Assignment-1

1 What is multiprogramming? What are its advantages?

Ans: Multiprogramming:

One of the most important aspects of operating system is the ability to multiprogram. A single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Single users frequently have multiple programs running. Multiprogramming increases CPU utilization by organizing Jobs so that the cpu always has one to execute.

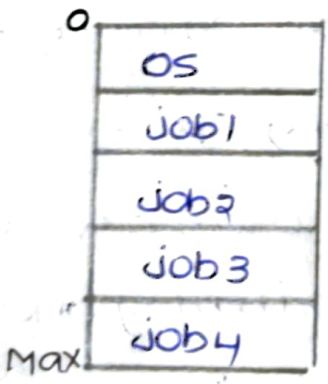


Fig: Memory layout for multiprogramming system

The idea is as follows: The operating system keeps several tracks Jobs in memory simultaneously. Since in general main memory is too small to accomodate all Jobs the Jobs are kept initially on the disk in Job pool. This pool consists of all processes residing on disk awaiting allocation of main memory.

The set of Jobs in memory can be a subset of the Jobs kept in Job pool. The OS picks and begins to execute Jobs in the memory. And other Job have to wait

such as I/O operation to complete. In non-multiprogrammed cpu will sit idle. In a multiprogrammed system the OS simply switches to and executes another job. If the job wants to idle cpu executes another job. As long as at least one job needs to execute, the CPU is never idle.

Multiprogrammed systems provides an environment in which various system resources are utilized effectively but they do not provide for user interaction with computer. The sharing is an logical extension of multi programming. In time sharing cpu executes several jobs by switching among them. users also can interact with this. A program loaded into memory and executes is called process. In operating system must ensure reasonable for response time.

Advantages of Multiprogramming

* Increases cpu utilization by organizing jobs.
* cpu never be idle until all jobs are executed.
* several jobs are placed in the main memory.
* processor switched from job to job needed to keep several jobs advancing keeping the peripheral devices in use.
* It is the first instance where operating system must make decisions for the users.

2. Explain different operating system services that helpful to the user? Explain

Ans operating system services

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided of course differ from one OS to another but we can identify common classes. These OS services provided for convenience of the programmer to make programming easier

One set of OS services provides functions that are helpful to the user.

* User interface
* Program executions
* I/O operations
* File-system manipulation
* communications
* Error detection

Another set of services that are not helpful for users but useful for the efficient operation of the system are

* Resource Allocation
* Accounting
* protection and security

1. User interface : Almost all operating systems have user interface. This interface take several forms. one is command line interface which use text commands and method for entering them. Another is Batch interface which commands and directories to control commands and enter into files.

2. Program Execution: The systems must be able to load a program into memory and to run program. The program must be

able to end its execution either normally or abnormally.

3. I/O operations: A running program may require I/O which may involve a file or an I/O device. For specific devices special functions may be desired.

4. File systems Manipulation: The file system of particular interest. Obviously programmers need to read and write files and directories. They also need to create and delete them by name, search for given file and list file information.

5. communication: There are many circumstances in which one process need to exchange information with another process. Such communication may occur processes that are executing on the same computer or between processes that are executing on different computer system.

6. Error detection: The operating system needs to be detecting and correcting errors constantly. Errors may occur in the CPU and memory hardware in I/O devices and in user progra

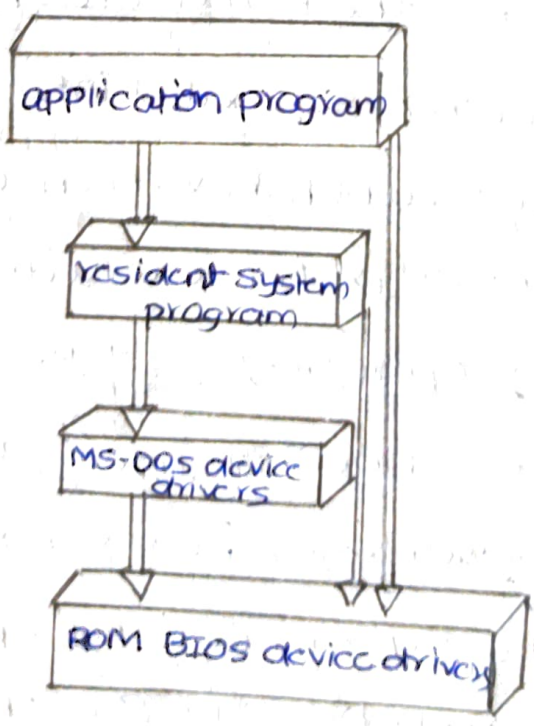
7. Resource Allocation: When there are multiple users or multiple Jobs running at the same time, resources must be allocated each of them. The OS manages all types of Resources.

