Hall Ticket Number:

II/IV B.Tech (Regular) DEGREE EXAMINATION

DECEMBER, 2019 — **Computer Science and Engineering**

Third Semester — **Operating Systems**

Time: Three Hours — **Maximum :** 50 Marks

Answer Question No.1 compulsorily. (1X10 = 10 Marks)

Answer ONE question from each unit. (4X10=40 Marks)

1. Answer all questions (1X10=10 Marks)

a Define Operating System
b Define Response Time
c Describe various process states
d Define threads
e Define Inter Process Communication(IPC)
f Define Semaphore
g Define Race Condition
h Define Page Fault
i Describe various Operations performed on files
j Define Access Matrix

UNIT – I

2.a Explain different services provided by the Operating Systems 6M
2.b Explain process state transition with neat sketch 4M

(OR)

3.a Explain Process Schedulers 5M
3.b Explain message passing mechanism in Inter Process Communication 5M

UNIT – II

4.a Define Critical Section. Explain Peterson's Solution 4M
4.b Explain Producer-Consumer Problem with the help of Semaphore 6M

(OR)

5. Consider the following set of processes, with the length of the CPU burst given in milliseconds: 10 M

Process	Burst Time	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all arrive at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF and non-preemptive priority (a larger priority number implies a higher priority)

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

c. What is the waiting time of each process for each of these scheduling algorithms?

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

UNIT – III

6.	Explain Deadlock Detection and its Recovery methods	10M

(OR)

7.	Consider the reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0,1 for a memory with three frames. Trace FIFO, optimal, and LRU page replacement algorithms	10M

UNIT – IV

8.a	Explain file system structure in detail	5M
8.b	Explain implementation of access matrix	5M

(OR)

9.	Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25 Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using i) FCFS ii)SSTF iii)SCAN iv)C-SCAN scheduling.	10M

SCHME

(1X10 = 10 Marks)

a Define Operating System

An operating system is a program that manages a computer's hardware. It acts as an intermediary between the computer user and the computer hardware

b Define Response Time

the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding.

c Describe various process states

As a process executes, it changes state. A process may be in one of the following states:

- New: The process is being created.
- Running: Instructions are being executed.
- Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
- Ready: The process is waiting to be assigned to a processor.
- Terminated: The process has finished execution.

d Define threads

A thread is the smallest unit of processing that can be performed in an OS. In most modern operating systems, a thread exists within a process - that is, a single process may contain multiple threads.

e Define Inter Process Communication(IPC)

Cooperating processes require an Inter Process Communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of Inter Process Communication: shared memory and message passing.

f Define Semaphore

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations wait and signal that are used for process synchronization.

g Define Race Condition

It is the condition where several processes tries to access the resources and modify the shared data concurrently and outcome of the process depends on the particular order of execution that leads to data inconsistency, this condition is called Race Condition

h Define Page Fault

A page fault is a type of exception raised by computer hardware when a running program accesses a memory page that is not currently mapped by the memory management unit (MMU) into the virtual address space of a process.

i Describe various Operations performed on files

The operating system can provide system calls to create, write, read, reposition, delete, and truncate files. There are six basic file operations within an Operating system.

j Define Access Matrix

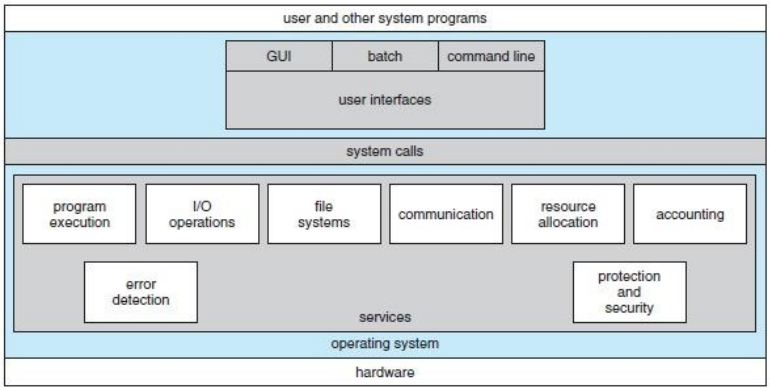
In computer science, an Access Control Matrix or Access Matrix is an abstract, formal security model of protection state in computer systems, that characterizes the rights of each subject with respect to every object in the system.

