CPU Scheduling Algorithm

- CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated first to the CPU. There are four types of CPU scheduling that exist.

FCFS Scheduling

SJF Scheduling

Priority Scheduling

Round Robin Scheduling

- These algorithms are either non-pre-emptive or pre-emptive. Non-pre-emptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time, whereas the pre-emptive scheduling is based on priority where a scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.

First Come, First Served Scheduling (FCFS) Algorithm

- This is the simplest CPU scheduling algorithm. In this scheme, the process which requests the CPU first, that is allocated to the CPU first.
- The implementation of the FCFS algorithm is easily managed with a FIFO queue.
- When a process enters the ready queue its PCB is linked onto the rear of the queue.
- The average waiting time under FCFS policy is quiet long.

FCFS Scheduling Algorithm

- Consider the following example:
- Let four processors having their CPU execution time as mentioned in the table coming into the ready queue. Now find the average waiting time and average turn around time of the processes if they enter into the queue in P1□P2□P3□P4 order.

Processes	CPU Time
P1	3
P2	5
P3	2
P4	4

Solution:

If the process arrived in the order

P1, P2, P3, P4 then according to the FCFS

the Gantt chart will be:

P1	P2	P3	P4
0 3	8		10 14

FCFS Scheduling Algorithm

The waiting time for process

P1 = 0, P2 = 3, P3 = 8, P4 = 10

then the turnaround time for process

P1 = 0 + 3 = 3, P2 = 3 + 5 = 8, P3 = 8 + 2 = 10, P4 = 10 + 4 =14.

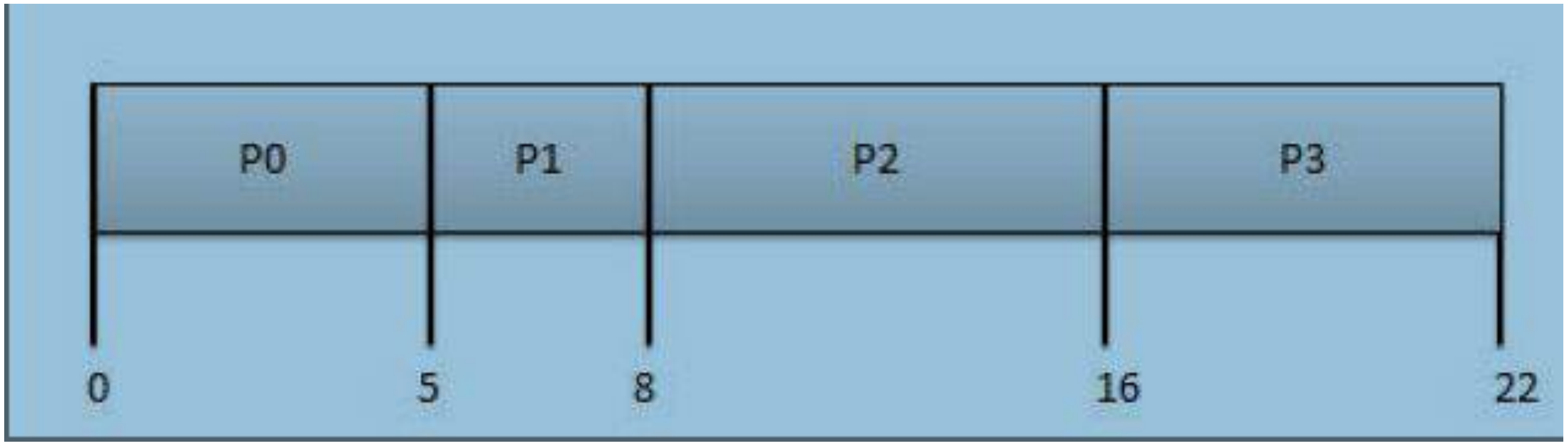
Then average waiting time = (0 + 3 + 8 + 10)/4 = 21/4 = 5.25

Average turnaround time = (3 + 8 + 10 + 14)/4 = 35/4 = 8.75

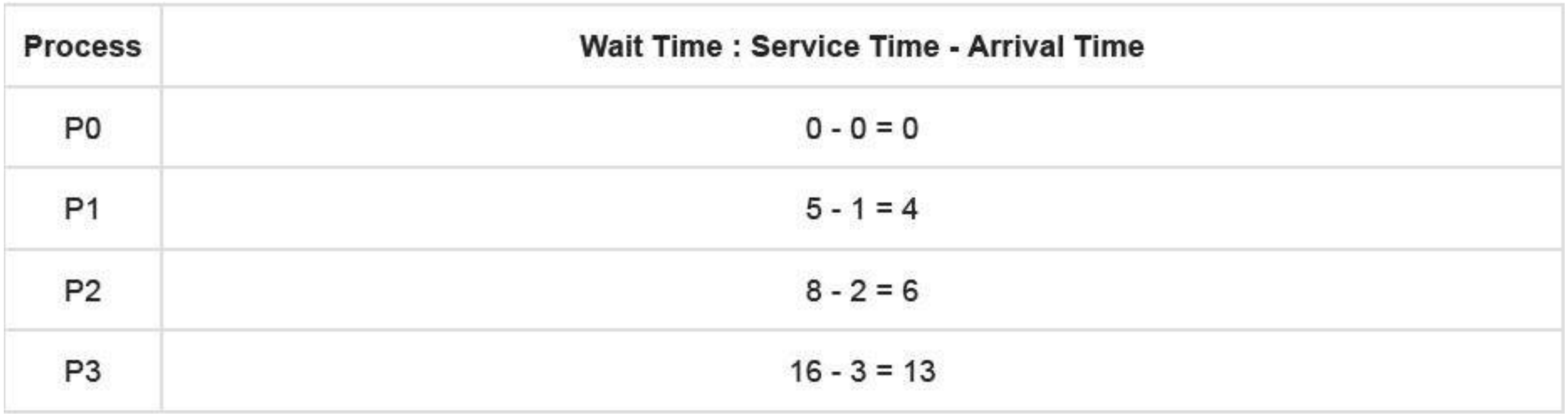
The FCFS algorithm is non preemptive means once the CPU has been allocated to a process then the process keeps the CPU until the release the CPU either by terminating or requesting I/O.

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

Example 2: The waiting time will be shown in the Gantt's chart as below



Wait time of each process is as follows −



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 = 5.75

Shortest Job First(SJF) Scheduling Algorithm

- This algorithm associates with each process if the CPU is available. This scheduling is also known as shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of the process rather than its total length
- This is a non pre-emptive / pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time a process will take.

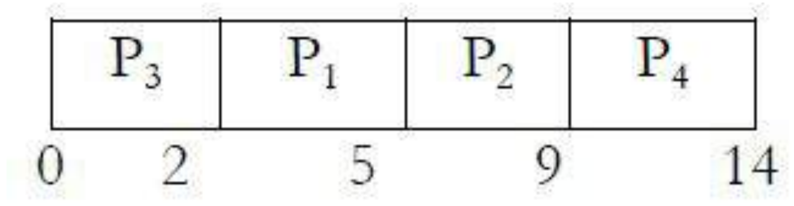
SJF Scheduling Algorithm

Example 1: Find the average waiting time & average turn around time.

Process	CPU time
P_1	3
P_2	5
P_3	2
P_4	4

Solution:

According to the SJF the Gantt chart will be



SJF Scheduling Algorithm

- The waiting time for process P3 = 0, P1 = 2, P2 = 5, P4 = 9
- The turnaround time for process P3 = 0 + 2 = 2, P1 = 2 + 3 = 5, P2 = 5 + 4 = 9, P4 = 9 + 5 =14.
- Then average waiting time = (0 + 2 + 5 + 9)/4 = 16/4 = 4
- Average turnaround time = (2 + 5 + 9 + 14)/4 = 30/4 = 7.5
- The SJF algorithm may be either preemptive or non preemptive algorithm.

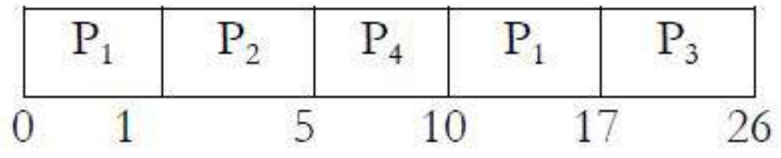
SJF Scheduling Algorithm

- The preemptive SJF is also known as shortest remaining time first.

Example 2:consider the following processes having CPU burst time are arriving into the ready queue as shown in the table. Find the average waiting time.

Process	Arrival Time	CPU time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

In this case the Gantt chart will be



The waiting time for process

P1 = 10 - 1 = 9

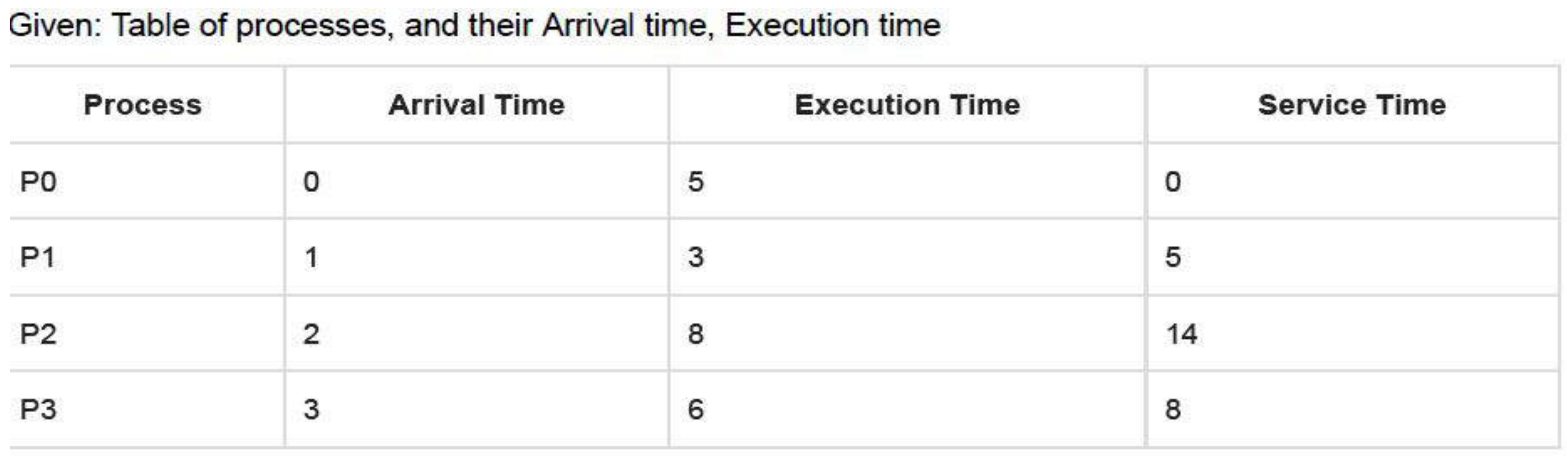
P2 = 1 – 1 = 0

P3 = 17 – 2 = 15

P4 = 5 – 3 = 2

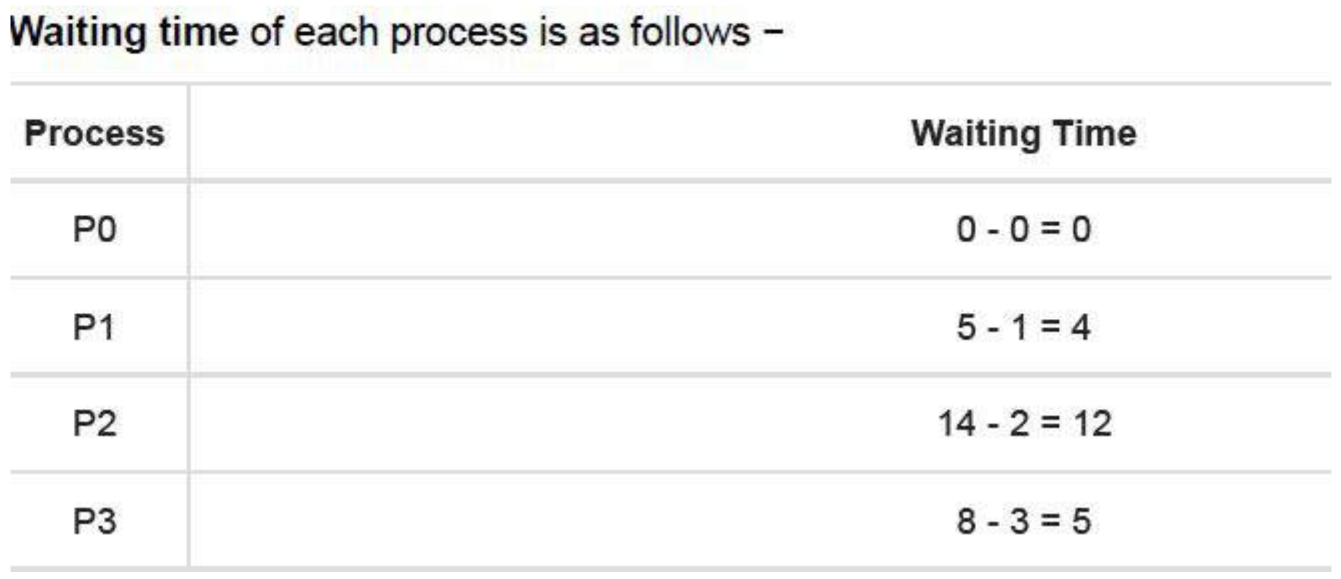
The average waiting time = (9 + 0 + 15 + 2)/4 = 26/4 = 6.5

Example: 3 Find Average waiting time.



Given: Table of processes, and their Arrival time, Execution time

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


Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5..25

Priority Scheduling Algorithm

- In this scheduling a priority is associated with each process and the CPU is allocated to the process with the highest priority.
- Equal priority processes are scheduled in FCFS manner.
- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- 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 served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

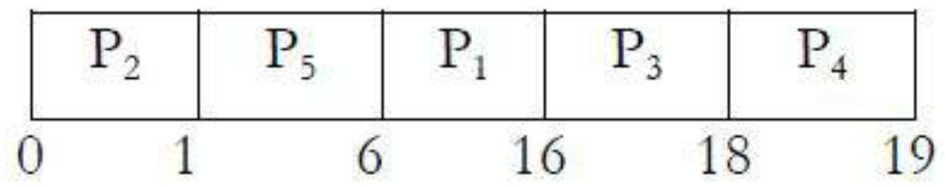
Examples:

- Find the waiting time of each process and average waiting time.(Least arrival time is highest priority).