8. Accounting: We want to keep track of which users use how much and what kinds of computer resources.

9 protection and security: The owners of information stored in a multiuser or networked computer system may wanto to control and use of that information.

3. Explain the MS-DOS layer structure with a neat diagram

Ans: Many operating systems do not have well defined structures. Frequently such systems started as small, simple and limited systems and then grew beyond their original scope. MS-DOS is an Example of such a system It was originally designed and implemented by a few people who had no idea that it would become so popular It was written to provide the most fonctionality in the least space, so it was not carefully divided into modules



MS-DOS layer structure

In MS-DOS the interfaces and levels of functionality are not well separated. For instance application programs are able to access the basic I/O routines to write directly to the display and disk drivers. Such freedom leaves MS-DOS structure vulnerable to errant programs, cause entire system crashes when user programs fail. Of course MS-DOS was also limited by the hardware of its era. Because the Intel 8088 for which it was written provides no dual mode and no hardware protection, the designers of MS-DOS had no choice but to leave the base hardware accessible.

Another Example of limited structuring is the original UNIX operating system. Like MS-DOS, UNIX initially was limited by hardware limitionally. It consists of two separatable parts: the kernel and the system programs. The kernel is further separated into a series of interfacy and device drivers, which have been added and expanded over the years as UNIX has evolved. We can view the traditional UNIX operating system as being layered to some extent, as shown in figure. It had a distinct performance advantage, however there is very little overhead in the system call interface or in communication with the kernel. We still see the evidence of this simple, monolithic structure in the UNIX, Linux and Windows operating system.

4. Explain various states and transitions between states with the help of a diagram?

Ans: A program in execution is called process.

A process is more than the program code which is sometimes known as the text section. It also represents the current activity of operating system was Job processing. By the value of program counter the programs are executed.

Let us see how the processes are stored in the memory.

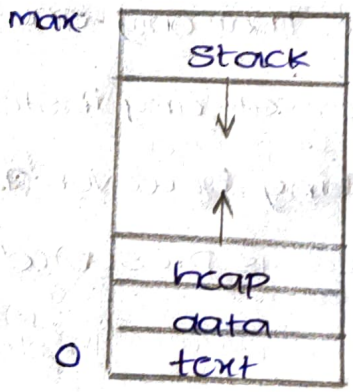


Fig: A process in memory

Process states

As a process executes it changes state. The state of a process is defined in part by the current activity of that process. 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.

These names are arbitary and they vary across operating systems. The states that they represent are found on all systems however certain operating system also more finely delienate process states. It is important to realise that only one process can be running or any processor at any instant. many processes may be ready and waiting however. The state diagram corresponding to these states is presented in following figure

