1. Why OS is also known as Resource Manager and an extended machine?

→

A computer system has many resources (hardware/software) which maybe required to complete a task. The commonly required resources are ip/op devices, memory, file storage space, CPU, etc. The operating system acts as manager of the above resources and allocates them to specific programs and users, whenever necessary to perform a particular task. Therefore, operating system is the resource manager i.e it can manage the resource of a computer system internally. The resources are processor, memory, files, and I/O devices. In simple terms, an OS is the interface between the user and the machine.

The computer system is design so that use of resource to maximize the user is performing. In this case the operating system is designed mostly for easy with good performance, the performance is important to the user but it doesn't matter if most of the system is sitting idle, waiting for the slow i/o speed of the user. The OS is design to maximize resource utilization.

The program that hides the truth about the hardware from the programmer and present and a nice simple view a name file that can be read & written as 'operating system'. OS shields the programmer from the interface, the abstraction offers by the operating system is ~~slow~~ slower

and easier to use than the underlying hardware. The main function of operating system is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than underlying hardware.

2. Write about history of OS.

→

Operating systems provide a set of functions and links needed to control and synchronize computer hardware. These are used by most application programs on a computer. The first computers had no OS. Every program needed the full hardware specification to run correctly and perform standard tasks, and its own drivers for peripheral devices. Considering the hardware system was evolving and becoming more complex, app programs become a necessity. In the 1960s Bell labs started working on the origins of UNIX, the first multi-tasking and multi-user functionality operating system. The first version was available in the 70s.

In the 1977s The AppleII series was born. This is a family of home computers which was the first highly successful micro-computer. It was an 8-bit computer with the first color-graphics. Apple Dos 3.3 was the final and most popular version of the software.

In 1981 May MS-DOS was launched by Microsoft and started on the basis of 86-DOS by a company called 'Seattle computer products' which was created by Tim Patterson. MS-DOS was launched shipped and user for the IBM personal computer, which was also licensed to IBM but called PC-DOS.

On May 22, 1990 Microsoft Windows launched Windows 3.0 The graphical environment was the 3rd major release

On Sept 17, 1991 Linux released its OS kernel. Linux is a free OS which is widely known for its distributions such as Ubuntu and it's commercial use like that of Redhat Hat enterprise Linux.

On June 25, 1998 Microsoft launched Windows 98, which sported a hybrid 16 bit and 32 bit GUI which makes it a graphical operating system.

On March 6, 2008 iPhone OSJ was the first ios for Apple's mobile operating system Apple stated that the iPhone ran on a version of its desktop OS macOS, then known as Mac OSX.

On May 2011 Google launched Chrome OS which is a Linux Kernel based OS. It is a free software which uses the Google Chrome web browser as it's a primary user interface (UI) and supports web application. Its User Data runs directly

off of the cloud, making it the first OS to be cloud-based. We are currently on the precipice of AI robotics, and blockchain and these sectors will lead us towards different dimensions of operating systems.

Chapter 2: Process / Thread Management

2.3 Processor Scheduling

Scheduling Techniques

1 First Come First Serve (FCFS/FIFO) Schedule

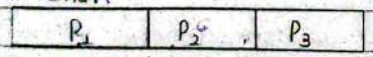
With this scheme, the process that requests the CPU first is allocated the CPU first. It is non-primitive in nature.

i) Eg: Consider following set of process that arrive at time 0 with length of CPU burst given in milliseconds.

Process	Burst time
P_1	24
P_2	3
P_3	3

Suppose process arrive in order P_1, P_2, P_3

Gantt Chart



Rikka Karanjit
171433

Waiting Time (Start - Arrival)

$P_1 = (0-0) = 0$

$P_2 = (24-0) = 24$

$P_3 = (27-0) = 27$

$\therefore$ Avg waiting Time $= \frac{0+24+27}{3} = \frac{51}{3} = 17$ ms

Turnaround Time (TAT) : W.T + B.T (Waiting Time + Burst Time)

$P_1 = 0+24 = 24$

$P_2 = 24+3 = 27$

$P_3 = 27+3 = 30$

$\therefore$ Average TAT $= \frac{24+27+30}{3} = \frac{81}{3}$

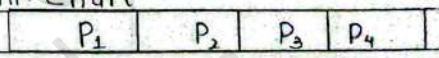
$= 27$ ms.

ii) Eg:

Process	Arrival Time	Burst Time (ms)
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Suppose process arrive in order P_1, P_2, P_3, P_4

Gantt Chart



Waiting Time [Start - Arrival] :

$P_1 = (0-0) = 0$

$P_2 = (8-1) = 7$

$P_3 = (12-2) = 10$

$P_4 = (21-3) = 18$

$\therefore$ Avg WT $= \frac{0+7+10+18}{4} = \frac{79}{4} = \frac{215}{4} = 8.75$

Turnaround Time (TAT) : WT + BT

$P_1 = 0+8 = 8$

$P_2 = 7+4 = 11$

$P_3 = 10+9 = 19$

$P_4 = 18+5 = 23$

$\therefore$ Avg TAT $= \frac{8+11+19+23}{4}$

$= 15.25$

2) Shortest - Job first (SJF) Scheduling

→ With this scheme, the process that has smallest CPU burst time is scheduled first.

→ If CPU burst of two process are same, FCFS is used to break tie.

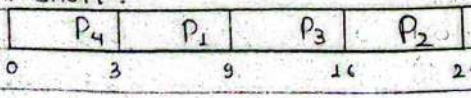
Examples:

i) Consider following set of process with CPU burst time given in ms (millisecond).

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Soln;

Gantt Chart:



Waiting time (Start - Arrival)

$P_1 = 3-0 = 3$

$P_2 = 16-0 = 16$

$P_3 = 9-0 = 9$

$P_4 = 0-0 = 0$

$\therefore$ Avg waiting time $= \frac{3+16+9+0}{4}$

$= 7$ millisecond

Turn around time (TAT) : WT + BT

$P_1 = 3+6 = 9$

$P_2 = 16+8 = 24$

$P_3 = 9+7 = 16$

$P_4 = 0+3 = 3$

$\therefore$ Avg TAT $= \frac{9+24+16+3}{4}$

$= 13$ ms

ii)

Process	Arrival-Time	Burst-Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Arr	B.T
P_2	4
P_3	9
P_4	5

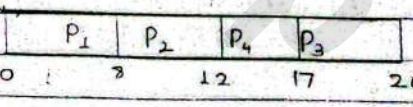
Soln;

Gantt Chart

In SJF, when arrival time is given take smallest of arrival first

∴ In such cases, we look at arrival time first.

Now,



Waiting-time (Start - Arrival)

$P_1 = 0 - 0 = 0$

$P_2 = 8 - 1 = 7$

$P_3 = 17 - 2 = 15$

$P_4 = 12 - 3 = 9$

$\therefore$ Avg waiting time $= \frac{0 + 7 + 15 + 9}{4}$

$= 7.75$ ms

Turn-around Time (TAT) : WT + BT

$P_1 = 0 + 8 = 8$

$P_2 = 7 + 4 = 11$

$P_3 = 15 + 9 = 24$

$P_4 = 9 + 5 = 14$

$\therefore$ Avg TAT $= \frac{8 + 11 + 24 + 14}{4}$

$= 14.25$ ms

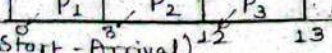
Q. Using FCFS and SJF, solve :

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

Solution →

i) Using FCFS

Gantt Chart,



Waiting-time (Start - Arrival)

$P_1 = 0 - 0 = 0$

$P_2 = 8 - 0.4 = 7.6$

$P_3 = 12 - 1.0 = 13$

$\therefore$ Avg waiting time $= \frac{0 + 7.6 + 13}{3}$

$= 6.87$ ms

Turn-around time (TAT) : W.T + B.T

$P_1 = 0 + 8 = 8$

$P_2 = 7.6 + 4 = 11.6$

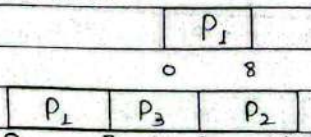
$P_3 = 13 + 1 = 14$

$\therefore$ Avg TAT $= \frac{8 + 11.6 + 14}{3}$

$= 11.2$ ms

ii) Using SJF

→ Gantt Chart



At 8,
P_2 4
P_3 1

Waiting time

$P_1 = 0 - 0 = 0$

$P_2 = 9 - 0.4 = 8.6$

$P_3 = 8 - 1.0 = 7$

$\therefore$ Avg WT $= \frac{0 + 8.6 + 7}{3} = 5.2$ ms

Turn around time (TAT)

$P_1 = 0 + 8 = 8$

$P_2 = 8.6 + 4 = 12.6$

$P_3 = 7 + 1 = 8$

$\therefore$ Avg TAT $= \frac{8 + 12.6 + 8}{3}$

$= 9.53$ ms

Pre-emptive

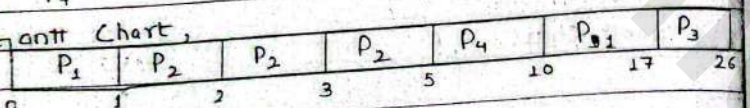
3) SJF (~~Primitive~~) | SRTF / SRTN / STRF / STRN

- SRTF = Shortest - Remaining - Time - First
- SRTN = Shortest - Remaining - Time - Next
- STRF = Shortest - Time - Remaining - First
- STRN = Shortest - Time - Remaining - Next

Example:

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

Gantt Chart,



At 1	B.T
P_1	8-1=7
P_2	4

At 2	B.T
P_1	7
P_2	4-1=3
P_3	9

At 3	B.T
P_1	7
P_2	4-2=2
P_3	9
P_4	5

Waiting-Time :

$P_1 = (0-0) + (10-1) = 9$ ms

$P_2 = (1-1) = 0$ ms

$P_3 = 17-2 = 15$ ms

$P_4 = 5-3 = 2$ ms

$\therefore$ Avg W.T $= \frac{9+0+15+2}{4}$

$= 6.5$ ms.

Turn-Around-Time = (W.T + B.T) :

$P_1 = 9+8 = 17$ ms

$P_2 = 0+4 = 4$ ms

$P_3 = 15+9 = 24$ ms

$P_4 = 2+5 = 7$ ms

$\therefore$ Avg TAT $= \frac{17+4+24+7}{4}$

$= 13$ ms

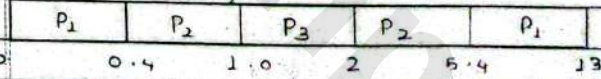
Qn

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

Calculate Avg W.T and Avg TAT using SJF (Primitive)

Soln;

Gantt Chart,



At 0.4	B.T
P_1	8-0.4 = 7.6
P_2	4

At 1.0 B.T

P_1 7.6

P_2 4-0.6 = 3.4

P_3 1

Waiting-Time :

P_1 : (0-0) + (5.4 - 0.4) = 5

P_2 : (0.4 - 0.4) + (2 - 1) = 1

P_3 : (1 - 1) = 0

Avg W.T = $\frac{5+1+0}{3}$ = 2 ms

Turn Around Time (TAT) = WT + BT

P_1 = 5 + 8 = 13

P_2 = 1 + 4 = 5

P_3 = 0 + 1 = 1

∴ Avg TAT = $\frac{13+5+1}{3}$

= 6.33 ms

Ritika Karanjit
171483

4) Priority Scheduling

↳ A priority is associated with each process and the CPU is allocated to the process with highest priority. Equal-priority process are scheduled in FCFS order. It can be both ~~prim~~ preemptive and non-preemptive.

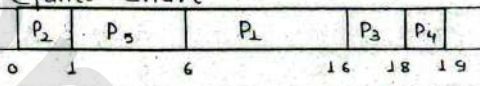
Example :

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

Let us assume 1 as highest priority & so on :

Priority (Non-preemptive)

Gantt Chart



W.T

P_1 = 6 - 0 = 6

P_2 = 0 - 0 = 0

P_3 = 16 - 0 = 16

P_4 = 18 - 0 = 18

P_5 = 1 - 0 = 1

Avg W.T = $\frac{6+0+16+18+1}{5}$

= 8.2

TAT

$P_1 = 6 + 10 = 16$

$P_2 = 0 + 1 = 1$

$P_3 = 16 + 2 = 18$

$P_4 = 18 + 1 = 19$

$P_5 = 1 + 5 = 6$

Avg TAT = $\frac{16+1+18+19+6}{5}$

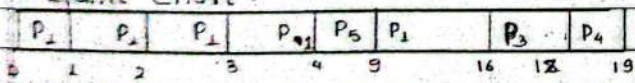
$= 12$

Eg 2

Process	Burst Time	Priority	Arrival Time
P_1	10	3	0
P_2	1	1	1
P_3	2	4	2
P_4	1	5	3
P_5	5	2	4

Priority (non - preemptive) :

Gantt Chart :



At 1	Priority
P_1	3
P_2	1

At 2	Priority
P_1	3
P_3	4

At 3	Priority
P_1	3
P_3	4
P_4	5

At 4	Priority
P_1	3
P_3	4
P_4	5

W.T

$P_1 = (10-0) + (2-1) + (9-4) = 0 + 1 + 5 = 6ms$

$P_2 = (1-1) = 0ms$

$P_3 = 16 - 2 = 14ms$

$P_4 = 18 - 3 = 15ms$

$P_5 = 4 - 4 = 0ms$

Avg W.T = $\frac{6+0+14+15+0}{5}$

$= 7ms$

TAT (WT + BT)

$P_1 = 6 + 10 = 16$

$P_2 = 0 + 1 = 1$

$P_3 = 14 + 2 = 16$

$P_4 = 15 + 1 = 16$

$P_5 = 0 + 5 = 5$

Avg TAT = $\frac{16+1+16+16+5}{5}$

$= 10.8ms$

5) Round-Robin Scheduling (RR)

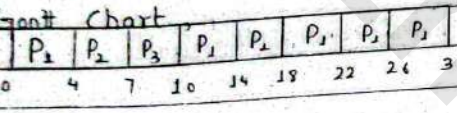
↳ The RR Scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling but pre-emption is added to the switch between processes. A small unit of time is called a time quantum or time slice is defined. The ready queue is treated as a circular queue. The CPU scheduler goes around the ready queue, also called allocating the CPU to each process for a time

interval of upto 1 time quantum.

