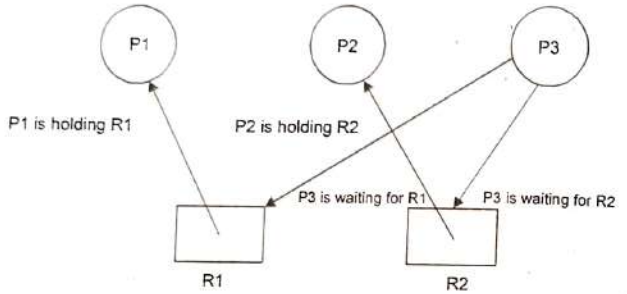


Fig. Single Instance Resource Type Without Deadlock

Here's another example, that shows Processes P1 and P2 acquiring resources R1 and R2 while process P3 is waiting to acquire both resources. In this example, there is no deadlock because there is no circular dependency. So cycle in single-instance resource type is the sufficient condition for deadlock.

Example 2 (Multi-instances RAG) –

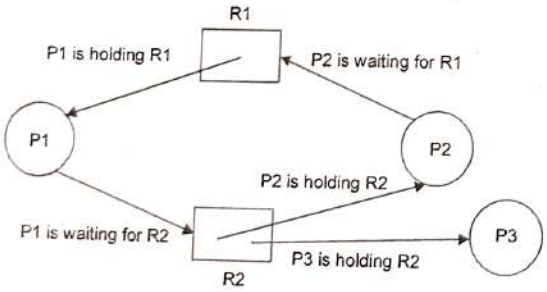


Fig. Multi Instance without deadlock

From the above example, it is not possible to say the RAG is in a safe state or in an unsafe state. So to see the state of this RAG, let's construct the allocation matrix and request matrix.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

• The total number of processes are three; P1, P2 & P3 and the total number of resources are two; R1 & R2.

Allocation matrix –

• For constructing the allocation matrix, just go to the resources and see to which process it is allocated.

• R1 is allocated to P1, therefore write 1 in allocation matrix and similarly, R2 is allocated to P2 as well as P3 and for the remaining element just write 0.

Request matrix –

• In order to find out the request matrix, you have to go to the process and see the outgoing edges.

• P1 is requesting resource R2, so write 1 in the matrix and similarly, P2 requesting R1 and for the remaining element write 0.

So now available resource is = (0, 0).

Checking deadlock (safe or not) –

Available = 0 0 (As P3 does not require any extra resource to complete the
P3 0 1 execution and after completion P3 release its own resource)

New Available = 0 1 (As using new available resource we can satisfy the requirement of
P1 1 0 process P1 and P1 also release its previous resource)

New Available = 1 1 (Now easily we can satisfy the requirement of process P2)
P2 0 1

New Available = 1 2

So, there is no deadlock in this RAG.Even though there is a cycle, still there is no deadlock.Therefore in multi-instance resource cycle is not sufficient condition for deadlock.

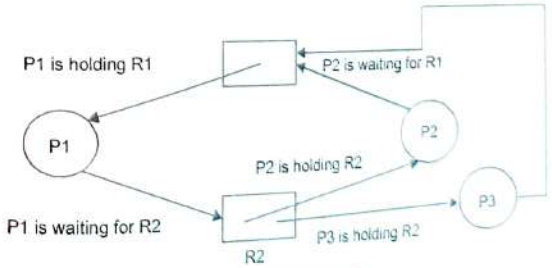


Fig. Multi Instances With Deadlock

Above example is the same as the previous example except that, the process P3 requesting for resource R1. So the table becomes as shown in below.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

So, the Available resource is = (0, 0), but requirement are (0, 1), (1, 0) and (1, 0).So you can't fulfill any one requirement. Therefore, it is in deadlock.

Therefore, every cycle in a multi-instance resource type graph is not a deadlock, if there has to be a deadlock, there has to be a cycle. So, in case of RAG with multi-instance resource type, the cycle is a necessary condition for deadlock, but not sufficient.

Q. 4. (b) Write an algorithm for deadlock detection for multiple instances of resources. (6.5)

Ans. The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following **Data structures** are used to implement the Banker's Algorithm:

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

• It is a 1-d array of size **'m'** indicating the number of available resources of each type.

• Available[j] = k means there are **'k'** instances of resource type $\mathbf{R_j}$

Max :

• It is a 2-d array of size **'n*m'** that defines the maximum demand of each process in a system.

• Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation :

• It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.

• Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need :

• It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.

• Need [i, j] = k means process $\mathbf{P_i}$ currently allocated **'k'** instances of resource type $\mathbf{R_j}$

• Need [i, j] = Max [i, j] – Allocation [i, j]

Allocation specifies the resources currently allocated to process Pi and Needi specifies the additional resources that process Pi may still request to complete its task.

Banker's algorithm consist of Safety algorithm and Resource request algorithm

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4...n

(2) Find an i such that both

(a) Finish[i] = false

(b) Needi <= Work if no such i exists goto step (4)

(3) Work = Work + Allocation

Finish[i] = true

goto step (2)

(4) if finish [i] = true for all *i*

then the system is in a safe state

Resource-Request Algorithm

Let Requesti be the request array for process Pi. Requesti [j] = k means process P_i wants k instances of resource type Rj. When a request for resources is made by process P_i, the following actions are taken:

(1) If $Request_i$ <= $Need_i$ Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

(2) If $Request_i$ <= Available Goto step (3); otherwise, P_i must wait, since the resources are not available.

(3) Have the system pretend to have allocated the requested resources to process P_i by modifying the state as follows: Available = Available – $Request_i$

$Allocation_i$ = $Allocation_i$ + $Request_i$ $Need_i$ = $Need_i$ – $Request_i$

Q. 5. Explain memory address binding method using paging. How the concept of associative memory is useful in the context? If main memory access time is 50 msec and associative memory access time is 20 nsec what will be the hit ratio if the maximum achievable improvement in main memory access time is 40%? (12.5)

Ans. Computer memory uses both logical addresses and physical addresses. Address binding allocates a physical memory location to a logical pointer by associating a physical address to a logical address, which is also known as a virtual address. Address binding is part of computer memory management and it is performed by the operating system on behalf of the applications that need access to memory.

Address binding is the process of mapping the program's logical or virtual addresses to corresponding physical or main memory addresses. In other words, a given logical address is mapped by the MMU (Memory Management Unit) to a physical address.

• **CPU generates the logical or virtual address** for an instruction/data to be fetched from RAM.

• The logical address undergoes translation by the MMU or address translation unit in particular.

• The output of this process is the appropriate physical address or the location of code/data in RAM.

The above described three-step process is also known as **execution or run-time address binding** where each physical memory reference is resolved only when the reference is made at run-time. Until a reference is made, address resolution or binding does not happen. This type of binding requires the compiler to generate **relocatable or offset based addresses from the source code.** The exact manner of carrying out the address mapping is dependent on the memory management scheme employed by the operating system. I will now briefly describe address binding in the context of various memory management schemes. The description of the schemes themselves is brief as the **focus is on address binding.**

Let's take the case of **contiguous memory allocation.** In this method, a process will occupy a contiguous main-memory area starting at some location "L" and extending to "L+X" where X is a byte offset relative to L. In this scheme, **address binding happens**

through a set of 2 registers — relocation/base register and **limit register**. In our example, relocation/base register will have address "L" as its value, and limit register will have offset "X" as its value. The purpose of these registers is two-fold:

- **Memory protection** - References made are checked if they lie within the process's address space or the contiguous memory area occupied by the process.
- **Address conversion** - Once the logical address reference is verified, it is simply added to the value in base or relocation register. The added value is the real physical address.

Next is paging. Contiguous memory allocation suffers from the problem of external fragmentation. Paging allows the process to be allocated physical memory wherever it is available. In short, with paging , the physical address space of a process can be non-contiguous. The main memory is divided into fixed size chunks or frames, and the logical memory is partitioned into fixed size blocks or pages(usually 4K in size).

In a paging based system, address binding/mapping happens through page tables. Page table maps a virtual page number to a physical frame number in main-memory. When the process is loaded into memory, the allocation is done in the form of frames or physical pages wherever they are available. Along with this, the page table is also populated with correct frame number.

Now coming back to address binding, the logical address generated by CPU is divided into two parts:

- Page Number (aka virtual page number)
- Offset or displacement within the page.

Page number is used to index into the page table, to get the corresponding physical frame/page number. The physical page number is them combined with the offset to get the complete physical address.

Next is **segmentation.** In this scheme, the process is allocated memory in the form of segments:

Code/Text Segment - containing instructions to be executed

Data Segment - containing the program's global and static data. Heap - space used to service dynamic memory allocation requests for the process.

Stack - space to be used during function calls.

All these **segments together form the physical address space of the process**. Logical to physical address mapping is done through a segment table. Each segment in memory has a segment number assigned by the loader. A **segment table** is used to keep the following information about each segment.

- **Segment Limit** - The offset range within the segment.
- **Segment Base** - The starting address of each segment in memory.

Logical address generated by the CPU is divided into two parts:

- Segment number.
- Offset or displacement within the segment.

Segment number is used to index into the segment table to get the values of segment limit and base. First we check that offset is within the range [0, limit]. If so, value of segment base is added to the offset to get the real physical memory address within the segment.

I hope the above explains the execution-time address binding in the context of different memory management schemes.

Apart from execution-time, address binding can be further classified into two different types:

Load-time: logical to physical address resolution happens at the time executable/program is loaded into main memory. Subsequently, all the logical addresses generated by the CPU are identical to physical addresses. **In both load and execution time binding, the executable has relocatable addresses for code/data.** The difference is that in this type of binding, loader translates these relocatable addresses to actual physical addresses whereas this translation happens at run-time or dynamically in execution-time binding.

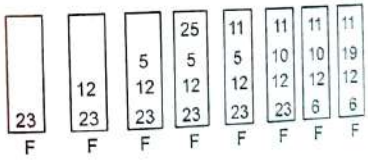
Compile-time: Usually at compile time we don't know where exactly the program is going to reside in the physical memory. But in case it is known, the real physical addresses can be generated at the compile time. Again, the logical addresses are same as physical memory addresses. Compile-time binding makes it virtually impossible to have two different programs (with binding done at compile-time) resident in the memory at the same time.

Q. 6. (a) Given the following set of page reference string, compute number of page faults for Optimal, LRU and second change LRU page replacement algorithms. Assume that four frames have been allocated to the process.

23, 12, 5, 25, 12, 11, 10, 10, 23, 12, 10, 11, 12, 6, 19

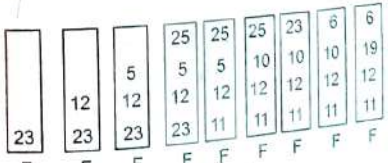
Ans. 23, 12, 5, 25, 12, 11, 10,10, 23, 12, 10, 11, 12, 6, 19

→ optimal page replacement (4 Frames)



8 Page Faults

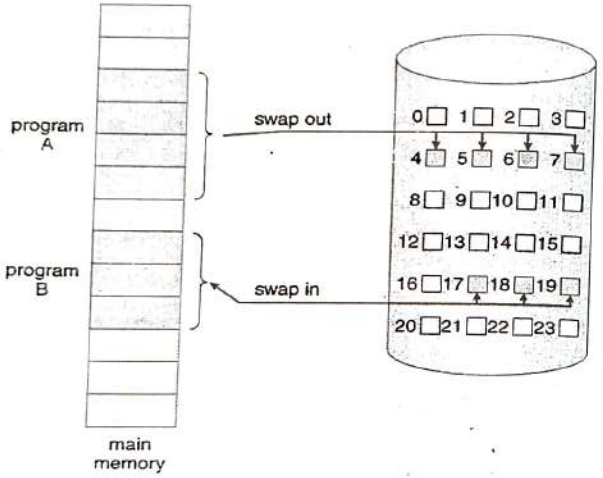
→ LRU



9 PAGE Faults

Q. 6. (b) Explain with suitable example that a program structure affects the performance of demand paging in virtual memory. **(6)**

Ans. Demand Paging The basic idea behind *demand paging* is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them. (on demand.) This is termed a ***lazy swapper***, although a ***pager*** is a more accurate term.

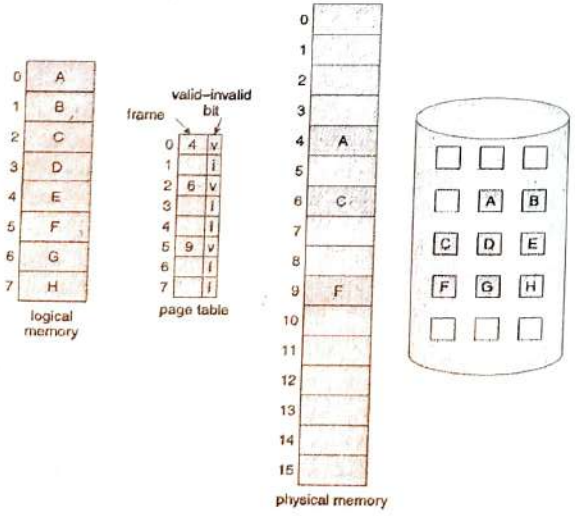


Transfer of a paged memory to contiguous disk space

• The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need (right away.)

• Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. (The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive.)

• If the process only ever accesses pages that are loaded in memory (***memory resident*** pages), then the process runs exactly as if all the pages were loaded in to memory.



Page table when some pages are not in main memory.

• On the other hand, if a page is needed that was not originally loaded up, then a ***page fault trap*** is generated, which must be handled in a series of steps:

1. The memory address requested is first checked, to make sure it was a valid memory request.

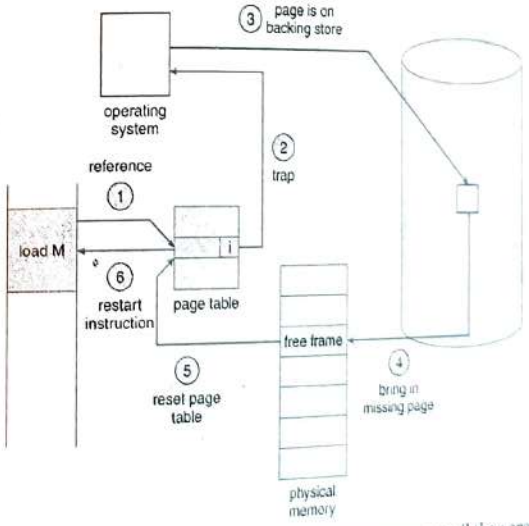
2. If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.

3. A free frame is located, possibly from a free-frame list.

4. A disk operation is scheduled to bring in the necessary page from disk. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)

5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.

6. The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU.)



• In an extreme case, NO pages are swapped in for a process until they are requested by page faults. This is known as ***pure demand paging***.

• In theory each instruction could generate multiple page faults. In practice this is very rare, due to ***locality of reference***, covered in section 9.6.1.

• The hardware necessary to support virtual memory is the same as for paging and swapping: A page table and secondary memory.

• A crucial part of the process is that the instruction must be restarted from scratch once the desired page has been made available in memory. For most simple instructions this is not a major difficulty. However there are some architectures that allow a single instruction to modify a fairly large block of data, (which may span a page boundary), and if some of the data gets modified before the page fault occurs, this could cause

problems. One solution is to access both ends of the block before executing the instruction, guaranteeing that the necessary pages get paged in before the instruction begins.

Performance of Demand Paging

• Obviously there is some slowdown and performance hit whenever a page fault occurs and the system has to go get it from memory, but just how big a hit is it exactly?