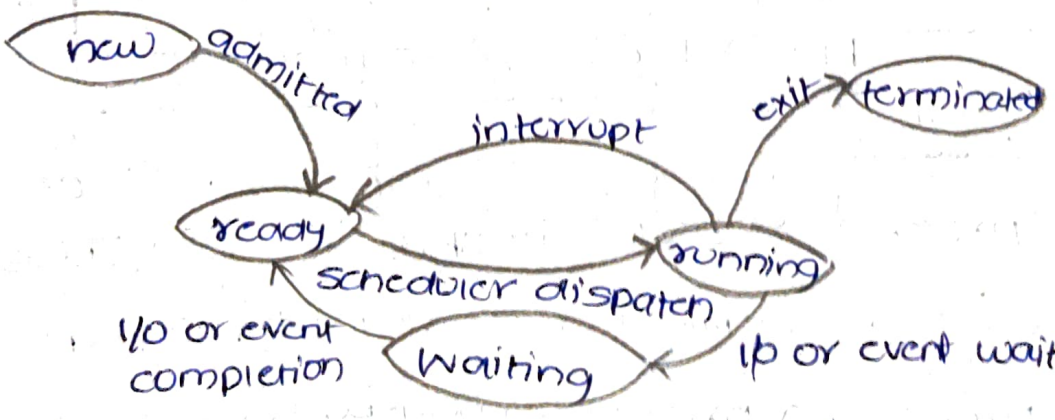


Fig: Diagram of process state

5 Explain the following scheduling algorithms with examples

(i) First come First serve

ii) Shortest Job First

ns i) First come first serve

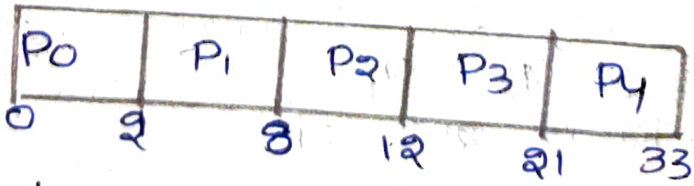
FCFS scheduling algorithm simply schedules the Jobs according to arrival time. The Job which comes first in the ready queue will get the cpu first. The lesser the arrival time of the Job the sooner will the Job will get the cpu. FCFS scheduling may causes the problem of Stravation if the burst time of first process is longest among all the Jobs.

Example

Process ID	Arrival time	Burst time	completion time	Turn around time	waiting time
0	0	2	2	2	0
1	1	6	8	7	1
2	2	4	12	10	6
3	3	9	21	18	9
4	6	12	33	29	17

Avg Waiting time = 31/5

Gantt chart:



Waiting time = turn around time - Burst time

turn around time = complete time - Arrival time

ii) Shortest Job First

Till now we were scheduling the processes according to their arrival time. However SJF scheduling algorithms schedules the processes according to their burst time.

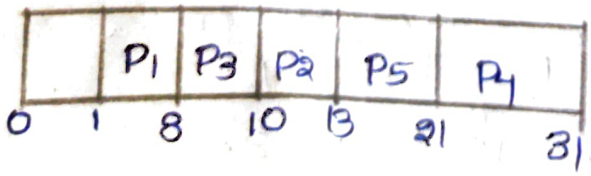
In SJF scheduling the process with the lowest burst time among the list of available processes in the ready queue is going to be scheduled next.

However it is very difficult to predict the burst time needed for a process. So this algorithm is very difficult to implement in the system.

Example:

PID	Arrival Time	Burst Time	completion Time	T.A.T	Waiting Time
1	1	7	8	7	0
2	3	3	13	10	7
3	6	2	10	4	2
4	7	10	31	24	14
5	9	8	21	12	4

Gantt chart



Avg waiting time = 27/5

TAT = CA - AT

WT = TAT - BT

6. What is semaphore? Give the solution for dining-philosophers problem using semphores.

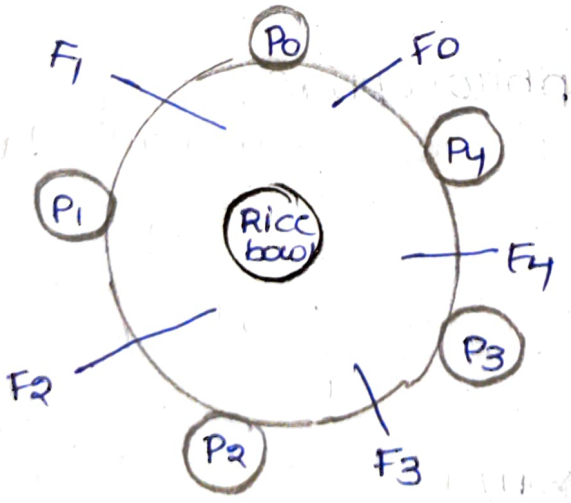
Ans: A semaphore 'S' is a synchronization tool which is an integer value that apart from initialization, is accessed only through two standard atomic operations; wait and signal. Semaphores can be used to deal with the n-process critical section problems. The classic definition of wait

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

The classic definition of 'signal'

```
Signal(S)
{
S++;
}
```

Solution for dining philosophers problem



'5' philosophers They have two states think
'5' Forks eat

Program

```
void philosopher(void)
 {
   While (true)
    {
      Thinking();
      take-fork(i);  ← left fork
      take-fork((i+1)%.N); → Right fork
      EAT;
      put fork(i);
      put fork((i+1)%. N);
      }
     }
```

case 1) Po comes

* Thinking
* take fork(0) = F0
* take fork((0+1)%. 5) = F(0+1) mod 5 = F1
* EAT
  * F0
  * F1