Eg

Process	Burst Time	Time quantum = 4ms
P_1	24	
P_2	3	
P_3	3	

Gantt Chart



Waiting Time,

$P_1 = (0-0) + (10-4) = 6ms$

$P_2 = 4-0 = 4ms$

$P_3 = 7-0 = 7ms$

$\therefore$ Avg WT $= \frac{6+4+7}{3}$

$= 5.66ms$

TAT = B.T + W.T

$P_1 = 6+24 = 30ms$

$P_2 = 4+3 = 7ms$

$P_3 = 7+3 = 10ms$

$\therefore$ Avg TAT $= \frac{30+7+10}{3}$

$= 15.66ms$

Example:

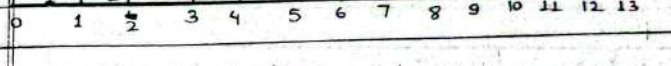
Process	Arrival Time	Burst Time
P_1	0	8
P_2	0.4	4
P_3	1	1

Time Quantum = 1

Ready Queue

P_1

Gantt Chart,



Waiting Time,

$P_1 = (0-0) + (3-1) + (5-4) + (7-6) + (9-8)$

$= 5ms$

$P_2 = (1-0.4) + (4-2) + (6-5) + (8-7)$

$= 4.6ms$

$P_3 = 2-1$

$= 1ms$

$\therefore$ Avg W.T $= \frac{5+4.6+1}{3}$

$= 3.53ms$

TAT = B.T + W.T

$P_1 = 8+5 = 13ms$

$P_2 = 4+4.6 = 8.6ms$

$P_3 = 1+1 = 2ms$

$\therefore$ Avg TAT $= \frac{13+8.6+2}{3}$

$= 7.86ms$

6) Highest-Response-Ratio-Next (HRRN / HRN)

↳ It is non-preemptive in nature. With this scheme we calculate response time and the process with highest response time is scheduled first.

$$\text{Response Time} = \frac{\text{Waiting Time} + \text{Burst Time}}{\text{Burst Time}}$$

Example:

Process	Arrival Time	Service Time (Burst Time)
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Gantt Chart,

A	B	C	E	D
0	3	9	13	15	20

At 3,

B

At 9,

C ~~; $R_T(C)$;~~

D

E

$$R_T(C) = \frac{(9-4)+4}{4} \qquad \frac{(9-A.T)+B.T}{B.T}$$

$$\therefore R_T(C) = 2.25$$

$$R_T(D) = \frac{(9-6)+5}{5}$$

$$\therefore R_T(D) = 1.6$$

$$R_T(E) = \frac{(9-8)+2}{2}$$

$$\therefore R_T(E) = 1.5$$

Since $R_T(E)$ is highest, we run C at 9

At 13, we have D and E

$$R_T(D) = \frac{(13-6)+5}{5}$$

$$\therefore R_T(D) = 2.21$$

$$R_T(E) = \frac{(13-8)+2}{2}$$

$$\therefore R_T(E) = 3.5$$

Since $R_T(E)$ is highest, we run E at 13.

At 15,

~~$R_T(E)$~~ = (15 − 8) → At 15, we run D.

Waiting Time:

A = 0 − 0 = 0ms

B = 3 − 2 = 1ms

C = 9 − 4 = 5ms

D = 15 − 6 = 9ms

E = 13 − 8 = 5ms

$$\text{Avg W.T} = \frac{0+1+5+9+5}{5} = 4\text{ms}$$

TAT : W.T + B.T

A = 3 + 0 = 3

B = 6 + 1 = 7

C = 4 + 5 = 9

D = 5 + 9 = 14

E = 2 + 5 = 7

$$\text{Avg TAT} = \frac{3+7+9+14+7}{5} = 8\text{ms}$$

Q. By Round Robin find WT & TAT.

Process	Arrival Time	Service Time (BT)
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

; Time quantum = 2ms

Soln.

Gantt Chart

A	B	A	C	B	D	C	E	B	D	D

0 2 4 5 7 9 11 13 15 17 19 20

RQ

A
B
A
C
B
D
C
E
B
D

Waiting Time:

A = (0-0) + (4-2) = 2

B = (2-2) + (7-4) + (15-9) = 9

C = (5-4) + (11-7) ~~+ (15-9)~~ = 5

D = (9-6) + (17-11) = 9

E = 13-8 = 5

Avg W.T = $\frac{2+9+5+9+5}{5}$

= 6ms

TAT = W.T + B.T

A = 3 + 2 = 5

B = 6 + 9 = 15

C = 4 + 5 = 9

D = 5 + 9 = 14

E = 2 + 5 = 7

Avg TAT = $\frac{5+15+9+14+7}{5}$

= 10ms

7) Multilevel Feedback Queue

Q. Consider a multilevel feedback queue scheduling (MLFBQ) with three queues q_1, q_2, and q_3. q_1 and q_2 use round-robin algorithm with time quantum (T_Q) = 5 and 4 respectively. q_3 uses FCFS algorithm. Find the average waiting time and average TAT for executing the process

Process	P_1	P_2	P_3	P_4
Burst Time	8	22	4	12

Soln;

	q_1	q_2	q_3
P_1	8	3	-
P_2	22	17	13
P_3	4	-	-
P_4	12	7	3

P_4 P_3 P_2 P_1

P_4 P_2 P_1

P_4 P_2

CPU utilization = $\frac{\text{Total B.T}}{\text{Total S.T}} \times 100$; Throughput = $\frac{\text{Total no. of proc}}{\text{Total S.T}}$

Gantt Chart (MLFBQ)

P_1	P_2	P_3	P_4	P_1	P_2	P_4	P_2	P_4

0 5 10 14 19 22 26 30 43 46

Waiting - Time

$P_1 = (0-0) + (19-5) = 14$

$P_2 = (5-0) + (22-10) + (30-26) = 21$

$P_3 = (10-0) = 10$

$P_4 = (14-0) + (26-19) + (43-30) = 34$

$\therefore$ Avg W.T = $\frac{14+21+10+34}{4}$

= 19.75 ms

TAT ~~= 14+8=22~~ (Total Time)

$P_1 = 14+8=22$

$P_2 = 21+22 = 43$

$P_3 = 10+4 = 14$

$P_4 = 34+12 = 46$

$\therefore$ Avg TAT = $\frac{22+43+14+46}{4}$

= 31.25 ms

Q. Find Avg WT & Avg TAT using SJF (Preemptive)

Process	A.T	W.T
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Soln: Gantt Chart

A	A	B	C	C	E	B	D

0 2 3 4 6 8 10 15 20

At 2	B.T
A	3-2=1
B	6

At 3	B.T
3	

At 4	B.T
B	6-1=5
C	4

At 6	B.T
B	5
C	4-2=2
D	5

At 8	B.T
B	5
D	5
E	2

W.T

A = (0-0) + (

B =

C =

D =

E =

2.4 Process Synchronization

Background
Producer and Consumer Problem
A producer process produces information that is consumed by a consumer process. For example,

Program
Two-Tasks Solution
Peterson's Algorithm
↳ Two process solution. It is much simpler way to achieve mutual exclusion.

```
#define FALSE 0
#define TRUE  1
#define N     2          /* no. of processes */

int turn;                /* whose turn is it? */
int interested[N];       /* all values initially 0 i.e False */
void enter_region(int process)   /* process is 0 or 1 */
{
    int other;           /* no. of other process */
    other = 1 - process;     /* opposite of process */
    interested[process] = TRUE;  /* show that you are interested */
    turn = process;          /* set flag */
    while (turn == process && interested[other] == TRUE)
                             /* null statement */
}
void leave_region(int process)   /* process who is leaving */
{
    interested[process] = FALSE;
}
```


* buffer : storing device
  - unbounded buffer: physical size is undefined
  - bounded buffer: physical size is defined

2.5 Deadlocks

Deadlock Avoidance
Eg: Consider the following snapshot of the system.

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

Answer the following using Banker's Algorithm
1. What is the content of the matrix need?
2. Is the system in safe state? Also find the safe sequence.
3. If the request from process P_1 arrives for (0,4,2,0) Can the request be granted immediately?

Answer;
1. Need Matrix (Max − Allocation)

	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

2. Available ≥ Need

Since available (1, 5, 2, 0) ≥ need of P_0 & P_3

So we can run either P_0 or P_3.

Run P_3, After completion it releases its allocated resource.

Run P_3

New Available = Available + Allocation of P_3

= (1, 5, 2, 0) + (0, 6, 3, 2)

= (1, 11, 5, 2)

Since New Available (1, 11, 5, 2) ≥ need of all remaining process. So, we can run any process in any order.

Run P_0:

After completion it releases its allocated resources.

∴ New Available = (1, 11, 5, 2) + (Allocation of P_0)

= (1, 11, 5, 2) + (0, 0, 1, 2)

= (1, 11, 6, 4)

Run P_1:

New Available = (1, 11, 6, 4) + (Allocation of P_1)

= (1, 11, 6, 4) + (1, 0, 0, 0)

= (2, 11, 6, 4)

Run P_2:

New Available = (2, 11, 6, 4) + (Allocation of P_2)

= (2, 11, 6, 4) + (1, 3, 5, 4)

= (3, 14, 11, 8)

Run P_4

New Available = (3, 14, 11, 8) + (Allocation of P_4)

= (3, 14, 11, 8) + (0, 0, 1, 4)

= (3, 14, 12, 12)

Since all processes are completed, so the system is in safe state. And one of the safe sequence is P_3, P_0, P_1, P_2, P_4.

3. Available ≥ request

Since available (1, 5, 2, 0) ≥ request (0, 4, 2, 0)

So, initially we can grant the request.

After granting request;

New Available = (Available) − (Request)

= (1, 5, 2, 0) − (0, 4, 2, 0)

= (1, 1, 0, 0)

And, allocation of P1 now becomes ~~(1, 0, 0)~~

= (1, 0, 0, 0) + (0, 4, 2, 0)

= (1, 4, 2, 0)

And need of P_1 = (0, 3, 3, 0)

Now,

Available (1, 1, 0, 0) ≥ Need of P_0 (0, 0, 0, 0)

So, we run P_0. After completion it releases its allocated resources.

New Available = (Available) + (Allocation of P_0)

= (1, 1, 0, 0) + (0, 0, 1, 2)

= (1, 1, 1, 2)

Since, Available (1, 1, 1, 2) ≥ Need of P_2

Run P_2;

New Available = (1, 1, 1, 2) + (Allocation of P_2)

= (1, 1, 1, 2) + (1, 3, 5, 4)

= (2, 4, 6, 6)

Since, Available (2, 4, 6, 6) ≥ need of P1 & P3.
We can run either P1 and P3.

Run P3:
New Available = (2, 4, 6, 6) + (Allocation of P3)
= (2, 4, 6, 6) + (0, 6, 3, 2)
= (2, 10, 9, 8)
Since, Available (2, 10, 9, 8) ≥ need of P1 & P4.
So we can run any process.

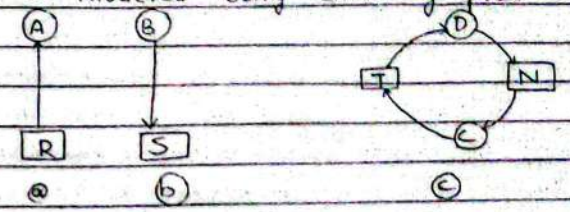
Run P1:
New Available = (2, 10, 9, 8) + (Allocation of P1)
= (2, 10, 9, 8) + (1, 4, 2, 0)
= (3, 14, 11, 8)

Run P4:
New Available = (3, 14, 11, 8) + (Allocation of P4)
= (3, 14, 11, 8) + (0, 0, 1, 4)
= (3, 14, 12, 12)

After granting the request for P1, the system is still in safe state. So, we can grant the request for P1 immediately. And one of the safe sequence after granting request is P_0, P_2, P_3, P_1, P_4.