• Suppose that a normal memory access requires 200 nanoseconds, and that servicing a page fault takes 8 milliseconds. (8,000,000 nanoseconds, or 40,000 times a normal memory access.) With a ***page fault rate*** of p, (on a scale from 0 to 1), the effective access time is now:

(1 - p) * (200) + p * 8000000

= 200 + 7,999,800 * p

which ***clearly*** depends heavily on p! Even if only one access in 1000 causes a page fault, the effective access time drops from 200 nanoseconds to 8.2 microseconds, a slowdown of a factor of 40 times. In order to keep the slowdown less than 10%, the page fault rate must be less than 0.0000025, or one in 399,990 accesses.

• A subtlety is that swap space is faster to access than the regular file system, because it does not have to go through the whole directory structure. For this reason some systems will transfer an entire process from the file system to swap space before starting up the process, so that future paging all occurs from the (relatively) faster swap space.

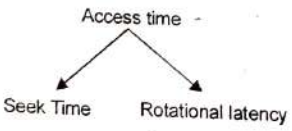
• Some systems use demand paging directly from the file system for binary code (which never changes and hence does not have to be stored on a page operation), and to reserve the swap space for data segments that must be stored. This approach is used by both Solaris and BSD Unix.

Q.7. (a) Explain SSTF and SCAN disk scheduling algorithms with suitable example. What are different parameters on which the performance of disk scheduling algorithms algorithms does depend? **(6.5)**

Ans. Disk scheduling

→ For using H/W efficiently

→ We need to achieve fast Access time and large disk bandwidth



$$\text{Bandwidth} = \frac{\text{Total no. of bytes transfered}}{\text{difference of first request} - \text{completion of last transfer}}$$

Access time and B.W can be improved by managing the order in which disk requests are serviced.

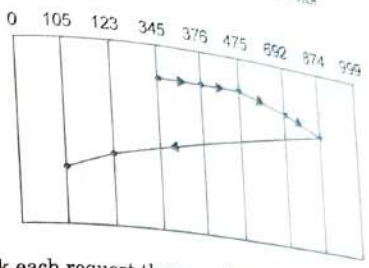
Disk with 1000 cylinders, 0 - 999, Calculate seek time, Assume last request was received At 345 (∴ the head is currently At 345). Head moves from 345 to 0

Track requests: 123, 874, 692, 475, 105, 376

(2) SSTF (shortest seek time first)

Same question As above

The request which will result in less seek time is granted first.



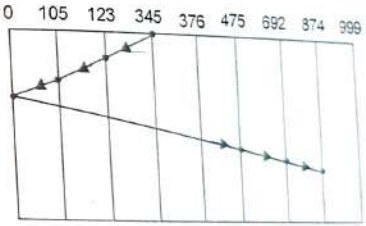
From head, check each request the one which has less difference will be granted first.

Total seek time = (376 – 345) + (475 – 376)
+ (692 – 475) + (874 – 692)
+ (874 – 123) + (123 – 105)
= 1298

SCAN (Elevator scheduling)

• From the head either go to the highest point (999) of the disk while processing the requests in the way or to the lowest point (0) doing the same.

• But service the requst only till the last request don't go the last extreme point (0 or 999) At the END



extreme point is 999 (lost request is left) if 345 moves in direction of 999 first instead of 0 then 0 will be left.

Total seek time = (345 – 123) + (123 – 105) +
(105 – 0) + (376 – 0) +
(475 – 376) + (692 – 475)
(874 – 692)
= 1219

Q. 7. (b) What do you mean by data stripping? How is RAID concept useful in disk reliability? **(6)**

Ans. Data striping is the technique of segmenting logically sequential data, such as a file, so that consecutive segments are stored on different physical storage devices.

Striping is useful when a processing device requests data more quickly than a single storage device can provide it. By spreading segments across multiple devices which can be accessed concurrently, total data throughput is increased. It is also a useful method for balancing I/O load across an array of disks. Striping is used across disk drives in redundant array of independent disks (RAID) storage.

Reliability is the ability of the disk system to accommodate a single- or multi-disk failure and still remain available to the users. Performance is the ability of the disks to efficiently provide information to the users.

Adding redundancy almost always increases the reliability of the disk system. The most common way to add redundancy is to implement a Redundant Array of Inexpensive Disks (RAID).

There are two types of RAID:

Hardware — The most commonly used hardware RAID levels are: RAID 0, RAID 1, RAID 5, and RAID 10. The main differences between these RAID levels focus on reliability and performance as previously defined.

Software — Software RAID can be less expensive. However, it is almost always much slower than hardware RAID, because it places a burden on the main system CPU to manage the extra disk I/O.

RAID 0 (Disk striping):

RAID 0 splits data across any number of disks allowing higher data throughput. An individual file is read from multiple disks giving it access to the speed and capacity of all of them. This RAID level is often referred to as striping and has the benefit of increased performance. However, it does not facilitate any kind of redundancy and fault tolerance as it does not duplicate data or store any parity information (more on parity later). Both disks appear as a single partition, so when one of them fails, it breaks the array and results in data loss. RAID 0 is usually implemented for caching live streams and other files where speed is important and reliability/data loss is secondary.

RAID 1 (Disk Mirroring):

RAID 1 writes and reads identical data to pairs of drives. This process is often called data mirroring and it's a primary function is to provide redundancy. If any of the disks in the array fails, the system can still access data from the remaining disk(s). Once you replace the faulty disk with a new one, the data is copied to it from the functioning disk(s) to rebuild the array. RAID 1 is the easiest way to create failover storage.

RAID 5 (Striping with parity):

RAID 5 stripes data blocks across multiple disks like RAID 0, however, it also stores parity information (Small amount of data that can accurately describe larger amounts of data) which is used to recover the data in case of disk failure. This level offers both speed (data is accessed

from multiple disks) and redundancy as parity data is stored across all of the disks. If any of the disks in the array fails, data is recreated from the remaining distributed data and parity blocks. It uses approximately one-third of the available disk capacity for storing parity information.

RAID 6 (Striping with double parity):

Raid 6 is similar to RAID 5, however, it provides increased reliability as it stores an extra parity block. That effectively means that it is possible for two drives to fail at once without breaking the array.

RAID 10 (Striping + Mirroring):

RAID 10 combines the mirroring of RAID 1 with the striping of RAID 0. Or in other words, it combines the redundancy of RAID 1 with the increased performance of RAID 0. It is best suitable for environments where both high performance and security is required.

Q. 8. What do you mean by various file allocation methods? Explain advantages and disadvantages of each method. (12.5)

Ans. The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide:

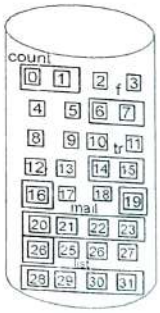
- Efficient disk space utilization.
- Fast access to the file blocks.

All the three methods have their own advantages and disadvantages as discussed below:

1. Contiguous Allocation: In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,....b+n-1*. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

The *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies 19, 20, 21, 22, 23, 24 blocks.



Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Advantages:

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

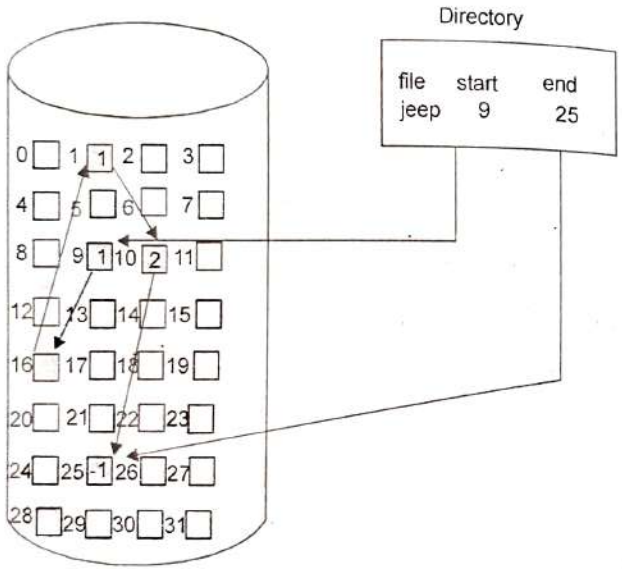
Disadvantages:

- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

• Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

2. Linked List Allocation: In this scheme, each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

• This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.

• This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

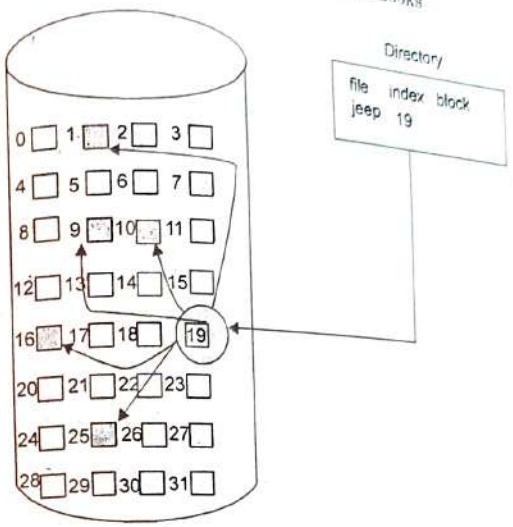
• Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.

• It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.

• Pointers required in the linked allocation incur some extra overhead.

3. Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:



Advantages:

• This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.

• It overcomes the problem of external fragmentation.

Disadvantages:

• The pointer overhead for indexed allocation is greater than linked allocation.

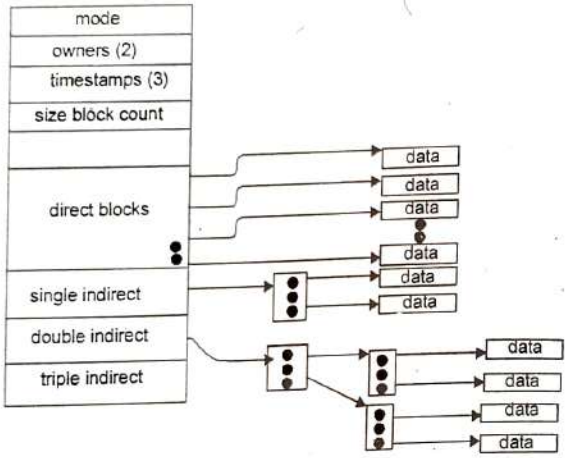
• For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

For files that are very large, single index block may not be able to hold all the pointers. Following mechanisms can be used to resolve this:

1. Linked scheme: This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.

2. Multilevel index: In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.

3. Combined Scheme: In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file *as shown in the image below.* The first few of these pointers in Inode point to the **direct blocks** i.e the pointers contain the addresses of the disk blocks that contain data of the file. The next few pointers point to indirect blocks. Indirect blocks may be **single indirect**, double indirect or triple indirect. **Single Indirect block** is the disk block that does not contain the file data but the disk address of the blocks that contain the file data. Similarly, **double indirect blocks** do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.



Q. 9. Write short note on the following:

(a) File Access Methods. (3)

Ans. There are three ways to access a file into computer system: Sequential Access, Direct Access, Index sequential Method.

1. Sequential Access – It is the simplest access method. Information in the file id processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file. A read operation *-read next-* read the next position of the file and automatically advance a file pointer, which keep track I/O location. Similarly for the write*write next* append to the end of the file and advance to the newly written material.

2. Direct Access – Another method is *direct access method* also known as *relative access method*. A filed-length logical record that allow program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for direct access file.

A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on.

3. Index sequential method – It is the other method of accessing a file. Which is built on the top of he direct access method. these methods we construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

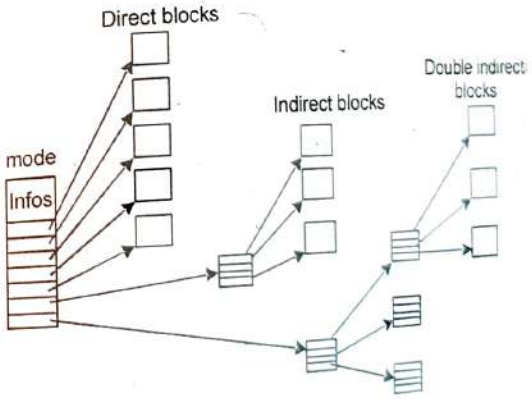
Q. 9. (b) Inode in UNIX operating system. (3)

Ans. I-node is a data structure that keeps track of all the information about a file. You keep your information in a file and the OS stores the information about a file in an I-node.

Data structures that contain information about files in UNIX file systems that are created when a file system is created. Each file has an I-node and is identified by an I-node number (I-number) in the file system where it resides. I-nodes provide important information on files such as user and group ownership, access mode (read, write, execute permissions) and type.

The I-node contains the following pieces of information:

- Mode/permission (protection)
- Owner ID
- Group ID
- Size of file
- Number of hard links to the file
- Time last accessed
- Time last modified
- Time I-node last modified

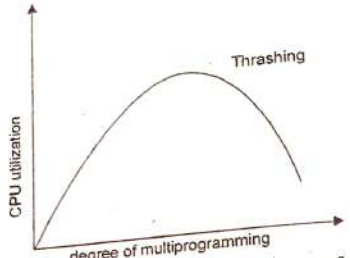


When a file is created inside a directory then the file-name and Inode number are assigned to file. These two entries are associated with every file in a directory. The user might think that the directory contains the complete file and all the extra information related to it but this might not be the case always. So we see that a directory associates a file name with its Inode number.

When a user tries to access the file or any information related to the file then he/she uses the file name to do so but internally the file-name is first mapped with its Inode number stored in a table. Then through that Inode number the corresponding Inode is accessed. There is a table (Inode table) where this mapping of Inode numbers with the respective Inodes is provided. (3.5)

Q. 9. (c) Thrashing

Ans. Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilization would fall drastically. The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory thereby increasing the degree of multiprogramming. This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing. (3)

Q. 9. (d) Time sharing system.

Ans. Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

END TERM EXAMINATION [MAY–JUNE 2019]
SECOND SEMESTER [MCA]
OPERATING SYSTEM [MCA-106]

Time : 3 hrs. **M.M. : 75**

Note :- *Attempt five questions in all including Q. No. 1 which is compulsory. Select one question from each unit.*

Q.1. (a) Operating system is also called Resource manager. Why ? (2.5)

Ans. A Computer system has many resources (hardware and software), which may required to complete a task. The commonly required resources are input/output devices, Memory; file storage space, CPU time and so on. The operating system acts as the manager of these resources and allocates them to specific programs and users as necessary for their tasks. Therefore we can say operating system as a resource allocator.

When a number of computers connected through a network and more than one computer trying for a common printer, or a common resource, then the operating system follow some order and manage the resources in an effective manner. In addition, users often need to share not only hardware, but information (files, databases, etc.) as well. In short, this view of the operating system holds that its primary task is to keep track of who is using which resource, to grant resource requests, to account for usage, and to mediate conflicting requests from different programs and users.

Resource management includes sharing resources in two ways: In time and in space. When a resource is time shared, different programs or users take turns using it. First one of them gets to use the resource, then another, and so on. For example, with only one CPU and multiple programs that want to run on it, the operating system first allocates the CPU to one program, and then after it has run long enough, another one gets to use the CPU, then another, and then eventually the first one again. Determining how the resource is time multiplexed who goes next and for how longis the task of the operating system. Another example of time sharing is sharing the printer. When multiple print jobs are queued up for printing on a single printer, a decision has to be made about which one is to be printed next.