This will repeat for all philosophers

S(i) five semaphore

S0 S1 S2 S3 S4

1 1 1 1 1

Po comes

code: void philosophr (void)

```
      {
        while (true)
          {
```

```
Thinking();
wait (take fork (Si))
wait (take fork (S(i+1) mod N))   } Eating
    EAT();
Signal (put fork(i))
Signal (put fork ((i+1) % N))
  }
}
```

Then P0 comes oth philosopher i=0

for execute P0 we need 2 semaphores

P0 → S0 S1

P1 → S1 S2

P2 : S2 S3

P3 : S3 S4

P4 : S4 S0 → Here S0 bcoz (4+1) mod 5
5 mod 5 = 0 => S0

S0	S1	S2	S3	S4	
↓	↓	↓	↓	↓	P0 comes
✗	✗	✗	✗	1	P1 comes we can't lower value
0	0	0	0		It is blocked.

Eat P0, P2 will be there — S(1) waits

P2 comes

Here deadlock situation occurs.

Left forks we will done. one side we will execute.

Second side we can't preempt. Bcoz they are blocked.

We can change the sequence of S4 as Right, left.

In this also block.

To remove the deadlock

wait (takeFork(SH1 mod N)

wait (takeFork(Si))

This way we will solve Dining philosopher problem by using semaphores.

Assignment-2

1. What is Page Fault? Explain the steps involved in handling a page fault?

Ans: * Accessing a page marked valid bit means page hit
* Accessing a page marked invalid bit means page fault.

SO Page Fault occurs whenever we access page marked invalid bit.

The procedure for handling this page fault is straight forward.

1. We check an internal table for this process to determine whether the reference was a valid or an invalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in
3. We find a Free Frame
4. We schedule a disk operation to read the desired page into the newly allocated Frame
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it

had always been in memory.

Let us see the diagrammatic representation to handling steps in a page Fault.