Deadlock Modeling Using Resource Allocation Graph:

↳ Can be modeled using direct graphs.



@ ⓑ ©

Fig 1: Resource allocation graphs
@: Holding a Resource
ⓑ: Requesting a Resource
©: Deadlock

Now, let us see how resource graphs can be used. We have 3 processes A, B & C and 3 resources R, S & T. The requests & releases of three processes are given in fig 1 @ - ©. The operating system is free to run any unblocked process at any instant, so it could decide to run A until A finishes all its work. Then run B to completion and finally run C. This ordering doesnot lead to any deadlocks but it also has no parallelism at all.

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R

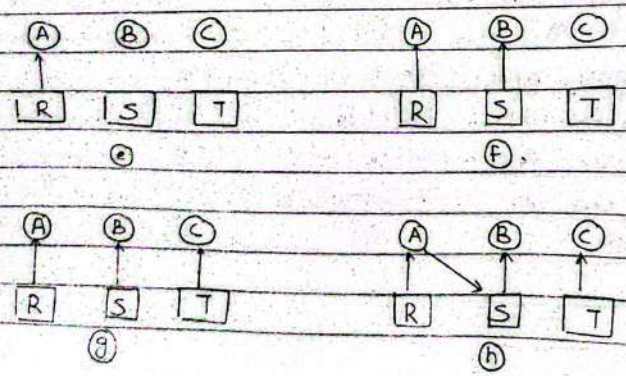
Fig 2) @ ⓑ ©

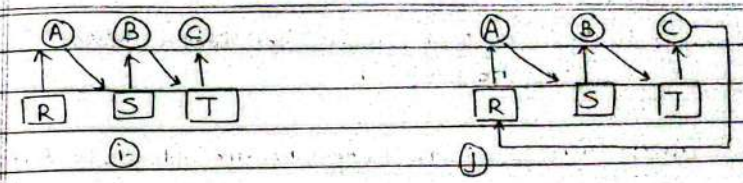
Let's suppose that the process does both I/O and computing. The resource requests might occur in order of fig 3.ⓓ If these six requests are carried out ~~request 4 has been made~~ in that order. The six resulting resource graphs are shown in fig 3 ⓔ-ⓙ. ~~After next two steps~~ After request 4 has been made, A blocks waiting for S, as shown in fig 3(h). In the next two steps B and C also block, ultimately leading to a cycle and the deadlock of fig 3(j). From this point on, the system is frozen.

1. A request R
2. B request S
3. C request T
4. A request S
5. B request T
6. C request R

Deadlock

Fig 3: ⓓ





In figure ⓓ, If OS knew about the impending deadlock, it could suspend B instead of granting it S. By running A & C, we would get the requests and release fig ⓚ, instead of fig 2.ⓓ. This sequence leads to the resource graphs of fig 1 ⓛ-ⓠ, which doesnot lead to deadLock.

1. A request R
2. C request T
3. A request S
4. C request R
5. A released R
6. A released S

No deadlock

ⓚ

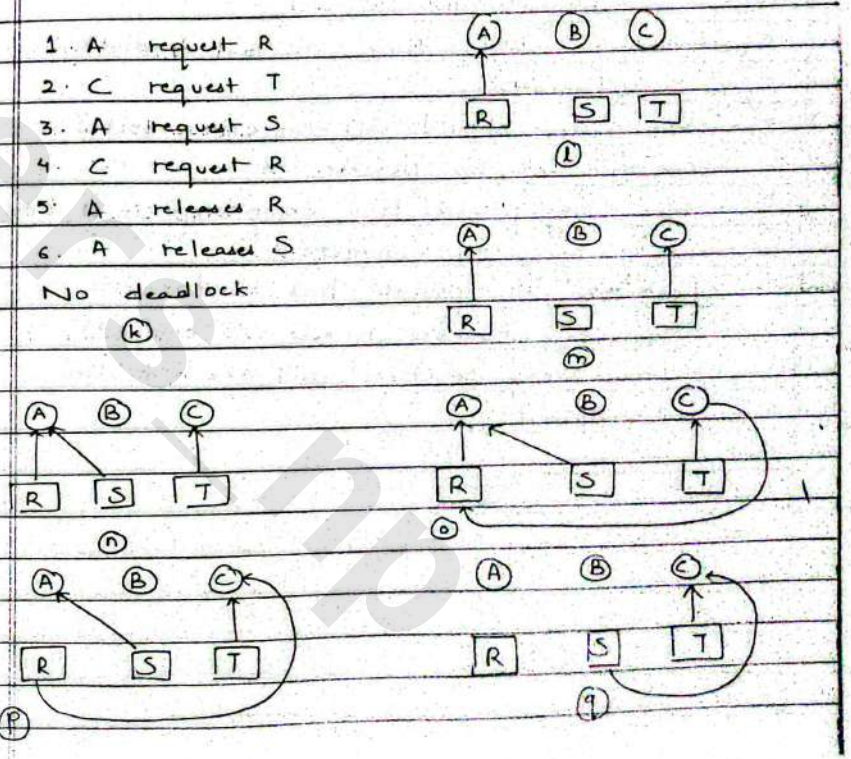


Fig 3: An example of how deadlock occurs and how it can be avoided.

After step (q), process can be granted S because A is finished and C has everything it needs. Even if B should eventually block when requesting T_1, no deadlock can occur. B will just wait until C is finished.

Recovery from Deadlock:

→ Process Termination

- Abort all deadlock processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - priority of the process
  - How long process has completed, and how much longer to complete
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.

→ Resource Preemption:

Page:
Date: / /

Assignment - 1

9.

Process ID	AT (ms)	CPU BT (ms)
A	0	6
B	5	5
C	6	7
D	8	4
E	12	3

→

STRF:

A	A	B	B	D	D	E	C

0 5 6 8 11 12 15 18 25

At 5,	BT
A	6-5=1
B	5

At	6	BT
B		5
C		7

At 8	BT
B	5-2=3
C	7
D	4

At 11	BT
C	7
D	4

At 12	BT
C	7
D	4-1=3
E	3

Since BT of D and E are same, apply FCFS to break the tie. ∴ We run D.

Page:
Date:

RR (Time Slice = 3 ms):

A	A	B	C	D	B	E	C	D	C

0 3 6 9 12 15 17 20 23 24 25

RQ	WT (Start - Arrival)
A	A = 0 - 0 = 0
A	B = (6 - 5) + (15 - 9)
B	= 1 + 6 = 7 ms
C	C = (9 - 6) + (20 - 12) + (24 - 23)
D	= 12 ms
B	D = (12 - 8) + (23 - 15) = 12 ms
E	E = (17 - 12) = 5 ms
C	
D	
C	

Avg W.T = 35/5 = 7 ms

TAT = WT + BT

A = 0 + 6 = 6 ms

B = 7 + 5 = 12 ms

C = 12 + 7 = 19 ms

D = 12 + 4 = 16 ms

E = 5 + 3 = 8 ms

Avg TAT = 61/5 = 12.2 ms

Chapter 3: Memory Management

Placement Strategies

- First Fit
- Best Fit
- Worst Fit
- Next Fit

Example: Given five memory partitions of 100KB, 500KB, 200KB, 300KB and 600KB (in order). How would each of the first-fit, best-fit, worst-fit and next-fit algorithms place processes of 212 KB, 417KB, 112KB & 426KB (in order)? Which algorithm makes the most efficient use of memory?

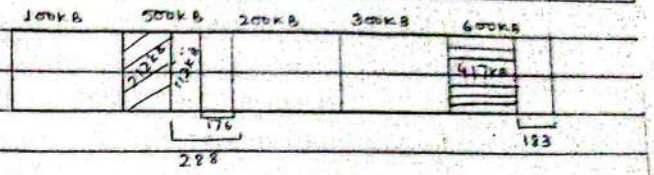
Soln;

Given memory partitions: 100KB, 500KB, 200KB, 300KB, 600KB

Process Size: 212KB, 417KB, 112KB, 426KB

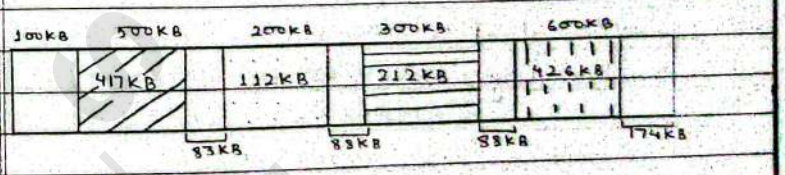
* First-fit:

- 212KB, is placed in 500KB
  - Remaining Size = 500 - 212
  - = 288 KB
- 417KB is placed in 600KB
  - Remaining Size = 600 - 417
  - = 183KB
- 112KB is placed in 288KB
  - Remaining Size = 288 - 112
  - = 176KB
- 426KB must wait



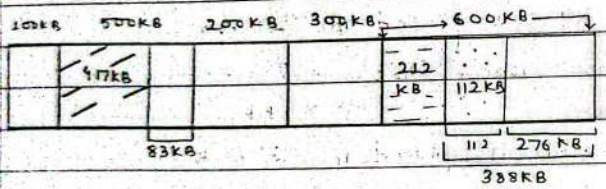
* Best-Fit:

- 212KB is placed in 300KB
  - Remaining Size = 300 - 212 = 88KB
- 417KB is placed in 500KB
  - Remaining Size = 500 - 417
  - = 83KB
- 112KB is placed in 200KB
  - Remaining Size = 200 - 112
  - = 88KB
- 426KB is placed in 600KB
  - Remaining Size = 600 - 426
  - = 174KB

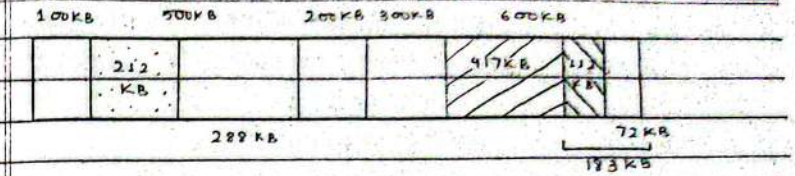


PS 212, 417, 112, 426
MS 100, 500, 200, 300, 600

* Worst-Fit
- 212 KB is placed in 600KB
  Remaining Size = 600 - 212
  = 388KB
- 417KB is placed in 500 KB
  Remaining Size = 500 - 417
  = 83 KB
- 112KB is placed in 388KB
  Remaining Size = 388 - 112
  = 276 KB
- 426 KB must wait.



* Next-Fit : {Fastest Search}
- 212KB is placed in 500KB
  Remaining Size = 500 - 212
  = 288 KB
- 417KB is placed in 600KB
  Remaining Size = 600 - 417
  = 183 KB
- 112KB is placed in 183KB
  Remaining Size = 183 - 112
  = 71KB
- 426 must wait



Best-fit is most efficient because all processes were allocated.

2017 Fall

6@ Given six memory partitions of 300KB, 600KB, 350KB, 200KB, 750KB and 125KB (in order). How would the first-fit, best-fit and worst-fit algorithms place processes of size 115KB, 500KB, 358KB, 200KB and 375KB (in order)? Rank the algorithms in terms of how efficiently they use memory.