The other kind of sharing is space sharing. Instead of the customers taking turns, each one gets part of the resource. For example, main memory is normally divided up among several running programs, so each one can be resident at the same time (for example, in order to take turns using the CPU). Assuming there is enough memory to hold multiple programs, it is more efficient to hold several programs in memory at once rather than give one of them all of it, especially if it only needs a small fraction of the total. Of course, this raises issues of fairness, protection, and so on, and it is up to the operating system to solve them. Another resource that is space shared is the (hard) disk. In many systems a single disk can hold files from many users at the same time. Allocating disk space and keeping track of who is using which disk blocks is a typical operating system resource management task.

Q.1. (b) Define preemption and non-preemption with the help of example. (2.5)

Ans. Preemption as used with respect to operating systems means the ability of the operating system to preempt (that is, stop or pause) a currently scheduled task in favour of a higher priority task. The resource being scheduled may be the processor or I/O, among others. Preemptive scheduling allows a process to be interrupted in the middle of its execution, taking the CPU away and allocating it to another process. Non-preemption ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst time.

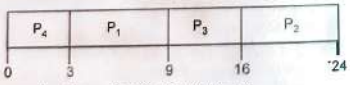
Non pre-Emptive Vs Preemptive Scheduling:

- **Non-Preemptive:** Non-preemptive algorithms are designed so that once a process enters the running state(is allowed a process), it is not removed from the processor until it has completed its service time (or it explicitly yields the processor). Context_switch is called only when the process terminates or blocks.
- **Preemptive:** Preemptive algorithms are driven by the notion of prioritized computation. The process with the highest priority should always be the one currently using the processor. If a process is currently using the processor and a new process with a higher priority enters, the ready list, the process on the processor should be removed and returned to the ready list until it is once again the highest-priority process in the system

Example of SJF non preemptive

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- SJF scheduling chart

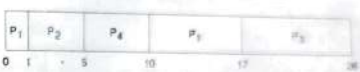


- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF PREEMPTIVE

PROCESS	BURST	ARRIVAL
P1	8	0
P2	4	1
P3	9	2
P4	5	3

- Average waiting time: (9 + 0 + 15 + 2)/4 = 6.5



- P1 = 10 – 1 = 9 – 0 = 9
- P2 = 1 – 1 = 0
- P3 = 17 – 2 = 15
- P4 = 5 – 3 = 2

Q.1. (c) Describe process Scheduling and process states. **(2.5)**

Ans. The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating system. Such operating systems allow more than one process to be loaded into the executable memory at a time and loaded process shares the CPU using time sharing.

The states of a process are:

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

2 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.

3 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.

4 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.

5 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.

Q.1. (d) What are cooperating processes ? **(2.5)**

Ans. A process is said to be a cooperating process if it can affect or be affected by **other** processes in the system. A process that shares data with other processes is cooperating.

Advantages of Cooperating Processes:

- **Information Sharing:** Several users may which to share the same information e.g. a shared file. The O/S needs to provide a way of allowing concurrent access.
- **Computation Speedup:** Some problems can be solved quicker by sub-dividing it into smaller tasks that can be executed in parallel on several processors.

• **Modularity:** The solution of a problem is structured into parts with well-defined interfaces, and where the parts run in parallel.

• **Convenience:** A user may be running multiple processes to achieve a single goal, or where a utility may invoke multiple components, which interconnect via a pipe structure that attaches the stdout of one stage to stdin of the next etc.

Q.1. (e) Mention the necessary conditions for a deadlock to occur. (2.5)

Ans. Necessary conditions for Deadlock to occur:

1. Mutual exclusion: The resources involved must be unshareable; otherwise, the processes would not be prevented from using the resource when necessary.

2. Hold and wait or partial allocation : The processes must hold the resources they have already been allocated while waiting for other (requested) resources. If the process had to release its resources when a new resource or resources were requested, deadlock could not occur because the process would not prevent others from using resources that it controlled.

3. No pre-emption : The processes must not have resources taken away while that resource is being used. Otherwise, deadlock could not occur since the operating system could simply take enough resources from running processes to enable any process to finish.

4. Resource waiting or Circular wait : A circular chain of processes, with each process holding resources which are currently being requested by the next process in the chain, cannot exist. If it does, the cycle theorem (which states that "a cycle in the resource graph is necessary for deadlock to occur") indicated that deadlock could occur. (2.5)

Q.1. (f) What is Belady Anomaly?

Ans. The Belady's anomaly occurs in case of the FIFO page replacement policy in the OS.

When FIFO is used, the numbers of page frames are increased. The frames that are required by the program varies in a large range (due to large no of pages) as a result of this the number of page faults increases with the number of frames.This anomaly doesn't occur in the LRU(least recently used) scheduling algorithm. (2.5)

Q.1. (g) Define internal and external Fragmentation.

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

Fragmentation is of two types –

External fragmentation: Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

Internal fragmentation: Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process. (2.5)

Q.1. (h) Compare program threats and system threats.

Ans. Program Threats: Operating system's processes and kernel do the designated task as instructed. If a user program made these process do malicious tasks, then it is known as Program Threats. One of the common example of program

threat is a program installed in a computer which can store and send user credentials via network to some hacker.

Following is the list of some well-known program threats.

• Trojan Horse – Such program traps user login credentials and stores them to send to malicious user who can later on login to computer and can access system resources.

• Trap Door – If a program which is designed to work as required, have a security hole in its code and perform illegal action without knowledge of user then it is called to have a trap door.

• Logic Bomb – Logic bomb is a situation when a program misbehaves only when certain conditions met otherwise it works as a genuine program. It is harder to detect.

• Virus – Virus as name suggest can replicate themselves on computer system. They are highly dangerous and can modify/delete user files, crash systems. A virus is generally a small code embedded in a program. As user accesses the program, the virus starts getting embedded in other files/ programs and can make system unusable for user

System Threats: System threats refers to misuse of system services and network connections to put user in trouble. System threats can be used to launch program threats on a complete network called as program attack. System threats creates such an environment that operating system resources/ user files are misused. Following is the list of some well-known system threats.

• Worm- Worm is a process which can choked down a system performance by using system resources to extreme levels. A Worm process generates its multiple copies where each copy uses system resources, prevents all other processes to get required resources. Worms processes can even shut down an entire network.

• Port Scanning - Port scanning is a mechanism or means by which a hacker can detects system vulnerabilities to make an attack on the system.

• Denial of Service- Denial of service attacks normally prevents user to make legitimate use of the system. For example, a user may not be able to use internet if denial of service attacks browser's content settings. (2.5)

Q.1. (i) Explain cryptography in OS ?

Ans. Cryptography is technique of securing information and communications through use of codes so that only those person for whom the information is intended can understand it and process it. Thus preventing unauthorized access to information. The prefix "crypt" means "hidden" and suffix "graphy" means "writing".

In Cryptography, the techniques which are use to protect information are obtained from mathematical concepts and a set of rule based calculations known as algorithms to convert messages in ways that make it hard to decode it. These algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on internet and to protect confidential transactions such as credit card and debit card transactions.

Techniques used For Cryptography: In today's age of computers cryptography is often associated with the process where an ordinary plain text is converted to cipher text which is the text made such that intended receiver of the text can only decode it and hence this process is known as encryption. The process of conversion of cipher text to plain text this is known as decryption. (2.5)

Q.1. (j) What are different file attributes and file operations in a typical file system.

Ans. File Attributes: A file is named for the convenience of its human users and is referred by its name. a name usually a string of characters such as stdio.h. a file has

certain other attributes which may carry from one O.S. to another but typically consist of these.

• Name :- The symbolic file name is only the information kept in human readable form. If is usually a string of character such as stdio.h. it must be uniquely.

• Type :- This information is used by those system that support different types.

• Location :- This information is a pointers to a device and the location of the file on that device.

• Size :- The current size of file (bytes, words or blocks) and possibly the maximum size allowed are included in this attributes.

• Protection :- Access control info controls who can do reading writing execution and so on.

• Time date and user identification :- This information may be kept for creation, last modification and last use. These data can be use full for protection and security.

File Operation: A file is an abstract data type. To define a file properly we need to consider the operations that can be performed on file.

There are six basic operations as follows.

i. Creating a file: Two steps are necessary to create a file first space in the file system must be found. Second an entry for the new file must be made in the directory.

ii. Writing a file: To write a file we need the name of the file and information to be written to the file. By name system search directory for location. The system must keep a write pointer of the location for next entry. This pointer is updated automatically.

iii. Reading a file: To read from a file we need name and location. The system needs to keep a read pointers to location in the file where the next read is to take place. Once the read has takes place once the read has takes place the read pointers is up dated. Generally a file is either read or write. Most system keep only onecurrent file position pointer for both (read and write) purpose for saving the space and reducing the system complexity.

iv. Searching: Here directory searched and pointer is set to a given value. If does not need any input/output. This operation is also known as a file seek (searching)

v. Deleting a File: To delete file we search the directory for the named file and release all file space (so that is can be reused by other files) and erase the directory entry.

vi. Truncating a file: Here a user wants the attributes of a file to remain the same. But wants to erase the contents of the file.

UNIT - I

Q.2. (a) What is operating system structure? Explain the different types of operating system with merits and demerits. (6.5)

Ans. Operating System Structure: The design of an operating system architecture traditionally follows the *separation of concerns* principle. This principle suggests structuring the operating system into relatively independent parts that provide simple individual features, thus keeping the complexity of the design manageable.

Monolithic Systems: A monolithic design of the operating system architecture makes no special accommodation for the special nature of the operating system. Although the design follows the separation of concerns, no attempt is made to restrict the privileges granted to the individual parts of the operating system. The entire operating system executes with maximum privileges. The communication overhead inside the monolithic operating system is the same as the communication overhead inside any other software, considered relatively low.

CP/M and DOS are simple examples of monolithic operating systems. Both CP/M and DOS are operating systems that share a single address space with the applications.

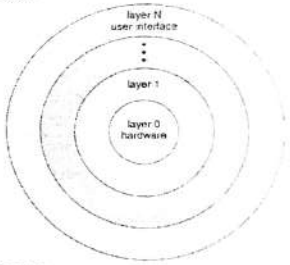
Layered Approach:

• Another approach is to break the OS into a number of smaller layers, each of which rests on the layer below it, and relies solely on the services provided by the next lower layer.

• This approach allows each layer to be developed and debugged independently, with the assumption that all lower layers have already been debugged and are trusted to deliver proper services.

• The problem is deciding what order in which to place the layers, as no layer can call upon the services of any higher layer, and so many chicken-and-egg situations may arise.

• Layered approaches can also be less efficient, as a request for service from a higher layer has to filter through all lower layers before it reaches the HW, possibly with significant processing at each step.



Types of Operating Systems:

1. Batch Operating System: This type of operating system does 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.

Advantages of Batch Operating System:

• It is very difficult to guess or know the time required by any job to complete. Processors of the batch systems know how long the job would be when it is in queue

• Multiple users can share the batch systems

• The idle time for batch system is very less

• It is easy to manage large work repeatedly in batch systems

Disadvantages of Batch Operating System:

• The computer operators should be well known with batch systems

• Batch systems are hard to debug

• It is sometime costly

• The other jobs will have to wait for an unknown time if any job fails

2. Time-Sharing Operating Systems: Each task is 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.

Advantages of Time-Sharing OS:

- Each task gets an equal opportunity
- Less chances of duplication of software
- CPU idle time can be reduced

Disadvantages of Time-Sharing OS:

- Reliability problem
- One must have to take care of security and integrity of user programs and data
- Data communication problem

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 distributed systems. These system's processors differ in size and function. 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.

Advantages of Distributed Operating System:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

Disadvantages of Distributed Operating System:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which are used are not well defined yet
- These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet

4. Network Operating System: These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions. These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network.

Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated to the system
- Server access are possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on central location for most operations
- Maintenance and updates are required regularly

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.

- **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.
- **Soft Real-Time Systems:** These OSs are for applications where for time-constraint is less strict.

Advantages of RTOS:

- Maximum Consumption: Maximum utilization of devices and system,thus more output from all the resources
- Task Shifting: Time assigned for shifting tasks in these systems are very less. For example in older systems it takes about 10 micro seconds in shifting one task to another and in latest systems it takes 3 micro seconds.
- Focus on Application: Focus on running applications and less importance to applications which are in queue.
- Real time operating system in embedded system: Since size of programs are small, RTOS can also be used in embedded systems like in transport and others.

Disadvantages of RTOS:

- Limited Tasks: Very few tasks run at the same time and their concentration is very less on few applications to avoid errors.
- Use heavy system resources: Sometimes the system resources are not so good and they are expensive as well.
- Complex Algorithms: The algorithms are very complex and difficult for the designer to write on.

Q.2. (b) Explain Process Control Block. Draw the block diagram of process transition states. **(6)**

Ans. Refer to Q.2. (a) from End Term Examination 2018. (Page No. 5-2018)

Q.3. Consider the processes listed in the following table:

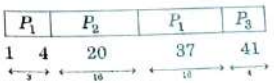
Processes	Arrival Time	Burst Time	Priority No.
P1	1	20	2
P2	4	16	1
P3	5	04	3

Note that the small value (of priority No.) indicates high order of priority Answer the following.

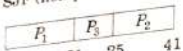
(a) Draw 3 Gantt charts for Priority (preemptive), SJF (non-preemptive) and RR (with the quantum 4). **(5)**

Ans. Priority (Preemptive)

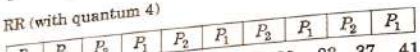
Gantt Chart



SJF (non-preemptive)



RR (with quantum 4)



Q.3. (b) Calculate avg. waiting time for each of the above scheduling. **(4)**

Ans. Waiting time

Process	Priority (Preemptive)	SJF (Non-preemptive)	RR
P_1	0 + 16 = 16	0	0 + 8 + 4 + 4 + 4 = 20
P_2	0	21	1 + 8 + 4 + 4 = 17
P_3	15 + 17 = 32	16	4

Average waiting time for priority (Preemptive) scheduling

$$= \frac{16 + 0 + 32}{3} = \frac{48}{3} = 16$$

Average waiting time for SJF (Non-preemptive) Scheduling

$$= \frac{0 + 21 + 16}{3} = \frac{37}{3} = 12.33$$

Average waiting time for RR

$$\text{Scheduling} = \frac{20 + 17 + 4}{3} = \frac{41}{3} = 13.66$$

Q.3. (c) Calculate avg. Turnaround time for each of the above scheduling. **(3.5)**

Ans. Turn-around time

Process	Priority (Preemptive)	SJF (Non-preemptive)	RR
P_1	20 + 16 = 36	20 + 0 = 20	20 + 20 = 40
P_2	16 + 0 = 16	16 + 21 = 37	16 + 17 = 33
P_3	4 + 32 = 36	4 + 16 = 20	4 + 4 = 8.

Average Turnaround time for priority (Preemptive)

$$= \frac{36 + 16 + 36}{3} = 29.33$$

Average Turnaround time for SJF (Non-preemptive),

$$= \frac{20 + 37 + 20}{3} = 25.66$$

Average Turnaround time for RR

$$= \frac{40 + 33 + 8}{3} = 27$$

UNIT - II

Q.4. (a) What is a semaphore? Which are operations done on semaphore? Give implementation of producer-consumer problem with bounded buffer using semaphore. **(5)**

Ans. A semaphore is a value in a designated place in operating system (or kernel) storage that each process can check and then change. Depending on the value that is found, the process can use the resource or will find that it is already in use and must wait for some period before trying again.

When we multiple threads running they will invariably need to communicate with each other in order *synchronise* their execution.

There are a few possible methods of synchronising threads:

- Mutual Exclusion (Mutex) Locks
- Condition Variables
- Semaphores

Two **standard operations** modify S: wait() and signal()

- Originally called P() and V()
- Also called down() and up()
- The value of S can only be accessed through wait()and signal()

```
wait (S)
{
signal (S)
  {
while S <= 0
S++;
; // no-op

  }
S– –;
}
```

With each semaphore there is an associated waitingqueue.

```
typedefstruct
{
tnt value;
struct process *list;
} semaphore;
```

Two operations on processes:

- block – place the process invoking the operation on the appropriate waiting queue.
- wakeup – remove one of processes in the waiting queue and place it in the ready queue.

Counting semaphore – integer value can range over an unrestricted domain

- Binary semaphore – integer value can range only between 0 and 1; can be simpler to implement
- Also known as mutex locks
- Can implement a counting semaphore S as a binary semaphore

- Provides mutual exclusion

```
Semaphore mutex; // initialized to 1
do {
wait (mutex);
// Critical Section
signal (mutex);
// remainder section
} while (TRUE);
```

Producer Consumer problem: The producer consumer problem is a synchronization problem. There is a fixed size buffer and the producer produces items and enters them into the buffer. The consumer removes the items from the buffer and consumes them.

A producer should not produce items into the buffer when the consumer is consuming an item from the buffer and vice versa. So the buffer should only be accessed by the producer or consumer at a time.

The producer consumer problem can be resolved using semaphores. The codes for the producer and consumer process are given as follows:

Producer Process

The code that defines the producer process is given below:

```
do {
    PRODUCE ITEM
wait(empty);
wait(mutex);
    PUT ITEM IN BUFFER
signal(mutex);
signal(full);
} while(1);
```

In the above code, mutex, empty and full are semaphores. Here mutex is initialized to 1, empty is initialized to n (maximum size of the buffer) and full is initialized to 0.

The mutex semaphore ensures mutual exclusion. The empty and full semaphores count the number of empty and full spaces in the buffer.

After the item is produced, wait operation is carried out on empty. This indicates that the empty space in the buffer has decreased by 1. Then wait operation is carried out on mutex so that consumer process cannot interfere.

After the item is put in the buffer, signal operation is carried out on mutex and full. The former indicates that consumer process can now act and the latter shows that the buffer is full by 1.

Consumer Process

The code that defines the consumer process is given below:

```
do {
wait(full);
wait(mutex);
    REMOVE ITEM FROM BUFFER
signal(mutex);
signal(empty);
    . CONSUME ITEM
} while(1);
```


The wait operation is carried out on full. This indicates that items in the buffer have decreased by 1. Then wait operation is carried out on mutex so that producer process cannot interfere.

Then the item is removed from buffer. After that, signal operation is carried out on mutex and empty. The former indicates that consumer process can now act and the latter shows that the empty space in the buffer has increased by 1.

Q.4. (b) Consider following Snapshot of a system. (7.5)

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Answer the following questions using banker's algorithm.

(i) What is the content of the matrix need? Is the system in a safe state?

(ii) If the request from process PI arrives for {1 0 2}, can the request be granted immediately?

Ans. (i) Content of the need matrix is calculated as Need = Max – Allocation

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

System is in safe state because resources are available (3, 3, 2)

(ii) As, we can see for process P_0, the need is (7, 4, 3) and the available is (3, 3, 2)

$\Rightarrow$ need $\geq$ available $\Rightarrow$ False

So, system will more for next process.

And the need for process P_1 is (1, 2, 2) and available is (3, 3, 2), so need $\leq$ available

$(1, 2, 2) \leq (3, 3, 2) \Rightarrow$ True

So, the request from process P_1 for (1, 0, 2) can be granted immediately.

Q.5. (a) Explain the process of logical to physical address translation in paging system. Give the respective block diagram. (5)

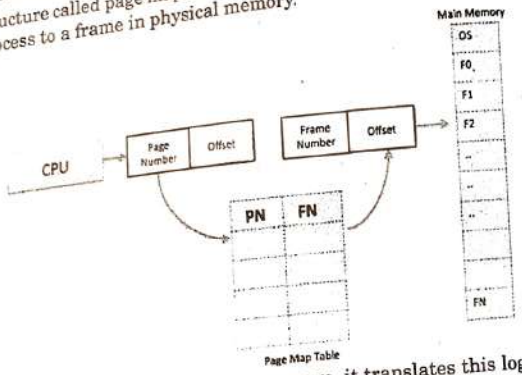
Ans. Page address is called logical address and represented by page number and the offset.

Logical Address = Page number + page offset

Frame address is called physical address and represented by a frame number and the offset.

Physical Address = Frame number + page offset

A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Q.5. (b) Consider the following page reference strings:-

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

How many page faults would occur for the following replacement algorithm, assuming four frames? Assuming all frames are initially empty and first unique pages will cost one fault each. (7,5)

(i) LRU replacement (ii) FIFO replacement (iii) Optimal replacement.

Ans. The given page reference strings :

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

(i) LRU replacement: Criteria is "Replace a page that has not been used for longest period of time"

Frame	1	2	3	4	2	1	5	2	3	7	6	3	5	1	2	3	6
0	1	1	1	1	1	1	1	1	1	7	7	7	7	1	1	1	1
1		2	2	2	2	2	2	2	2	2	2	2	5	5	5	5	6
2			3	3	3	3	5	5	5	5	6	6	6	6	2	2	2
3				4	4	4	4	4	3	3	3	3	3	3	3	3	3
	×	×	×	×	✓	✓	×	✓	×	×	×	✓	×	✓	×	✓	×

No. of page faults in LRU Replacement = 12

(ii) FIFO replacement:

Frame	1	2	3	4	2	1	5	2	3	7	6	3	5	1	2	3	6
0	1	1	1	1	1	1	5	5	5	5	5	5	5	1	1	1	1
1		2	2	2	2	2	2	2	2	7	7	7	7	7	2	2	2
2			3	3	3	3	3	3	3	3	6	6	6	6	6	6	6
3				4	4	4	4	4	4	4	4	3	3	3	3	3	3
	×	×	×	×	✓	✓	×	✓	✓	×	×	×	✓	×	×	✓	✓

No. of page faults in FIFO replacement = 10.

Optimal Replacement:

Frame	1	2	3	4	2	1	5	2	3	7	6	3	5	1	2	3	6
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2
1		2	2	2	2	2	2	2	2	7	6	6	6	6	6	6	6
2			3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3				4	4	4	5	5	5	5	5	5	5	5	5	5	5
	×	×	×	×	✓	✓	×	✓	✓	×	×	✓	✓	✓	×	✓	✓

No. of page faults in optimal replacement = 8.

UNIT - III

Q.6. (a) Define the device management. Explain various techniques for device management. (5)

Ans. All modern operating systems have a subsystem called the device manager. The device manager is responsible for detecting and managing devices, performing power management, and exposing devices to userspace. Since the device manager is a crucial part of any operating system, it's important to make sure it's well designed. The device manager also performs power management. Power management is a feature of hardware that allows for the power consumption of the system and devices to be controlled. Each device managed by the device manager should provide functions to set their power state.

The main role of the device manager is detecting devices on the system. Usually, devices are organized in a tree structure, with devices enumerating their children. Device detection should begin with a "root bus driver".

Each device that is detected should contain a list of resources for the device to use. Examples of resources are I/O, memory, IRQs, DMA channels, and configuration space. Devices are assigned resources by their parent devices. Devices should just use the resources they're given, which provides support for having the same device driver work on different machines where the resource assignments may be different, but the programming interface is otherwise the same. Drivers are loaded for each device that's found. When a device is detected, the device manager finds the device's driver. If not loaded already, the device manager loads the driver. It then calls the driver to initialize that device. How the device manager matches a device to a device driver is an important choice.

Device management has three standard approaches

Three types of device drivers:

(i) Programmed I/Os by polling from each device its the service need from each device.

(ii) Interrupt(s) from the device drivers device- ISR

(iii) Device uses DMA operation used by the devices to access the memory.

Most common is the use of device driver ISRs

Programmed I/O: Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device.

The CPU issues a command then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as Polling. As a result, the level of the performance of the entire system is severely degraded.

Interrupt: The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

Direct Memory Access (DMA): Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles. **(7.5)**

Q.6. (b) Write short notes on the following:

(i) Disk Reliability (ii) Buffering (iii) Block Multiplexing

Ans. (i) It is important to understand the terms reliability and performance as they pertain to disks. Reliability is the ability of the disk system to accommodate a single- or multi-disk failure and still remain available to the users. Performance is the ability of the disks to efficiently provide information to the users.

Adding redundancy almost always increases the reliability of the disk system. The most common way to add redundancy is to implement a Redundant Array of Inexpensive Disks (RAID).

There are two types of RAID:

• **Hardware** — The most commonly used hardware RAID levels are: RAID 0, RAID 1, RAID 5, and RAID 10. The main differences between these RAID levels focus on reliability and performance as previously defined.

• **Software** — Software RAID can be less expensive. However, it is almost always much slower than hardware RAID, because it places a burden on the main system CPU to manage the extra disk I/O.

Advangates of RAID technology

(1) High data transfer rates.

(2) Increased system throughout.

(3) Rate of read request i.e., number of reads per unit time, is also doubled as read requests can be sent to any of the multiple disks in parallel.

Disadvantages of RAID technology

1. Having more disks, reduces overall system reliability.

The performance improvement can be done by two methods-

(a) Data stripping (or parallelism).

(b) Using Redundancy.

(ii) The main memory has an area called buffer that is used to store or hold the data temporarily that is being transmitted either between two devices or between a device or an application. Buffering is an act of storing data temporarily in the buffer. It helps in matching the speed of the data stream between the sender and receiver. If speed of the sender's transmission is slower than receiver, then a buffer is created in main memory of the receiver, and it accumulates the bytes received from the sender and vice versa. The buffering overlaps input/output of one job with the execution of the same job.

(iii) On conventional 370 selector channels, the techniques of independent device operation and buffering are applicable only to the last CCW of a channel program. The channel and control unit remain connected to the device for the duration of the channel program. A 370 block multiplexor channel can be servicing multiple channel programs at the same time (e.g., eight channel programs). When the channel encounters an I/O operation such as a buffered or independent device operation that does not need the channel for awhile, it automatically switches to another channel program that needs immediate servicing. In essence, a block multiplexor channel represents a hardware implementation of multiprogramming for channel programs.

This type of channel multiprogramming could be simulated on a selector channel by making all channel programs only one command long. The device management software routines could then reassign the channel as neccessary. However, this approach is not very attractive unless the central processor is extremely fast, since the channel switching must be done very frequently and very quickly.

Q.7. (a) How swap space is managed by the operating system? Explain (5)

Ans. Swap-space management is another low-level task of the operating system. Virtual memory uses disk space as an extension of main memory. Since disk access is much slower than memory access, using swap space significantly decreases system performance. The main goal for the design and implementation of swap space is to provide the best throughput for the virtual memory system.

Swap-Space Use : Swap space is used in various ways by different operating systems, depending on the memory-

management algorithms in use. For instance, systems that implement swapping may use swap space to hold an entire process image, including the code and data segments. Paging systems may simply store pages that have been pushed out of main memory. The amount of swap space needed on a system can therefore vary depending on the amount of physical memory, the amount of virtual memory it is backing, and the way in which the virtual memory is used. It can range from a few megabytes of disk space to gigabytes.

If a process needs to bring a virtual page into physical memory and there are no free physical pages available, the operating system must make room for this page by discarding another page from physical memory.

If the page to be discarded from physical memory came from an image or data file and has not been written to then the page does not need to be saved. Instead it can be discarded and brought back into memory from the original image or data file if it is needed again.

However, if the page has been modified, the operating system must preserve the contents of that page so that it can be accessed at a later time. This type of page is known as a dirty page. When dirty pages are removed from memory, they are saved in a special sort of file called the swap file. Since access to the swap file takes a long time relative to the speed of the processor and physical memory, the operating system must juggle the need to write pages to disk with the need to retain them in memory.

If the swap algorithm, which is used to decide which pages to discard or swap is not efficient, then a condition known as thrashing occurs. In the case of thrashing, pages are constantly being written to and read back from disk. This causes the operating system to be too busy to perform enough real work. If, for example, physical page frame number 1 in Figure 3.1 is being regularly accessed then it is not a good candidate for swapping to hard disk. The set of pages that a process is currently using is called the working set. An efficient swap scheme would make sure that all processes have their working set in physical memory.

Linux uses a Least Recently Used (LRU) page aging technique to fairly choose pages which might be removed from the system. This scheme involves every page in the system having an age which changes as the page is accessed. The more that a page is accessed, the younger it is; the less that it is accessed, the older and more stale it becomes. Old pages are good candidates for swapping.

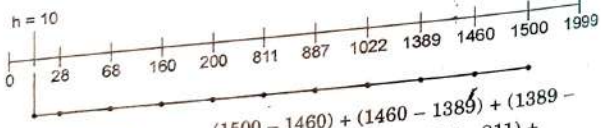
Q.7. (b) Suppose that a disk drive has 2000 cylinders numbered to 0 to 1999. The drive is currently at 10 and previous record was at 140. The queue of pending requests in FIFO is 68,1460,811,200,1500,1022,28,1389,887,160. Starting from current head position, what is total distance that disk moves to satisfy all pending request for each algorithm. (7.5)

(i) SSTF (ii) C-SCAN (iii) Look

Ans. (i) SSTF

Initial head position = 10

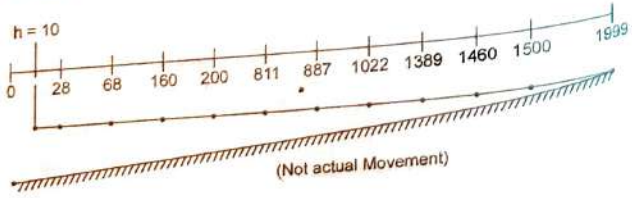
Track 68, 1460, 811, 200, 1500, 1022, 28, 1389, 887, 160



Total Head Movement = (1500 – 1460) + (1460 – 1389) + (1389 – 1022) + (1022 – 887) + (887 – 811) + (811 – 200) + (200 – 160) + (160 – 68) + (68 – 28) + (28 – 10)

= _____ Ans.

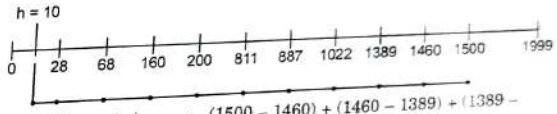
(ii) C-SCAN



Total Head Movement = (1999 – 1500) + (1500 – 1460) + (1460 – 1389) + (1389 – 1022) + (1022 – 887) + (887 – 200) + (200 – 160) + (160 – 68) + (68 – 28) + (28 – 10)

= _____ Ans.

(c) LOOK



Total Head Movement = (1500 – 1460) + (1460 – 1389) + (1389 – 1022) + (1022 – 887) + (887 – 811) + (811 – 200) + (200 – 160) + (160 – 68) + (68 – 28) + (28 – 10)

= _____ Ans.

UNIT - IV

Q.8. (a) Explain the sequential and direct file access methods. How can a sequential file be simulated on a direct access file? Explain. (6)

Ans. 1.Sequential Access – It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

Read and write make up the bulk of the operation on a file.

A read operation -read next- read the next position of the file and automatically advance a file pointer, which keeps track I/O location.

Similarly, for the write – write next append to the end of the file and advance to the newly written material.

Key points:

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file
- Such a method is reasonable for tape.

2. Direct Access – Another method is direct access method also known as relative access method. A filed-length logical record that allows the program to read and write record rapidly, in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.

A block number provided by the user to the operating system is normally a relative block number, the first relative block of the file is 0 and then 1 and so on.

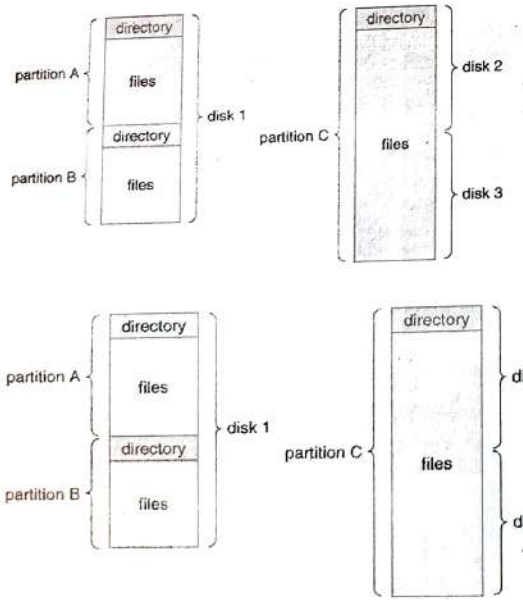
A sequential file can be simulated on a direct access file: A variable cp is used to indicate the current position in the file. The variable cp is incremented whenever a read or write operation is performed as shown below:

Sequential access	Implementation for direct access
reset	ep ← 0
read next	read ep ep ← ep + 1
write next	write ep ep ← ep + 1

Q.8. (b) Compare different directory structure implementation in a file system. (6.5)

Ans. Directory Structure: Storage Structure

- A disk can be used in its entirety for a file system.
- Alternatively a physical disk can be broken up into multiple **partitions, slices, or mini-disks**, each of which becomes a virtual disk and can have its own filesystem. (or be used for raw storage, swap space, etc.)
- Or, multiple physical disks can be combined into one **volume**, i.e. a larger virtual disk, with its own filesystem spanning the physical disks.



Directory Overview

- Directory operations to be supported include:

→ Search for a file

→ Create a file - add to the directory

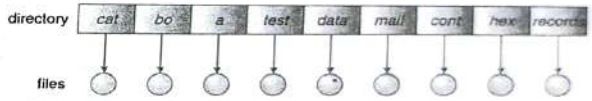
→ Delete a file - erase from the directory

→ List a directory - possibly ordered in different ways.

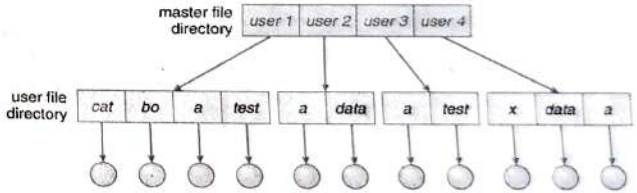
→ Rename a file - may change sorting order

→ Traverse the file system.

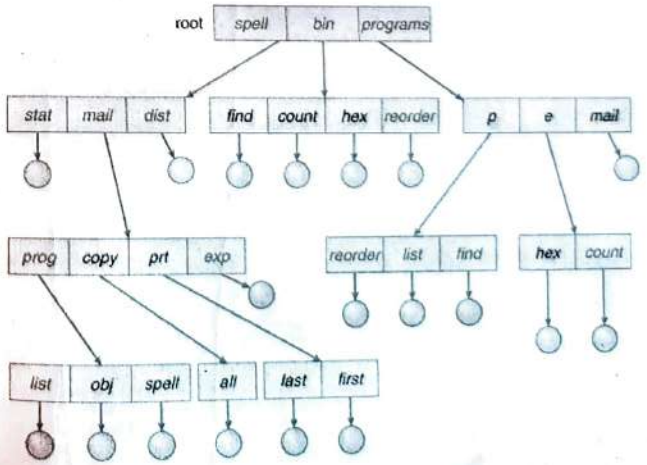
Single level directory :



Two-level directory:

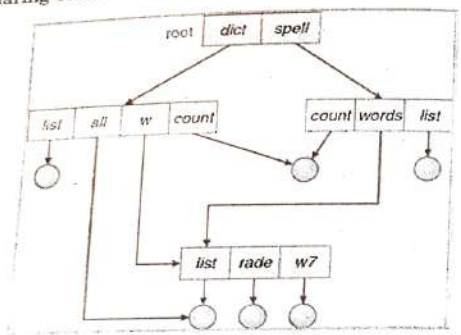


Tree structure directory:



Acyclic-Graph Directorie

- Allow sharing of subdirectories and files

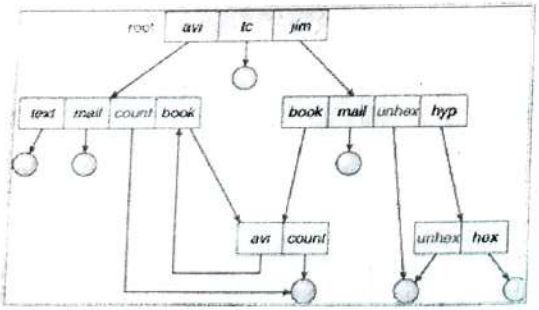


- Two different names (aliasing)
- If dict deletes list dangling pointer

Solutions:

- Backpointers, so we can detete all pointers Variable size records a problem
- Backpointers using a daisy chain organization
- Reference count for each file
- New directory entry type
- **Link–** another name (Pointer) to an existing file
- **Resolve the link–** follow pointer to locate the file

General Graph Directory



- How do we guarantee no cycles?

→ Allow only link to files not subdirectories

→ Garbage collection

→ Every time a new link is added use a cycle detection algorithm to determine whethre it is is OK.

Q.9. List the differences between Linux Operating System and Windows XP operating system. **(12.5)**

Ans. Linux is a free and open source operating system based on Unix standards. It provides programming interface as well as user interface compatible with Unix based systems and provides large variety applications. A Linux system also contains many separately developed elements, resulting in Unix system which is fully compatible and free from proprietary code.

The traditional monolithic kernel is employed in Linux kernel for performance purpose, but its modular feature allows most drivers to dynamically loaded and unloaded at runtime. Linux protects processes and is a multiuser system. Inter-process communication is supported by both of mechanisms such as message queue, shared memory and semaphore.

An abstract layer is used in Linux to govern the different file systems, but to users, the file system looks like a hierarchical directory tree. It also supports networked, device-oriented and virtual file systems. Disk storage is accessed through a page cache which is unified with the virtual memory systems. To minimize the duplication of shared data among different processes the memory management system uses page sharing and copy-on-write.

Windows is a licensed operating system in which source code is inaccessible. It is designed for the individuals with the perspective of having no computer programming knowledge and for business and other commercial users. It is very simple and straightforward to use.

Windows is extensible, portable and assists multiple operating environments, symmetric multiprocessing and client-server computing. It offers integrated caching, virtual memory, and preemptive scheduling.

Basis	Linux	Windows
Cost	Free of cost	Expensive
Open source	Yes	No
Customizable	Yes	No
Security	More secure	Vulnerable to viruses and malware attacks.
Booting	Either primary or logical partition.	Only primary partition.
Separation of the directories using	Forward slash	Back slash
File names	Case sensitive	Case insensitive
File system	EXT2, EXT3, EXT4, Reisers FS, XFS and JFS	FAT, FAT32, NTFS and ReFS
Type of kernel used	Monolithic kernel	Microkernel
Efficiency	Effective running efficiency	Lower than Linux

Some more key differences are:

- Linux has access to source code and alters the code as per user need whereas Windows doesnot have access to source code.

• Linux will run faster than windows latest editions even with a modern desktop environment and features of the operating system whereas windows are slow on older hardware.

• Linux distributions don't collect user data whereas Windows collect all the user details which lead to privacy concern.

• Linux is more reliable then windows as in Linux we can kill application if they hung through x kill command whereas, in windows, we need to try multiple times to kill it.

• Linux supports a wide variety of free software's than windows but windows have a large collection of video game software.

• In Linux software cost is almost free as all programs, utilities, complex applications such as open office are free but windows also have many free programs and utilities but most of the programs are commercial.

• Linux is highly secure because it's easy to identify bugs and fix whereas Windows has a large user base and becomes a target for developers of viruses and malware.

• Linux is used by corporate organizations as servers and operating system for security purpose at Google, Facebook, twitter.etc. whereas windows are mostly used by gamers and business users.

• Linux and windows have same priority over hardware and driver support in the present situation.

END TERM EXAMINATION [FEB. 2023]
FIRST SEMESTER [MCA]
OPERATING SYSTEMS WITH LINUX [MCA-105]

Time: 3 Hrs. **Max. Marks: 75**

Note: Attempt five questions in all including Q. No. 1 which is compulsory. Select one question from each unit.

Q.1. Attempt all questions: **(10 × 2.5 = 25)**

Q.1. (a) List all the differences between internal fragmentation and external fragmentation.

Ans. Internal fragmentation and external fragmentation are two types of memory management issues that occur in computer systems. Here are the main differences between them:

Internal Fragmentation:

1. **Definition:** Internal fragmentation occurs when memory is allocated in fixed-sized blocks, and the allocated memory block is larger than the actual data size.
2. **Cause:** It is caused by the memory allocation technique where fixed-sized blocks are used, and the allocated memory block is not fully utilized.
3. **Location:** Internal fragmentation happens within a single memory partition or block.
4. **Memory wastage:** The unused memory within the allocated block results in memory wastage.
5. **Impact:** Internal fragmentation reduces the overall memory efficiency by occupying memory with unused space.
6. **Mitigation:** Techniques like memory compaction or dynamic memory allocation algorithms can be used to minimize internal fragmentation.
7. **Memory allocation method:** Internal fragmentation is common in fixed partitioning and fixed-size allocation schemes.

External Fragmentation:

1. **Definition:** External fragmentation occurs when free memory blocks are scattered throughout the memory, making it difficult to allocate contiguous memory blocks for larger programs.
2. **Cause:** It is caused by the memory allocation and deallocation processes that leave small free memory blocks scattered across the memory space.
3. **Location:** External fragmentation occurs across different memory partitions or blocks.
4. **Memory wastage:** The fragmented free memory blocks cannot be utilized effectively, leading to wasted memory.
5. **Impact:** External fragmentation reduces the available memory for allocation and can eventually lead to memory allocation failures.
6. **Mitigation:** Techniques like memory compaction, memory paging, or dynamic memory allocation algorithms (e.g., best fit, worst fit) can be used to mitigate external fragmentation.
7. **Memory allocation method:** External fragmentation is common in dynamic partitioning, variable-sized allocation schemes, or when using virtual memory systems.

In summary, internal fragmentation refers to wasted memory within a single memory block, while external fragmentation refers to the scattering of free memory

blocks throughout the memory space. Internal fragmentation occurs within a block, while external fragmentation occurs across multiple blocks or partitions.

Q.1. (b) Explain the function of the ready queue?

Ans. The ready queue is a crucial component of process scheduling in operating systems. It serves as a temporary storage area where processes that are ready to be executed are held until they are assigned to the CPU for execution. Here are the main functions of the ready queue:

1. Process Scheduling: The ready queue allows the operating system to manage and schedule the execution of multiple processes efficiently. It acts as a pool of ready-to-run processes that are waiting for CPU time.

2. Process Prioritization: The ready queue helps in prioritizing processes based on their priority levels or scheduling algorithms. Processes with higher priority are typically given precedence over those with lower priority when selecting the next process to be executed.

3. Context Switching: The ready queue plays a vital role in facilitating context switching between processes. When a currently running process completes its time slice or gets blocked, the operating system selects the next process from the ready queue for execution, and a context switch occurs.

4. Fairness and Resource Utilization: By organizing processes in the ready queue, the operating system can ensure fairness in CPU allocation. It allows each process to have an opportunity to execute and utilize system resources effectively.

5. Process Synchronization: In multi-threaded or multi-core systems, the ready queue can help manage synchronization between different threads or cores. Synchronization mechanisms like mutexes or semaphores can be used to control access to the ready queue, ensuring that only one thread or core accesses it at a time.

6. Pre-emption: The ready queue enables pre-emption of running processes when a higher-priority process becomes available. If a process with higher priority enters the ready queue, it can preempt the currently running process, allowing for priority-based scheduling.

Q.1. (c) What is multi – threading?

Ans. Multithreading is a programming and execution model where multiple threads of execution run concurrently within a single process. A thread can be thought of as a lightweight subunit of a process that can perform tasks independently. In a multithreaded program, multiple threads execute within the same memory space and share the same resources, such as the CPU, memory, and file system.

Multithreading is widely used in various applications, such as server applications, graphical user interfaces, real-time systems, and computationally intensive tasks, to achieve better performance, responsiveness, and resource utilization.

Q.1. (d) What are the basic requirements of any solution to the critical section problem?

Ans. To provide a solution to the critical section problem, which involves multiple processes or threads accessing shared resources concurrently, certain basic requirements must be met. These requirements aim to ensure safety, progress, and fairness in accessing the critical section. The basic requirements for a solution to the critical section problem are as follows:

1. Mutual Exclusion: Only one process or thread can access the critical section at a time. This requirement ensures that concurrent access to shared resources does not lead to data inconsistency or corruption.

2. Progress: If no process is currently executing in the critical section and some processes are waiting to enter, the selection of the next process to enter the critical section should be made in a fair manner. This requirement avoids the scenario where a process is indefinitely blocked from entering the critical section.

3. Bounded Waiting: There should be a limit on the number of times other processes can enter the critical section while a particular process is waiting. This requirement prevents starvation, ensuring that every process eventually gets a fair chance to access the critical section.

Q.1. (e) Differentiate between multiprogramming and multiprocessing.

Ans. Multiprogramming and multiprocessing are both techniques used in computer systems to achieve better resource utilization and improved system performance. However, they differ in how they utilize and manage multiple tasks or programs. Here's a comparison between the two:

Multiprogramming:

1. Definition: Multiprogramming is a technique where multiple programs are loaded into main memory concurrently, and the CPU switches between them for execution.

2. CPU Utilization: Multiprogramming aims to maximize CPU utilization by keeping the CPU busy with some task at all times, even if a program is waiting for I/O operations or other resources.

3. Context Switching: The CPU performs context switching to switch between different programs. When one program is waiting, the CPU can quickly switch to another ready program.

4. Resource Sharing: Programs share resources such as memory, I/O devices, and peripherals. The CPU schedules different programs based on their priority and availability of resources.

5. Memory Management: Memory is divided into fixed partitions or variable-sized partitions. Each program is loaded into a separate partition, and the CPU switches between them.

6. Communication: Communication between programs is typically limited. They may share files or use inter-process communication mechanisms like pipes or shared memory.

7. Example: In a multiprogramming system, a computer may have multiple programs running simultaneously, and the CPU time is divided among them for efficient utilization.

Multiprocessing:

1. Definition: Multiprocessing involves the use of multiple CPUs or processor cores to execute multiple tasks or programs simultaneously.

2. CPU Utilization: Multiprocessing aims to improve system performance by dividing the workload among multiple processors, allowing tasks to execute in parallel and achieve faster execution times.

3. Task Distribution: Each CPU or core is assigned a specific task or program for execution. Multiple tasks can be executed concurrently without the need for frequent context switching.

4. Resource Allocation: Each CPU or core has its own set of resources, such as registers, cache, and execution units. Tasks are allocated to different CPUs or cores based on the available resources and load balancing strategies.