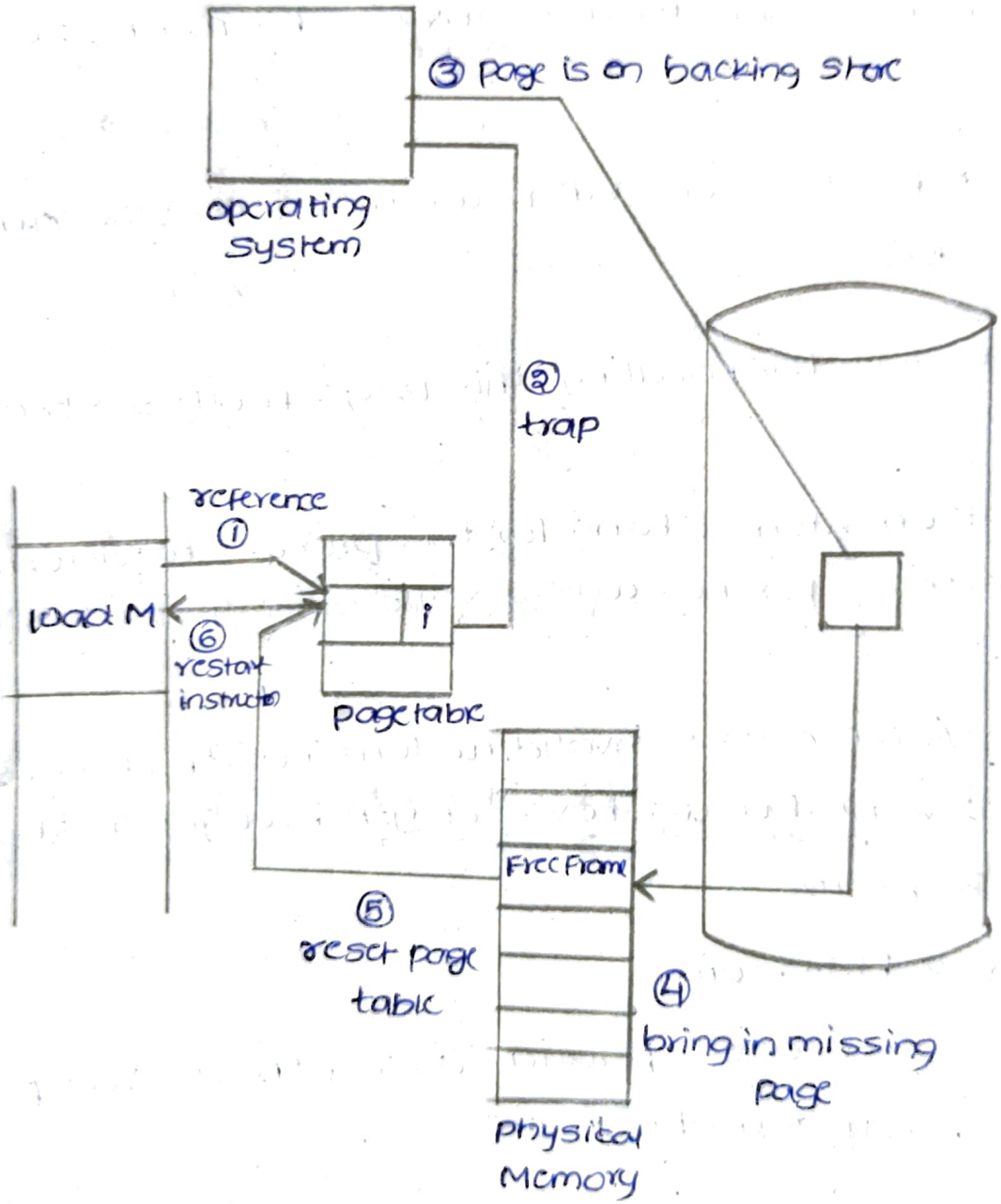


Fig: Steps in handling a Page Fault

2. Explain about page Replacements algorithms?

Ans The page replacement algorithms decides which memory Page is to be replaced. The process of replacement is sometimes called swap out or write to disk. page replacement is done when the requested page is not found in the main memory.

Virtual Memory

Frame Allocation	Page Replacement
(How many Frame are to be allocated to the process)	(Which page is to be Replaced)

There are two main stages of virtual Memory. Frame allocation and page Replacement.

Types of page Replacement Algorithms

There are various page Replacement algorithms. Each algorithm has a different method by which the page can be replaced.

1) Optimal Page Replacement
2) Least Recently used Page Replacement
3) FIFO page Replacement

Whenever there is a page fault and all Frames are Filled with pages then the requested page has to replace

the existing page.

Basic page Replacement

1. Find the location of the desired page on the disk.

2. Find a free frame

a. If there is a free frame, use it

b. If there is no free frame, use a page-replacement algorithm to select a victim frame.

c. Write the victim frame to the disk, change the page and frame tables accordingly

3. Read the desired page into the newly freed frame, change the page and frame tables.

4. Continue the user process from where the page fault occored.

FIFO page Replacement

* oldest page in main memory is the one which will be selected for environment replacement.

* Easy to implement

optimal page algorithm

* An optimal page-replacement algorithm has the lowest page fault rate of all algorithms.

LRU Algorithm

* page which has not been used for the longest time in main memory is the one which will be selected for replacement

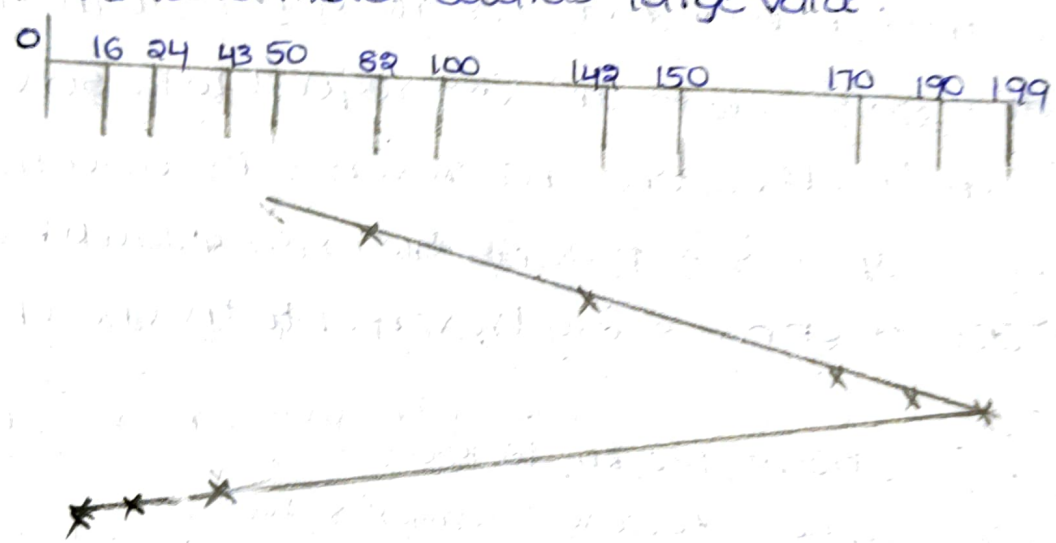
* Easy to implement, keep a list, replace pages looking back into time