Process	CPU time	Arrival time
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

Sol:

According to the priority scheduling the Gantt chart will be



The waiting time for process

P1 = 6

P2 = 0

P3 = 16

P4 = 18

P4 = 1

The average waiting time

= (0 + 1 + 6 + 16 + 18)/5 = 41/5 = 8.2

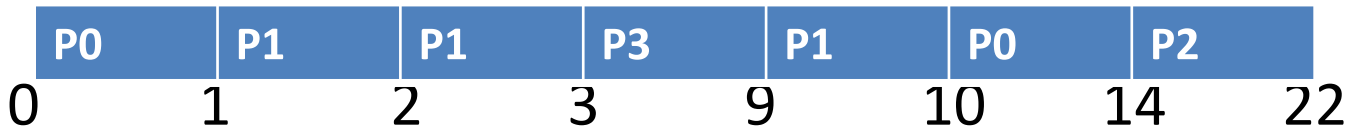
Examples:

- Draw the Gantt chart and find the waiting time of each process and average waiting time

Process	CPU time	Arrival time	Priority
P0	5	0	1(Lowest)
P1	3	1	2
P2	8	2	1
P3	6	3	3(highest)

Sol:

According to the priority scheduling the Gantt chart will be



The waiting time for process

P0 = 10-0-1=9

P1=9-1-2=6

P2 = 14-2-0=12

P3 = 3-3-0=0

The average waiting time

= (9 + 6 + 12 + 0)/4 = 27/4 = 6.75

Round Rabin Scheduling Algorithm

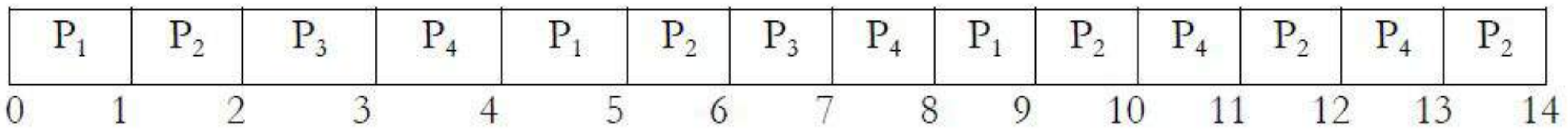
- This type of algorithm is designed only for the time sharing system. It is similar to FCFS scheduling with preemption condition to switch between processes.
- A small unit of time called quantum time or time slice is used to switch between the processes.
- Context switching is used to save states of pre-empted processes.
- Once a process is executed for a given time period, it is pre-empted and other process executes for a given time period.
- The average waiting time under the round robin policy is quiet long.

Example:

Draw the Gantt chart and find the waiting time of each process and also the average waiting time(Tme Slice=1 Milli Sec.).

Process	CPU time
P1	3
P2	5
P3	2
P4	4

Sol: The Gantt chart is given below



The waiting time for process

P1 = 0 + (4 – 1) + (8 – 5) = 0 + 3 + 3 = 6

P2 = 1 + (5 – 2) + (9 – 6) + (11 – 10) + (13 – 12) = 1 + 3 + 3 + 1 + 1 = 9

P3 = 2 + (6 – 3) = 2 + 3 = 5

P4 = 3 + (7 – 4) + (10 – 8) + (12 – 11) = 3 + 3 + 2 + 1 = 9

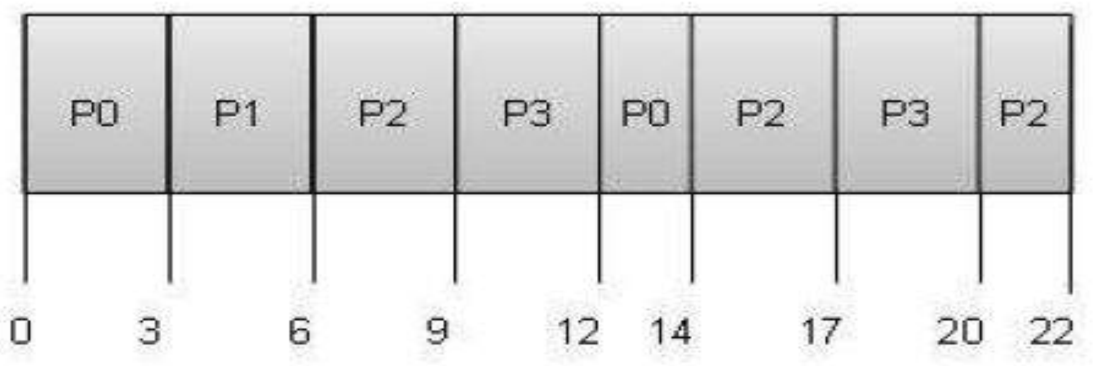
The average waiting time = (6 + 9 + 5 + 9)/4 =29/4= 7.25

Example:

Draw the Gantt chart and find the waiting time of each process and also the average waiting time(Tme Slice=3 Milli Sec.).

Process	CPU time	Arrival Time
P0	5	0
P1	3	1
P2	8	2
P3	6	3

Sol: The Gantt chart is given below



The waiting time for process

P0=(0 - 0) + (12 - 3) = 9

P1 = (3 - 1) = 2

P2 = (6 - 2) + (14 - 9) + (20 - 17) = 12

P3 =(9 - 3) + (17 - 12) = 11

The average waiting time = (9 + 2+ 12 + 11)/4 =34/4= 8.5

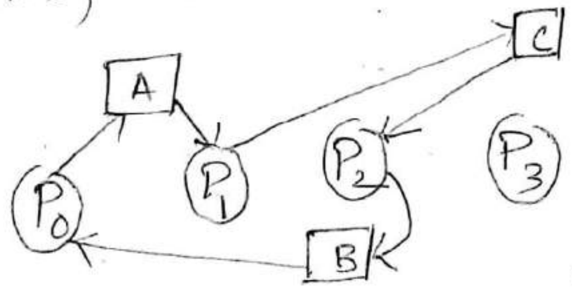
LECTURE NOTE ON DEADLOCK

1- Concept of deadlock.
2- System Model.
3- Deadlock detection.
4- Resource Allocation Graph.
5- Methods of Deadlock handling.
6- Recovery & Prevention
7- Banker's Algo^m and Safety Algo^m.

Deadlock

A deadlock is a situation where each process of a group is waiting for resources which is held by another process in the same group.

For Example.
Let ~~b~~ P_0, P_1 P_2 P_3 are four processes in a group G and they are competing for getting resources like A, B, C. Then the situation like.



In this situation P_0 is holding B and waiting for A which is held by P_1 and P_1 is waiting for C which is held by P_2 and P_2 is waiting for B. In this situation, the processes are on deadlock.

Necessary condition:-
A system is in deadlock of following Conditions hold simultaneously.

1- Mutual Exclusion:- At least one resource must be held in a non-shwable mode.

2- Hold and wait:- A process is currently holding at least one resource and requesting additional resource which are being held by other ~~[illegible]~~ process.

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

4. Circular wait:- Each 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.

Handling Deadlock:-

Deadlock can be handled by

1- Ignoring.
2- Deadlock prevention.
3- Deadlock Avoidance.
4- Deadlock detection and recovery.

1) Ignoring :- In this approach the system will assume that the deadlock will never occure. ~~So the~~ The operating system will ignore the deadlock if occure. ~~So all the processes are in deadlock~~ Initially a deadlock occure with small no. of processes. Gradually, all the processes fall in the deadlock and the system will not proceed further. At that moment just restart the system.

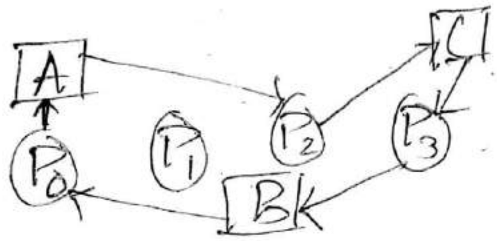
2) Deadlock detection and Recovery :-
In this approach, deadlock are allowed to occure. Then the o/s will examined the state, to detect that a deadlock has occured by using Resource Allocation graph algorithm. ~~The~~ Once detected, the operating system runs some Recovery procedure to remove the detected deadlock.

Resource-Allocation graph :- Here the resource scheduler has all knowledge about which resources are allocated ~~to~~ to or requested ~~to~~ by whom. By observing the distribution of resources it plots a graph known as resource-Allocation graph. If ~~a~~ the o/s found a closed graph. then a dead lock ~~will be~~ is path detected where the processes lies in the closed path will ~~are~~ be in deadlock.

7/17

Ex :- Resources are represented by □
processes are represended by O
request is represented by a ⟶
from process to resource.
Allocation is represented by a ⟵
from resource to process.

let A B C are 3 resorces and
~~to~~ P_0, P_1 P_2 P_3 are 4 ~~resorces~~ processes.
The Graph ~~is~~ of the allocation ~~at~~ at some time
is



Here a closed path occure with sequence.

$P_0 \longrightarrow P_2 \rightarrow P_3 \longrightarrow P_0$

so It is a deadlock.

Recovery :- once a deadlock is detected, it can be corrected by using one of the following Methods.

i) Process Termination :- In this approach one or more processes involved in the deadlock may be aborted.
The o/s will abort all the processes involved in the deadlock. But this will incure costly because all the partially processed data may be lost.

Another approach is the o/s could choose to abort one process at a time until the deadlock is resolved.

ii) Resource preemption :- Here resources allocated to various processes may be successively preempted and allocated to other process until the deadlock is broaken.

3) Deadlock Prevention :-

This approach works by preventing one of following four condition from occurring.

i) Mutual Enclusion.

ii) hold and wait.

iii) No preemption.

iv) Circular wait.

i) Mutual Enclusion :- preventing Mutual Enclusion means no process will have exclusive access to a resource. But this is impossible because some resources are non-preempting or not - shareble.

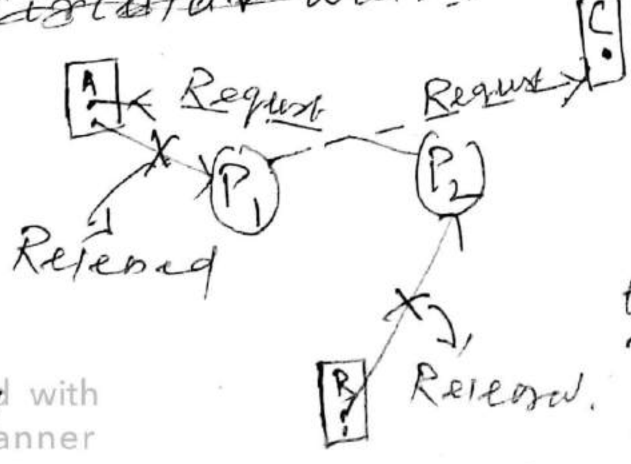
ii) No- preemption :- preventing this condition is also dibbicult or impossible since some resources are not sharable like printer. If the o/s forcefully preempted the process holding a non-preempting resource then the non-preemting resource will roll-back and lost all particelly processed data.

iii) hold & wait :- This condition can be removed by allocating all the resources. to a process request for well in advance. But predicting the resources requested by a process well in advance is pretty difficult and it allocated, then the resources are utilized ~~ineffici~~ in efficiently.

Another approach is that the process can request resources only when it has none. i.e before requesting for resources, it must release ~~the~~ all currently held resources.
This approach may be fine but not efficient since allocated resources may be unused for a long time. Another difficulty is, the processes requesting for popular resources must wait for a long time which is known as resource starvation.

~~iv) Circular Wait:-~~

eg.



here P_1 and P_2 holding an instance of A and B respectively. If P_1 ~~...~~ and P_2 request for resources C and A respectively, then P_1 & P_2 should release instance of A and B respectively.

10/17

(v) Circular wait :- In this approach, the O/S ~~will~~ disable interrupts during circuital sections and using a hirarchy to determine a partial ordering of resources. If no obvious hirarchy ~~exist~~ exists, even ~~though~~ the memory address of resources ~~are~~ has been used to determine ordering and resources are requested in increasing order of the enumeration.

Ex :- Each resource has certain number associated with it.

Let A B C D are 4 resources having number.

$A \rightarrow 4$

$B \rightarrow 2$

$C \rightarrow 9$

$D \rightarrow 12$

Then if P_0 be a process holding B then it can request for A, C, D (increasing order) But if P_0 is holding C(9) then it can request for D(12) only not for A(4), B(2). If P_0 is requesting for A(4) then it must release all the resource having number greater then A(4).

4) Deadlock Avoidance :- Here the operating system has the knowledge of total no of available resources, ~~Allocated resources~~, Maximum resources each process can request, the total number of resources allocated to each process and also remaing resourses may need ~~[illegible]~~ by a process in future. By having this knowledge the O/S can ~~allocate or deallocate~~ accept or ~~[illegible]~~ reject request ~~[illegible]~~ of process to avoid deadlock. The operating system always seen in a safe state.

The system maintains two vectors. Work and Finsh whose lengths are. m and respectivily.
where work is the n vector of number of Available ~~[illegible]~~ instances of each resoure type. and
Finesh is the vector which containes. number of resowrs need by each processes.
Deadlock Avoidance can be done through Banker's Algorithm.

Banker's Algorithm

1) Let work and Finish be ~~the~~ two vectors of size m and n, respectivily. Initialize the work with available resources and Finish[i] is false for i = 1 to n.

2) Find an i such that both
   a) Finish[i] = false.
   b) Need$_i$ ≤ work

If no such i exist then goto step 4.

3) work = work + Allocation$_i$
   Finish[i] = True.
   goto step 2.

4) If Finish[i] = True for all i then
   the system is in safe state.

<u>Ex</u> :-
Let we have 5 processes P_0 to P_4 and 4 resources A, B, C, D. The processes ~~P~~ can request for resours as per the ~~following~~ processing progress.
~~A P~~ Maximum number of resources each process can request is represented by

MAX =	A	B	C	D
P_0	0	0	1	2
P_1	1	7	5	0
P_2	2	3	5	6
P_3	0	6	5	2
P_4	0	6	5	6

At Any Moment the number of each resources allocated to each process can be represente

as

	A	B	C	D
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

= ~~Avail~~
= Allocation.

And the Avaliable resources at that moment can be represented as.

A	B	C	D	
1	5	2	0	= Available.

i) Calculate the Matrix Need.

ii) Is the System in Safe state.

iii) if P_1 arrives with request (0,3,2,0) can it be granted immidiately.

<u>Ans !-</u>

Here n = # processes = 5 = P_0 to P_4
m = # resources = 4 = A, B, C, D.

Available[M] = Size = 4

Allocation[$n \times m$] = 5×4 = 20 Size.

Max[$n \times m$] = 5×4 = 20 size.

~~Now~~ Need[$n \times m$] = 5×4 = 20 size
= Max − Allocation

~~A~~ Need

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

ii) For Po

since Po has no need resources so Po can be allowed to execute. After execution it's resources are released.