UNIT – I

2.a Explain different services provided by the Operating Systems 6M

Operating System Services

Following are the five services provided by operating systems to the convenience of the users.



Program Execution

The purpose of computer systems is to allow the user to execute programs. So the operating system provides an environment where the user can conveniently run programs. Running a program involves the allocating and de allocating memory, CPU scheduling in case of multiprocessing.

I/O Operations

Each program requires an input and produces output. This involves the use of I/O. So the operating systems are providing I/O makes it convenient for the users to run programs.

File System Manipulation

The output of a program may need to be written into new files or input taken from some files. The operating system provides this service.

Communications

The processes need to communicate with each other to exchange information during execution. It may be between processes running on the same computer or running on the different computers. Communications can occur in two ways: (i) shared memory or (ii) message passing

Error Detection

An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation operating system constantly monitors the system for detecting the errors. This relieves the user of the worry of errors propagating to various part of the system and causing malfunctioning.

Following are the three services provided by operating systems for ensuring the efficient operation of the system itself.

Resource allocation

When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system.

Accounting

The operating systems keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.

Protection

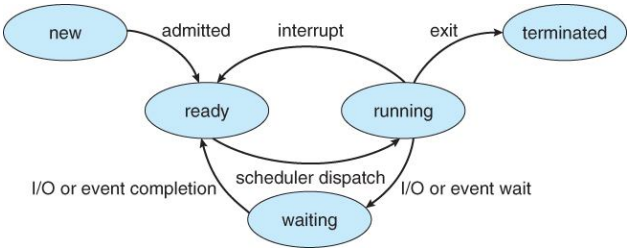
When several disjointed processes execute concurrently, it should not be possible for one process to interfere with the others, or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with each user having to authenticate him to the system, usually by means of a password, to be allowed access to the resources.

2.b Explain process state transition with neat sketch 4M
[Diagram-2M and explanation- 2M]

Process: A process is program under execution.

It's a program during execution i.e., executable file loaded into main memory. It's an active entity with program counter (in PCB) to specify the next instruction to be executed & set of associated resources.

A process includes heap, stack, data & text sections.

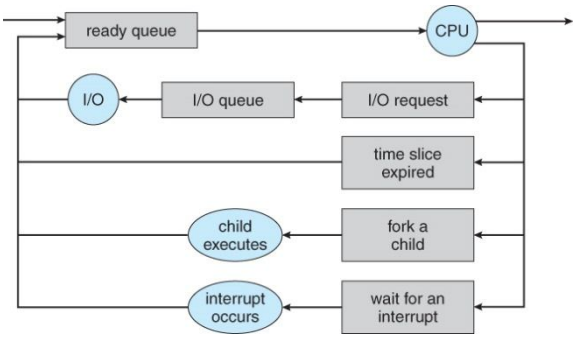


(OR)

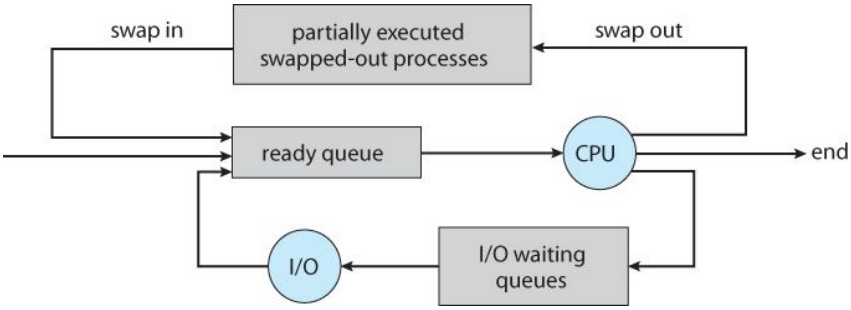
3.a Explain Process Schedulers 5M

A long-term scheduler is typical of a batch system or a very heavily loaded system. It runs infrequently, (such as when one process ends selecting one more to be loaded in from disk in its place), and can afford to take the time to implement intelligent and advanced scheduling algorithms.

The short-term scheduler, or CPU Scheduler, runs very frequently, on the order of 100 milliseconds, and must very quickly swap one process out of the CPU and swap in another one.



Queueing-diagram representation of process scheduling

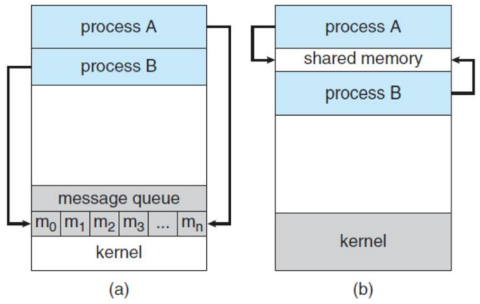


Addition of a medium-term scheduling to the queueing diagram

3.b Explain message passing mechanism in Inter Process Communication 5M

Processes executing concurrently in the operating system may be either independent processes or cooperating processes. A process is independent if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent. A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

Cooperating processes require an inter process communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of inter process communication: shared memory and message passing. In the **shared-memory model**, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region. In the **message-passing model**, communication takes place by means of messages exchanged between the cooperating processes.



igure 3.12 Communications models. (a) Message passing. (b) Shared memory.

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. It is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network. For example, an Internet chat program could be designed so that chat participants communicate with one another by exchanging messages. A message-passing facility provides at least two operations:
send(message) receive(message)
If processes P and Q want to communicate, they must send messages to and receive messages from each other: a communication link must exist between them. This link can be implemented in a variety of ways.

- Direct or indirect communication
- Synchronous or asynchronous communication
- Automatic or explicit buffering

Naming:
Processes that want to communicate must have a way to refer to each other.They can use either direct or indirect communication. Under direct communication, each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as:
• send(P, message)—Send a message to process P.
• receive(Q, message)—Receive a message from process Q.
With indirect communication, the messages are sent to and received from mailboxes, or ports. A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed. Each mailbox has a unique identification.
The send() and receive() primitives are defined as follows:
• send(A, message)—Send a message to mailbox A.
• receive(A, message)—Receive a message from mailbox A.

Unit-II

4.a. Define Critical Section. Explain Peterson's Solution 4M

Critical Section: a critical section is a code segment that accesses shared variables and has to be executed as an

atomic action. The critical section problem refers to the problem of how to ensure that at most one process is executing its critical section at a given time.

a classic software-based solution to the critical-section problem known as Peterson's solution. Because of the way modern computer architectures perform basic machine-language instructions, such as load and store, there are no guarantees that Peterson's solution will work correctly on such architectures.

```
do{
flag[i] = true;
Turn=j;
While(flag[j] && turn ==j);
Critical section
Flag[i]=false;
Remainder section
} while (true);
```

Peterson's solution is restricted to two processes that alternate execution between their critical sections and remainder sections.

Peterson's solution requires the two processes to share two data items:

```
int turn;
boolean flag[2];
```

The variable turn indicates whose turn it is to enter its critical section. That is, if turn == i, then process Pi is allowed to execute in its critical section. The flag array is used to indicate if a process is ready to enter its critical section. For example, if flag[i] is true, this value indicates that Pi is ready to enter its critical section. With an explanation of these data structures complete, we are now ready to describe the algorithm shown in To enter the critical section, process Pi first sets flag[i] to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section, it can do so. If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time. Only one of these assignments will last; the other will occur but will be overwritten immediately. The eventual value of turn determines which of the two processes is allowed to enter its critical section first.

4.b. Explain Producer-Consumer Problem with the help of Semaphore 6M

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.

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

Process	Burst Time	Priority
P1	2	2
P2	1	1
P3	8	4
P4	4	2
P5	5	3

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, all arrive at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, non-preemptive priority (a larger priority number implies a higher priority),

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

c. What is the waiting time of each process for each of these scheduling algorithms?

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

FCFS Scheduling Algorithm:

Gantt Chart:

P1	P2	P3	P4	P5

0 2 3 11 15 20

Process	Burst Time	Completion Time	TAT=CT-AT	Waiting Time=TAT-BT
P1	2	2	2	0
P2	1	3	3	2
P3	8	11	11	3
P4	4	15	15	11
P5	5	20	20	15
			Average	41/5=8.2

SJF Scheduling Algorithm:

Gantt Chart:

P2	P1	P4	P5	P3

0 1 3 7 12 20

Process	Burst Time	Completion Time	TAT= CT-AT	Waiting Time= TAT-BT
P1	2	3	3	1
P2	1	1	1	0
P3	8	20	20	12
P4	4	7	7	3
P5	5	12	12	7
			Average	23/5=4.6

Priority Scheduling Algorithm:
Gantt Chart:

P3	P5	P1	P4	P2

0 8 13 15 19 20

Process	Burst Time	Completion Time	TAT= CT-AT	Waiting Time= TAT-BT
P1	2	15	15	13
P2	1	20	20	19
P3	8	8	8	0
P4	4	19	19	15
P5	5	13	13	8
			Average	55/5=11

Overall minimum average waiting time is :4.6 (Shortest Job First Scheduling Algorithm)

Unit-III

6. Explain Deadlock Detection and its Recovery methods

Deadlock Detection

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment,the system may provide:

• An algorithm that examines the state of the system to determine whether a deadlock has occurred
• An algorithm to recover from the deadlock

The detection algorithm described here simply investigates every possible allocation sequence for the processes that remain to be completed.

1. LetWork and Finish be vectors of length m and n, respectively.
   Initialize Work = Available. For i = 0, 1, ..., n–1,
   if Allocationi = 0, then
   Finish[i] = false. Otherwise,
   Finish[i] = true.
2. Find an index i such that both
   a. Finish[i] == false
   b. Requesti ≤Work

If no such i exists, go to step 4.

3. Work =Work + Allocationi
Finish[i] = true
Go to step 2.
4. If Finish[i] ==false for some i, 0≤i<n, then the system is in a deadlocked state. Moreover,
   if Finish[i] == false, then process Pi is deadlocked.

This algorithm requires an order of m × n^2 operations to detect whether the system is in a deadlocked state.

instances. Suppose that, at time T_0, we have the following resource-allocation state:

Allocation Request Available

	A B C	*A B C*	*A B C*
*P*0	0 1 0	0 0 0	0 0 0
*P*1	2 0 0	2 0 2	
*P*2	3 0 3	0 0 0	
*P*3	2 1 1	1 0 0	
*P*4	0 0 2	0 0 2	

We claim that the system is not in a deadlocked state. Indeed, if we execute our algorithm, we will find that the sequence <*P*0, *P*2, *P*3, *P*1, *P*4> results in ***Finish***[*i*] == *true* for all *i.*

Suppose now that process *P*2 makes one additional request for an instance of type *C.* The ***Request*** matrix is modified as follows:

Request

A B C

*P*0 0 0 0

*P*1 2 0 2

*P*2 0 0 1

*P*3 1 0 0

*P*4 0 0 2

We claim that the system is now deadlocked. Although we can reclaim the resources held by process *P*0, the number of available resources is not sufficient to fulfill the requests of the other processes. Thus, a deadlock exists, consisting of processes *P*1, *P*2, *P*3, and *P*4.

Recovery from Deadlock

When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock. One is simply to abort one or more processes to break the circular wait. The other is to preempt some resources from one or more of the deadlocked processes.

Process Termination:

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated

Aborting a process may not be easy. Many factors may affect which process is chosen, including

1) What the priority of the process is
2) How long the process has computed and how much longer the process
3) will compute before completing its designated task
4) Howmanyandwhat typesof resources theprocesshasused(for example,
5) whether the resources are simple to preempt)
6) How many more resources the process needs in order to complete
7) How many processes will need to be terminated
8) Whether the process is interactive or batch

Resource Preemption: To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

If preemption is required to deal with deadlocks, then three issues need to be addressed:

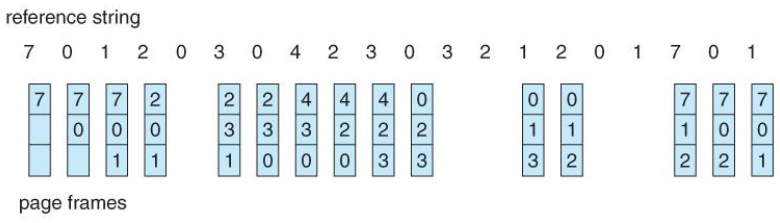
1. Selecting a victim
2. Rollback
3. Starvation

7. Consider the reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0,1 for a memory with three frames. Trace FIFO, optimal, and LRU page replacement algorithms

FIFO Page Replacement

A simple and obvious page replacement strategy is FIFO, i.e. first-in-first-out.

As new pages are brought in, they are added to the tail of a queue, and the page at the head of the queue is the next victim. In the following example, 20 page requests result in 15 page faults:

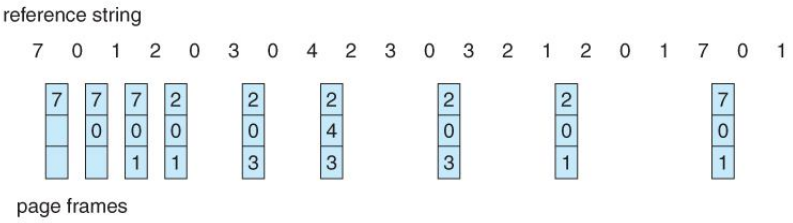


FIFO page-replacement algorithm.

Optimal Page Replacement

The discovery of Belady's anomaly lead to the search for an optimal page-replacement algorithm, which is simply that which yields the lowest of all possible page-faults, and which does not suffer from Belady's anomaly.

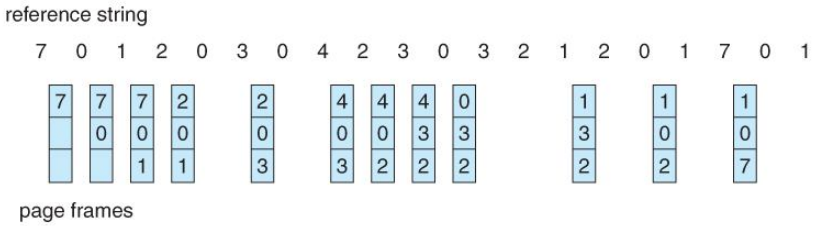
Such an algorithm does exist, and is called OPT or MIN. This algorithm is simply "Replace the page that will not be used for the longest time in the future."



LRU Page Replacement

The prediction behind LRU, the Least Recently Used, algorithm is that the page that has not been used in the longest time is the one that will not be used again in the near future. (Note the distinction between FIFO and LRU: The former looks at the oldest load time, and the latter looks at the oldest use time.)

Some view LRU as analogous to OPT, except looking backwards in time instead of forwards. (OPT has the interesting property that for any reference string S and its reverse R, OPT will generate the same number of page faults for S and for R. It turns out that LRU has this same property.)



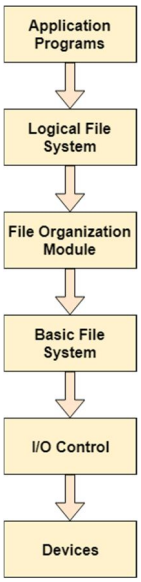
Unit-IV

8a. Explain file system structure in detail 5M

File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.

Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.

The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

8b. Explain implementation of access matrix 5M

Access Matrix:

- View protection as a matrix (access matrix)
- Rows represent domains
- Columns represent objects
- Access(i, j)is the set of operations that a process executing in $Domain_i$ can invoke on $Object_j$

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

Implementation of Access matrix:

Generally, a sparse matrix implemented

Option 1 –Global table

- Store ordered triples < domain, object, rights-set> in table
- A requested operation M on object Ojwithin domain Di-> search table for < Di, Oj, Rk>
- with M €Rk

- But table could be large -> won't fit in main memory
- Difficult to group objects (consider an object that all domains can read)

Option 2 –Access lists for objects

- Each column implemented as an access list for one object
- Resulting per-object list consists of ordered pairs < domain, rights-set > defining all domains with no-empty set of access rights for the object
- Easily extended to contain default set -> If M default set, also allow access
- Each column = Access-control list for one object Defines who can perform what operation
  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read
- Each Row = Capability List (like a key)For each domain, what operations allowed on what objects

  Object F1 –Read
  Object F4 –Read, Write, Execute
  Object F5 –Read, Write, Delete, Copy

Option 3 –Capability list for domains

- Instead of object-based, list is domain based
- Capability list for domain is list of objects together with operations allows on them
- Object represented by its name or address, called a capability
- Execute operation M on object Oj, process requests operation and specifies capability as parameter
- Possession of capability means access is allowed
- Capability list associated with domain but never directly accessible by domain
- Rather, protected object, maintained by OS and accessed indirectly
- Like a "secure pointer"
- Idea can be extended up to applications

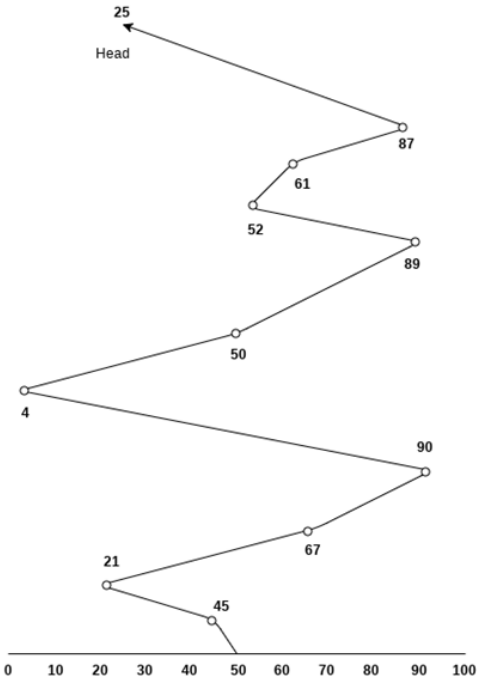
Option 4 –Lock-key

- Compromise between access lists and capability lists
- Each object has list of unique bit patterns, called locks
- Each domain as list of unique bit patterns called keys
- Process in a domain can only access object if domain has key that matches one of the locks

9 Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25 Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using i) FCFS ii)SSTF iii)SCAN iv)C-SCAN scheduling. 10M

i)FCFS:

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.



Number of cylinders moved by the head
= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)
= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62
= 376

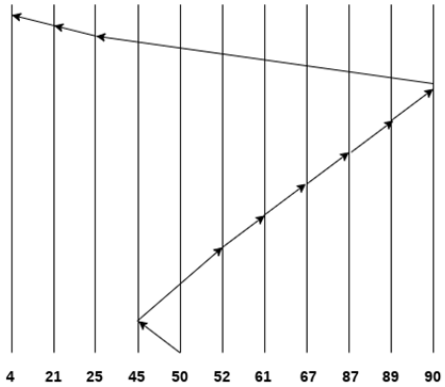
ii)Shortest Seek Time First:

Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS.

It allows the head to move to the closest track in the service queue.
Consider the following disk request sequence for a disk with 100 tracks

45, 21, 67, 90, 4, 89, 52, 61, 87, 25

Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

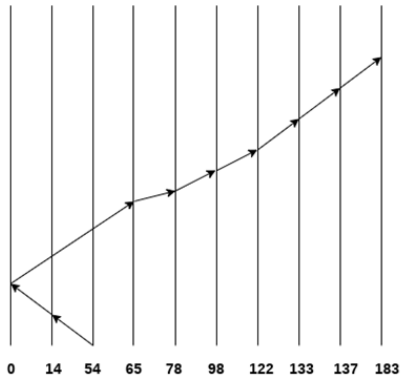


Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

iii) Scan Algorithm

It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path,and then it turns backand moves in the reverse direction satisfying requests coming in its path.
It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.
Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using

SCAN scheduling.

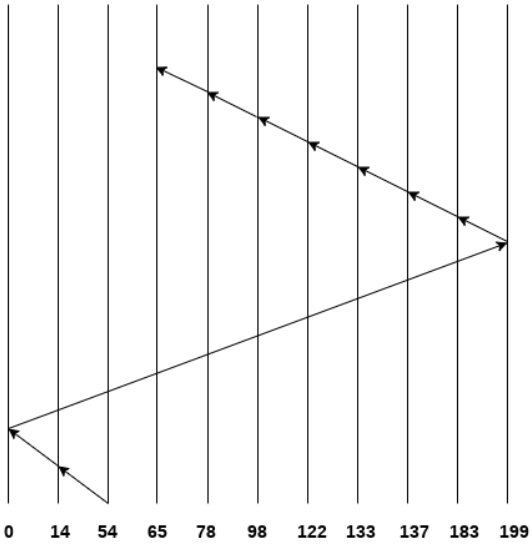


Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

iv)C-SCAN algorithm

In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

	Name of the faculty	**Signature**
Question Paper & Scheme Prepared by	Dr.Shaik Nazeer, Prof., Dept. of CSE	
Question Paper & Scheme Verified by	Prof. V Chakradhar, 2nd Year Class Coordinator, Dept. of CSE	