3 Explain differences between SCAN, C-SCAN, LOOK and C-LOOK disk Scheduling Algorithms with Examples?

Ans SCAN : In scan algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk it reverses its direction and again services the request arriving in its path. So this algorithm works as an elevator

Ex: Requests: 82, 170, 43, 140, 24, 16, 190
Arm at position: 50

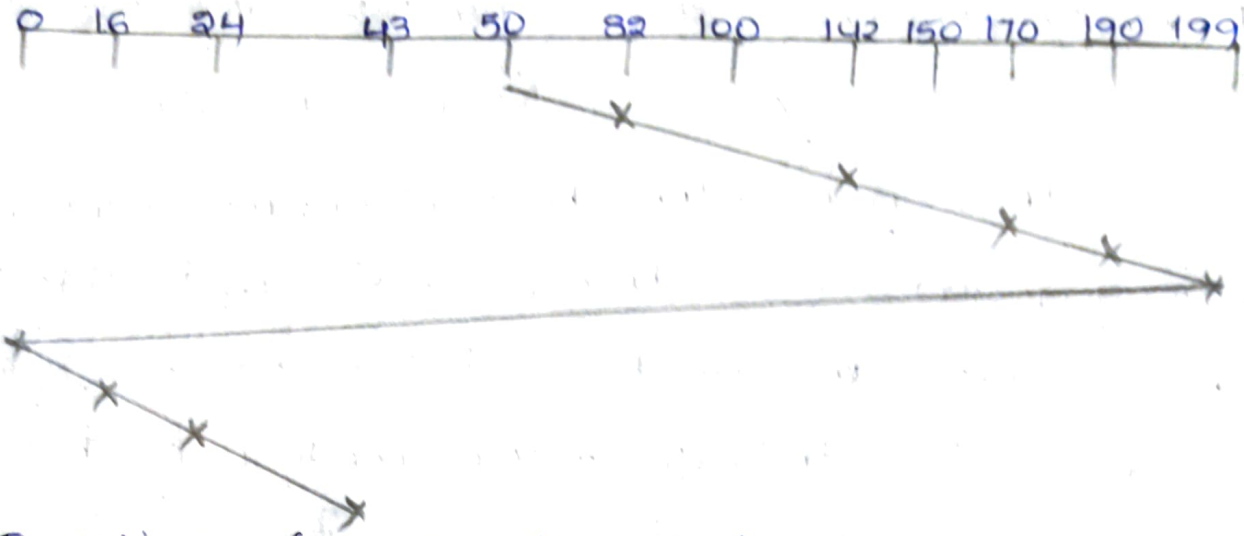
Arm should move towards large value.



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

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

Ex: Requests: 82, 170, 43, 140, 24, 16, 190
Arm is at 50th position. move towards larger value.