5. Memory Management: Memory is typically shared among all processors, allowing them to access a common address space. Memory access coordination and synchronization mechanisms are required to prevent conflicts.

6. Communication: Inter-process communication mechanisms, such as message passing or shared memory, are commonly used to enable communication and data sharing between tasks running on different processors or cores.

7. Example: In a multiprocessing system, tasks or programs are divided among multiple processors or cores, allowing for parallel execution and faster processing of multiple tasks simultaneously.

Q.1. (f) Differentiate between kernel level and user level threads.

Ans. Kernel-level threads and user-level threads are two approaches to implementing threads in an operating system. They differ in how they are managed, their relationship to the operating system, and their overhead. Here's a comparison between kernel-level and user-level threads:

Kernel-Level Threads:

1. Managed by the Kernel: Kernel-level threads are created, scheduled, and managed directly by the operating system kernel.

2. Operating System Support: The operating system provides native support for kernel-level threads, treating each thread as a separate entity.

3. Thread Creation and Scheduling: Thread creation and scheduling involve system calls that interact with the kernel. The kernel maintains thread-specific information and makes scheduling decisions based on thread priorities and other factors.

4. Concurrency and Parallelism: Kernel-level threads can achieve true concurrency and parallelism, as each thread can be scheduled to run on a separate CPU or core simultaneously.

5. Blocking Operations: If a kernel-level thread performs a blocking operation, such as I/O or waiting for a resource, the entire process is blocked, including all associated threads.

6. Context Switching Overhead: Context switching between kernel-level threads incurs relatively higher overhead, as it involves transitioning between user mode and kernel mode.

7. Scalability: Kernel-level threads may provide better scalability for applications that require intensive multitasking or heavily rely on system resources.

User-Level Threads:

1. Managed by User-Level Libraries: User-level threads are implemented and managed by user-level libraries or runtime environments without direct involvement from the operating system kernel.

2. Limited Operating System Support: The operating system is typically unaware of user-level threads and treats the associated process as a single entity.

3. Thread Creation and Scheduling: Thread creation and scheduling are handled within the user-level library or runtime environment, using thread management routines provided by the library.

4. Concurrency and Parallelism: User-level threads do not achieve true concurrency or parallelism at the kernel level, as multiple threads of a process are scheduled and executed within a single kernel-level thread.

5. Blocking Operations: If a user-level thread performs a blocking operation, it does not block the entire process. Other user-level threads within the same process can continue execution.

6. Context Switching Overhead: Context switching between user-level threads incurs lower overhead, as it typically does not involve transitioning to kernel mode. Switching between user-level threads can be done within the same process without OS intervention.

7. Portability and Flexibility: User-level threads offer portability across different operating systems and flexibility in thread management, as it is independent of the kernel threading model.

Q.1. (g) What is meant by context switch?

Ans. A context switch refers to the process of saving and restoring the state of a running process or thread so that another process or thread can be scheduled and resume execution. It involves the operating system's ability to switch the context from one executing entity to another, allowing for multitasking and concurrent execution of multiple processes or threads. Here's an explanation of the context switch process:

1. Process or Thread State: Each process or thread has its own execution context, which includes information such as program counter (PC), register values, stack pointers, and other necessary data related to the execution of the process or thread.

2. Trigger for Context Switch: Context switches can be triggered by various events, including time slices, interrupts, I/O operations, synchronization mechanisms, or the operating system's scheduling algorithm determining that another process or thread needs to be executed.

3. Saving the Current Context: When a context switch is triggered, the operating system saves the current context of the running process or thread. It involves saving the values of registers, program counter, stack pointer, and other relevant information into a data structure known as the process or thread control block (PCB/TCB).

4. Loading the Next Context: After saving the context of the current process or thread, the operating system selects the next process or thread to be executed. It loads the saved context from the PCB/TCB of the selected process or thread back into the CPU registers and memory, preparing it for execution.

5. Context Switch Overhead: Context switching introduces some overhead due to the time required to save and restore the context of processes or threads. This overhead includes saving and restoring registers, updating data structures, and performing necessary bookkeeping tasks.

6. Resuming Execution: Once the context of the new process or thread has been loaded, the CPU resumes execution from the point where it left off before the context switch. The new process or thread continues its execution as if no interruption occurred.

Context switches are an essential mechanism for multitasking and enabling concurrent execution of multiple processes or threads on a single CPU or across multiple CPUs or cores. They allow the operating system to allocate CPU time to different processes or threads, ensuring fairness, responsiveness, and efficient resource utilization.

Q.1. (h) What is the use of fork and exec system calls?

Ans. The **fork()** and **exec()** system calls are commonly used in operating systems to create new processes and execute different programs within those processes. Here's an explanation of each system call and their respective uses:

1. fork() System Call:

• **Purpose:** The fork() system call is used to create a new process, known as the child process, from an existing process, known as the parent process.

• **Process Creation:** When fork() is called, the operating system creates a new child process that is a duplicate of the parent process. The child process receives a copy of the parent's memory, file descriptors, and other relevant resources.

• **Return Values:** In the parent process, the fork() system call returns the process ID (PID) of the newly created child process, while in the child process, it returns 0 to indicate that it is the child process.

• **Use Cases:** fork() is often used in scenarios where a new process needs to be created to perform a separate task, such as running a different program, implementing parallel processing, or creating a multi-process architecture.

2. exec() System Call:

• **Purpose:** The exec() system call is used to replace the current process's memory and code with a new program, effectively loading and executing a different executable file.

• **Process Replacement:** When exec() is called, the operating system loads the specified program into the current process's memory space, overwriting the existing code and data. The new program starts execution from its entry point.

• **Arguments and Environment:** exec() takes arguments that specify the path to the executable file and command-line arguments to be passed to the new program. It also sets up the environment variables for the new program's execution.

• **Return Value:** The exec() system call does not return if successful, as the current process is replaced by the new program. If exec() fails, it returns -1, indicating an error.

• **Use Cases:** exec() is commonly used to launch different programs or scripts from within a process. It allows for dynamic program execution and is often used in scenarios like process spawning, command execution, and script interpretation.

By combining the fork() and exec() system calls, it is possible to create a new process using fork() and then replace its memory with a different program using exec(). This sequence is commonly used to create new processes and execute various programs or scripts within an operating system.

Q.1. (i) Differentiate between pre-emptive and non – pre-emptive scheduling.

Ans. Pre-emptive and non-pre-emptive scheduling are two different approaches to task scheduling in operating systems. They determine how the CPU is allocated to different processes or threads. Here's a comparison between pre-emptive and non-pre-emptive scheduling:

(a) Pre-emptive Scheduling: Pre-emptive scheduling is a scheduling policy where a running process or thread can be interrupted and forcibly removed from the CPU by the operating system. The CPU is then allocated to another process or thread based on priority or a predefined scheduling algorithm. Pre-emptive scheduling involves frequent context switching as the operating system can interrupt a running process at any time and switch to another process. This allows for fair allocation of CPU time and responsiveness. Common pre-emptive scheduling algorithms include Round Robin, Shortest Remaining Time First (SRTF), and Priority Scheduling.

(b) Non-pre-emptive Scheduling: Non-pre-emptive scheduling is a scheduling policy where a running process or thread retains the CPU until it voluntarily relinquishes control, such as by blocking for I/O or completing its execution. The CPU is allocated to another process only when the running process explicitly releases it. Non-pre-emptive scheduling involves fewer context switches since processes or threads run until they finish or explicitly yield the CPU. This can lead to longer response times for high-priority tasks if a low-priority task is running for an extended period. Non-pre-emptive scheduling algorithms include First-Come, First-Served (FCFS), Shortest Job Next (SJN), and Priority Scheduling without preemption.

The choice between pre-emptive and non-pre-emptive scheduling depends on factors such as system responsiveness requirements, priority management, fairness considerations, and the nature of the tasks or processes being scheduled. Pre-emptive scheduling is often preferred in real-time systems or scenarios requiring fairness and responsiveness, while non-pre-emptive scheduling can be simpler and more suitable for certain environments where predictability and context switch overhead need to be minimized.

Q.1. (j) Differentiate between long term and short term scheduling.

Ans. Long-term scheduling and short-term scheduling are two distinct phases in the process of scheduling tasks or processes in an operating system. They serve different purposes and operate at different stages of the overall scheduling process. Here's a comparison between long-term and short-term scheduling:

Long-Term Scheduling (Admission Scheduling): Long-term scheduling is responsible for selecting which processes or tasks from the job queue should be brought into the system for execution. It decides when to admit a process from the pool of waiting processes to the main memory. Long-term scheduling occurs infrequently, typically when a new process enters the system, or when an existing process completes or terminates. It determines the overall degree of multiprogramming or the number of processes allowed to execute simultaneously. Long-term scheduling algorithms include the First-Come, First-Served (FCFS) algorithm, the Shortest Job Next (SJN) algorithm, or more sophisticated approaches like the Lottery Scheduling or Multilevel Queue Scheduling algorithms.

Short-Term Scheduling (CPU Scheduling): Short-term scheduling, also known as CPU scheduling, determines the order in which ready processes or threads in the memory queue are allocated CPU time for execution. It focuses on optimizing CPU utilization and responsiveness. Short-term scheduling occurs frequently, usually on a millisecond or microsecond timescale. It decides which process or thread should execute on the CPU at a given moment. Short-term scheduling algorithms include Round Robin, Shortest Remaining Time First (SRTF), Priority Scheduling, and Multilevel Feedback Queue Scheduling.

Long-term scheduling focuses on admitting processes into the system and managing the degree of multiprogramming, while short-term scheduling determines which processes or threads are allocated CPU time for execution. Long-term scheduling operates on a longer timescale, whereas short-term scheduling occurs more frequently with a focus on optimizing CPU utilization and responsiveness.

UNIT - 1

Q.2. (a) List five services provided by an operating system. Explain how each provide convenience to the users. Explain also in which cases it would be impossible for user – level programs to provide these services. (6)

Ans. Five services provided by an operating system are:

1. Process Management:

• Convenience: The operating system manages processes, which are instances of executing programs. It provides services like process creation, termination, scheduling, and synchronization. This convenience allows users to run multiple programs simultaneously, switch between them, and ensure fair allocation of resources.

• User-Level Limitation: User-level programs do not have direct control over the underlying system resources or scheduling decisions. They lack the necessary privileges and mechanisms to manage processes at a low level or perform operations like context switching or memory allocation.

2. Memory Management:

• Convenience: The operating system handles memory allocation and management, ensuring that processes have access to the required memory resources. It provides services like memory allocation, deallocation, and virtual memory management. This convenience allows users to run programs without worrying about memory limitations or conflicts.

• User-Level Limitation: User-level programs do not have direct control over physical memory allocation or virtual memory management. They rely on the operating system to provide the necessary abstractions and services for efficient memory utilization.

3. File System Management:

• Convenience: The operating system provides a file system that organizes and manages files and directories. It offers services like file creation, deletion, read, write, and permission management. This convenience allows users to store, access, and organize their data in a hierarchical and structured manner.

• User-Level Limitation: User-level programs do not have direct control over low-level file system operations or access control. They rely on the operating system's file system services and permissions to manage files securely and efficiently.

4. Device Management:

• Convenience: The operating system manages and provides services for accessing input/output (I/O) devices such as keyboards, printers, disks, network interfaces, and more. It offers device drivers, input/output services, and handles device synchronization. This convenience allows users to interact with various devices and perform I/O operations without dealing with low-level device details.

• User-Level Limitation: User-level programs do not have direct control over device drivers or the ability to manage I/O devices at a low level. They rely on the operating system's device management services and drivers to interact with devices effectively.

5. Security and Protection:

• Convenience: The operating system enforces security and protection mechanisms to ensure the integrity, confidentiality, and availability of user data and system resources. It provides authentication, access control, encryption, and auditing services. This convenience allows users to trust that their data is secure and protected from unauthorized access or malicious activities.

• User-Level Limitation: User-level programs do not have direct control over system-level security mechanisms or access control. They rely on the operating system's security services and policies to maintain the confidentiality and integrity of their data.

In cases where user-level programs attempt to provide these services without the support of the operating system, they would face several limitations:

• Lack of direct access to low-level hardware resources, making it impossible to manage processes, allocate memory efficiently, or interact with devices effectively.

• Inability to enforce security mechanisms at the system level, leading to potential vulnerabilities and unauthorized access to data.

• Lack of coordination and synchronization between user-level programs, resulting in conflicts and resource contention.

• Limited control over system-wide services and policies, leading to inefficient resource allocation and potential system instability.

• Difficulty in managing system-level tasks such as context switching, interrupt handling, or virtual memory management, which require privileged access and deeper control over the system's internal mechanisms.

Q.2. (b) What is directory? What are the different ways to implement a directory? (6.5)

Ans. A directory, also known as a folder, is a container or organizational structure used to hold and manage files and subdirectories in a file system. It provides a hierarchical structure for organizing and locating files, making it easier for users and programs to navigate and manage their data. A directory can contain files and other directories, forming a tree-like structure.

Different ways to implement a directory include:

1. Single-Level Directory:

• In this implementation, all files are stored in a single directory without any subdirectories. Files are given unique names, and conflicts may arise if multiple files have the same name. This approach is simple but lacks organization and scalability.

2. Two-Level Directory:

• This implementation introduces a separate directory for each user or entity. Each user has their own directory, and files within a user's directory can have the same name without conflict. This approach provides better organization and user isolation but may result in duplicate file names across different user directories.

3. Tree-Structured Directory:

• The tree-structured directory is the most common and widely used implementation. It allows for the creation of a hierarchical directory structure, with a root directory at the top and subdirectories branching out below. Each directory can contain files and other directories. The tree structure provides a clear organization and allows for efficient file management.

4. Acyclic-Graph Directory:

• In this implementation, directories can have multiple parents, forming a directed acyclic graph (DAG) structure. This allows for shared directories or symbolic links where a directory can be accessed from multiple paths. It provides flexibility but can lead to complexities in managing links and potential issues with loops.

5. Generalized Tree Directory:

• This implementation extends the tree-structured directory by allowing a file or directory to have multiple names or aliases. It provides the ability to create hard links or junction points where one file or directory appears in multiple locations within the directory structure. This approach allows for efficient space utilization but requires additional bookkeeping to manage links and prevent inconsistencies.

The choice of directory implementation depends on the requirements of the file system and the desired level of organization, access control, and scalability. Most modern file systems, such as NTFS (New Technology File System) in Windows and ext4 (Fourth Extended File System) in Linux, utilize a tree-structured directory implementation to provide efficient and organized file management.

Q.3. (a) Differentiate how distributed operating systems differ from multi – programmed and time -shared operating system? Give key features of each. (6)

Ans. Distributed operating systems, multi-programmed operating systems, and time-shared operating systems are different types of operating systems that serve distinct purposes and have unique characteristics. Here's a comparison between these types:

1. Distributed Operating Systems:

- **Key Features:**
- **Distribution of Resources:** Distributed operating systems consist of multiple interconnected computers or nodes that work together as a unified system. Resources, such as processing power, memory, and storage, are distributed across these nodes.
- **Transparency:** Distributed operating systems aim to provide transparency to users and applications, making the distributed nature of the system transparent. This includes transparency in accessing remote resources, location transparency, and failure transparency.
- **Concurrency and Coordination:** Distributed operating systems handle concurrent processes running on different nodes and provide mechanisms for inter-process communication and synchronization across the distributed system.
- **Fault Tolerance:** Distributed operating systems often incorporate fault-tolerant mechanisms to ensure system reliability. This includes replication, error detection and recovery, and load balancing.
- **Scalability:** Distributed operating systems can scale horizontally by adding more nodes to the system, allowing for increased processing power and storage capacity.

2. Multi-programmed Operating Systems:

- **Key Features:**
- **Concurrent Execution:** Multi-programmed operating systems support the concurrent execution of multiple programs or processes. The CPU is shared among these processes, and the operating system switches between them based on scheduling algorithms.
- **Resource Management:** Multi-programmed operating systems efficiently manage system resources, such as CPU time, memory, and I/O devices, to maximize overall system utilization and throughput.
- **Process Scheduling:** These systems employ process scheduling algorithms to determine the order in which processes are executed and to allocate CPU time fairly.
- **Process Isolation:** Each process runs independently and is isolated from other processes. This ensures that one process cannot interfere with the execution of another process.
- **Efficiency:** Multi-programmed operating systems aim to keep the CPU busy and reduce idle time by allowing other processes to execute when some processes are waiting for I/O or other events.

3. Time-shared Operating Systems (also known as Timesharing or Interactive Operating Systems):

- **Key Features:**
- **Time Division Multiplexing:** Time-shared operating systems allocate CPU time in small time slices or quanta, allowing multiple users or processes to share the CPU. Each user or process is given a fair share of CPU time.
- **Interactive Response:** Time-shared systems prioritize user interaction and provide quick response times. They ensure that each user receives a responsive environment for their interactive tasks.
- **Terminal Handling:** Time-shared operating systems provide terminal handling mechanisms, enabling multiple users to interact with the system through terminals simultaneously.
- **Resource Sharing:** These systems manage shared resources, such as memory and I/O devices, effectively among multiple users or processes, ensuring fair allocation and preventing conflicts.
- **Security and Protection:** Time-shared operating systems enforce security measures to protect user data and ensure that one user cannot access or interfere with another user's files or processes.

Q.3. (b) Explain the following: (i) Multitasking System (ii) Real-time System. (6.5)

Ans. (i) Multitasking Systems: A multitasking system, also known as a multitasker or a multitasking operating system, is an operating system that allows multiple tasks or processes to run concurrently on a single computer system. It provides the illusion of parallel execution, enabling users to execute multiple programs simultaneously and efficiently share system resources.

In a multitasking system, the CPU time is divided among different tasks or processes through a process scheduling algorithm. Each task is allocated a time slice or quantum, during which it can execute. The operating system performs context switching to save the state of a running task and load the state of the next task, allowing tasks to appear to run simultaneously.

Multitasking systems have become commonplace, and modern operating systems, such as Windows, macOS, and Linux, extensively utilize multitasking capabilities. They enable efficient resource sharing, enhance productivity, and provide users with a versatile computing experience by allowing the execution of multiple tasks concurrently.

(ii) Real – time systems: A real-time system is a computer system or software that is designed to respond to events or inputs within strict timing constraints. It is characterized by the need to provide timely and predictable responses to external stimuli, typically in the context of critical applications where timing is crucial. Real-time systems have specific timing requirements, known as deadlines, which must be met. There are two types of real-time systems: hard real-time systems, where missing a deadline can result in catastrophic consequences, and soft real-time systems, where occasional missed deadlines are tolerable. Examples of real-time systems include aircraft control systems, industrial automation systems, medical monitoring systems, traffic control systems, and real-time financial trading systems.

Developing and managing real-time systems require careful consideration of timing requirements, resource allocation, and system design. Specialized operating systems and software frameworks, such as real-time operating systems (RTOS), are often used to provide the necessary features and guarantees for real-time applications.

Q.4. (a) What is a semaphore? Explain busy waiting semaphores. (6)

Ans. A semaphore is a synchronization primitive used in concurrent programming to control access to shared resources or coordinate the execution of multiple processes or threads. It acts as a signaling mechanism, allowing threads or processes to block or proceed based on the availability of resources.

A semaphore typically consists of an integer value and two fundamental operations: "wait" (also known as "P" or "down") and "signal" (also known as "V" or "up").

The "wait" operation decreases the value of the semaphore by one. If the resulting value is negative, the calling thread or process is blocked, and it is put into a waiting state until the semaphore value becomes positive. This ensures that resources are not overused, and only a limited number of threads can access the shared resource simultaneously.

The "signal" operation increases the value of the semaphore by one. If any threads were waiting due to a negative semaphore value, one of them is allowed to proceed, and the semaphore value becomes non-negative.

Busy waiting, also known as spinning, occurs when a thread continuously checks the value of a semaphore in a loop while waiting for it to become positive. In the context of busy waiting semaphores, a thread repeatedly executes a "wait" operation until the semaphore value becomes positive, without actually blocking or relinquishing the CPU. This means the thread consumes CPU time even when it is not actively performing useful work.

Busy waiting semaphores have certain drawbacks:

1. Wasted CPU Time: Busy waiting consumes CPU cycles even when a thread is not doing useful work. It can lead to inefficient resource utilization, particularly in scenarios where threads need to wait for longer periods.

2. Resource Contentions: Busy waiting can result in resource contentions and increased contention for CPU time, especially in scenarios where multiple threads are spinning on the same semaphore simultaneously. This contention can lead to decreased overall system performance.

3. Power Consumption: Continuously spinning and checking the semaphore value can consume significant power, especially on devices with limited battery life or in energy-efficient systems.

Despite these drawbacks, busy waiting semaphores can be useful in certain scenarios, such as when the waiting period is expected to be short, or when the system has sufficient CPU resources available. However, in most cases, it is preferable to use more efficient synchronization mechanisms, such as blocking semaphores or condition variables, that allow threads to yield the CPU and avoid wasting resources during waiting periods.

Q.4. (b) Explain the differences with diagram between multilevel queue and multilevel feedback queue scheduling. (6.5)

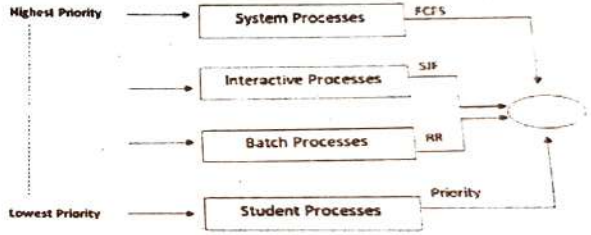
Multilevel queue scheduling and multilevel feedback queue scheduling are two different approaches to scheduling processes in a computer system. Here's an explanation of the differences between these two scheduling algorithms, along with diagrams to illustrate their structures:

1. Multilevel Queue Scheduling:

- In multilevel queue scheduling, processes are divided into separate queues based on their characteristics, such as priority, job type, or other criteria.
- Each queue is assigned a different priority level, and processes in higher priority queues are given precedence over those in lower priority queues.

- Processes are permanently assigned to a specific queue, and once they enter a queue, they remain in that queue until they complete or are terminated.
- Each queue can have its own scheduling algorithm, such as First-Come-First-Served (FCFS), Round Robin (RR), or Priority scheduling.
- The scheduler selects a process from the highest priority queue that contains runnable processes and allocates the CPU to that process.

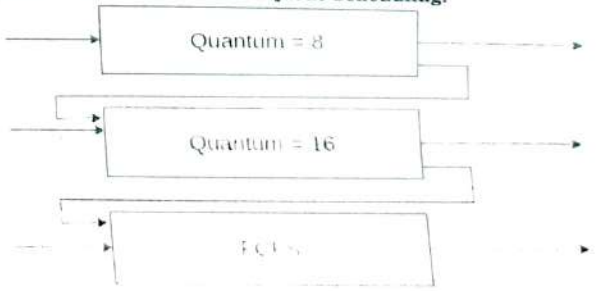
Diagram of Multilevel Queue Scheduling:



2. Multilevel Feedback Queue Scheduling:

- In multilevel feedback queue scheduling, processes are initially assigned to a default queue based on some criteria, such as priority or arrival time.
- Each queue has a different priority level, and processes can move between queues based on their behavior and performance.
- If a process uses up its time quantum (a fixed amount of CPU time) in the current queue, it is demoted to a lower priority queue.
- If a process performs I/O operations or exhibits interactive behavior, it can be promoted to a higher priority queue.
- This dynamic movement of processes between queues allows for adaptation to varying workload characteristics and helps prioritize CPU-bound or I/O-bound processes.
- The scheduler selects a process from the highest priority queue that contains runnable processes and allocates the CPU to that process.

Diagram of Multilevel Feedback Queue Scheduling:



The main difference between multilevel queue scheduling and multilevel feedback queue scheduling lies in their dynamic behaviour and process movement between queues. Multilevel queue scheduling assigns processes to fixed priority queues, while multilevel feedback queue scheduling allows processes to move between priority queues based on their behaviour and performance. This flexibility in process movement enables better adaptation to varying workload characteristics and helps optimize system responsiveness and resource utilization.

Q.5. (a) How semaphores help in process synchronization? What is the difference between binary and counting semaphores? (6)

Ans. Semaphores are synchronization primitives used to coordinate and control access to shared resources in concurrent programming. They help ensure that multiple processes or threads cooperate correctly and avoid race conditions or conflicts when accessing shared resources. Semaphores provide a mechanism for processes to signal and wait for each other, enabling synchronization and mutual exclusion.

The main purpose of semaphores in process synchronization is to enforce the following properties:

1. Mutual Exclusion: Semaphores allow only one process or thread to access a shared resource at a time, ensuring that multiple processes do not interfere with each other while accessing the resource concurrently.

2. Synchronization: Semaphores enable processes or threads to synchronize their execution by coordinating their activities. They allow processes to wait until a certain condition is met or until a resource becomes available before proceeding further.

3. Deadlock Avoidance: Semaphores can be used to prevent deadlock situations where processes are stuck waiting indefinitely for resources. By properly managing semaphore operations and resource allocation, deadlock scenarios can be avoided.

Now, let's discuss the difference between binary semaphores and counting semaphores:

1. Binary Semaphores: A binary semaphore, also known as a mutex (short for mutual exclusion), has two possible values: 0 and 1. It serves as a simple lock that allows only one process or thread to enter a critical section at a time. When a process acquires a binary semaphore, it sets the semaphore value to 0, indicating that the critical section is occupied. If another process attempts to acquire the semaphore while it is already locked (value 0), it will be blocked and put in a waiting state until the semaphore is released (value 1) by the owning process. Binary semaphores are commonly used for mutual exclusion and protecting critical sections.

2. Counting Semaphores: A counting semaphore can have any non-negative integer value. It keeps track of the number of available resources and allows multiple processes or threads to access a resource concurrently, up to a certain limit defined by the semaphore value. Each time a process acquires the semaphore, the value is decremented, and when the process releases the semaphore, the value is incremented. If the semaphore value reaches zero, indicating that all resources are currently in use, subsequent processes attempting to acquire the semaphore will be blocked and put in a waiting state. Counting semaphores are useful for resource allocation and synchronization scenarios where multiple resources can be used concurrently up to a certain limit.

Q.5. (b) Process P1, P2 and P3 arrive for execution at times indicated. Using non pre-emptive scheduling, answer the questions below: (6.5)

Process	Arrival Time	Burst time
P1	0.0	8
P2	0.4	4
P3	0.8	1

(i)What is average turnaround time for these processes with FCFS scheduling?

(ii)What is the average turnaround time for these processes with SJF scheduling?

(iii)What is the average turnaround time if the CPU is left idle for first 1 unit and then SJF scheduling is used?

Ans. To calculate the average turnaround time for the given processes using different scheduling algorithms, we need to consider the arrival time and burst time of each process. Let's calculate the average turnaround time for each case:

(i) FCFS Scheduling:

Process	Arrival Time	Burst time	Completion time	Turnaround time
P1	0.0	8	8	8
P2	0.4	4	12	11.6
P3	0.8	1	13	12.2

Average Turnaround Time = (8 + 11.6 + 12.2) / 3 = 10.6

(ii) SJF Scheduling:

Process	Arrival Time	Burst time	Completion time	Turnaround time
P1	0.0	8	12	12
P2	0.4	4	6	5.6
P3	0.8	1	1.8	1

Average Turnaround Time = (12 + 5.6 + 1) / 3 = 6.2

(iii) CPU Idle for 1 Unit, followed by SJF Scheduling:

Process	Arrival Time	Burst time	Completion time	Turnaround time
P2	1.4	4	5	4.6
P3	1.8	1	2.8	2

Average Turnaround Time = (4.6 + 2) / 2 = 3.3

In the third case, since the CPU is idle for the first 1 unit, the arrival times of all processes are shifted by 1 unit.

UNIT - III

Q.6. (a) What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem? (6)

Ans. Thrashing is a phenomenon that occurs in operating systems when a system is overwhelmed with excessive paging or swapping activity, resulting in a significant decrease in overall system performance. It happens when the system spends more time on paging or swapping operations than on actual useful work, leading to a state of constant high disk I/O and low CPU utilization.

The primary cause of thrashing is when the system is trying to allocate more memory to processes than is physically available. This situation typically arises when the system is overloaded with too many processes, and each process requires a large

amount of memory. As a result, the operating system starts swapping pages between main memory and secondary storage (such as the hard disk) frequently, in an attempt to provide memory to all active processes.

To detect thrashing, the system can use various monitoring techniques:

1. Page Fault Rate Monitoring: The system can monitor the rate of page faults, which indicates the frequency at which processes are requiring pages that are not currently in main memory. If the page fault rate becomes excessively high, it could be an indication of thrashing.

2. CPU Utilization Monitoring: If the CPU utilization is consistently low while the system is under heavy load, it could suggest that the system is spending more time on paging or swapping operations than executing processes. This low CPU utilization combined with high I/O activity is a sign of thrashing.

Once thrashing is detected, the system can take several measures to eliminate or mitigate the problem:

(a)Locality-Based Page Replacement Algorithms: The system can use page replacement algorithms that prioritize pages with higher locality, such as the Least Recently Used (LRU) algorithm. By keeping frequently accessed pages in main memory, the system reduces the need for excessive paging.

(b)Working Set Model: The system can utilize the working set model, which identifies the minimum set of pages required by a process to run efficiently. By monitoring and adjusting the working set size for each process, the system can minimize thrashing by ensuring that essential pages remain in memory.

(c)Resource Allocation Adjustment: The system can dynamically adjust resource allocations, such as reducing the number of active processes or limiting the memory allocated to each process. By better managing resource utilization, the system can prevent overloading and reduce the chances of thrashing.

(d)Process Prioritization: The system can prioritize critical processes or processes with high priority to ensure their efficient execution. By focusing on executing important processes first, the system can prevent them from being affected by thrashing.

(e)Adding Physical Memory: Increasing the amount of physical memory available to the system can alleviate thrashing by reducing the need for excessive paging or swapping. This involves upgrading the hardware configuration of the system to accommodate the memory requirements of the processes.

Q.6. (b) Consider the following page reference string: **(6.5)**

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

How many page faults would occur for the following replacement algorithms assuming three frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

(i) LRU replacement

(ii) FIFO replacement

(iii) Optimal replacement

Ans. To calculate the number of page faults for the given page reference string using different page replacement algorithms, we need to simulate the execution and track the state of the page frames. Let's calculate the page faults for each algorithm with three frames:

Given page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

(i) LRU Replacement: Initially: | - | - | - |

1 (fault) | 1 | - | - | 2 (fault) | 1 | 2 | - | 3 (fault) | 1 | 2 | 3 | 4 (fault) | 4 | 2 | 3 | 2 (fault) | 4 | 2 | 3 | 1 (fault) | 4 | 1 | 3 | 5 (fault) | 4 | 1 | 5 | 6 (fault) | 4 | 1 | 6 | 2 | 4 | 2 | 6 | 1 | 4 | 1 | 6 | 2 | 4 | 2 | 6 | 3 (fault) | 3 | 2 | 6 | 7 (fault) | 3 | 7 | 6 | 6 | 3 | 7 | 6 | 3 | 3 | 7 | 6 | 2 | 3 | 7 | 2 | 1 | 1 | 7 | 2 | 2 | 1 | 2 | 7 | 3 | 1 | 2 | 3 | 6 (fault) | 1 | 6 | 3 |

Total Page Faults: 14

(ii) FIFO Replacement: Initially: | - | - | - |

1 (fault) | 1 | - | - | 2 (fault) | 1 | 2 | - | 3 (fault) | 1 | 2 | 3 | 4 (fault) | 4 | 2 | 3 | 2 (fault) | 4 | 2 | 3 | 1 (fault) | 4 | 2 | 1 | 5 (fault) | 5 | 2 | 1 | 6 (fault) | 5 | 6 | 1 | 2 | 5 | 6 | 2 | 1 | 5 | 6 | 2 | 2 | 5 | 6 | 2 | 3 (fault) | 3 | 6 | 2 | 7 (fault) | 3 | 7 | 2 | 6 | 3 | 7 | 6 | 3 | 3 | 7 | 6 | 2 | 3 | 7 | 2 | 1 | 1 | 7 | 2 | 2 | 1 | 2 | 7 | 3 | 1 | 2 | 3 | 6 (fault) | 1 | 6 | 3 |

Total Page Faults: 16

(iii) Optimal Replacement: Initially: | - | - | - |

1 (fault) | 1 | - | - | 2 (fault) | 1 | 2 | - | 3 (fault) | 1 | 2 | 3 | 4 (fault) | 4 | 2 | 3 | 2 | 4 | 2 | 3 | 1 | 4 | 1 | 3 | 5 (fault) | 4 | 1 | 5 | 6 (fault) | 4 | 1 | 6 | 2 | 4 | 1 | 6 | 1 | 4 | 1 | 6 | 2 | 4 | 1 | 6 | 3 (fault) | 3 | 1 | 6 | 7 (fault) | 3 | 1 | 7 | 6 | 3 | 1 | 7 | 3 | 3 | 1 | 7 | 2 | 3 | 1 | 2 | 1 | 3 | 1 | 2 | 2 | 3 | 1 | 2 | 3 | 3 | 1 | 2 | 6 (fault) | 3 | 1 | 6 |

Total Page Faults: 15

Therefore, the number of page faults for each replacement algorithm with three frames is as follows: (i) LRU replacement: 14 page faults (ii) FIFO replacement: 16 page faults (iii) Optimal replacement: 15 page faults

Q.7. (a) Write Banker's algorithm for deadlock avoidance. **(6)**

Ans. The Banker's algorithm is a deadlock avoidance algorithm used in operating systems to manage resource allocation and avoid the possibility of deadlock. It ensures that the system only grants resource requests if the resulting state is safe, meaning that there is a sequence of resource allocation that will allow all processes to complete without deadlock.

Here's the pseudocode for the Banker's algorithm:

1. Initialize the following data structures:

- Available: Vector of length M representing the number of available instances of each resource.
- Max: Matrix of size N × M representing the maximum resource requirement of each process.
- Allocation: Matrix of size N × M representing the currently allocated resources for each process.
- Need: Matrix of size N × M representing the remaining resource need for each process.

2. Calculate the Need matrix: Need[i, j] = Max[i, j] - Allocation[i, j] for all i in range N and j in range M.

3. Define the work and finish vectors:

- Work: A vector of length M initially set to the Available vector.
- Finish: A vector of length N initially set to false for all processes.

4. Find a process *i* such that:

- Finish[i] is false.
- Need[i, j] <= Work[j] for all j in range M.

5. If such a process *i* is found, it means there is a safe sequence, and we can simulate granting the resources to process i:

- Work[j] += Allocation[i, j] for all j in range M.
- Finish[i] = true.

6. Repeat steps 4 and 5 until no more processes can be found.

7. If all processes are marked as finished, the system is in a safe state, and the resources can be allocated. Otherwise, the system is in an unsafe state, and resources cannot be allocated without the risk of deadlock.

The Banker's algorithm ensures that resources are allocated in a way that prevents deadlock by checking for safety before granting resource requests. By maintaining the Available, Max, Allocation, and Need matrices, the algorithm can determine if a safe sequence exists and make resource allocation decisions accordingly.

Q.7. (b) What is virtual memory? How demand paging supports the virtual memory? Explain in detail. **(6.5)**

Ans. Virtual memory is a memory management technique used in operating systems to provide an illusion of a larger and more capable main memory than what is physically available. It allows processes to use more memory than is physically installed on the system by using a combination of main memory (RAM) and secondary storage (typically a hard disk).

Virtual memory works by dividing the logical address space of a process into fixed-size units called pages. Similarly, the physical memory is divided into fixed-size blocks called frames. The virtual memory manager maps pages of the process to frames in physical memory, and when a process accesses a page that is not currently in physical memory, a page fault occurs.

Demand paging is a strategy employed by virtual memory systems to optimize memory usage and improve overall system performance. It supports virtual memory by loading only the necessary pages into physical memory when they are demanded by a process, rather than loading the entire process into memory at once.

When a process requests a page that is not present in physical memory, a page fault is triggered, and the operating system retrieves the required page from secondary storage (disk) and loads it into an available frame in physical memory. The process is then resumed, and it can access the page that caused the fault.

Demand paging provides several benefits in supporting virtual memory:

1. Efficient Memory Utilization: Demand paging allows processes to use more memory than the physical memory capacity, as only the necessary pages are loaded into memory when needed. This improves memory utilization and enables the execution of larger programs that may not fit entirely in physical memory.

2. Faster Process Startup: By loading only the essential pages into memory initially, the process startup time is reduced. Only the pages required for the process's immediate execution are loaded, while the remaining pages are loaded on-demand as the process progresses.

3. Reduced I/O Overhead: Demand paging minimizes the amount of data transferred between secondary storage and physical memory. Only the requested pages are loaded, which reduces the I/O overhead compared to loading the entire process into memory.

4. Increased Multiprogramming: Demand paging allows for efficient multiprogramming, enabling multiple processes to coexist in memory concurrently. The operating system can manage the available physical memory effectively by swapping pages in and out as needed, optimizing resource utilization.

5. Simplified Memory Management: With demand paging, the operating system can manage memory more flexibly. It can allocate physical memory frames dynamically as needed and prioritize the allocation based on page faults and process requirements.

However, demand paging also introduces some challenges:

1. Page Faults: Demand paging introduces the overhead of page faults, as accessing a page not currently in physical memory requires loading it from disk. This can cause some delays in process execution due to the time required for I/O operations.

2. Thrashing: If the demand for pages exceeds the available physical memory, and the system spends a significant amount of time swapping pages between memory and disk, it can lead to thrashing. Thrashing occurs when the system is continuously busy with swapping, resulting in low CPU utilization and poor overall performance.

Virtual memory provides an expanded address space for processes, and demand paging optimizes memory usage by loading pages into physical memory only when they are needed. This allows for efficient memory utilization, faster process startup, reduced I/O overhead, increased multiprogramming, and simplified memory management. However, it also introduces the challenge of page faults and the potential for thrashing if not managed effectively by the operating system.

UNIT - IV

Q.8. (a) Explain about file directories and its attributes in detail. (6)

Ans. File directories, also known as folders, are a fundamental component of file systems in operating systems. They serve as organizational structures for storing and managing files. Directories provide a hierarchical structure that allows for the organization and categorization of files into a logical and easily navigable system.

Attributes of File Directories:

1. Name: Every directory has a unique name that identifies it within the file system. The name is typically alphanumeric and may include special characters depending on the file system.

2. Path: The path of a directory specifies its location within the file system hierarchy. It consists of a series of directory names separated by slashes ("/") or backslashes ("\"). For example, "/home/user/documents" represents the path to the "documents" directory located within the "user" directory, which is in turn within the "home" directory.

3. Parent Directory: Each directory, except for the root directory, has a parent directory. The parent directory is the directory that contains the current directory. It is denoted by ".." and allows for navigation to the parent directory.

4. Subdirectories: Directories can contain other directories, forming a nested structure. These nested directories are referred to as subdirectories. Subdirectories provide a way to organize files and create a hierarchical structure within the file system.

5. Files: Directories can contain files, which are data entities stored on a storage medium. Files within a directory can be uniquely identified by their names within that directory.

6. Permissions: Directories, like files, can have permissions associated with them. Permissions control who can access, modify, and execute files and directories. Common permissions include read, write, and execute permissions for the owner, group, and other users.

7. Timestamps: Directories often have associated timestamps that indicate when certain operations were performed on them. Common timestamps include the creation time, last access time, and last modification time.

8. Size: The size of a directory represents the amount of storage space it occupies on the storage medium. This size is typically calculated based on the combined sizes of the files and subdirectories within the directory.

9. Special Directories: Some file systems have special directories with predefined names and purposes. For example, the root directory ("/") is the top-level directory in Unix-like systems, and it serves as the starting point for the file system hierarchy.

File directories play a crucial role in organizing and managing files within a file system. They provide a structured and logical organization that simplifies file access and management. By using directories, users can easily navigate through the file system, locate specific files, and maintain a well-organized file storage system.

Q.8. (b) What is RAID? Explain various RAID levels. **(6.5)**

Ans. RAID (Redundant Array of Independent Disks) is a technology used in computer storage to combine multiple physical disk drives into a single logical unit. RAID offers improved performance, fault tolerance, and data redundancy by distributing data across multiple disks or mirroring data across multiple drives.

There are several RAID levels, each offering different configurations and characteristics. The most common RAID levels are:

1. RAID 0 (Striping):

- Data is distributed across multiple drives in equal-sized chunks or stripes.
- Offers improved performance as data can be read and written in parallel across multiple drives.
- No redundancy or fault tolerance, as there is no data duplication.
- If one drive fails, the entire RAID 0 array may fail, resulting in data loss.

2. RAID 1 (Mirroring):

- Data is mirrored across multiple drives, creating an exact copy of the data on each drive.
- Provides data redundancy and fault tolerance as data is duplicated.
- If one drive fails, the mirrored drive can continue to serve data.
- Read performance can be improved, as data can be read from either drive in parallel.
- Typically, the capacity is limited to the size of a single drive, as the data is duplicated.

3. RAID 5 (Striping with Parity):

- Data and parity information are distributed across multiple drives.
- Provides fault tolerance by calculating and storing parity information, which can be used to reconstruct data in case of a drive failure.
- Offers a balance between performance and data redundancy.
- Requires a minimum of three drives.
- If one drive fails, the parity information can be used to rebuild the data on a replacement drive.

4. RAID 6 (Striping with Double Parity):

- Similar to RAID 5, but with two sets of parity information.
- Offers higher fault tolerance compared to RAID 5 as it can withstand the failure of two drives.
- Requires a minimum of four drives.
- Provides data redundancy and the ability to reconstruct data even if two drives fail simultaneously.

5. RAID 10 (Combination of RAID 1 and RAID 0):

- Combines mirroring (RAID 1) and striping (RAID 0).
- Data is mirrored across pairs of drives, and then the mirrored pairs are striped.
- Offers both performance improvement and fault tolerance.
- Provides high read and write performance due to the striping.
- Requires a minimum of four drives, with an even number of drives.

These are just a few examples of common RAID levels. There are other RAID levels such as RAID 2, RAID 3, RAID 4, RAID 7, and more, each with its own specific configurations and characteristics. The choice of RAID level depends on factors such as performance requirements, data redundancy needs, cost considerations, and the number of available drives.

Q.9. (a) What are the three methods for allocating disk space? Explain. **(6)**

Ans. The three methods for allocating disk space are:

1. Contiguous Allocation:

- In contiguous allocation, files are stored in consecutive blocks on the disk.
- Each file occupies a contiguous block of disk space, meaning the blocks are physically adjacent to each other.
- The starting block address and the length of the file are stored in the file's directory entry.
- Contiguous allocation provides fast and efficient file access since files are stored contiguously.
- However, it suffers from the drawback of external fragmentation, where free space becomes scattered across the disk as files are created, modified, and deleted. This fragmentation can lead to inefficient disk space utilization over time.

2. Linked Allocation:

- Linked allocation represents each file as a linked list of disk blocks.
- Each block contains a pointer to the next block in the file.
- The last block of the file has a null pointer, indicating the end of the file.
- The directory entry of a file contains the address of the first block of the file.
- Linked allocation avoids external fragmentation since files can be allocated in any available free space on the disk.
- However, it suffers from the overhead of traversing the linked list to access data, which can impact performance, especially for large files.

3. Indexed Allocation:

- Indexed allocation uses an index block to store pointers to all the blocks of a file.
- Each file has its own index block, which contains a fixed number of pointers.
- The index block serves as a lookup table, mapping logical block addresses to physical block addresses.
- The directory entry of a file contains the address of its index block.
- Indexed allocation provides fast direct access to any block of a file by using the index block.
- It eliminates external fragmentation, as files can be allocated in any available free space.
- However, it introduces the overhead of maintaining the index block and requires additional disk space for the index blocks.

Each allocation method has its advantages and disadvantages. Contiguous allocation provides efficient access but suffers from external fragmentation. Linked allocation avoids fragmentation but introduces traversal overhead. Indexed allocation allows for direct access and eliminates fragmentation but requires additional space for the index blocks. The choice of allocation method depends on factors such as file size, access patterns, and disk space utilization requirements. Modern file systems often use a combination of these methods or variants to optimize file allocation and space utilization.

Q.9. (b) Describe various file access methods. (6.5)

Ans. Various file access methods are used to read and write data from files in an operating system. The commonly used file access methods are:

1. Sequential Access:

- In sequential access, data is read or written sequentially from the beginning of the file to the end.
- Reading or writing starts at the beginning of the file and proceeds sequentially until the desired data is reached.
- It is suitable for accessing files that are processed in a serial manner, such as log files or tape drives.
- Sequential access is simple and efficient for accessing data sequentially but inefficient for random access or searching specific data within the file.

2. Direct Access:

- Direct access, also known as random access or indexed access, allows direct and non-sequential access to any location within the file.
- Each record in the file has a unique identifier or record number, and records can be read or written directly by specifying their record number.
- Direct access is useful when quick access to specific records is required, such as in databases.
- It requires an index or a file allocation table to map record numbers to physical locations in the file.

3. Indexed Sequential Access Method (ISAM):

- ISAM combines sequential and direct access methods.
- The file is accessed sequentially, but an index is used to provide direct access to specific records.
- The index contains key fields from the records along with pointers to their physical locations in the file.
- ISAM provides fast access to specific records while still allowing sequential access when needed.
- It is commonly used in database systems for efficient data retrieval and manipulation.

4. Hashed Access:

- Hashed access is a method used for direct access to files using a hashing technique.
- Each record in the file is assigned a unique key, which is used to compute a hash value.
- The hash value is then used to determine the location of the record within the file.
- Hashed access provides fast retrieval of records based on their keys but may suffer from collisions if multiple records have the same hash value.
- It is commonly used in hash tables and databases where quick access to specific records is crucial.

These file access methods offer different trade-offs in terms of efficiency, flexibility, and suitability for different types of applications. The choice of file access method depends on factors such as the type of data, access patterns, required performance, and the specific requirements of the application or system.