Soln;

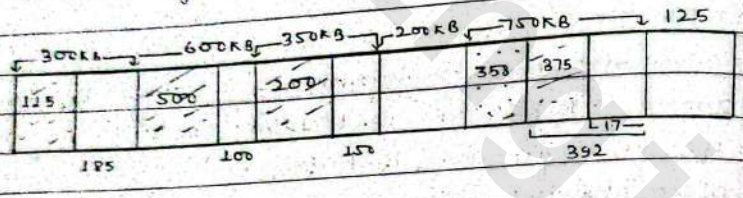
Given memory partitions : 300KB, 600KB, 350KB, 200KB, 750KB, 125KB

Process Size : 115KB, 500KB, 358KB, 200KB, 375KB

* First-fit :
- 115KB is placed in 300KB
  Remaining Size = 300 - 115
  = 185 KB
- 500KB is placed in 600KB
  Remaining Size = 600 - 500
  = 100KB
- 358 KB is placed in 750 KB
  Remaining Size = 750 - 358
  = 392 KB

• 200KB is placed in ~~392KB~~ 350KB
Remaining Size = 350 - 200
= 150KB

• 375KB is placed in 392KB
Remaining Size = 17KB



• Best - Fit

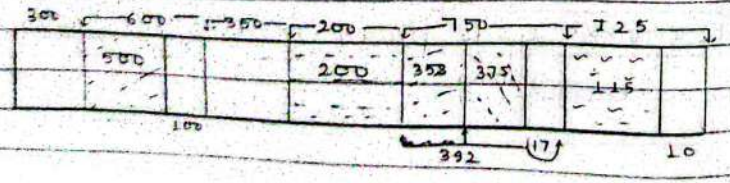
• 115KB is placed in 125KB
Remaining Size = 125 - 115
= 10KB

• 500KB is placed in 600KB
Remaining Size = 600 - 500
= 100KB

• 358KB is placed in 750KB
Remaining Size = 750 - 358
= 392KB

• 200KB is placed in 200KB

• 375KB is placed in 392KB
Remaining Size = 392 - 375
= 17KB



• Worst - Fit

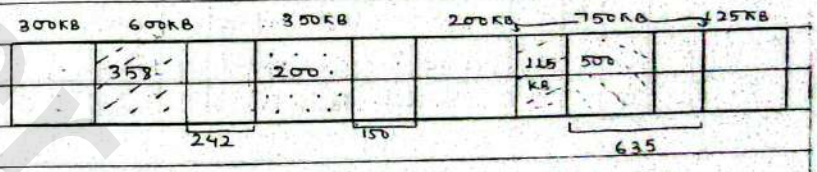
• 115KB is placed in 750KB
Remaining Size = 750 - 115KB
= 635KB

• 500KB is placed in 635KB
Remaining Size = 635 - 500
= 135KB

• 358KB is placed in 600KB
Remaining Size = 600 - 358
= 242KB

• 200KB is placed in 350KB
Remaining Size = 350 - 200
= 150KB

• 375KB must wait.



∴ The ranking in terms of efficiency is First - Fit, Best - Fit and Worst - Fit.

Page:
Date:

In A:

Page Fault
Page Replacement Algorithm

total hit = 5
total miss = 15

Q. Reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

→ Here,
Page replacement algorithm
1. FIFO (First In First out)

A. No. of frame = 3

7	0	1	2	0	3	0	4
7	7	7	2	No	2	2	4
	0	0	0	page	3	3	3
		1	1	fault	1	0	0

2	3	0	3	2	1	2
4	4	0	No	No	0	0
2	2	2	page	page	1	1
0	3	3	fault	fault	3	2

0	1	7	0	1
No	No	7	7	7
page	page	1	0	0
fault	fault	2	2	1

Total number of page fault = 15
Fault Ratio = Total no. of page fault / Total no. of reference string
= 15/20
= 0.75
hit ratio = total no. of hit / total ref string = 5/20 =

B. → No. of frame = 4

7	0	1	2	0	3	0	4	2
7	7	7	7	No	3	No	3	No
	0	0	0	page	0	page	4	page
		1	1	fault	1	fault	1	fault
			2		2		2	

3	0	3	2	1	2	0	1
No	3	No	No	3	2	No	No
page	4	page	page	4	4	page	page
fault	0	fault	fault	0	0	fault	fault
	2			1	1		

7	0	1
3	No	No
7	page	page
0	fault	fault
1		

Total number of page fault = 10
Fault Ratio = Total no. of page fault / Total no. of reference string
= 10/20
= 0.5

FIFO Anomolies:

Generally, if we increase no. of frame, the page fault decreases but in some rare cases the page fault increases

FIFO Anamolies (Belady's Anamolies)

→ Certain page reference pattern increases no. of page fault if no. of page size is increased.

Reference String:

A B C D A B E A B C D E

→ Frame Size = 3

A	B	C	D	A	B	E	A	B
A	A	A	D	D	D	E	No page fault	No page fault
	B	B	B	A	A	A		
		C	C	C	B	B		
✓	✓	✓	✓	✓	✓	✓	x	x

C	D	E
E	E	no page fault
C	C	
B	D	
✓	✓	x

✓ → Page Fault

x → No page fault

Total no. of page fault = 9

Fault Ratio = $\frac{9}{12}$

= 0.75

→ Frame Size = 4

A	B	C	D	A	B	E	A
A	A	A	A	No page fault	No page fault	E	E
	B	B	B			B	A
		C	C			C	C
			D			D	D

B	C	D	E
E	E	D	D
A	A	A	E
B	B	B	B
D	C	C	C

Total no. of page fault = 10

Fault Ratio = $\frac{10}{12}$

= 0.83

2. Least Recently Used (LRU)

Check left side

Reference String:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	0	1	2	0	3	0	4
7	7	7	2	No	2	No	4
	0	0	0	page	0	page	0
		1	1	fault	3	fault	3

2	3	0	3	2	1	2
4	4	0	No	No	1	No
0	3	3	page	page	3	page
2	2	2	fault	fault	2	fault

0	1	7	0	1
1	No	1	No	No
0	page	0	page	page
2	fault	7	fault	fault

∴ Total no. of page fault = 12

Fault Ratio = $\frac{12}{20}$

= 0.6

Page Replacement Algorithm (Contd..)

3 Optimal Page Replacement : Check right side

Eg: Reference String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Frame Size = 3. Initially all frame are empty

7	0	1	2	0	3	0	4	2	3
7	7	7	2	No	2	No	2	No	No
	0	0	0	page	0	page	4	page	page
	1	1	1	fault	3	fault	3	fault	fault

0	3	2	1	2	0	1	7	0	1
2	No	No	2	No	No	No	7	No	No
0	page	page	0	page	page	page	0	page	page
3	fault	fault	1	fault	fault	fault	1	fault	fault

Total no. of page fault = 9

∴ fault rate / ratio = $\frac{9}{20}$ = 0.45

4 Most Frequently Used (MFU)

Eg: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	0	1	2	0	3	0
7	7	7	2	No	2	2
	0	0	0	page	3	3
		1	1	fault	1	0

R.S	Counter
7	1 2
0	1 2 3
1	1 2 3
2	1 2 3 4
3	1 2 3
4	1

4	2	3	0	3	2	1	2	0	1
2	No	No	0	No	2	2	No	0	No
3	page	page	3	page	3	1	page	1	page
4	fault	fault	4	fault	4	4	fault	4	fault

7	0	1
7	7	7
1	0	1
4	4	4

Total no. of page = 14

∴ Fault rate / ratio = $\frac{14}{20} = 0.7$

5. Least ~~used~~ Frequently Used (LFU)

Eg: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	0	1	2	0	3	0	4
7	7	7	2	No	2	No	4
	0	0	0	page	0	page	0
		1	1	fault	3	fault	3

R.S	Counter
7	✗ 2
0	✗ 2 ✗ 4 ✗ 6
1	✗ 2 ✗ 4
2	✗ 2 ✗ 4
3	✗ 2 3
4	1

2	3	0	3	2	1	2	0
4	3	No	No	No	3	3	No
0	0	page	page	page	0	0	page
2	2	fault	fault	fault	1	2	fault

1	7	0	1
1	7	No	1
0	0	page fault	0

Total no. of page fault = 13

∴ Fault rate / ratio = $\frac{13}{20} = 0.65$

2019 Spring

4@ How many page faults occur for the following reference strings for 3 page frame:
2, 3, 4, 5, 4, 2, 5, 3, 6, 3, 2, 3, 4, 5. Using MFU, LRU and Optimal page replacement algorithm.

Soln; Frame Size = 3

Initially all frames are empty.

MFU;

2	3	4	5	4	2	5	3	6	4
2	2	2	5	No	5	No	No	6	5
	3	3	3	page	3	page	page	2	6
		4	4	fault	2	fault	fault	3	

RS	Counter
2	✗ 2 ✗ 3
3	✗ 2 ✗ 3 ✗ 4
4	✗ 2 3
5	✗ ✗ 2 3
6	1

3	2	3	5	4	5
No	No	No	6	6	
page	page	page	4	4	
fault	fault	fault	3	5	

Total page fault = 9

Fault Ratio = $\frac{9}{14} = 0.64$

LRU;

2	3	4	5	4	2	5	3	6
2	2	2	5	No	5	No	No	5
	3	3	3	page	3	page	page	3
		4	4	fault	2	fault	fault	6

3	2	3	4	5
No	5	5	5	No
page	2	2	2	page
fault	6	3	4	fault

Total page fault = 9

Fault / ratio = $\frac{9}{14}$ = 0.64

FIFO :

2	3	4	5	4	2	5	3	6	3
2	2	2	5	No	5	No	5	6	No
	3	3	3	page	2	page	2	2	page
		4	4	fault	4	fault	3	3	fault

2	3	4	5
No	No	6	6
page	page	4	4
fault	fault	3	5

Total page fault = 9

Fault / ratio = 9/14 = 0.64

Optimal Page Replacement :

2	3	4	5	4	2	5	3	6
2	2	2	2	No	No	No	2	2
	3	3	5	page	page	page	3	3
		4	4	fault	fault	fault	4	6

3	2	3	4	5
No	No	No	4	4
page	page	page	3	5
fault	fault	fault	6	6

Total page fault = 8

Fault / ratio = $\frac{8}{14}$ = 0.57

6 Second Chance Page Replacement Algorithm (SCA)

Eg 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 7, 0, 1

Frame Size = 3, Initially are frame are empty

R.B	7	0	1	2	0	3	0
0	7	7	7	2	2	2	2
	0	0	0	0	0	0	0
			1	1	1	3	3
	✓	✓	✓	✓	✗	✓	✗

✓ - Page Fault

✗ - No page fault

4	2	3	0	3	2	1	2
4	4	4	0	0	0	1	1
0	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
✓	✓	✗	✓	✗	✗	✓	✗