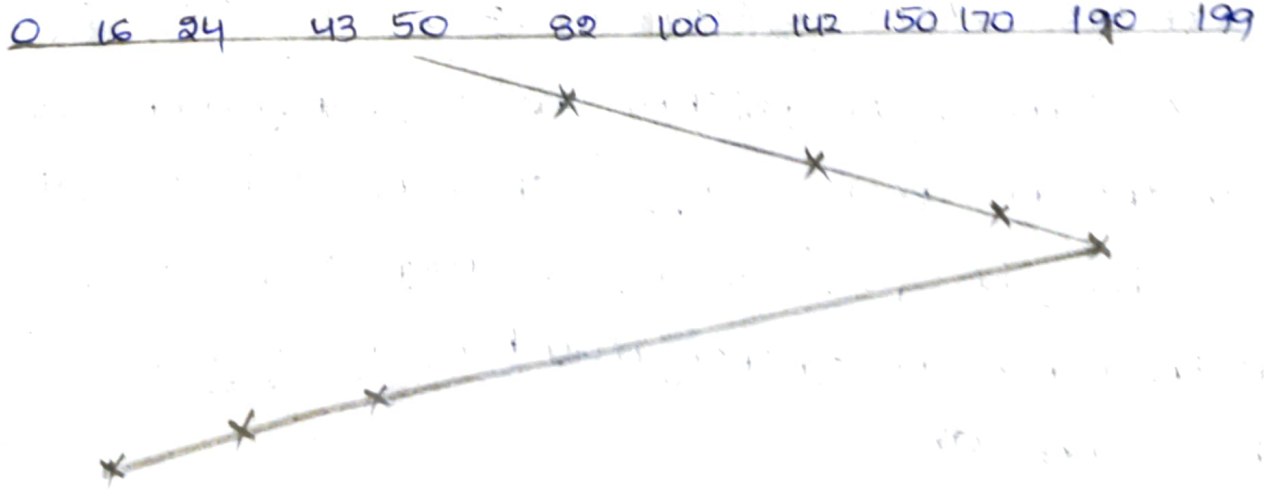
Seek time = (199 - 50) + (199 - 0) + (43 - 0)

= 391

LOOK It is similar to SCAN disk algorithm except for the difference that the disk arm inspite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occured due to unnecessary traversal to the end of the disk.

Ex: requests: 82, 170, 43, 140, 24, 16, 190

Arm position: 50 "towards larger value"



Seek time = (190 - 50) + (190 - 16)