Work	Finish
A 1+0 = 1	0 — ~~F~~ → T
B 5+0 = 5	1 — F
C 2+1 = 3	2 — F
D 0+2 = 2	3 — F
	4-F

For P_1

since P_1 (0, 7, 5, 0) ~~does~~ need 7 & 5 instances of B and C respectively & we have only 5 & 3 instances of B & C so it's request can't be made.

Work	Finish
A — 1	0 — T
B — 5	1 — F
C — 3	2 — F
D — 2	3 — F
	4 — F

For P_2

It's need is (1, 0, 0, 2) since we have 1 & 2 resources of A & D are availiable so request of P_2 is granted. After execution it's resources are released.

Work	Finish
A – 1 + 1 = 2	0 – T
B – 5 + 3 = 8	1 – F
C – 3 + 5 = 8	2 – T
D – 2 + 4 = 6	3 – F
	4 – F

For P_3

It need (0, 0, 2, 0), Sine we have 8 no. of instances of C so it's request is granted. After execution it's resources are released.

Work	Finish
A — 2 + 0 = 2	0 – T
B — 8 + 6 = 14	1 – F
C — 8 + 3 = 11	2 – T
D — 6 + 2 = 8	3 – T
	4 – F

For P_4.

It needs (0, 6, 4, 2) which can be made since we have (2, 14, 11, 8) resources. So After execution the resources are released.

Work	Finish
A – 2 + 0 = 2	0 – T
B – 14 + 0 = 14	1 – F
C – 11 + 1 = 12	2 – T
D – 8 + 4 = 12	3 – T
	4 – T

Now for P_1

If needs (0 7, 5, 0) which can be made since we have (2, 14, 12, 12) resources.

Work	Finish
A - 2 + 1 = 3	0 - T
B - 14 + 0 = 14	1 - T
C - 12 + 0 = 12	2 - T
D - 12 + 0 = 12	3 - T
	4 - T

[All Finish$_i$ is true]

So the sequence of execution is $\langle P_0, P_2, P_3, P_4, P_1 \rangle$ is in safe state.

— x —

DEVICE MANAGEMENT

1. Techniques for Device Management
    a. Dedicated
    b. Shared
    c. Virtual
2. Device allocation
    a. considerations I/O traffic control & I/O Schedule
    b. I/O Device handlers.
3. SPOOLING.

---

1. **Device management:** Device management is responsible for managing all the hardware devices of the computer system. It includes the management of the storage device as well as the management of all the input and output devices of the computer system. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

There are three basic techniques for implementing a device for policy.

a) Dedicated devices: These are devices that are assigned to one process at a time and the process only releases the device once it is completed. This includes devices like plotters and tape drives. The problem with this is that one user is using at a time and it might be inefficient if the device is not being used 100% of time that it is being locked by the user.

b) Shared devices: these are devices that can be shared by many processes. This includes devices like hard disk which is interleaving between different process requests. One difficulty is that all conflict for a device need to be resolved and pre-determined policies to determine which request is made first.

c) Virtual devices : These devices are combination of dedicated and shared devices. Its Combination of dedicated devices that have been transformed into shared devices. The device like printer is a dedicated device but when used spooling technique is used then it can transferred into shared device.

2. **Spooling:** The Spooling (Simultaneous peripheral output online) is a process in which data is temporarily held to be used and executed by a temporary buffer on the system. Data is sent to and stored in the memory or other volatile memory until the program or computer requests it for execution. Spooling" is the computing term for a method of copying data from one device to another when the speed of one device is considerably greater than another.

    A common usage is "print spooling". A program sending data to a printer is usually capable of sending that data far faster than the printer can actually print it out. Thus, instead of requiring the program that needs something printed to wait for the printer to finish printing it, the print data is "spooled": it is sent to a program whose job it is to interact with the printer. The spooling program confirms to the original program that it has received the data and queued it to print. If something happens so that the printing can't be completed, it is now the job of the print spooler to notify someone, and to make sure the job is not deleted until it has successfully printed.

3. **Device allocation:**

19-12-09
Saturday

I/o system :-

Two main jobs of a computer are I/o operation and processing. The role of o.s in computer I/o is to manage and control I/o operations and I/o devices. There are different types of I/o devices with their functionality and speed, so variety of methods are needed to control them.

I/o Hardware :-

A variety of I/o devices that can be used with a computer, but the basic hardware elements involve in I/o are buses, device controller and the devices.

Buses are used widely in computer architecture. It connects the processor-memory subsystem to the fast devices and also it connects slow devices such as keyboard and serial and parallel ports.

A device controller is a collection of electronics that can operate a port, a bus and a device.

A device communicates with a computer system by sending signals through a cable. The device communicates with the machine through a connection point, that is a serial port. If one or more devices use a common set of wires, then the connection is called a bus. ~~Th~~

The processor communicates with a device controller by reading and writing bit patterns in the register of cpu.

An I/O port consists of 4 registers i.e

1>. Status register.
2>. control register.
3>. Data-in register.
4>. Data-out register.

1>. Status register→

The status register contains bits that indicates the current states such as whether the current command has completed, whether a byte is available to be read, etc.

2> Control register→

A control register contains certain bits of a serial port for performing the data transmissing operation, parity checking and half duplex and fullduplex transmission.

3>. Data-IN register→

The data-in register is used to give input data to the system.

4>. Data-OUT Register→

The data-out register is used to send output data to the display devices.

Buffer.

It is a region of physical memory (usually in storage) used to temporarily hold data while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an I/p device or just before it is sent to an O/p device.
e.g. print command.

Bounded-buffer (or Producer Consumer) Problem

~~Th~~ This problem is a classical example of multiprocess synchronization problem. This problem describes two process producer & 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 onto the buffer & start again. At the same time the consumer is consuming the data (i.e. removing it from the buffer) one piece at a time. This problem is to make sure that the producer won't try to add data onto the buffer if it is full and the consumer won't try to remove the data if it is empty,

The soln for the producer is to either go sleep or discard the 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 sleep if it finds the buffer is be empty. The next time the producer puts the data into the buffer, it it wakes up the sleeping consumer. This soln can be reached by IPC typically using semaphores.

Process Synchronization.

```
# buffer_size 5
typedef struct
  {
    ...
  } item;
item  buffer [buffer_size];
int in=0;  out=0;  count=0;
```

Bounded buffer:

Producer:

```
while (true)
{
  /* produce an item and
  put in nextProduced */
while (count == buffer_size);
    do nothing;
buffer [in] = next Produced;
  in = (in+1) % buffer_size;
  count++;
}
```

consumer:

```
while (true)
{ while (count == 0);
    // do nothing
  nextconsumed = buffer[out]
  out = (out+1) % buffer_size
  count --;
  /* consume the item
  in nextconsumed */
}
```

-: Disk scheduling Algorithm:-

When a process wants to do disk I/o, an O.S call is made. This may takes some time. The process is put in a block state and the I/O request is sent to the device driver. If the disk is idle, then the operation is started. If the disk is busy, then another request is added to the queue.

There are no. of disk scheduling algorithm techniques are available, i.e →

1> FCFS scheduling. (First come First serve)
2>. SSTF scheduling (shortest seek Time First).
3>. Scan scheduling.
4>. C-scan scheduling. (circular-scan scheduling)
5>. Look scheduling.

(If the disk is busy, then we go for disk scheduling techniques).

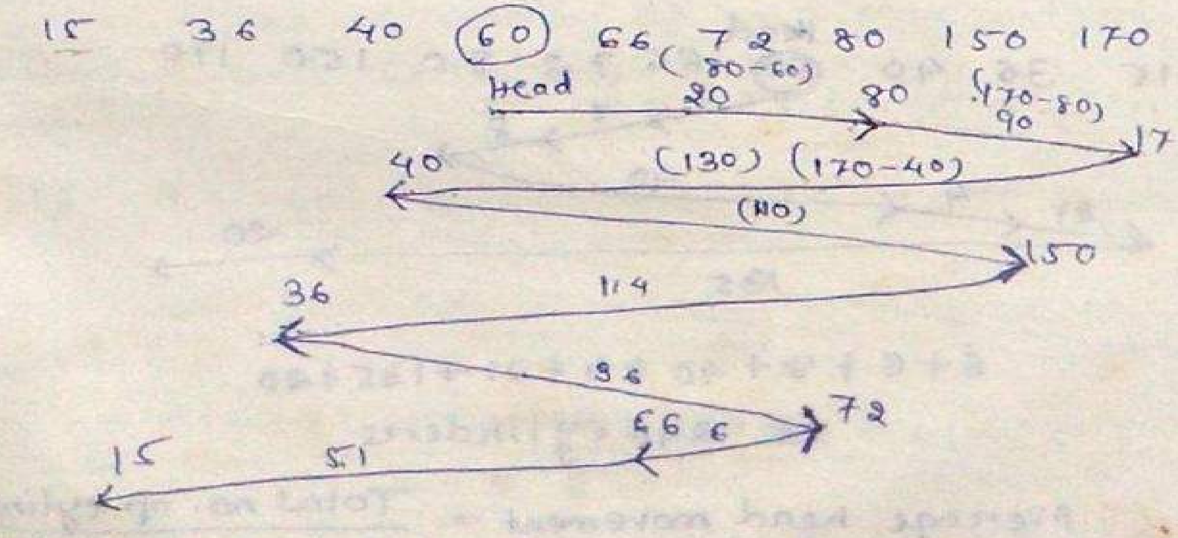
6> C-Look scheduling.

-12-09 Saturday

① FCFS Disk scheduling Algorithm:-

Queue→[80, 170, 40, 150, 36, 72, 66, 15]

Initially head is at cylinder 60.



20+90+130+110+114+36+6+51 = 557 cylinders

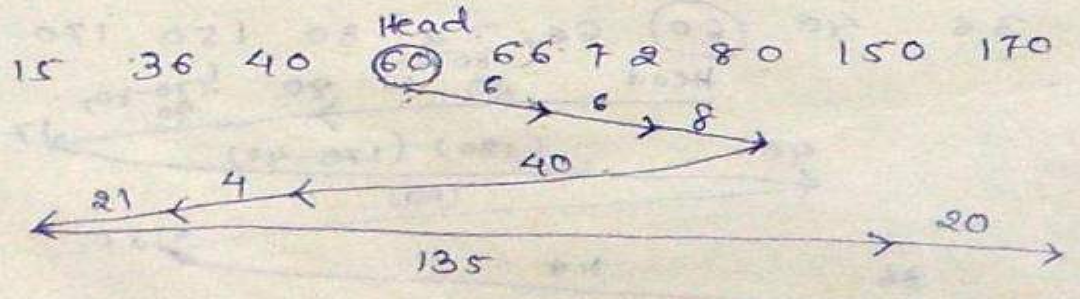
$= \frac{557}{8} = 69.625$
is the ~~be~~ average head movement.

Average head movement = $\frac{\text{Total no. of cylinders}}{\text{Total no. of head movements}}$

It is the simplest disk scheduling algorithm but it doesn't provide fastest service for the process. In this case, the head of the disk move from one cylinder position to another cylinder position to perform the I/O operation for the processes at the time of execution.

This algorithm is very simple to implement but the performance is not satisfactory because the average head movements are very high.

② Shortest Seek Time First Scheduling (SSTF)



6+6+8+40+4+21+135+20
= 240 cylinders

Average head movement = $\frac{\text{Total no. of cylinders}}{\text{Total no. of head movements}}$

$= \frac{\overset{30}{\cancel{240}}}{8} = 30$

~~If~~ same distance from a particular head occurs, then we go towards that direction towards which the head is placed previously.

* If the initial distances are same from the current head posⁿ, then we use FCFS scheduling to move towards another cylinder.

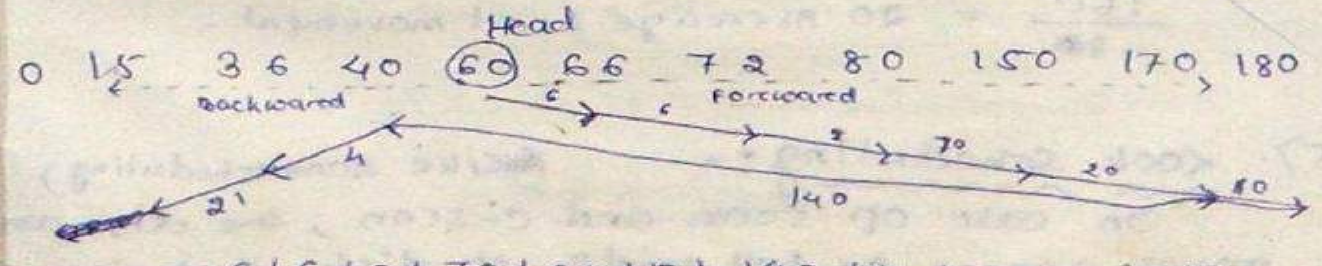
This algorithm selects the request with minimum seek time from the current head position. Here, in this case, to calculate the distance from one head posⁿ to its left and right cylinder posⁿ. The minimal distance cylinder will be selected to move the head because minimal distance having minimal seek time.

Comparing the algorithm with FCFS, it gives an improvement performance.

③ Scan Scheduling:-

In this case, the disk arm starts at one end of the disk and move towards the other end, while in the mean time, all requests are servicing until it gets the other end of the disk. At the other end, the direction of head movement is reversed and servicing continuously.

If t 180 → Range is given



$= \frac{6+6+8+70+20+10+140+4+21}{8} = \frac{285}{8}$

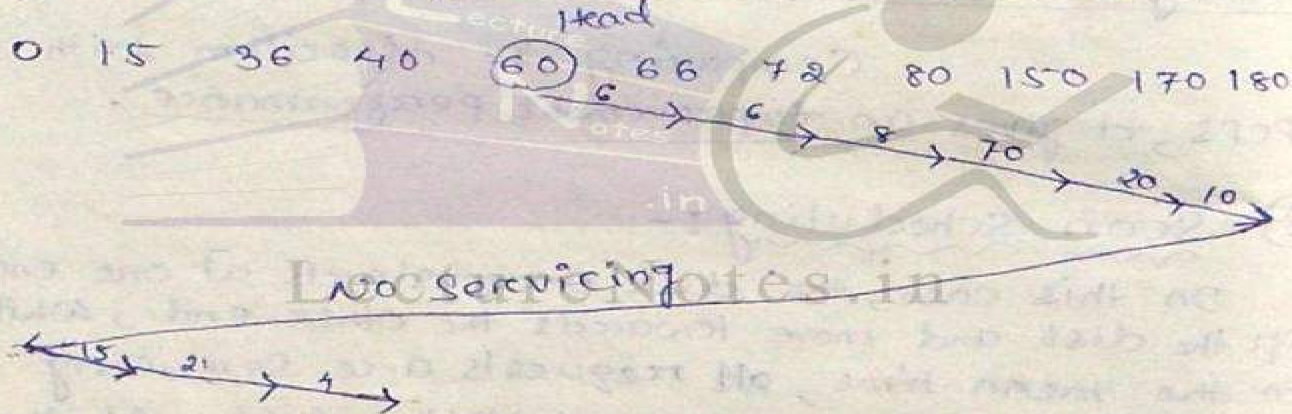
$= 35.625$ head movements

✓* Scan scheduling is better than FCFS but not than SSTF.

4) C-Scan Scheduling :-

The main drawback of scan scheduling is the waiting time. Some requests are waiting and some requests are servicing immediately. We can overcome this drawback with circular-scan scheduling algorithm. It is designed to provide a more uniform wait time. In this algorithm, moves the head from one end to the other end of the disk, servicing the request along the way. When the head reaches the other end, it immediately returns to the begining of the disk without servicing any request on the return trip.

✓* Always in forward direction movement.

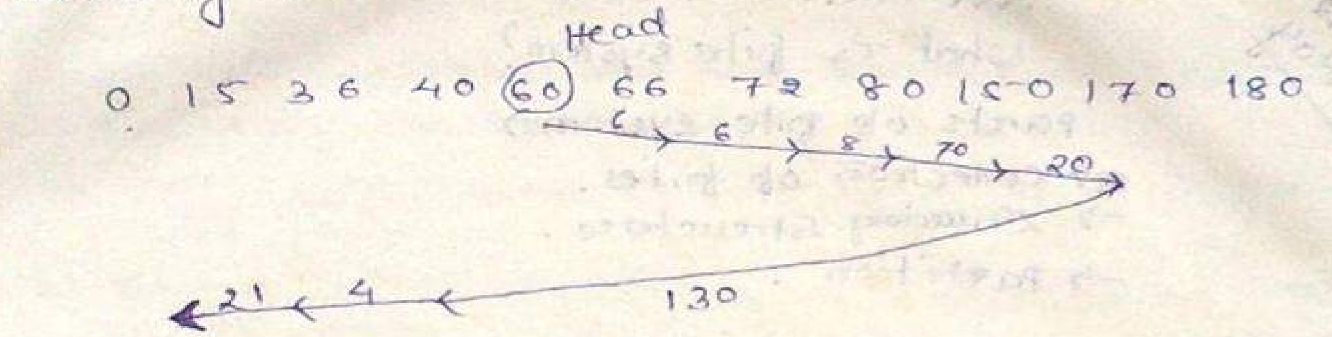


$6+6+8+70+20+10+15+21+4 = 160$ cylinders

$\frac{160}{8} = 20$ Average head movement.

5) Look Scheduling :- (Like scan scheduling)

In case of scan and c-scan, the disk arm moves across the full width of disk. But in case of look scheduling, the arm goes only as per as the final request in each direction. Then it reverses the direction immediatly, without reaching the end of the disk.



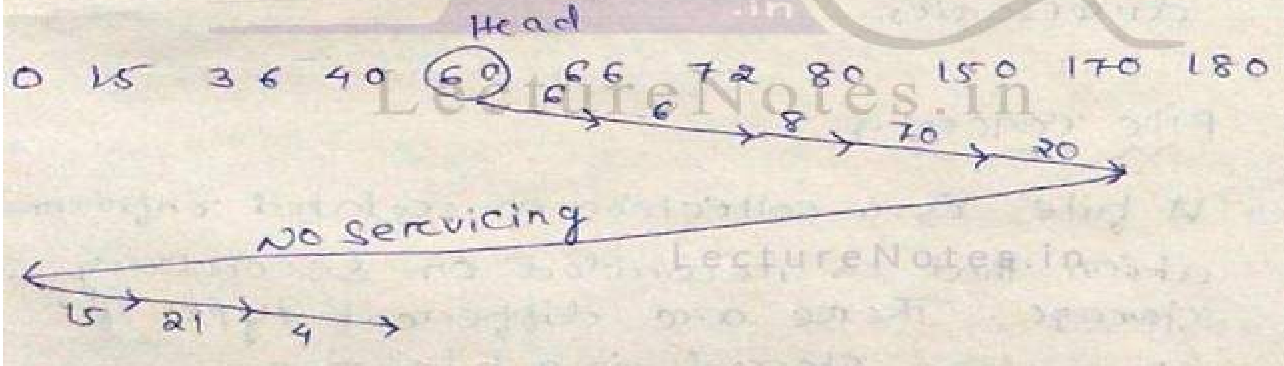
6+6+8+70+20+130+4+21 = 265 cylinders

$\frac{265}{8}$ = 33.125 average head movement.

(Both direcc^n can be used)

67. C-Look Scheduling :-

It is a variation of Look scheduling, where requests are satisfied only when the head moves outwards as in C-scan. No request is satisfied while the head moves inwards after determining that no requests are there, beyond the current point.



6+6+8+70+20+15+21+4 = 150 Cylinders

$\frac{150}{8}$ = 18.75 Head movement.

~~(Both direcc^n can be used)~~

-12-09
Monday

-: File system interface :-

What is file system?

Parts of file system →
→ collection of files.
→ Directory structure.
→ partition.

File system is the most visible aspect of an O.S. It provides the mechanism for storage and access of both data and programs of the O.S and all the users of the computer system. The file system consists of different parts i.e –

(i). A collection of file, each storing related data.
(ii). A directory structure, which organizes and provides informn about all the files in the system.
(iii) partition, which are used to separate physically or logically large collection of directories.

File concept →

A file is a collection of related information that is recorded on secondary storage. There are different types of information stored in a file i.e

→ source program.
→ object program.
→ Executable.
→ Numeric data.
→ Text.
→ Graphics images
→ sound Recording etc...

A read operation reads the records from the file and automatically advances the file quantum. Similarly, a write operation appends to the end of the file and advances to the end of the ~~the~~ newly written material. Such a file can be reset to the beginning and a program may be able to skip forward or backward inside the file.
eg→ Magnetic tape.

2). Direct Access Method→

Another method is direct access or relative access. A file is made up of fixed length logical records that allows programs to read & write records rapidly in no particular order. It is based on the disk model of file, since disks allows random access to any file block. For direct access, the file is viewed as a numbered sequence of blocks or records. A direct access file allows blocks to be read or written. Thus, we may read block no. 14, then block no. 53 and then write in block no 7. There are no restrictions on the order of reading or writing for a direct access file.

Direct access files are of great use or immediate access, for large amounts of information.

3). Other Access Method / Index Direct Access Method→

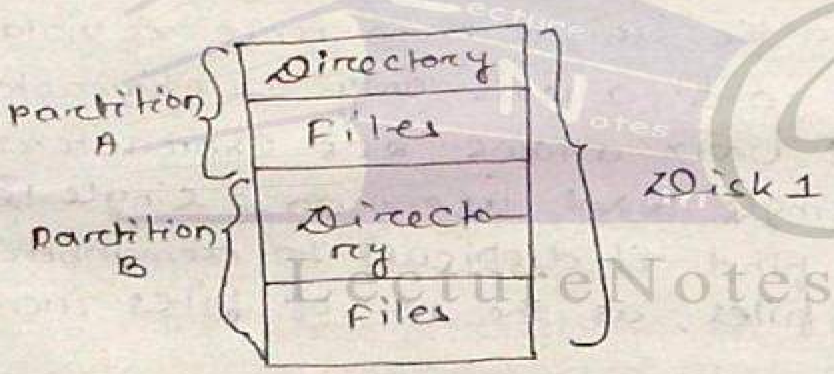
Other access methods can be built on the top of the direct access method. These methods involves the construction of an index for the file. The index contains pointers to the various blocks. To find a record in a file, first search the index and then use the pointer to access the file directly and then find the desired record.

Directory structure :-

Computer systems stores millions of files on the disk. To manage all these files, we need to organize them. This orgⁿ is done in 2 parts →

1) Disk are split into one ore more partitions. Each disk on a system contains at least one partition, where all the files and directories are reside. Sometimes partitions are used to provide several separate areas within one disk and each treated as a separate storage device.

2) Each partition contains informⁿ about files within it. This informⁿ is kept in a device directory. The device directory records informⁿ such as name of file, location of file, size of file and type of file on that partition.



When considering a particular directory structure, some operations that are to be performed on a directory i.e →

(i) search for a file (ii) create a file.
(iii) Delete a file (iv) List a directory.
(v) Rename a file. (vi) Traverse the filesystem.

10-12-09
Tuesday

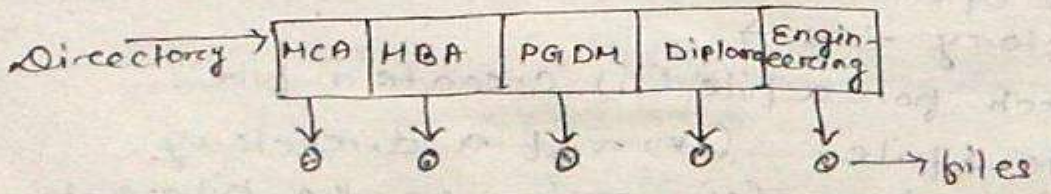
Types of Directory Structure :-

(1) Single - level Directory.
(2) Two - level Directory.
(3) Tree Directory.
(4) Acyclic Directory.

(1) Single - Level Directory →

This directory is one of the simplest directo structure. All files are contained in the same directory, which is easy to support and understan A single level directory has significant limitations, when the no. of files increases. Since, all the files are in the same directory, they must have unique names. If 2 users call their data file with same name, then the unique name rule is violated. Generally, MS-DOS O.S allows only 11 characters file name, while UNIX allows 255 characters file name. A single ~~level~~ user on a single level directory may find it difficult to remember the names of all the files, as the no. of files increases.

The structure of single-level directory can be represented as -

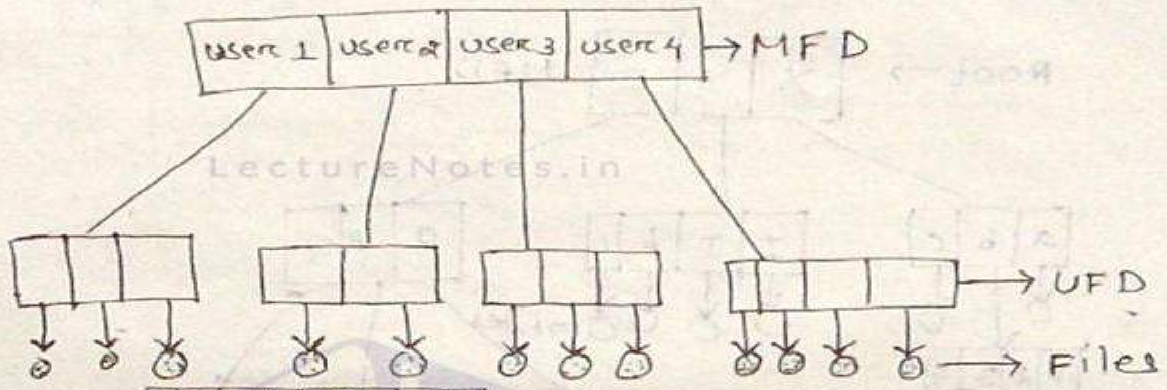


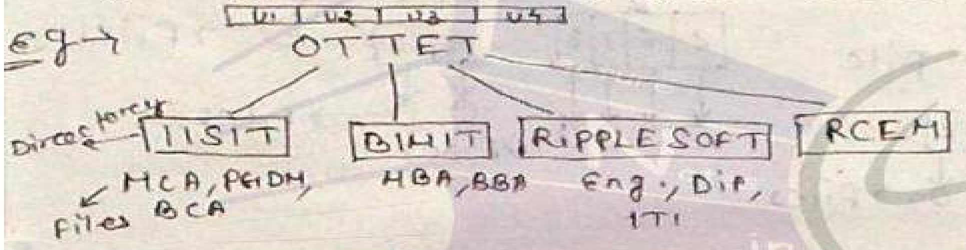
(2) Two - Level Directory →

~~The major disadvantage to~~ A single - level directory is the confusion of file names between different users. To avoid this confusion, create a separate directory for each user.

In Two-level directory

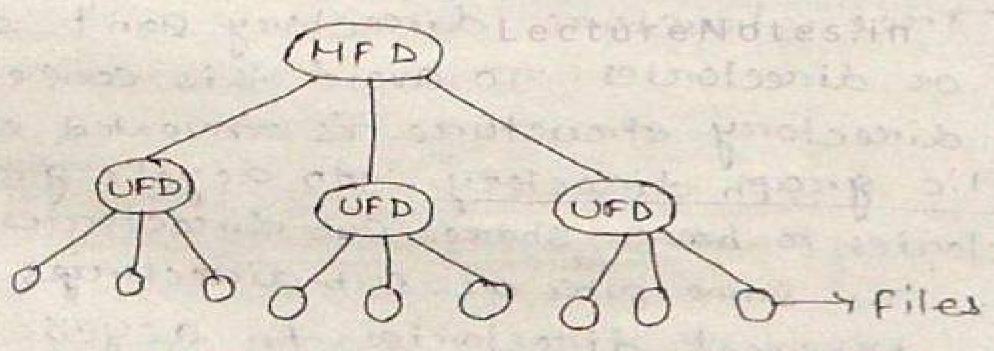
structure, each user has its own user file directory (UFD). Each UFD has a similar structure. When a user job starts, the system's master file directory (MFD) is searched. When a user refers to a particular file, his own UFD is searched. The structure of Two-level directory can be represented as →





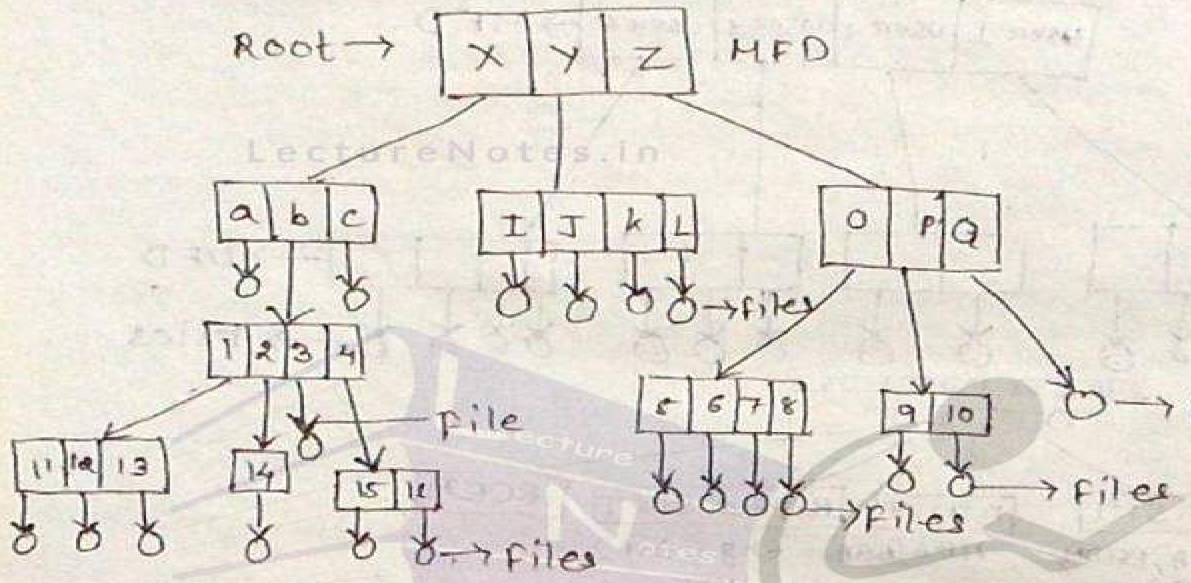
One advantage of this directory structure is to solve the collision problems.

The tree structure of Two-level directory structure can be represented as →



③. The Tree Structure Directory →

We can extend the directory structure to a tree of arbitary height. This allows user to create their own sub-directories and organize their files accordingly. A tree is the most common directory structure which consists of the root directory. every file system has a unique path name. A path name is the path from root, through all sub-directories to a specified file. A directory contains a set of files and sub-directories. All the sub-directories have the same internal format. The structure for this can be represented as →

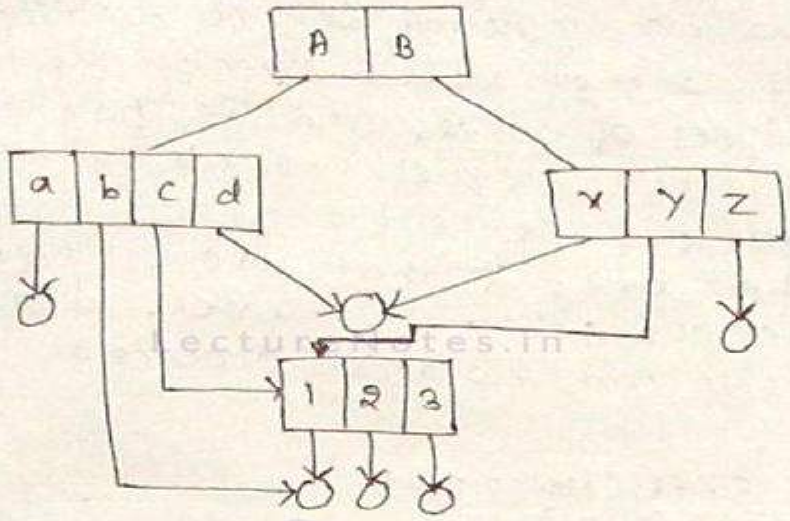


eg → Root / X / b / 2 / 14 (path name)

④. Acyclic Graph Directory →

Tree structure directory can't share files or directories. To avoid this concept, another directory structure is invented called acyclic graph directory. An acyclic graph allows directories to have shared sub-directories and files. The same files or sub-directories may be in 2 different directories. An acyclic graph is more ~~complex~~ flexible than simple tree structure, but is more ~~complex~~ complex. Another problem with this structure is deletion operation. To avoid this problem, some system doesn't allow shared directory.

The structure for this can be represented as →



11-12-09
Friday

Protection :-

When information is kept in a computer system, we want to keep it safe from physical damage and improper access. Physical damage is provided by duplicate copies of files. File systems can be damaged by hardware problems, power failures etc.

(reliability) (protection)

Improper access can be provided in many ways. For a small single user system, we provide protection by physically removing the disk and locking them. But in multiuser systems, other mechanisms are needed.

Types of Access :-

To protect files is a direct result of the ability to access files. Systems that donot permit access to the files of other users donot need protection. Thus, we can provide complete protection by prohibiting access.

Protection mechanism provides controlled access by limiting the type of access that can be made. Several types of operations may be controlled i.e →

1) Read, write, execute, append, delete, list, etc.

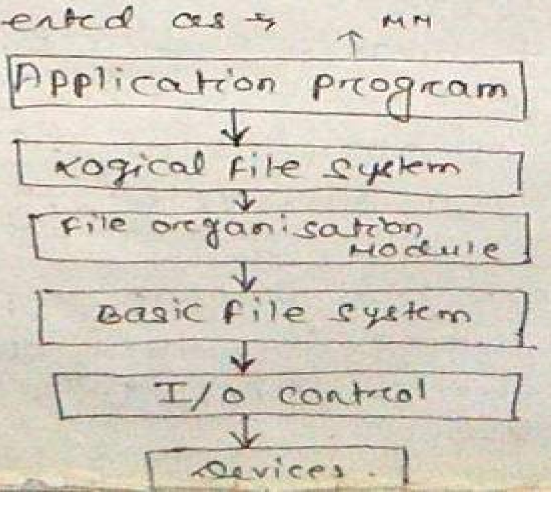
Access Control :-

User needs different types of access to a file or directory. The most general scheme to implement is ~~oo~~ access control which specify the user name and the types of access allowed for each user. When a user request to access a particular file, the O.S checks the access list associated with that file. If that user listed for a requested access, then the access allowed. otherwise, protection violation occures.

File System Structure :-

File system provides the mechanism for online storage and access to file contains contents including data and programs. The file system resides permanently on secondary storage, which is designed to hold a large amount of data permanently.

To provide an efficient and convenient access to the disk, the O.S imposes one or more file systems to allow the data to be stored, located and retrieved easily. The file system itself composed of many different levels and the level structure of file system can be represented as → MM



Memory Management

1. Memory allocation techniques
    a. Contiguous memory allocation
    b. Non contiguous memory allocation
2. Swapping
3. Paging
    a. Segmentation, virtual memory using paging
4. Demand paging, page fault handling

Memory Management

Some of the issues involved in the main memory.

Allocation: First of all the processes that are scheduled to run must be resident in the memory. These processes must be allocated space in main memory.

Swapping, fragmentation and compaction: If a program is moved out or terminates, it creates a hole, (i.e. a contiguous unused area) in main memory. When a new process is to be moved in, it may be allocated one of the available holes. It is quite possible that main memory has far too many small holes at a certain time. In such a situation none of these holes is really large enough to be allocated to a new process that may be moving in. The main memory is too fragmented. It is, therefore, essential to attempt compaction. Compaction means OS re-allocates the existing programs in contiguous regions and creates a large enough free area for allocation to a new process.

Garbage collection: Some programs use dynamic data structures. These programs dynamically use and discard memory space. Technically, the deleted data items (from a dynamic data structure) release memory locations. However, in practice the OS does not collect such free space immediately for allocation. This is because that affects performance. Such areas, therefore, are called garbage. When such garbage exceeds a certain threshold, the OS would not have enough memory available for any further allocation. This entails compaction (or garbage collection), without severely affecting performance.

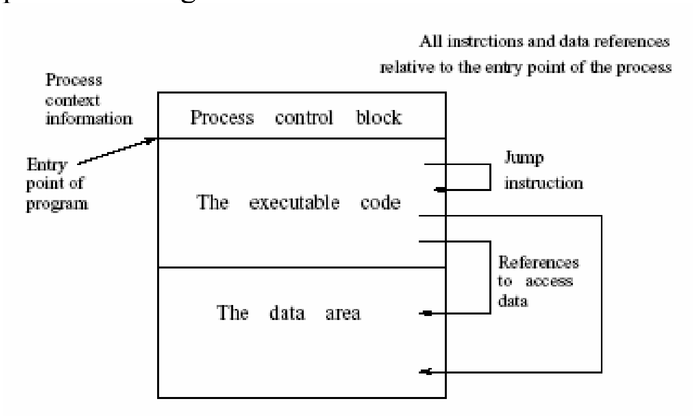
Protection: With many programs residing in main memory it can happen that due to a programming error (or with malice) some process writes into data or instruction area of some other process. The OS ensures that each process accesses only to its own allocated area, i.e. each process is protected from other processes.

Virtual memory: Often a processor sees a large logical storage space (a virtual storage space) though the actual main memory may not be that large. So some facility needs to be provided to translate a logical address available to a processor into a physical address to access the desired data or instruction.

IO support: Most of the block-oriented devices are recognized as specialized files. Their buffers need to be managed within main memory alongside the other processes. The considerations stated above motivate the study of main memory management. One of the important considerations in locating an executable program is that it should be possible to relocate it any where in the main memory.

Memory allocation

Let us assume the main memory is a linear map or one-dimensional array. If the address is known then its content can be fetched. So, a process residing in the main memory, set the program counter to an absolute address of its first instruction and can initiate its run. Also, if the locations of data is known then it can be fetched. This means that a process can be load with only absolute addresses for instructions and data, only when those specific addresses are free in main memory. But This will loose flexibility with regard to loading a process. For instance, we cannot load a process, if some other process is currently occupying that area which is needed by this process. This may happen even though we may have enough space in the memory. To avoid this processes are generated to be relocatable.


All instrctions and data references relative to the entry point of the process
Process context information
Process control block
Entry point of program
The executable code
Jump instruction
References to access data
The data area

Initially, all the addresses in the process are relative to the start address. With this flexibility the OS can allocate any area in the memory to load this process. Its instruction, data, process context (process control block) and any other data structure required by the process can be accessed easily if the addresses are relative. Suppose a process created a hole on moving out. If non-relocatable addresses are to be used then severe problem can occure.
When the process moves back in, that particular hole (or area) may not be available any longer. In case we can relocate, moving a process back in creates no problem. This is so because the process can be relocated in some other free area.

Contiguous memory allocation:

Contiguous memory allocation is a classical memory allocation model that assigns a process consecutive memory blocks (that is, memory blocks having consecutive addresses). When a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process. If sufficient contiguous memory is found, the process is allocated memory to start its execution. Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available.

The contiguous memory allocation scheme can be implemented in operating systems with the help of two registers, known as the base and limit registers. When a process is executing in main memory, its base register contains the starting address of the memory location where the process is executing, while the amount of bytes consumed by the process is stored in the limit register. A process does not directly refer to the actual address for a corresponding memory location. Instead, it uses a relative address with respect to its base register. All addresses referred by a program are considered as virtual addresses. The CPU generates the logical or virtual address, which is converted into an actual address with the help of the memory management unit (MMU). The base address register is used for address translation by the MMU. Thus, a physical address is calculated as follows:

Physical Address = Base register address + Logical address/Virtual address

The address of any memory location referenced by a process is checked to ensure that it does not refer to an address of a neighboring process. This processing security is handled by the underlying operating system.

One disadvantage of contiguous memory allocation is that the degree of multiprogramming is reduced due to processes waiting for free memory.

Non-contiguous memory allocation:

Noncontiguous memory allocation assigns the separate memory blocks at a different location in memory space in a nonconsecutive manner to a process requesting for memory. The noncontiguous memory allocation also **reduces** the **memory wastage** caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.

Difference between contiguous and non contiguous memory allocation

BASIS	Contiguous memory allocation	Noncontiguous memory allocation
Basic	Allocates consecutive blocks of memory to a process	Allocates separate blocks of memory to a process.
Overheads	Contiguous memory allocation does not have the overhead of address translation while execution of a process	Noncontiguous memory allocation has overhead of address translation while execution of a process.
Execution rate	A process executes faster in contiguous memory allocation	A process executes quite slower comparatively in noncontiguous memory allocation.
Solution	The memory space must be divided into the fixed-sized partition and each partition is allocated to a single process only	Divide the process into several blocks and place them in different parts of the memory according to the availability of memory space available.
Table	A table is maintained by operating system which maintains the list of available and occupied partition in the memory space	A table has to be maintained for each process that carries the base addresses of each block which has been acquired by a process in memory.

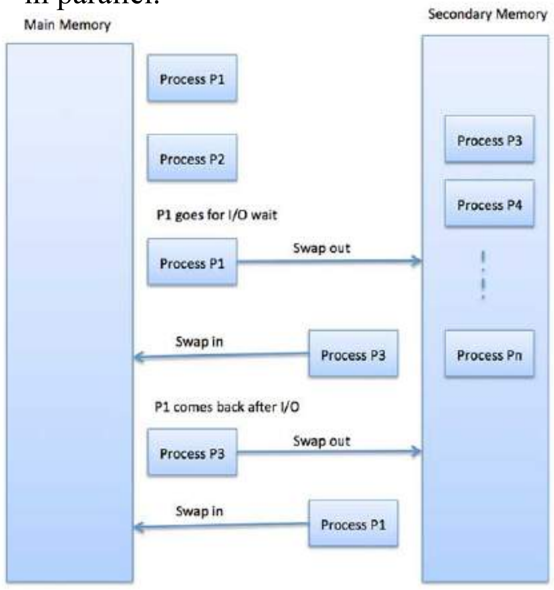
Types of memory allocation:

1. **Best fit memory allocation:** In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.
2. **Worst fit memory allocation:** Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.
3. **first fit memory allocation:** This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory with space more than or equal to it's size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient size.

2. SWAPPING :

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds
```

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

3.FRAGMENTATION :

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.

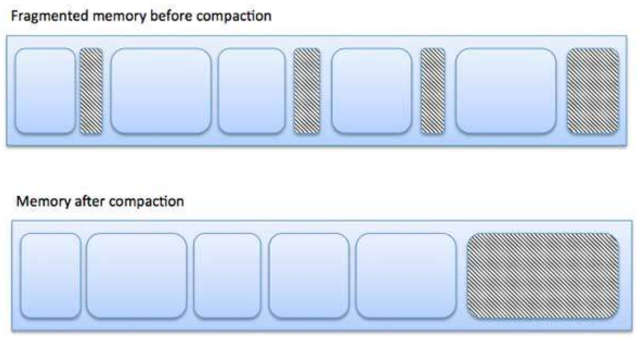
Fragmentation is of two types −

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.

Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.



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.

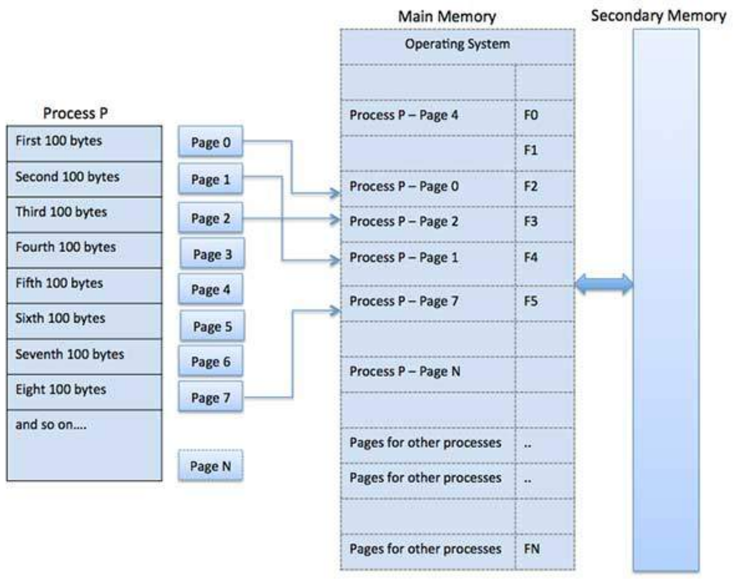
The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

4. **PAGING** :

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



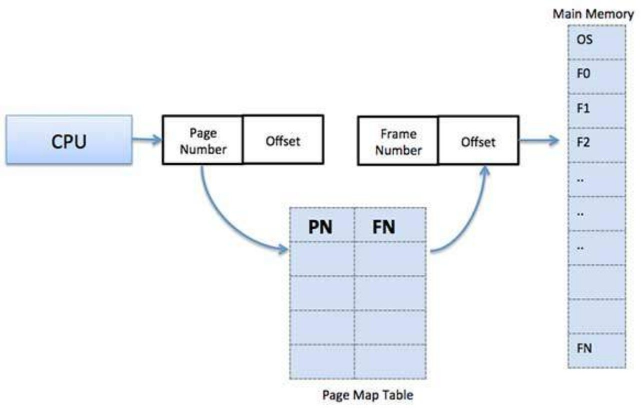
Address Translation:

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.

Advantages and Disadvantages of Paging:

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

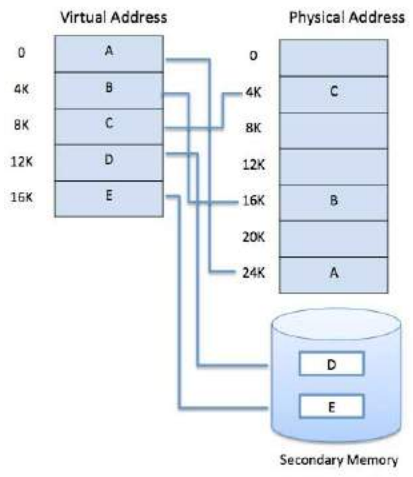
5. VIRTUAL MEMORY:

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

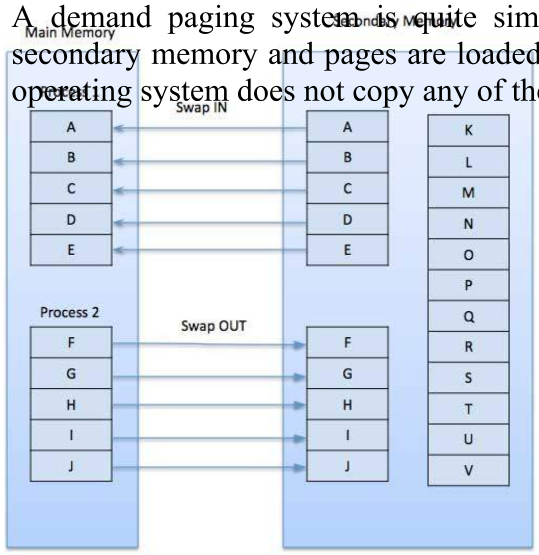


Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses which s shown in the figure.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

6. DEMAND PAGING:

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages:

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages:

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

7. PAGE FAULT AND PAGE FAULT HANDLING :

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :

- The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
- An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
- Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Some times hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
- Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
- If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
- If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.

- As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
- Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
- Assembly Routine reloads register and other state information, returns to user space to continue execution.

8. PAGE REPLACEMENT ALGORITHM:

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

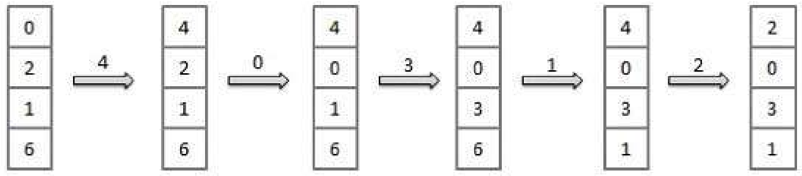
- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

8.1 First In First Out (FIFO) algorithm:

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses :x x x x x x x x x



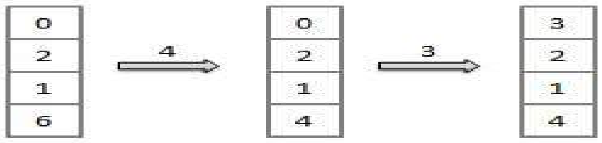
Fault Rate = 9 / 12 = 0.75

8.2 Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses :x x x x x x



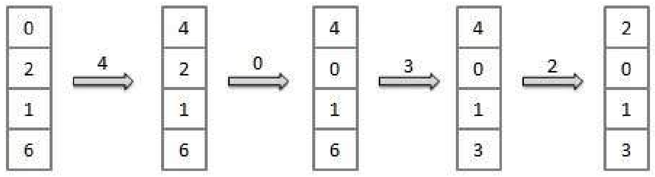
Fault Rate = 6 / 12 = 0.50

8.3 Least Recently Used (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 by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses :x x x x x x x x



Fault Rate = 8 / 12 = 0.67

8.4 Page buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

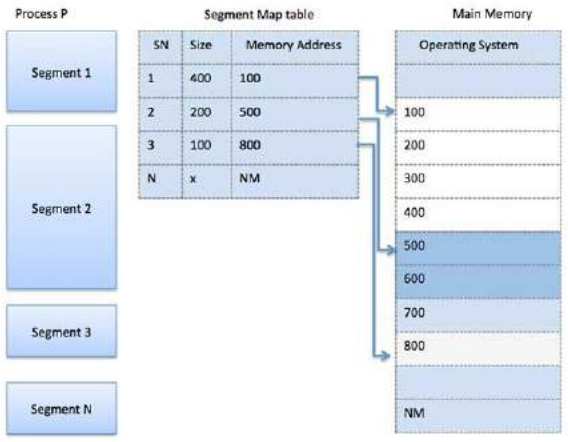
9. SEGMENTATION:

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



Operating System - Processes

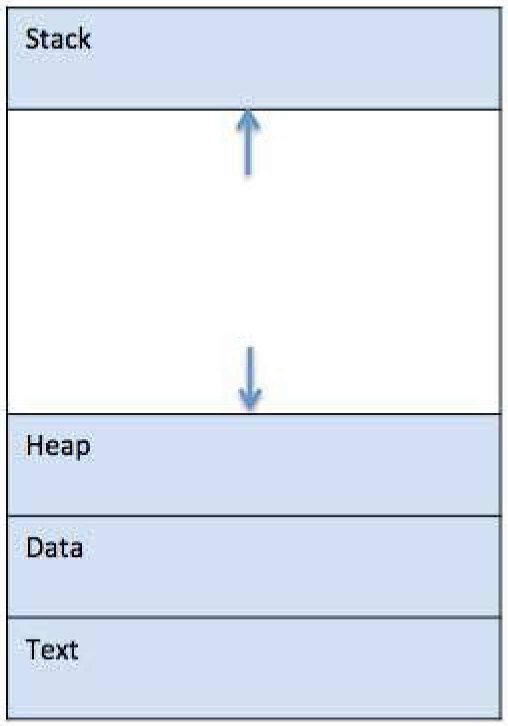
Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

> A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory −



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text** This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language −

```c
#include <stdio.h>

int main() {
   printf("Hello, World! \n");
   return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

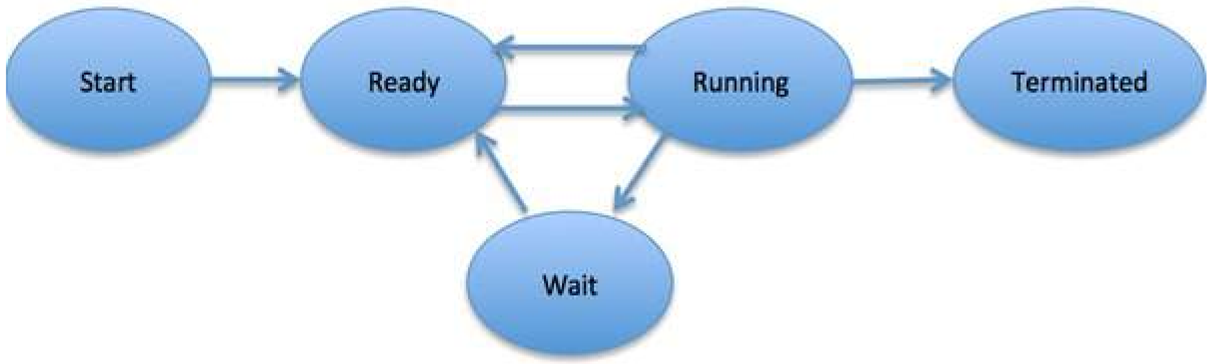
A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
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.

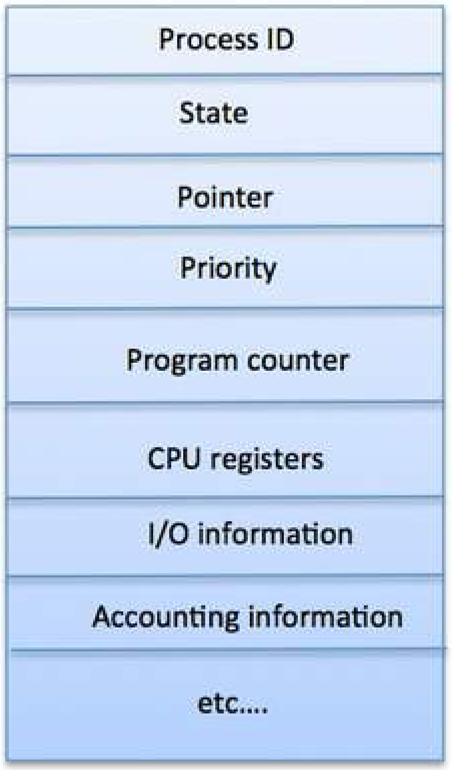


Process Control Block (PCB)

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

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



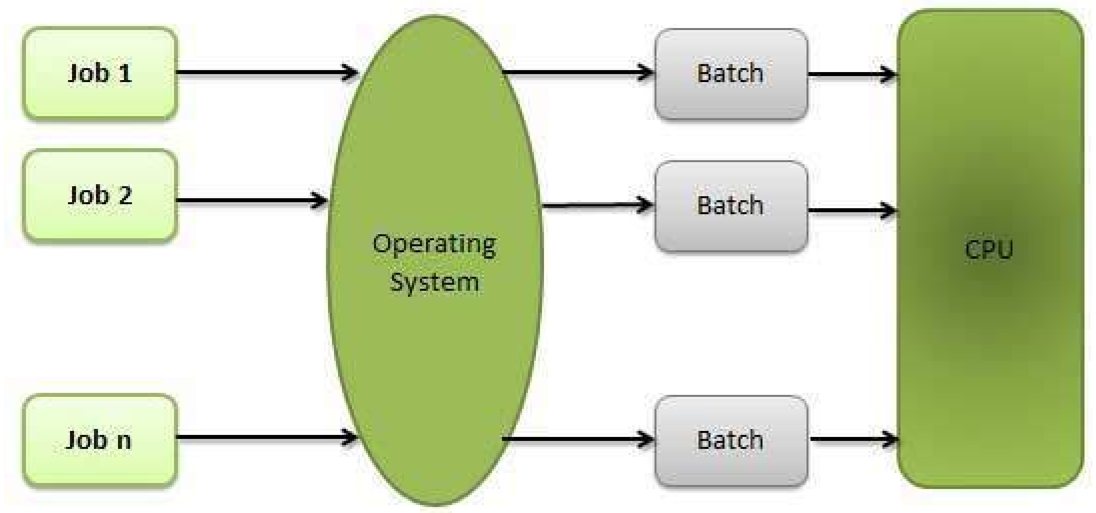
The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Operating System - Properties

Batch processing

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts. An operating system does the following activities related to batch processing −

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a jobs in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.



Advantages

- Batch processing takes much of the work of the operator to the computer.
- Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention.

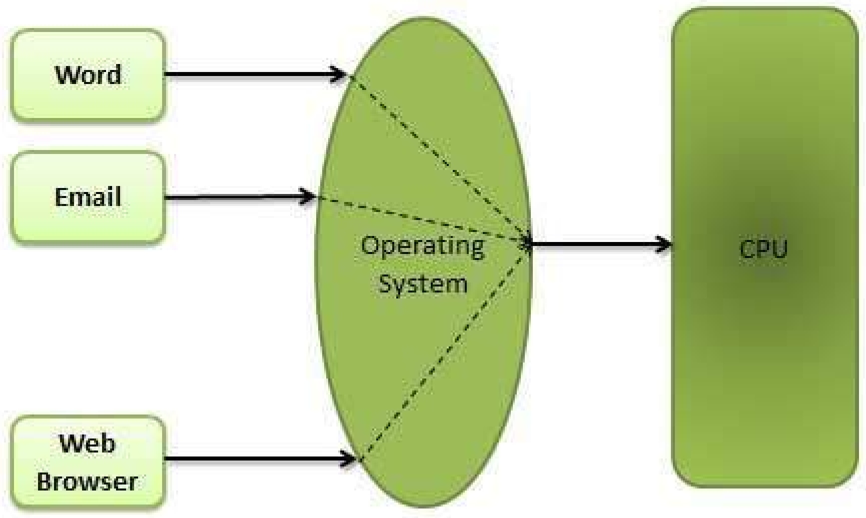
Disadvantages

- Difficult to debug program.
- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Multitasking

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking −

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.

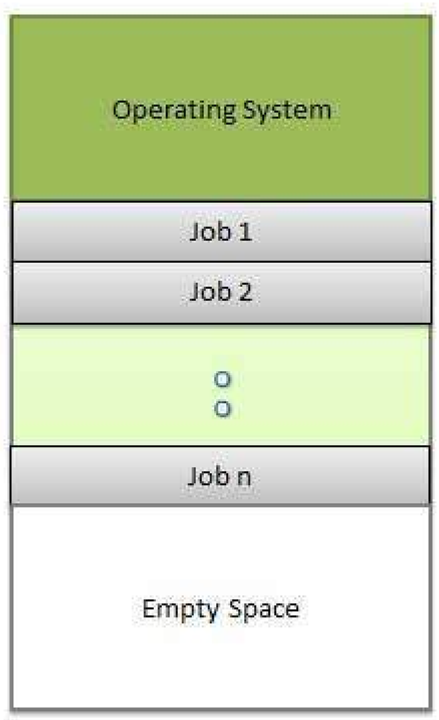


- A program that is loaded into memory and is executing is commonly referred to as a **process**.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprogramming

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.

Interactivity

Interactivity refers to the ability of users to interact with a computer system. An Operating system does the following activities related to interactivity −

- Provides the user an interface to interact with the system.
- Manages input devices to take inputs from the user. For example, keyboard.
- Manages output devices to show outputs to the user. For example, Monitor.

The response time of the OS needs to be short, since the user submits and waits for the result.

Real Time System

Real-time systems are usually dedicated, embedded systems. An operating system does the following activities related to real-time system activity.

- In such systems, Operating Systems typically read from and react to sensor data.
- The Operating system must guarantee response to events within fixed periods of time to ensure correct performance.

Distributed Environment

A distributed environment refers to multiple independent CPUs or processors in a computer system. An operating system does the following activities related to distributed environment −

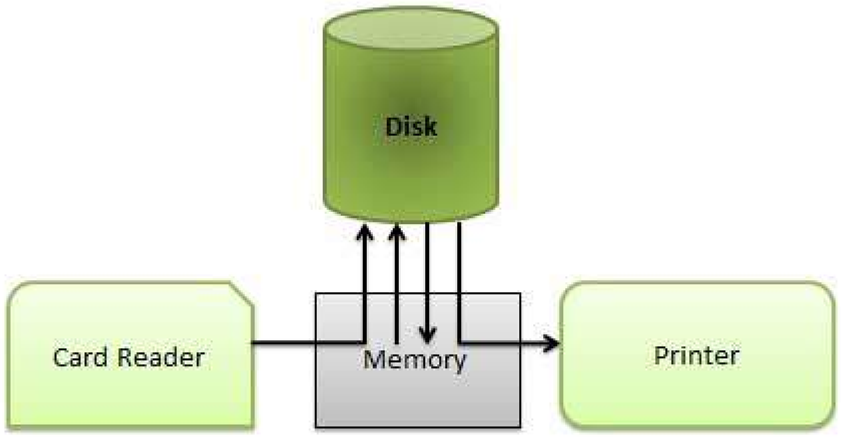
- The OS distributes computation logics among several physical processors.
- The processors do not share memory or a clock. Instead, each processor has its own local memory.
- The OS manages the communications between the processors. They communicate with each other through various communication lines.

Spooling

Spooling is an acronym for simultaneous peripheral operations on line. Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices.

An operating system does the following activities related to distributed environment −

- Handles I/O device data spooling as devices have different data access rates.
- Maintains the spooling buffer which provides a waiting station where data can rest while the slower device catches up.
- Maintains parallel computation because of spooling process as a computer can perform I/O in parallel fashion. It becomes possible to have the computer read data from a tape, write data to disk and to write out to a tape printer while it is doing its computing task.



Advantages

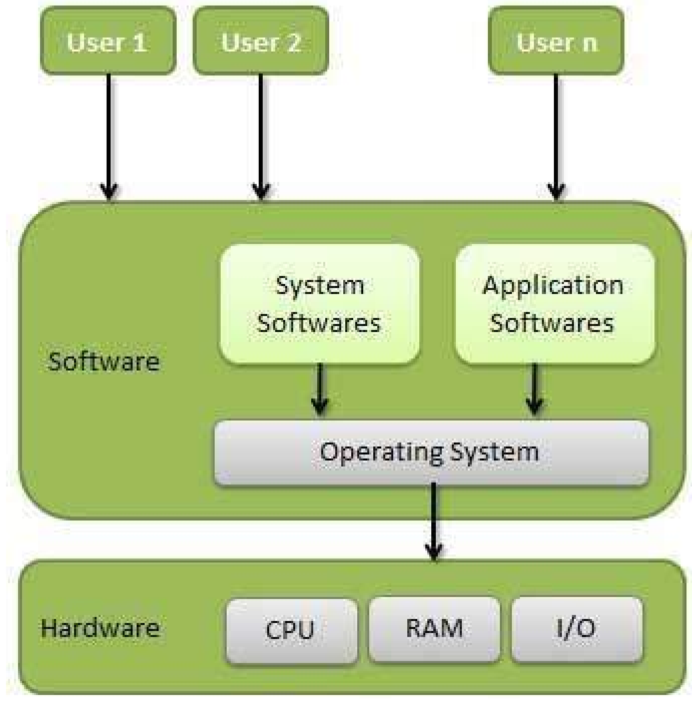
- The spooling operation uses a disk as a very large buffer.
- Spooling is capable of overlapping I/O operation for one job with processor operations for another job.

Operating System – Overview

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.
Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



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

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.
Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management:

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.

- In multiprogramming, the OS decides which process will get memory when and how much.

- Allocates the memory when a process requests it to do so.

- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management:

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.

- Allocates the processor (CPU) to a process.

- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management:

- Keeps tracks of all devices. The program responsible for this task is known as the **I/O controller**.

- Decides which process gets the device when and for how much time.

- Allocates the device in the most efficient way.

- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management:

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.

- Decides who gets the resources.

- Allocates the resources.

- De-allocates the resources.

Other Important Activities

Following are some of the important activities that an Operating System performs:

- **Security** -- By means of password and similar other techniques, it prevents unauthorized access to programs and data.

- **Control over system performance** -- Recording delays between request for a service and response from the system.

🎬 **Job accounting** -- Keeping track of time and resources used by various jobs and users.

🎬 **Error detecting aids** -- Production of dumps, traces, error messages, and other debugging and error detecting aids.

🎬 **Coordination between other software and users** -- Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

OPERATING SYSTEM STRUCTURE - I

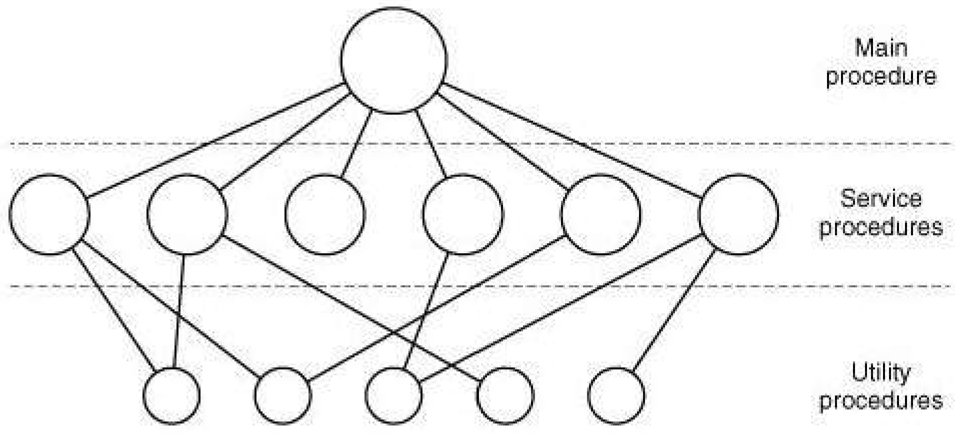
Operating System Structure:

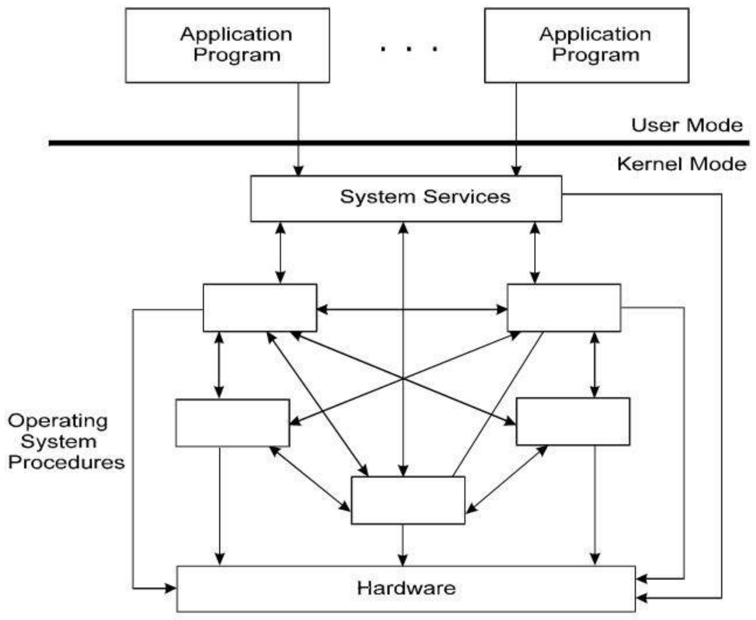
The structure of an operating system is dictated by the model employed in building them. An operating system model is a broad framework that unifies the many features and services the operating system provides and tasks it performs. Operating systems are broadly classified into following categories, based on the their structuring mechanism as follows:

a. Monolithic System
b. Layered System
c. Virtual Machine
d. Exokernels
e. Client-Server Model

Monolithic System

The components of monolithic operating system are organized haphazardly and any module can call any other module without any reservation. Similar to the other operating systems, applications in monolithic OS are separated from the operating system itself. That is, the operating system code runs in a privileged processor mode (referred to as kernel mode), with access to system data and to the hardware; applications run in a non-privileged processor mode (called the user mode), with a limited set of interfaces available and with limited access to system data. The monolithic operating system structure with separate user and kernel processor mode is shown in Figure.



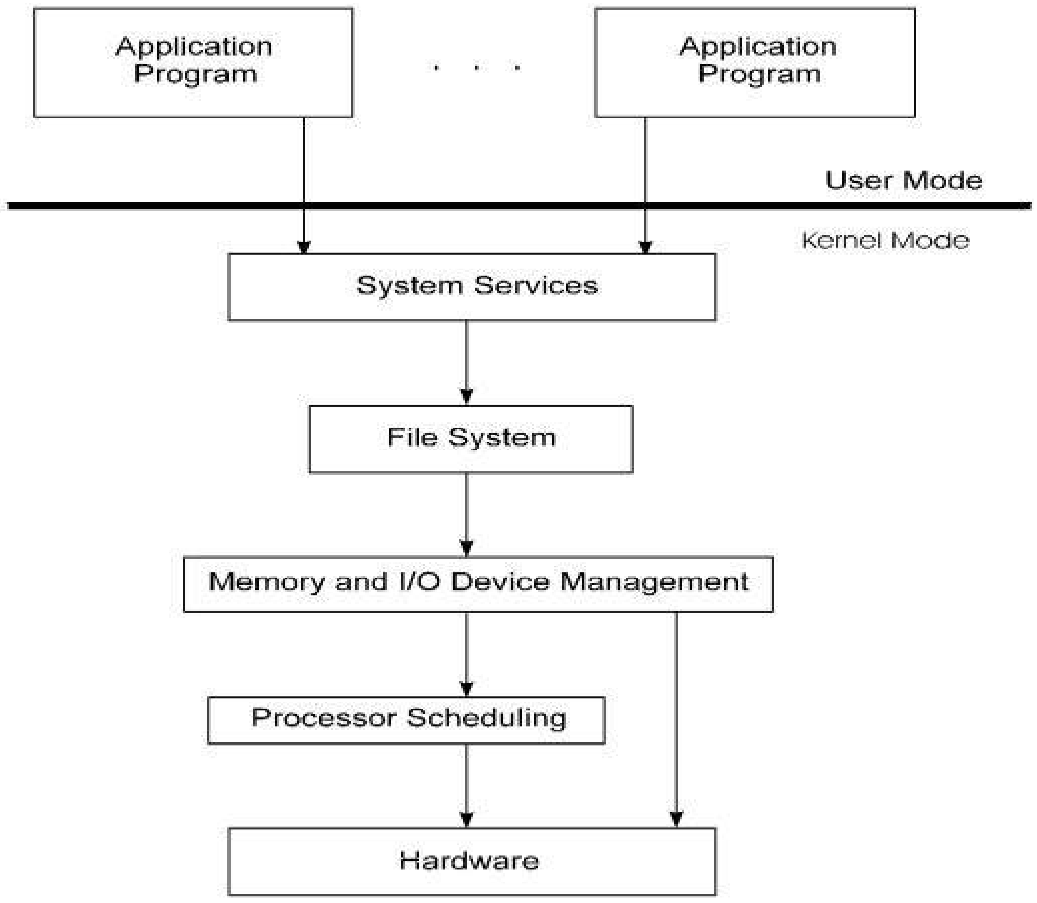


This approach might well be subtitled "The Big Mess." The structure is that there is no structure. The operating system is written as a collection of procedures, each of which can call any of the other ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one, if the latter provides some useful computation that the former needs.
Example Systems: CP/M and MS-DOS

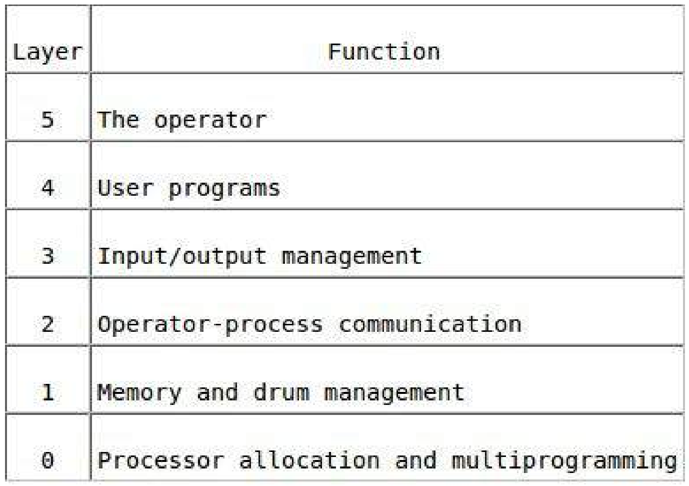
Layered Operating System

The layered approach consists of breaking the operating system into the number of layers(level), each built on the top of lower layers. The bottom layer (layer 0) is the hardware layer; the highest layer is the user interface.
The main advantages of the layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and

system verifications. That is in this approach, the Nth layer can access services provided by the (N-1)th layer and provide services to the (N+1)th layer. This structure also allows the operating system to be debugged starting at the lowest layer, adding one layer at a time until the whole system works correctly. Layering also makes it easier to enhance the operating system; one entire layer can be replaced without affecting other parts of the system.



The layer approach to design was first used in the THE operating system at the Technische Hogeschool Eindhoven. The THE system was defined in the six layers , as shown in the fig below.

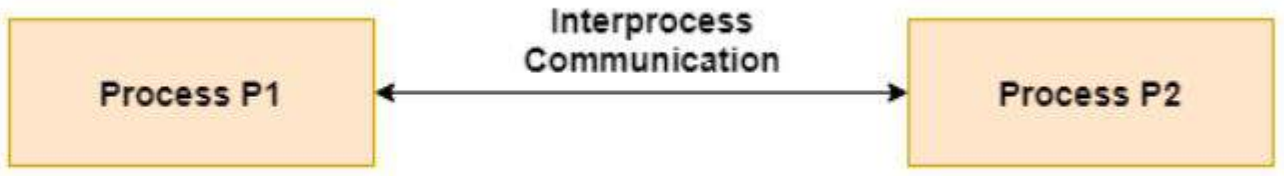


Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Example Systems: VAX/VMS, Multics, UNIX

PROCESS COMMUNICATIONS

- Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the transferring of data from one process to another.

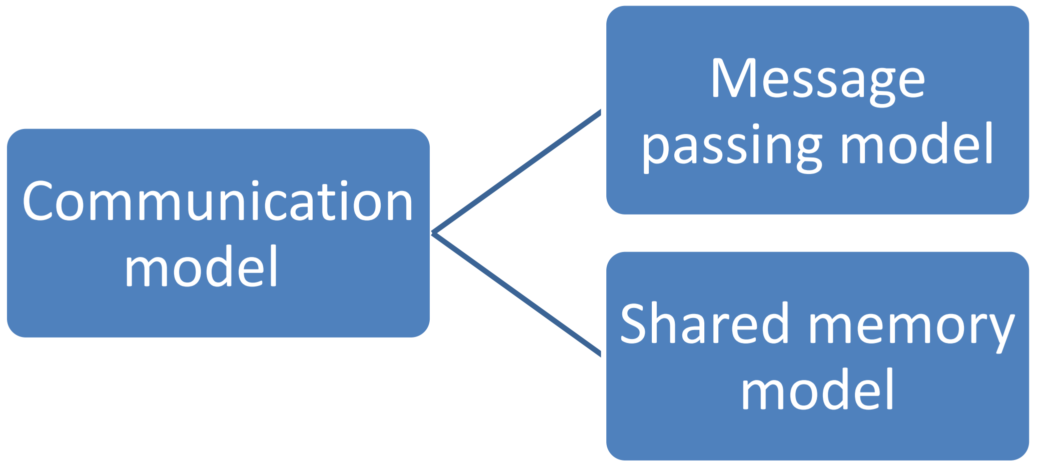


PROCESS COMMUNICATIONS

Processes can communicate with themselves in two ways.

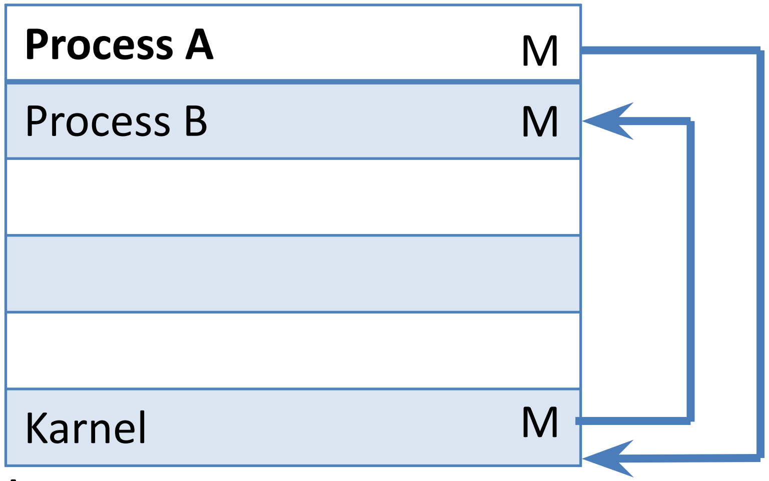
Message-passing model

Shared memory model



PROCESS COMMUNICATIONS

Message-passing model: Here the information is exchanged through an interprocess communication which is provided by the operating system.



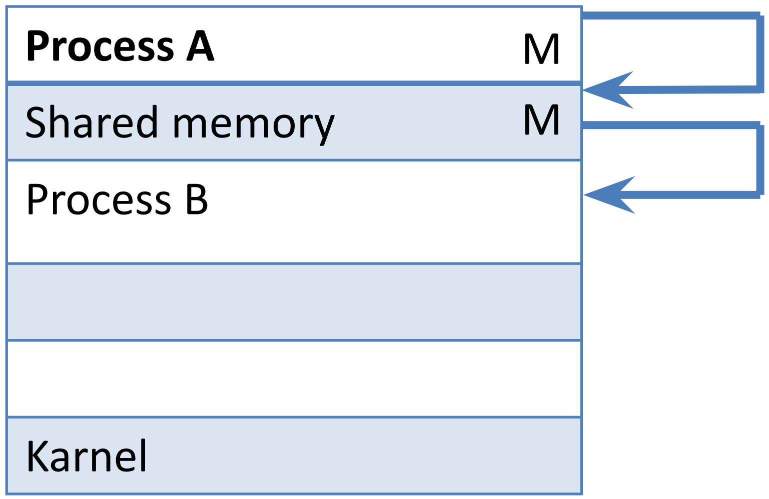
- The communication among processes takes place by system calls get hostid, get processid, open connection, close connection, accept connection, wait for connection, read message and write message.

- Before the communication, a connection should be opened between communicating processes
- The name of the communicating processes must be known to operating system at priori.
- Each computer in a network has a host name and each process has a process name.
- The process wants to communicate first execute the open connection system call and then get hostid and get processid call of the recipient process and send it to the karnel along with message.
- The karnel sends a request for a connection to the recipient process. If the recipient process is not ready then it must wait.

- When the recipient process is ready , then the recipient process execute accept connection.
- Once the connection between karnel and the recipient process is open, the message is then send.
- Once the message exchange completed, both the process execute close connection system call.
- Message passing communication is useful when small number of data needs to be exchanged.

PROCESS COMMUNICATIONS

Shared memory model: Here the information is exchanged through a memory shared by both the processors.



- The communication among processes takes place by executing map memory system call.

- Before communication through shared memory , both the processor should agree to share their memory.
- They may then exchange information by reading and writing data in these shared memory.
- The form of data and the location are determined by these processes and are not under the control of operating system.
- Shared memory communication is useful when large number of data needs to be exchanged with maximum speed.

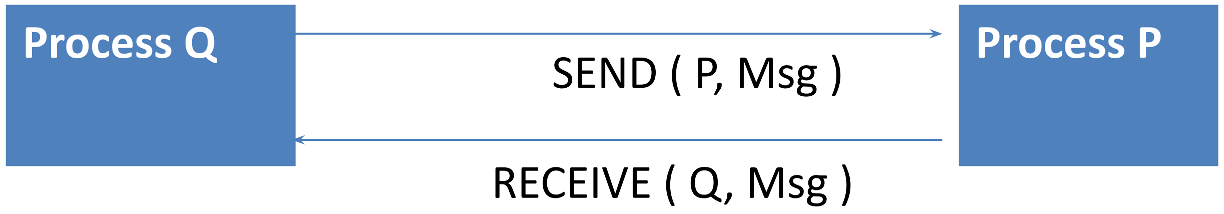
Process Communications

In addition to above , logically several methods can be implemented to established link between two processes and send , receive operations to achieve interprocess communication.

1. Direct Communication
2. Indirect Communication
3. Symmetric communication
4. Asymmetric communication
5. Automatic buffering communication

Direct Communication

- Here each process that wants to communicate must explicitly address the recipient or sender of the communication i.e. this direct communication shows symmetry in addressing. Hence this type of communication also known as Symmetric communication.
- A link is established automatically between each pair of process who wants to communicate. i.e. each process needs to know the identity of other process to communicate.
- Between each pair of processes , there exists exactly one link.
- The link may be unidirectional or bidirectional.

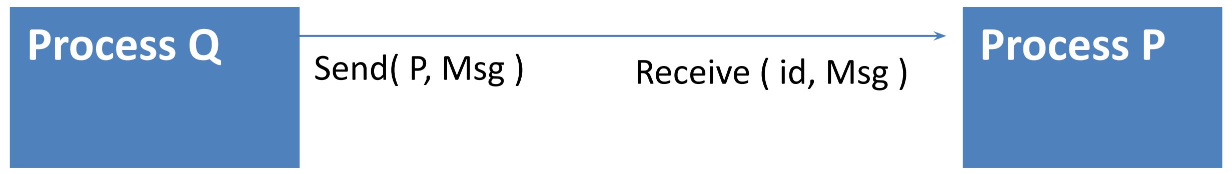


- The sender process execute the send(P , message) operation to send the message to the process P.
- The receiver process execute receive(Q , message) operation to receive a message from process Q which notify the process Q that the message has been consumed.

Asymmetric communication

- Here only the sender process explicitly address the recipient.
- A link is established automatically between each pair of process who wants to communicate. Here the sender must know the identity of the recipient.
- Between each pair of processes , there exists exactly one link.
- The link may be unidirectional.

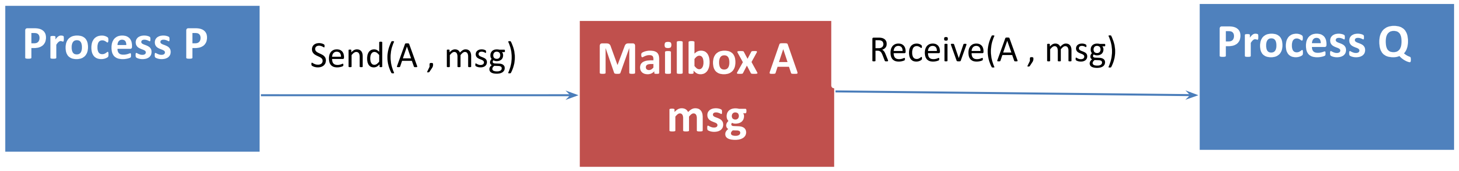
- The sender process execute the send(P , message) operation to send the message to the process P.
- The receiver process execute receive(id , message) operation to receive a message from any process.



- Both symmetric and Asymmetric communication has limited modularity i.e. changing the name of one process may necessitate examining all other process definitions.

Indirect Communication:

- Here communication can be done through a mailbox which can be viewed abstractly as an object into which message can be placed by a process and from which message can be removed.
- Each mailbox has unique identification.
- A link will be established between at most a single pair of processes who shared the mailbox.



- This link can be unidirectional or bidirectional.

- Now the process who wants to send message can send the message to the mailbox by calling send(A , msg) where A is the id of mailbox.
- The receiver will receive the message from Mailbox A by calling receive(A, msg).

A mailbox can be owned either by a process or by the operating system.

Mailbox owned by process:

- Here the mailbox is attached to a process.
- The process to whom the mailbox is attached is declared as the owner who can only receive message through this mailbox.
- Other processes who use this mailbox can only send message into this mailbox.

- When the owner of the mailbox terminates , the mailbox will disappear and all other processes who share the mailbox will be notified that this mailbox is no longer exist.

Mailbox owned by operating system:

- Here mailbox is independent and not attached to any process.
- The operating system allow a process to create the mailbox and that process become the owner of that mailbox.
- The owner process can only receive the message.
- This ownership and receive privilege can be passed to another process through proper system call.
- When the mailbox is no longer used by any process, the operating system should reclaim the space provided to mailbox by calling garbage collector.

Automatic buffering:

- Here a link will be established between the processes who wants to communicate.
- The link may reside the message temporarily.
- A process wants to communicate, sends a message to other process. The message will be send without any delay.
- After receiving, the process acknowledge the sender with proper signal.

Let Process P wants to send message to Process Q.

process P executes the sequence

- Send (Q, Msg)
- Receive(Q,msg)

Then process Q executes the sequence

- Receive (P, Msg)
- Send(P, 'Ack')

In the case of Automatic Buffering, the link may have Zero capacity, Bounded capacity or Unbounded capacity.

Zero capacity link-

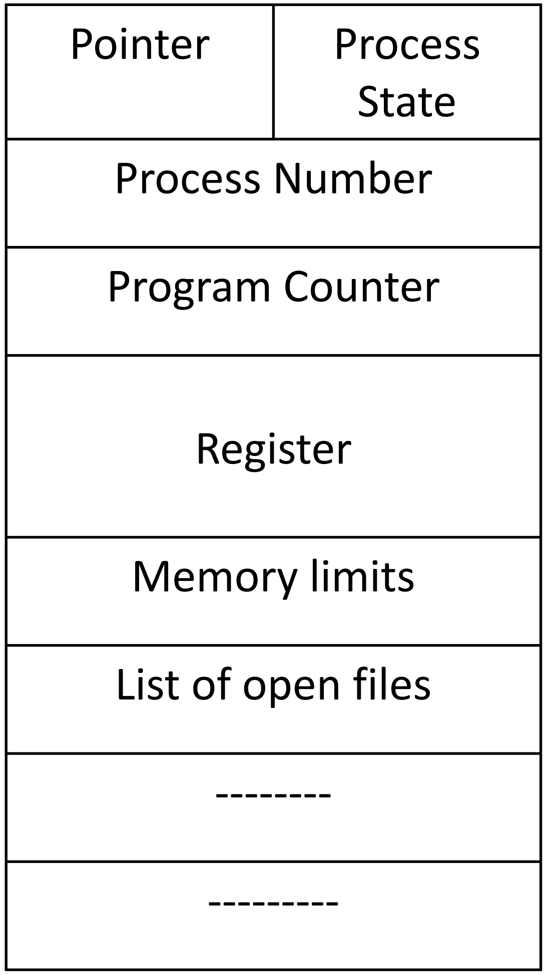
- This is also known as automatic buffering .
- The queue has maximum length 0 i.e. the link can not have any message waiting in it. Here the sender and receiver must synchronized for a message to be transferred.

Bounded capacity & Unbounded capacity:

- Here the queue has finite length *n* but it is infinite length for unbounded capacity
- if the queue is not full, the sender is sending the message continuously (Bounded).
- If the queue is full, then the sender must wait until space is available in the queue (Bounded).
- In case of Unbounded capacity any number of message can wait in the link but the sender never delayed.

Process Control Block

- Each process is represented in the operating system by a Process Control Block (PCB) or Task Control Block.
- A Process Control Block is a data structure maintained by the Operating System for every processes.
- The PCB is identified by an integer process ID (PID) which keeps all the information needed to keep track of a process.



Process State- The current state of the process i.e. whether it is new, ready, running, waiting, halted and so on.

- Pointer -A pointer to parent process.
- Process Number - Unique identification for each of the process in the operating system.
- Program counter- The counter indicates the address of the next instruction to be executed for this process.
- Memory management information - This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
- Register- The registers may vary in numbers and types, depending on the computer architecture. They may be accumulator, index register, stack pointer, general purpose register etc. these registers are being used to save the state information of a process when an interrupt occurs, and allow the process to be continued correctly afterwards.

- List of open files – These are the files which are currently being used by the process in execution.

Beyond these other information may also stored in the PCB.

CPU scheduling information – this includes a process priority, pointer to scheduling queue and other scheduling parameters.

Accounting Information – It includes the amount of CPU and real time used, time limits, account numbers, job or process number and so on.

I/O status information – It includes the list of I/O devices allocated to this process, list of open files and so on.