0	1	7	0	1
1	1	1	1	1
2	2	2	0	0
0	0	0	7	7
✓	✗	✓	✓	✗

Total Page fault = 12

∴ Fault Rate / ratio = $\frac{12}{20}$

= 0.6

Total no. of hit = 8

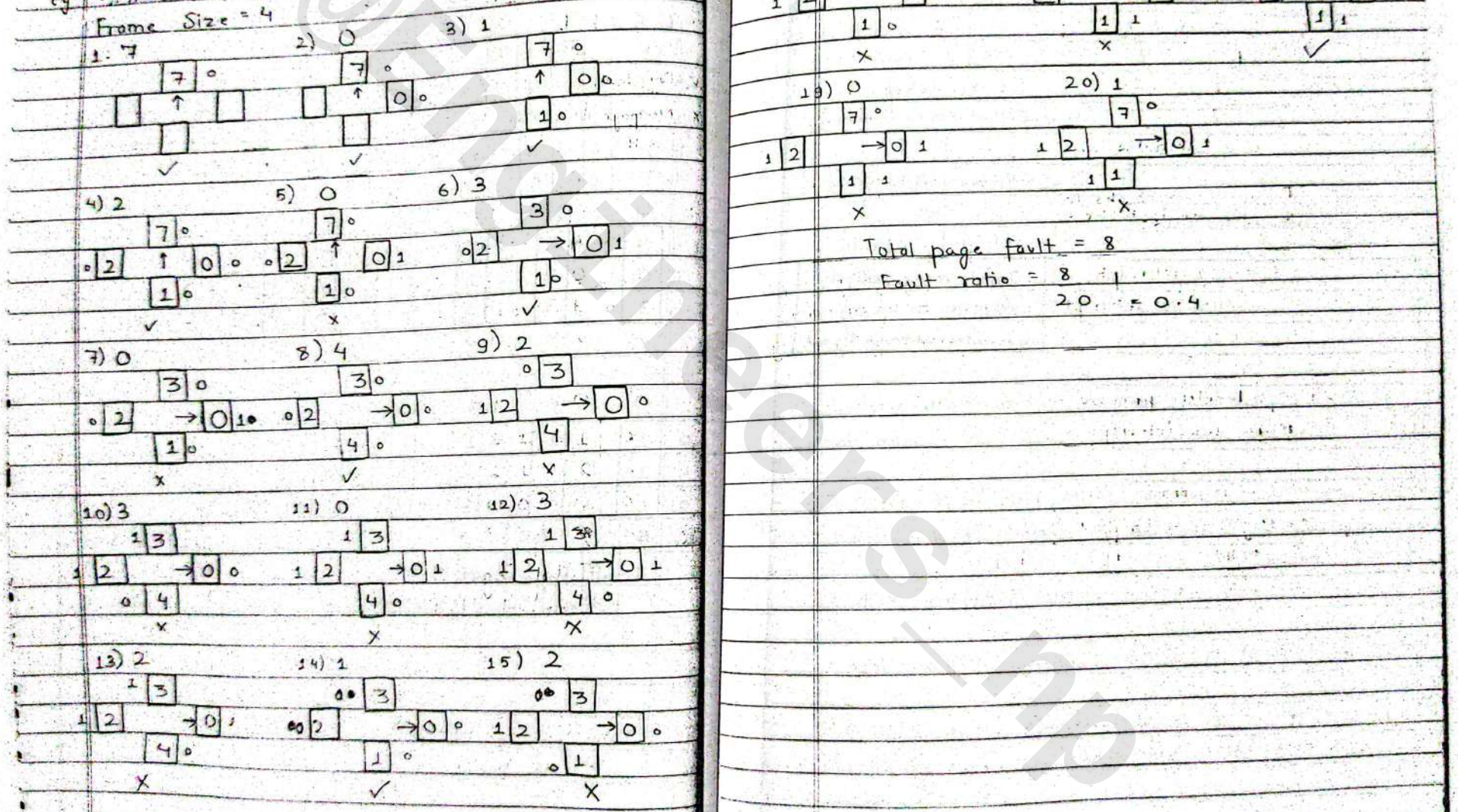
∴ Hit ratio = $\frac{8}{20}$

= 0.4

7. Clock Page Replacement

Eg: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Frame Size = 4

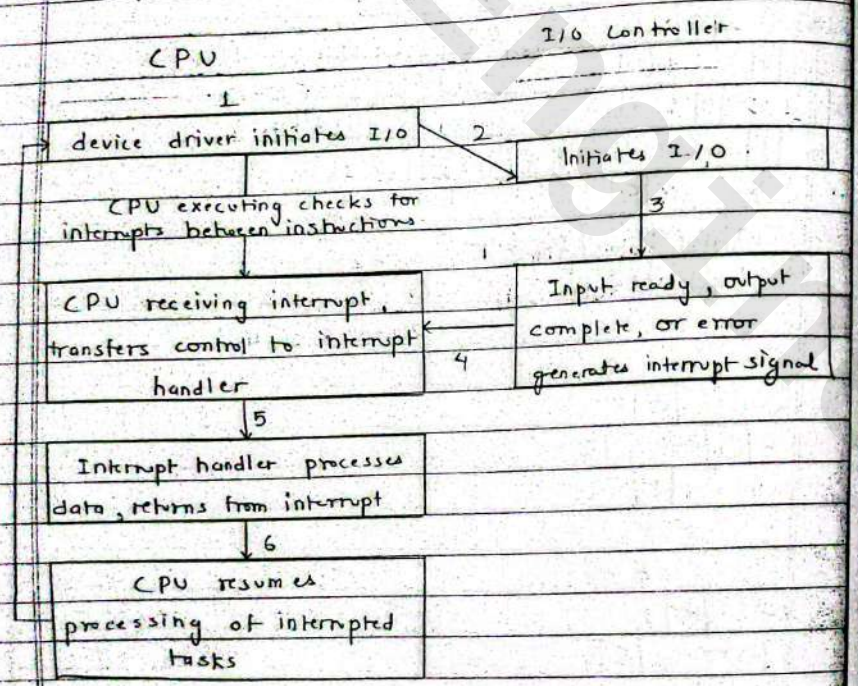


Total page fault = 8

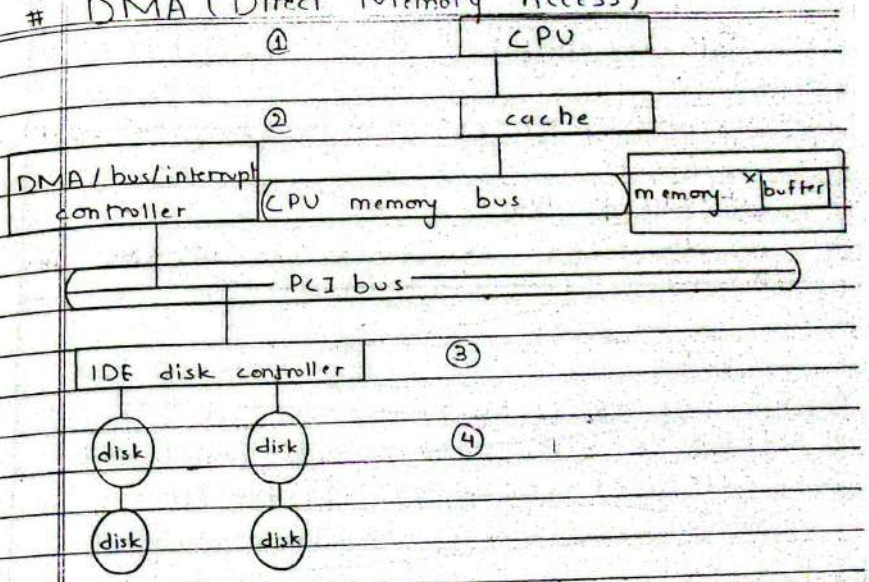
Fault ratio = 8 / 20 = 0.4

Chapter 4: Input / Output Management

Interrupt - Driven I/O Cycle



DMA (Direct Memory Access)



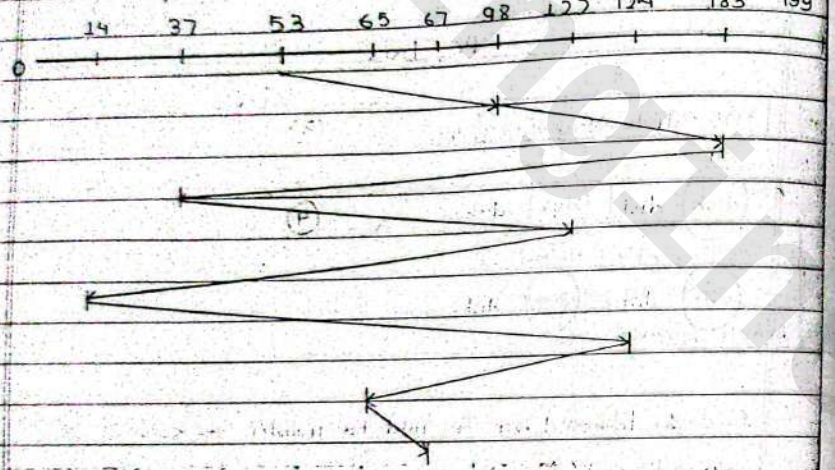
1. device driver is told to transfer disk data to buffer at address X.
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X.
3. disk controller initiates DMA transfer.
4. disk controller sends each byte to DMA controller.
5. DMA controller transfers bytes to buffer X, increasing memory addressing and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion

Disk Scheduling Algorithm

1. FCFS

Required Queue: 98, 183, 37, 122, 14, 124, 65, 67

Header pointer: 53



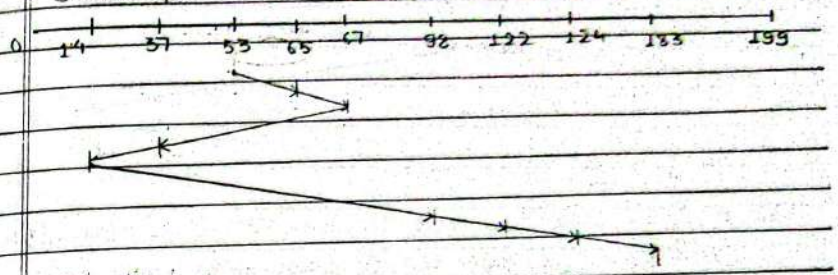
Seek Order: 53, 98, 183, 37, 122, 14, 124, 65, 67

Total Seek distance = |53-98| + |98-183| + |183-37| + |37-122| + |122-14| + |14-124| + |65-67|

= 640 cylinder.

2. SSTF (Shortest Seek Time First)

Same qn:



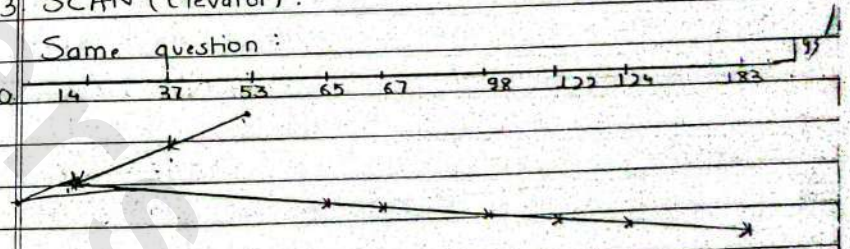
Seek order: 53, 65, 67, 37, 14, 98, 122, 124, 183

Seek distance = |53-65| + |65-67| + |67-37| + |37-14| + |14-98| + |98-122| + |122-124| + |124-183|

∴ Cylinder moved = 236 cylinder.

3. SCAN (Elevator):

Same question:



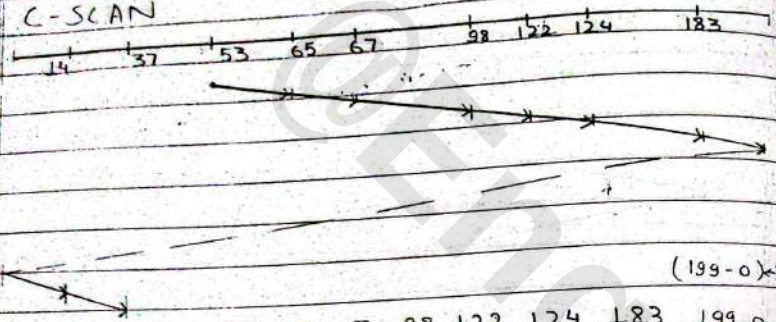
Seek order: 53, 37, 14, 65, 67, 98, 122, 124, 183

Total Seek distance: |53-37| + |37-14| + |14-65| + |65-67| + |67-98| + |98-122| + |122-124| + |124-183|

|14-0| + |0-65|

upward - right ; outward - right
downward - left ; inward - left

4. C-SCAN



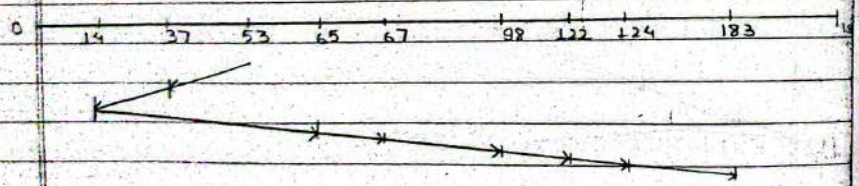
(199-0)×X

Seek-order: 53, 65, 67, 98, 122, 124, 183, 199, 0, 14, 37

Total seek distance: |53-65| + |65-67| + |67-98| + |98-122| + |122-124| + |124-183| + |183-199| + ~~|199-0|~~ + |0-14| + |14-37|

= 183 cylinder

5. LOOK

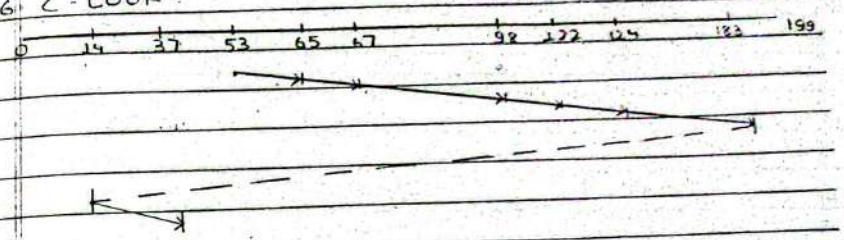


Seek-order: 53, 37, 14, 65, 67, 98, 122, 124, 183

Total-Seek Distance: |53-37| + |37-14| + |14-65| + |65-67| + |67-98| + ~~|122-98|~~ + |98-122| + |122-124| + |124-183|

= 208 Cyclinder

6. C-LOOK:



Seek-order: 53, 65, 67, 98, 122, 124, 183, 14, 37

Total-Seek distance: |53-65| + |65-67| + |67-98| + |98-122| + |122-124| + |124-183| + |14-37|

= 153 Cylinder

2019 Spring

5 a) Disk request come to the disk drive for cylinder 16, 18, 12, 6, 25, 38, 7 and 36 in that order A seek takes 2 micro sec per cylinder move. How much seek time is needed for

i) FCFS ii) Closest Cylinder Next

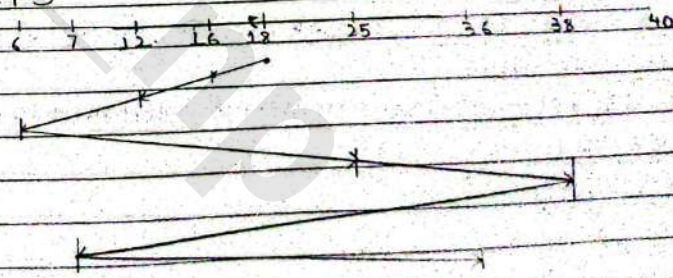
iii) C-SCAN ~~iv)~~ (initially moving upward)

iv). SCAN (initially moving downward)

Initially cylinder is at 18.

Answer;

i) FCFS



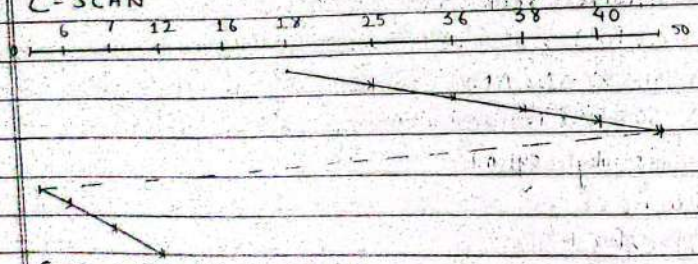
ii) Closet Cylinder Next

Seek order : 18, 16, 12, 7, 6, 25, 36, 38, 40

Total Seek Distance : |18-16| + |16-12| + |12-7| + |7-6| + |6-25| + |25-36| + |36-38| + |38-40|

= 46

iii) C-SCAN



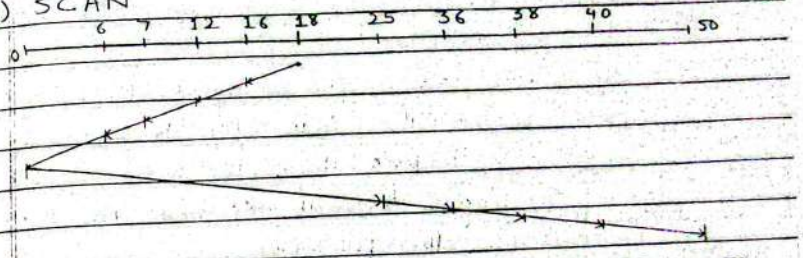
Seek order : 18, 25, 36, 38, 40, 50, 0, 6, 7, 12

total seek distance :

|18-25| + |25-36| + |36-38| + |38-40| + |40-50| + |0-6| + |6-7| + |7-12|

= 44 cylinder

iv) SCAN



Seek order : 18, 16, 12, 7, 6, 0, 25, 36, 38, 40, 50

Total Seek Distance : |18-16| + |16-12| + |12-7| + |7-6| + |6-0| + |0-25| + |25-36| + |36-38| + |38-40| + |40-50|

= 68

Interleaving

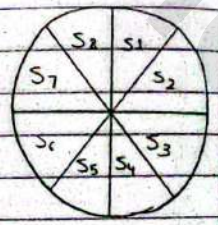
Example: A disk has 8 sectors per track and spins at 600 rpm. It takes the controller 10ms from the end of one I/O operation before it can issue a subsequent one. How long does it take to read all 8 sectors using the following interleaving systems?

ⓐ No interleaving
ⓑ Single interleaving
ⓒ Double interleaving

Soln;

ⓐ When there is no interleaving then sectors would be like this

$S1 > S2 > S3 > S4 > S5 > S6 > S7 > S8$



We know,
In 60 sec disk revolve 600 rotations [600 RPM]

∴ 1 sec disk revolve $\frac{600}{60} = 10$ rotation.

For 10 rotations it requires 1000 ms | 1 sec = 1000 ms

∴ For 1 rotation, it requires $\frac{1000}{10} = 100$ ms.

∴ Total time spend on each sector $= \frac{100}{8}$

$= 12.5$ ms

& the time it takes to load another request is 10ms. That is less than 12.5ms.

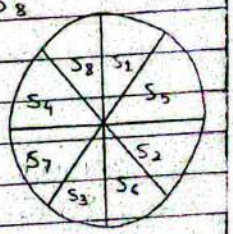
In first request it read sector S1, then while the disk is spinning the controller went for to fetch another request. And 10ms are wasted out of 12.5 ms which means more than half of S2 had already passed. Therefore, heads to rotate again to fetch S2. So, because of all 8 sectors meaning it will take 8 revolutions

∴ Total time = 8 × 100
= 800 msec.

ⓑ When there is single interleaving then sectors will be arranged in order,

$S_1 > S_5 > S_2 > S_6 > S_3 > S_7 > S_4 > S_8$

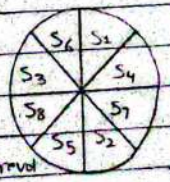
∴ In 1 revolution 4 sectors will be read and,
In 2 revolution 4×2 = 8 sectors will be read.



For 1 rotation it takes 100 ms
∴ For 2 rotation = 100×2 = 200 ms

ⓒ When double interleaving,

$S_1 > S_4 > S_7 > S_2 > S_5 > S_8 > S_3 > S_6$



∴ Only 3 sectors will be read in 1 revol
But since 8th sector is read at 3/4 of rev disk,
∴ Total revolution required = 2 + 3/4 of rotation
For 1 rotation it requires 100 ms
∴ Total time = 2×100 + 3/4×100
= 275 ms

Reference bit = R

Assignment

Q8) A computer has four page frames. The time of loading, time of last access, and the R and M bits for each page are as shown (the times are in clock ticks):

Page	Loaded	Last ref	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

(a) Which page will FIFO replace?

→ Page 3 because its loaded time is smallest i.e 110

(b) Which page will NRU replace?

→ Page 2 because it has R=0 and M=0

(c) Which page will LRU replace?

→ Page 1 because its Reference time is smallest i.e 265.

(d) Which page will second chance replace?

→ Page 2. Here, according to FIFO first victim is page 3 but it has R=1, so it gets second chance. The next victim is page but it also has R=1, so it also get second chance. Now, another victim is page 2 and it has R=0. So, it will be replaced.

2018 - Fall - 2b)

Process	A.T	B.T
P1	0	3
P2	3	6
P3	5	4
P4	6	5

RR, Quantum = 2

RQ: ~~P1~~ P3 ~~P1~~ ~~P2~~ ~~P1~~ ~~P4~~ ~~P3~~ P4 ~~P2~~ ~~P4~~

P1	P1	P2	P3	P2	P4	P3	P2	P4	P4
0	2	3	5	7	9	11	13	15	17	18

• ASSIGNMENT

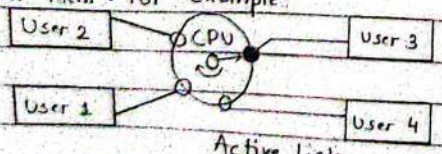
1. What is an operating system? Describe time-sharing, real-time and distributed operating systems in brief.

Answer

An operating system or 'OS' is software that communicates with the hardware and allows other programs to run. An OS acts as an intermediary between the user of a computer and the computer hardware. It hides hardware complexity, manages computational resources and provides isolation and protection. Most importantly, it directly has privilege access to the underlying hardware. Some common desktop operating systems include Windows, OS X, Linus, etc whereas Android, IOS are mobile operating systems.

• Time-Sharing Operating Systems

It is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing. Multiple jobs are executed by the CPU by switching between them. For example:



Active Link

In the figure, user 3 is active but user 1, 2 are in waiting state whereas user 4 is in ready status. As soon as the time slice of user 3 is completed, the control moves on the next ready user ie user 4. In this state user 2 is waiting and user 1 is ready. The process continues in the same way and so on.

Advantages

1. Provides the advantage of quick response
2. Avoids duplication of software
3. Reduces CPU idle time.

Disadvantages

1. Problem of reliability
2. Problem of data communication
3. Question of security and integrity of user programs and data.

• Real-Time Operating Systems

In this OS, the response time is already fixed. It focuses on accomplishing a computational task before its specified deadline. No computer resources are shared and there is a single application at a time. Modification of the program is not possible and the user must get the response within the defined time constraint. The real-time OS not only require accurate results but also the timely results, which means along the correctness of the results it must be produced in a time limit otherwise the system will fail. It is basically implemented in the applications which involve the control devices such as medical

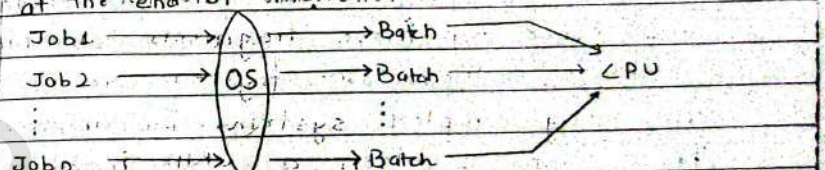
imaging systems, industrial control systems, automobile engine fuel injection system, weapon system, etc.

• Advantages
1. Maximum consumption
2. Deterministic Behaviour

• Disadvantages
1. Sometime cost is more
2. Not easy to program

• Distributed Operating Systems
It uses multiple central processors to serve multiple real-time applications and multiple users. The data processing jobs are distributed among the processors accordingly. The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Examples: www (world wide web), the internet, intranets, email, etc.
A distributed system is designed to tolerate failure of individual computers so the remaining computers keep working and provide services to the users.
Advantages
1. All systems are independent of each other, so the failure of one will not affect the others.
2. Electronic mail increased the data exchange speed.
3. Delay in data processing reduces.
Disadvantages
1. Failure of main network will stop the entire communication.
2. These type of systems are not readily available as they are very expensive.

• Batch Operating System
The users of batch OS don't interact with the computer directly. It is an OS in which the same type of processes are batched together for execution. Batch processing is the execution of a series of programs (jobs) on a computer without manual intervention. Jobs are set up so they can run to completion without human interaction. Batch is good for executing large jobs that need little interaction. Example: Bank Statements; at the end of month the bank makes statement for each account holder, so it can be made by batch system at the end of month.



Advantages
1. Multiple users can share the batch system.
2. Easy to manage large work repeatedly.
3. Idle time is less.
Disadvantage
1. Lack of interaction between the user and job.
2. Difficult to provide the desired priority.

- Personal-Computer Operating Systems

These are widely used for word processing, internet access and spreadsheets. It provides a good interface to a single user. PC-OS are made only for personal. We can say laptop, tablets, desktop are the PC and OS such as windows 7, windows 10, android are the PC-OS. At home, the most popular use for PC's is for accessing the internet, games and for ~~enta~~ entertainment.

- Parallel System

They are designed to speed up the execution of program by dividing the program into multiple fragments and processing these fragments simultaneously. Such ~~sy~~ multiprocessor system are also known as tightly coupled systems. The architecture of the software is often a UNIX-based platform which allows it to coordinate distributed loads between multiple computers in a network. Parallel OS are able to use software to manage all of the different resources of the computers running in parallel, such as memory, caches, storage space

Advantage

1. Massive data storage and quick data communication
2. Solve larger problems in a short time.

Disadvantage

1. Huge power consumptions
2. Difficult to implement

2. What is CPU bound and I/O bound process? Why inter process communication is required? Explain

Answer ↳

CPU bound means the rate at which, process progresses is limited by the speed of the CPU. A program is CPU bound if it would go faster, if the CPU is faster. A task that performs calculations on a small set of numbers, for example multiplying small matrices is likely to be CPU bound

I/O bound means the rate, at which, a process progres is limited by speed of I/O subsystem. A program is I/O bound if it would go faster if I/O subsystem is faster. A task that ~~processes~~ data from disk, for example counting number of lines in a file is likely to be I/O bound.

Inter process communication (IPC) is set of programming interfaces that allow a programmer to coordinate activities among different program process that can run concurrently in an OS. This allows a program to handle many user requests at same time. IPC is used for exchanging data between multiple threads in one or more process or programs. The process maybe running on single or multiple computers connected by a network. Since every single user may result in multiple processes running in operating system, the process may require to communicate with eachother. Each IPC protocol approach has its own advantage and limitation, so it is not unusual for single program to use all of the

IPC methods include pipes, message queing, semaphores, shared memory and sockets. IPC facilitates efficient message transfer between processes. The idea of IPC is based on Task Control Architecture (TCA). It is flexible technique that can send and receive variable length arrays, data structures and lists. It has the capability of using publish/subscribe and client/server data transfer paradigms while supporting wide range of operating systems & languages. There are numerous reasons for using interprocess communication process for information sharing which are:

ⓐ It helps to speedup modularity.
ⓑ Computation speedup
ⓒ Privilege Separation
ⓓ Modularity
ⓔ Convenience
ⓕ Helps OS to communicate with eachother and synchronizing their actions.

3 What is system call? Define major services of OS.

Answer ↳

A system call is how a program requests a service from an operating system's kernel (central or most important part of something). System calls provide the interface between a running program and the operating system. System call provides the services of the OS to the user programs via Application Program Interface (API). The whole process may include hardware related services, creating and executing new processes, and

communicating with integral kernel services. The services provided by system calls are;

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling (I/O)
5. Protection
6. Networking, etc.

System calls can be roughly grouped into 5 major categories;

1. Process Control
→ load
→ execute
→ create process
→ terminate process
→ get/set process attributes
→ wait for time, wait for event, signal event
→ allocate, free memory

2. File Management
→ create file, delete file
→ open, close
→ read, write, reposition
→ get/set attributes

3. Device Management
→ request device, release device
→ read, write, reposition
→ get/set device attributes
→ logically attach or detach devices

4. Information Maintainence
→ get/set time or date
→ get/set system data
→ get/set process, file or device attributes

5. Communication
→ create, delete communication connection
→ send, receive messages
→ transfer status information
→ attach or detach remote devices.

3. The services provided by the operating systems are;

i) Program execution

OS loads a program into memory & executes program. The program must be able to end its execution, either normally or abnormally.

ii) I/O Operation

Program may require any I/O device while running. So OS must provide the required I/O.

iii) File System Manipulation

Program needs to read or write a file. The operating system gives permission to program for operation on file.

iv) Communication

Data transfer between two process is required for some time. Both the processes are on one computer or on different computers but connected through computer network

v) Error Detection

Error may occur in CPU, I/O devices or in memory hardware. OS constantly needs to be aware of possible errors. It should take appropriate actions to ensure correct and consistent computing.

Additional operating system functions:

i) Resource Allocation

allocating resources to multiple users or multiple jobs running at same time.

ii) Accounting

Keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics

iii) Protection

Ensuring that all access to system resources is controlled.

4. Explain the advantages of multithreading. What are different multithreading models? Explain

Ans

↳ The advantages of multithreading are;

i) Responsiveness

Program responsiveness allows a program to run even if part of it is blocked using multithreading. This can also be done if the process is performing a lengthy operation. For example: An web browser with multithreading can use one thread for user contact and another for image loading at the same time.

ii) Resource Sharing

All the threads of a process share its resources such as memory, data, files, etc. A single application can have different threads within the same address space using resource sharing.

iii) Utilization of Multiprocessor Architecture

In a multiprocessor architecture, each thread can run on different processor in parallel using multithreading. This

increases concurrency of the system. This is in direct contrast to a single processor system, where only one process or thread can run on a processor at a time.

iv) Economy

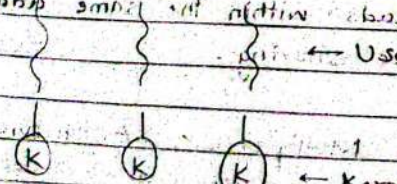
It is more economical to use threads as they share the process resources. Comparatively, it is more expensive and time-consuming to create processes as they require more memory and resources. The overhead for process creation and management is much higher than thread creation and management.

The multithreading models are:

i) One to One Model

The one to one model maps each of the user thread to a kernel threads. This means that many threads can run in parallel on multiprocessors and other threads can run when one thread makes a blocking system call. Examples: Linux, family of windows OS. Its disadvantage is that the creation of a user thread requires a corresponding kernel thread. Since, each of kernel threads burden the system, there is a restriction on the number of threads in the system.

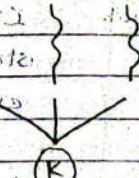
Figure:



ii) Many to one Model

It maps many of the user threads to a single kernel thread. This model is quiet efficient as the user space manages the thread management.

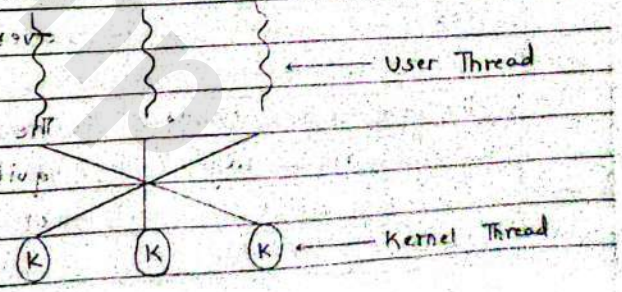
Its disadvantage is that a thread blocking system call blocks the entire process. Also, multiple threads cannot run in parallel as only one thread can access the kernel at a time. Figure:



iii) Many to Many Model

It maps many of the user threads to a equal number or less than kernel threads. The no. of kernel threads depends on the application and machine. It does not have disadvantages of the other two models as there can be as many user threads as required and their corresponding kernel threads can run in parallel on a multiprocessor.

Figure:



5. Differentiate between program and progress. Explain process control block (PCB) in detail.

Answer:

Program	Process
1. It is a set of instructions	When a program is executed it is known as process or a program in execution.
2. It has a longer life span. Stored on the hard disk in the computer.	It has a limited life span. Created when execution starts and terminates when execution is finished.
3. A program is passive or static entity.	A program is dynamic or active entity.
4. A program needs memory space on disk to store all instructions.	A process contains many resources like memory address, disk, printer, etc.
5. No duplication is needed.	New process requires duplication of the parent process.
6. No significant overhead cost.	Process have considerable overhead.
7. The program only needs memory for storage.	The resource requirement is quite high in case of a process.

Process Control Block (PCB)

PCB is a data structure used by computer OS to store all the information about a process. It is also known as a process descriptor. When a process is created (initialized or installed), the operating system creates a corresponding process control block. Information in a PCB is updated during the transition of process states. When the process terminates, its PCB is returned to the pool from which new PCBs are drawn. Each process has a single PCB.

A PCB contains 5 blocks such as	Process Id
Process Id, Process State, Priority,	Process State
Accounting Information, Program	Process Priority
Counter, CPU Register, etc.	Accounting Information
1. Process Id	Program Counter
2. Various process will be running	CPU Register
simultaneously with each process	PCB Pointers

has its unique ID which Process Control Block helps to identify the processes. This ID helps system in scheduling the process and is provided by the PCB.

2. Process State

The process states can be New, Running, Waiting, Executing, Blocked, Suspended, Terminated. PCB is used to define the process state of any process. The PCB's 4 field (process state) holds the current state of the resp. process. Example: If the process is currently executing, the process will hold the running state for that process.

3. Process Priority

It is a numeric value. Lesser the value, greater is the priority of that process. The priority of the process can be assigned externally by the user or by the operating system itself. The process is assigned the priority at the time of its creation. The priority of the process may get changed over its lifetime depending on the various parameter.

4. Process Accounting Information

The field of PCB gives the account/description of the resources used by that process such as the amount of CPU time, real-time used, connect time

5. Program Counter

It holds the pointer to an instruction in the program or code that is to be executed next. This field contains the address of the instruction that will be executed next in the process.

6. List of open files (CPU Registers, PCB Pointer, Process PID status information, Event information, etc)

It contains the information of all the files that is required by the program its execution. It helps the OS to close all the opened files which aren't closed explicitly at the termination of the program.

a. CPU Registers: Used to hold the temporary values & information

b. PCB Pointer: Here, the pointer has an address of the next PCB, whose process state is ready.

6. What is dispatcher and dispatch latency? Explain short term scheduler, medium term scheduler and long term scheduler.

Ans

↳ Dispatcher is the module that gives control of the CPU to the process selected by the short-time schedulers (selects from among the processes that are ready to execute). The function involves switching context, switching to user mode, jumping to the proper location in the user program to restart that program.

Dispatch latency is the time taken by the dispatcher to stop one process and start another.

Short Term Scheduler

→ Main objective is increasing system performance in accordance with chosen set of criteria.

→ It is CPU scheduler.

→ Speed is very fast

→ Less control over degree of multiprogramming.

→ Also known as dispatcher.

→ Minimal in time sharing system

→ It select from among the processes that are ready to execute

→ Process state is (Ready to Running)

→ Select a new process for a CPU quite frequently.

Medium Term Scheduler

→ It removes the process from the memory.

→ It is swapping.

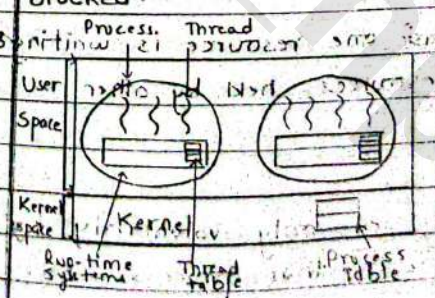
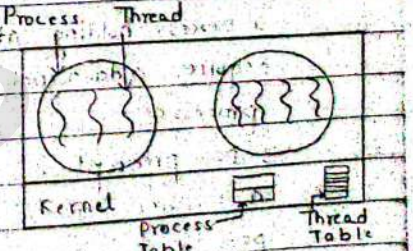
→ Speed is in between both.

→ Reduce the degree of multiprogramming.
→ Time sharing system use medium term scheduler.
→ Process can be reintroduced into memory and its execution can be continued.

Long-Term Scheduler:
→ It is job scheduler.
→ Speed is less than short-term scheduler.
→ determines which programs are admitted to the system for processing.
→ It controls degree of multiprogramming.
→ Absent or minimal in time sharing systems.
→ It selects processes from pool and loads them into memory for execution.
→ Process state is New to Ready from.
→ Select a good process mix of I/O bound & CPU bound.

7. How process differs from thread? Explain. Differentiate between user and kernel thread. Draw figures to explain.

Answer	Process	Thread
1	Process means any program is in execution.	Thread means a segment of a process.
2	Takes more time to terminate.	Takes less time to terminate.
3	Takes more time for creation.	Takes less time for creation.
4	Takes more time for context switching.	Takes less time for context switching.
5	It is less efficient in communication.	It is more efficient in communication.
6	Consumes more resources.	Consumes less resources.
7	Is isolated.	Shares memory.
8	Is heavy weight process.	Is light weight process.
9	Switching uses interface in operating system.	Switching doesn't require to call a OS and cause an interrupt to the kernel.

	User level Thread	Kernel level Thread
1	Implemented by users.	Implemented by OS.
2	Not recognized by OS.	Recognized by OS.
3	Implementation is easy.	Implementation is complicated.
4	Context switch time is less.	Context switch time is more.
5	Context switch requires no hardware support.	Hardware support is needed.
6	Are designed as dependent threads.	Are designed as independent threads.
7	If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
8		
9.	Example: JAVA thread POSIX	Example: Window Solaris

8. Do you feel deadlock is to great enemy of computer system? If yes, why? Also write protection mechanism for deadlock.

Answer

↳ A set of process is said to be deadlock if each process in set is waiting for an event that only another process in the set can cause. Since, Deadlock is a situation that occurs in OS when any process enters a waiting state because another waiting process is holding the demanded resource. It is a common problem in multi-processing where several processes share a specific type of mutually exclusive resource known as a software. Since, all process are waiting, none of them will ever cause any of the event that would make up any of the other members of the deadlock and all process continue to wait forever. This stops the whole process of executing the task and getting it completed and thus this hampers the system. Deadlock can arise if four conditions hold simultaneously.

1. Mutual exclusion:
only one process at a time can use a resource.

2. Hold and wait:
a process holding atleast one resource is waiting to acquire additional resources held by other processes.

3. No preemption:
a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait
there exists a set {P0, P1 ... P0} of waiting processes such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource that is held by P1, and P0 is waiting for a resource that is held by P0.

Methods for handling deadlocks

- Ensure that the system will never enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most OS, including UNIX.

Deadlock can be prevented / protected as:
Can be avoided at least one of the four conditions.

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

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

3. No Preemption
Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

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

9. What is mutual exclusion? Show how mutual exclusion can be achieved using Peterson's solution.

Answer

↳ Mutual exclusion is a way of making sure that if one process is using a shared variable or file, the other process will be excluded from doing the same things. Example: When two threads work on the same data at the same time, it acts as a lock and is th When a thread tries to acquire a mutual exclusion, it gains mutex if it is available, otherwise the thread is set to sleep condition. or If process P_i is executing in its critical section, then no other processes can be executing in its critical section

Peterson's Solution:

→ There are only 2 processes P_0 and P_1 (two process solution)

→ Assume that the LOAD and STORE instructions are atomic (i.e cannot be interrupted)

→ The two processes share two variables

int turn;

Boolean flag[2]

→ The variable turn indicates whose turn it is to enter the critical section.

→ The flag array is used to indicate if a process is ready to enter critical section

→ flag[i] = true implies that process P_i is ready!

Algorithm 1:

. shared variables

boolean flag[2];

initially flag[0] = flag[1] = FALSE

flag[i] = TRUE ⇒ Pi wants to enter its critical section

. Process Pi

do {

flag[i] = TRUE;

while (flag[j]);

critical section

flag[i] = FALSE;

remainder section

} while (TRUE);

→ satisfies mutual exclusion, but not progress requirements.

Algorithm 2:

shared variables

initially turn = 0

turn = i ⇒ Pi can enter its critical section

Process Pi

do {

while (turn != i);

critical section

turn = j;

remainder section

} while (TRUE);

→ satisfies mutual exclusion, but not progress.

Algorithm 3 (Peterson's Algorithm)

- Combined shared variables of algorithms 1 & 2
- initially flag[0] = flag[1] = False
- Process: Pi

```
do{
    flag[i] = true;
    turn = j;
    while (flag[j] and turn=j);
    // critical section
    flag[i] = false;
    // remainder section
    while(1);
```

- meets all three requirements, solves critical-section problem for two processes.

10. What is process synchronization needed? What problem may occur if they are not synchronized? Describe Peterson's algorithm in detail.

Answer

↳ Process Synchronization means sharing system resources by processes in such a way that concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of operating processes. It was introduced to handle problem that arose while multiple process executions. The need for synchronization originates when process need to execute concurrently. The main purpose is the cordination of process interaction in an operating system.

The problems that may occur if they're not synchronized are

1. Principle of Concurrency
2. Producer - Consumer Problem
3. Race Condition
4. Mutual Exclusion.

Peterson's Algorithm.

```
    flag[j] = true;
    turn = i;
    while (flag[i] == true && turn == i)
    {   flag[j] = false;
        flag[i] = true;
        turn = j;
    }
    while (flag[j] == true && turn == j)
    {   flag[i] = false; }
```

Peterson's algorithm is used to synchronize 2 processes. It uses two variables, a bool array flag of size 2 and an int variable turn to accomplish it. In the solution i represents the consumer and j represents the producer. Initially the flags are false. When a process wants to execute it's critical section, it sets the flag to true and turn as the index of the other processes. This means that the process wants to execute but it will allow the other process to run first. The process performs busy waiting until the other process has finished it's own critical section. After this the current process enters its critical section and adds or removes a random number from the shared buffer. After completing the critical

section, it sets it's own flag to false, indication it doesnot wish to execute anymore.
The programs runs for a fixed amount of time before exiting. This time can be changed by changing value of the macro RT.

11. How do you determine the state of system is safe or unsafe using Banker's algorithm for multiple resource type?

Ans

→ Banker's algorithm:

The resource-allocation graph algorithm is not applicable to resource-allocation system with multiple instances of each resource type. This is applicable to such a system, but is less efficient than resource-allocation graph scheme.

- Multiple Instances
- Each process must have a prior claim max use
- When process gets all its resources it must return them in a finite amount of time.

Data structures for Banker's algorithm (Multiple Resou

- Let n = no. of process, & m = no. of resource types
- Available: Vector of length m. If available[j] = k, there are k instances of resource type Rj available
- Max: n x m matrix. If Max[i,j] = k, then process Pi may request at most k instances of resources type Rj.
- Allocation: n x m matrix. If allocation[i,j] = k then it is currently allocated k instances of Rj

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

Need[i,j] = Max[i,j] - Allocation[i,j]

→ Safety Algorithm

1. Let work and finish be vectors of length m & n
   @ Initialize: Work = Available
   ⓑ Finish[i] = false for i = 0, 1, ..., n-1
2. Find i such that both
   @ Finish[i] = false
   ⓑ (Need)i ≤ Work
3. If no such i exists, go to step ④
   @ Work = Work + (Allocation)i
   ⓑ Finish[i] = true
   ⓒ goto step②
4. If Finish[i] = true for all i, then system is in safe

→ Resource-Request Algorithm for Process Pi

Request = request vector for process Pi. If Request i[j] = k then process Pi wants k instances of resource type Rj

1. If (Request)i ≤ (Need)i goto step②. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If (Request)i ≤ Available goto step ③, otherwise Pi must wait since resource are not available
3. Pretend to allocate requested resources to Pi by modifying the state as follows:
   @ Available = Available - Request i;
   ⓑ Allocation = (Allocation)i + (Request)i;
   ⓒ (Need)i = (Need)i - (Request)i;

If safe → the resources are allocated to P_i

If unsafe → P_i must wait, and old resource-allocation state is restored.

12. What are requirements for the solution of critical selection ~~solution~~ problem? Explain software solution for critical section problem.

→

Critical solution is a code segment that can be accessed by only one process at a time. Critical section contain shared variables which need to be synchronized to maintain consistency of data variables. At given point of time only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.

Any solution to critical section problem must satisfy three requirements:

1. Mutual Exclusion

If a process is executing in its critical section, then no other process is allowed to execute in the critical section.

2. Progress

If no process is executing in critical section & other processes are waiting outside critical section, then only those processes that aren't executing in their remainder section can participate in deciding which will enter critical section next, and the selection cannot be postponed indefinitely.

3. Bounded waiting:

A bound must exist on number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section & before the request is granted.

Assumptions:

- Assume that a variable can only have one value; never "between values".
- If process A & B write a value to "same time", either value from A or value from B will be written rather than some scrambling of bits

→ Peterson's algorithm

13. What is semaphore? How producer-consumer problem is solved using semaphore. Explain with pseudo code.

→

A semaphore is special kind of integer value which can be initialized and can be accessed only through two atomic operations P & V. If S is semaphore variable,

P operation: Wait for semaphore to become positive and then decrement	V operation: Increment
P(S):	V(S)
while (S ≤ 0)	S = S + 1;
do no-op;	
S = S - 1	

Properties of Semaphores:
1. Its simple and always have non-negative integer value
2. Works with many processes.
3. Can have many different critical sections with different semaphores.
4. Each critical section has unique access semaphores.
5. Can permit multiple processes into critical section at once, if desirable.

Semaphores implementation without busy waiting:
Implementation of wait

```
wait(s)
{   value--;
    if (value < 0)
    {   add this process to waiting queue
        block();
    }
}
```

Implementation of signal:

```
~~wait(s)~~ signal(s)
{   value++;
    if (value <= 0)
    {   remove a process P from the waiting queue
        wakeup(P);
    }
}
```



```
#define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer(void)
{   int item;
    while (TRUE)
    {   item = produce_item();
        down(&empty);
        down(&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}
void consumer(void)
{   int item;
    while (TRUE)
    {   down(&full);
        down(&mutex);
        item = remove_item();
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

The soln uses three semaphores:
i) Full: For counting no. of slots that are full, initially 0.
ii) Empty: For counting no of slots that are empty, initially equal to no. of slots in the

iii) Mutex: To make sure that producer & consumer do not access buffer at same time, initially 1.

14. Discuss in detail the use of translation, lookside buffer (TLB) in prosess of paging.

→ A TLB is a memory cache that stores recent translations of virtual memory to physical address for faster retrieval. Each entry in TLB consists of two parts: a key (or tag) and a value.

When associative memory is presented with an item, the item is compared with all keys simultaneously. If item is found, the corresponding value field is returned. The search is fast; a TLB lookup in modern hardware is part of instruction pipeline, essentially adding no performance penalty. To be able to execute search within a pipeline step, however the TLB must be kept small. It is typically betn 32 and 1024 entries in size. The TLB is used with page-table entries. When logical address is generated by the CPU, its page no. is presented to TLB. If page no. is found its frame no. is immediately available and is used to access memory. These steps are executed as part of instructions pipeline within the CPU, adding no performance penalty compared with system that doesnot implement paging.

If page no. is not in the TLB (known as TLB miss), a memory reference to page table must be made. Depending on the CPU, this maybe done automatically in hardware or via an interrupt to operating system. When frame no. is obtained, we can use it to access memory. In addition, we add page no. & frame no. to the TLB, so that they will be found quickly on next reference. If TLB is already full of entries, an existing entry must be selected for replacement.

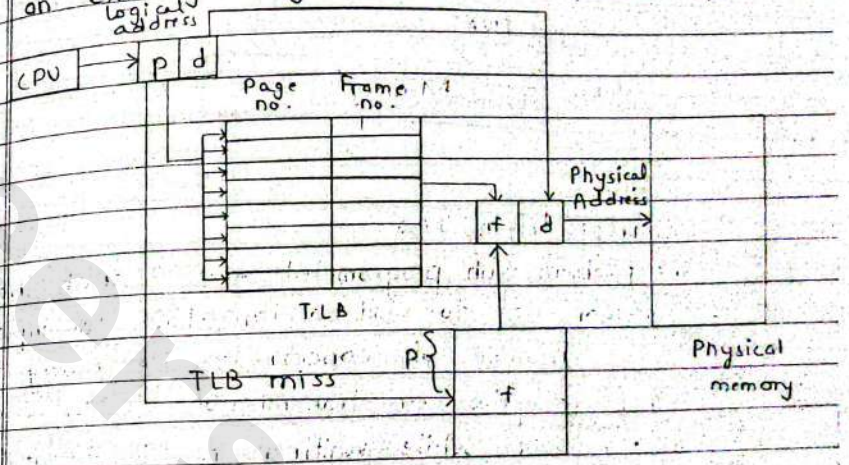
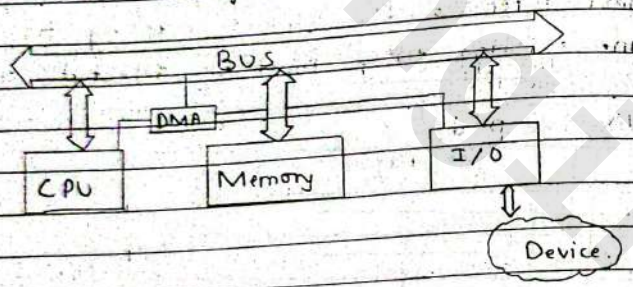


Fig: Paging hardware with TLB

15. What are different ways to input output ? Explain Interrupt driven i/o with diagram.

→

The ways are
i) Programmed I/O
ii) Interrupt - driven
iii) Direct Memory Access



Interrupt driven I/O :
The problem with programmed I/O is that the processor has to wait a long-time for input/output module of concern to be ready for either reception or transmission of more data. The processor while waiting must repeatedly interrogate the status of I/O module. As a result, the level of performance of entire system is degraded.
An alternative approach for this is interrupt-driven I/O. The processor issue an I/O command to module & then go on to do some other useful work. The i/o will then interrupt the processor to request service, when it is ready to exchange data with processor. The processor then executes the data transfer as before and then resumes its former processing. Interrupt driven I/O still consumes alot of time because every data has to pass with processor.

Hardware	Software
Device controller or other system hardware issues an interrupt	Save remainder of process state info
Processor finishes execution of current instruction	Process Interrupt
Processor signals acknowledgement of interrupt	Restore process state info
Processor pushes PSW & PC onto control stack	Restore old PSW and PC
Processor loads new PC value based on interrupt	

16. How data is maintained and managed using RAID technology? Is there any chance of losing data using it ? Describe each of available modes in detail.

→

RAID (Redundant Array of Independent Disks) is a storage technology that combines multiple disk drive components into logical unit. Data is distributed across the drivers in one of several ways called 'RAID levels', depending on what level of

redundancy and performance is required.

RAID Levels:

(a) RAID level 0 - striping:

In RAID 0, system data are split up into blocks that get written across all drives in the array. By using multiple disks at same time, this offers superior I/O performance. This performance can be enhanced further by using multiple controllers, ideally one controller per disk.

RAID 0 offers great performance, both in read & write operations. There is no overhead caused by parity controls. All storage capacity is used, there is no overhead. The technology is easy to implement. RAID 0 is not fault tolerant. If one drive in RAID 0 fails, all data are lost.

(b) RAID Level 1 - Mirroring:

Data are stored twice by writing them in both data drive and mirror drive. If ~~advice~~ active fails, the controller uses either the data drive or mirror drive for data recovery and continued operation. You need atleast 2 drives for a RAID 1 array. RAID 1 offers excellent read speed and write speed that is comparable to that of single drive. In case drive fails, data donot have to be rebuild, they just have to be copied to the replacement drive. RAID is very simple technology. The main

disadvantage is that effective storage capacity is only half of total drive capacity because all data get written twice.

(c) RAID Level 2 Bit - level Striping:

All disk spindle rotation is synchronized and data is striped such that each sequential bit is on different drive. Hamming code parity is calculated across corresponding bits and stored atleast one parity device.

(d) RAID Level 3: Byte - level Striping:

All disk spindle rotation is synchronized and data is striped so each sequential byte is on different drive. Parity is calculated across corresponding bytes and stored on dedicated parity drive.

(e) RAID Level 4: Block level Striping:

Identical to RAID 5, but confines all parity data to single drive. In this setup, files maybe distributed between multiple drives. Each driver operates independently, allowing I/O requests to be performed in parallel. However use of dedicated parity drive could create performance bottleneck.

(f) RAID Level 5

It is most common secure RAID level. It requires atleast 3 drives but can work with upto 16. Data blocks are stripped across drives and on one drive a parity check sum of all block data

is written. The parity data aren't written to a fixed device, they are spread across all drives. Using parity data, computer can recalculate data of one of other data blocks, should those data no longer be available, that means RAID 5 array can withstand a single drive failure without losing data or access to the data. Read data transactions are very fast while data transactions are somewhat slower. If drive fails, you still have access to all data, even when failed drive is being replaced and storage controller rebuilds data on the new drive. Drive failures have an efficient on throughput, although this is still acceptable.

17. What are different file access methods? Explain.

→ When a file is used, info is read and accessed into computer memory and there are several methods to access this information of file. Some system provide only one access method for files. Other systems such as those of IBM, support many access methods, and choosing right one for a particular application is major design problem.
There are mainly three ways to access a file into computer system;

ⓐ Sequential Access
It is the simplest access method. Information in the file is processed in ~~other~~ order, one record after the other. This mode of access is by far the most common. For example, editor and compiler usually access the file in this fashion. Data is accessed one record right after another record in an order. When we use read command, it moves ahead pointer by one. When we use write command, it will allocate memory and move the pointer to the end of the file. Such a method is reasonable for tape.

ⓑ Direct Access
Also known as relative access method. A filed length logical record that allows the program to read and write record rapidly in no particular order. The direct access is based on disk model of a file since disk allows random access to any file block. For direct access, the file is viewed

18. What are different file operation? Explain contiguous and linked list allocation in file system.

→ File exist to store information and allow it to be retrieved later. Different systems provide different operations to allow storage and retrieval.
Typical file operations are:
i) Create: A new file is defined and positioned within structure of files