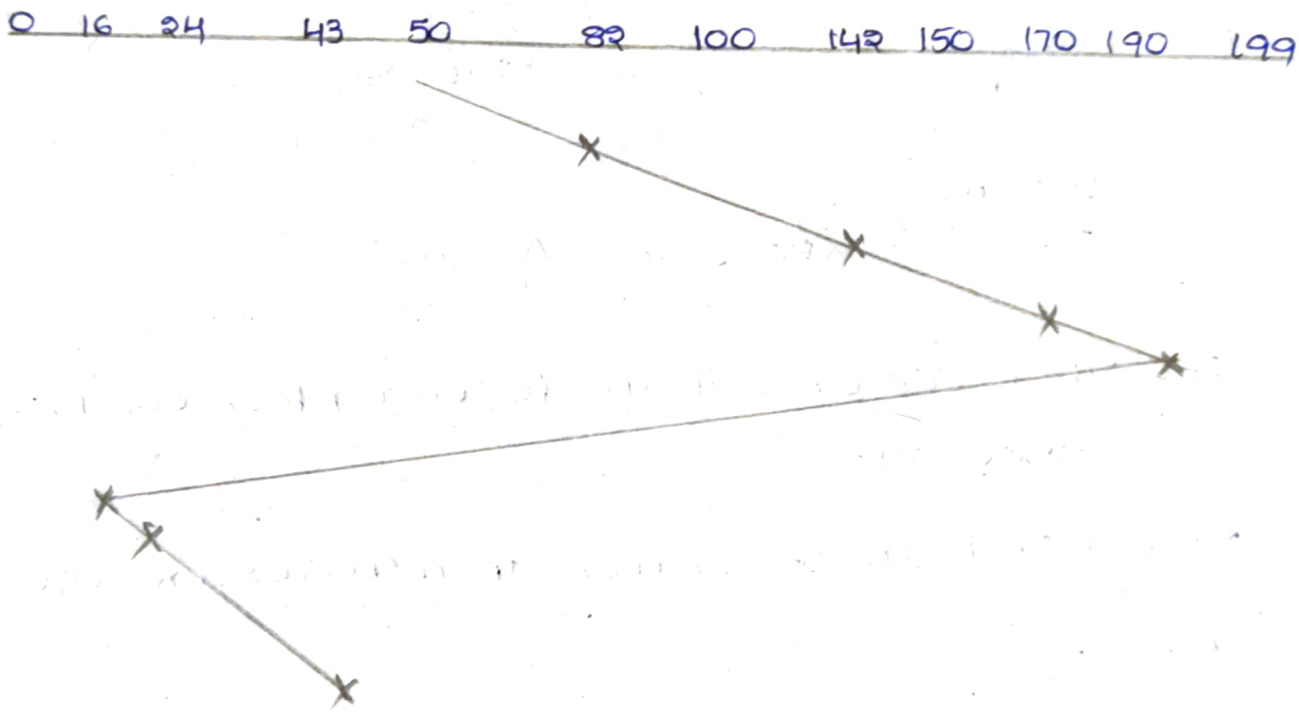
= 314

CLOOK : A Look is similar to SCAN algorithm in similar way CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last required to be serviced in front of the head and then from there goes to be the other end's last request.

Ex: Requests 82, 170, 43, 140, 24, 16, 190

Disk arm position 50

towards larger value



Seek time is calculated as

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

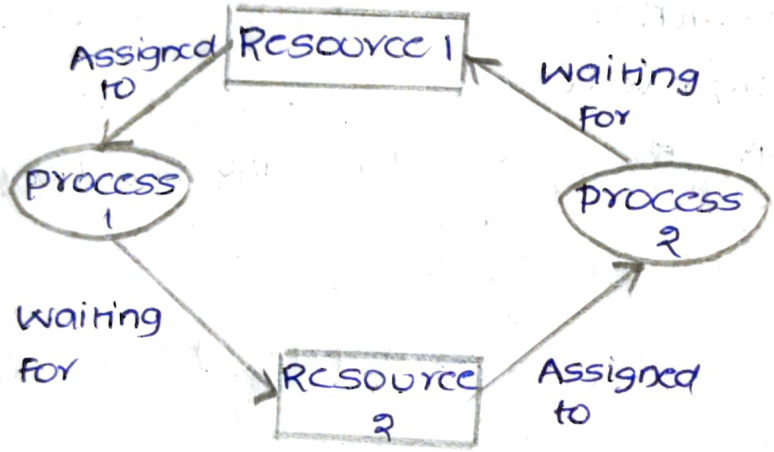
$= 341$

Therefore seek time = 341

(P.T.O)

4. What are the causes of deadlock? Discuss in detail about deadlock prevention?

Ans: Deadlock: Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process



Deadlock can arise if the following four conditions hold simultaneously.

Mutual Exclusion: one or more than one resource are non-sharable

Hold and wait: A process is holding at least one resource and waiting for resources.

No preemption: A Resource cannot be taken from a process unless the process releases the resource.

circular wait: A set of processes are waiting for each other in circular form.

Deadlock prevention

By ensuring that at least one of these conditions cannot hold. We can prevent the occurence of a deadlock.

Mutual Exclusion

The mutual Exclusion condition must hold. That is at least one resource must be non-sharable, sharable Resource in contrast does not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are a good Example of a sharable Resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource.

Hold and Wait:

To ensure that the hold and wait condition never occurs in the system, we must guarantee that whenever a process requests a resource. It does not hold another resources. One protocol that we can use require each process to request and the allocated all its resources before it begins Execution. We can implement this provision by requiring that system calls requesting resources for a process Precede all other System calls.

No preemption:

The third neccessary condition for deadlocks is that there be no preemption of resource that cannot be immediately allocated to it then all resources the process is currently

holding are preempted. The process will be restarted only which it can regain it sold resources as well as the new one that it is requesting.

Circular wait:

The Fourth and final condition for deadlocks is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol: If a process is holding some resources and requests another resource that cannot be immediately allocated to it then all resources the process is currently holding are preempted.

5. Write short notes on Access Matrix.

Ans: Access Matrix is a security model of protection state in computer system. It is represented as a matrix. Access matrix is used to define the rights of each process Executing in the domain with respect of each object. The rows of matrix represent of access rights which are given to the processes of domain means each entry (i,j) defines the set of operations that a process Executing domain D_i can invoke on object O_j.

Entries within the matrix indicate what access that domain has to that Resource

View protection as a Matrix

Rows Represents domains

Columns Represents Objects

Access(i,j) is the set of operations that a process Execution in Domain i can invoke on object

object / domain	F_1	F_2	F_3	printer
D_1	Read		Read	
D_2				print
D_3		read	Execute	
D_4	read write		read write	

6. Explain about program threats, system and network threats.

Ans: