20 | 2020 Wednesday May | 141-225 • WK 21

VARUN VERMA
CS-B1 (Roll no - 38)
1801010170

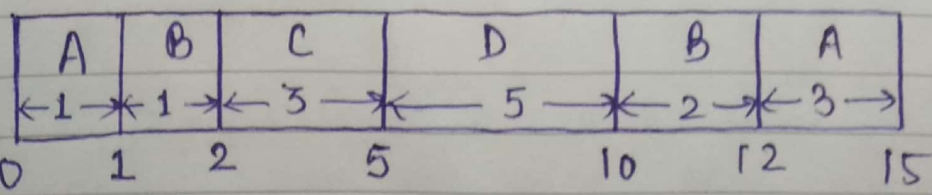
Operating System Assignment - 4

Ques 1 → Four processes A, B, C and D require CPU burst time of 4, 3, 3 and 5 units with priority 3, 4, 6 and 5 respectively. Draw Gantt chart and compute the average TAT and AWT of these processes for the priority scheduling algorithms : i) Preemptive
ii) Non-preemptive.

Solution → i) Preemptive

Process	B.T	A.T.	Priority	TAT (CT-AT)	WT (TAT-BT)	CT
A	4	0	3	15	11	15
B	3	1	4	11	8	12
C	3	2	6 (H)	3	0	5
D	5	3	5	7	2	10

Gantt chart :-



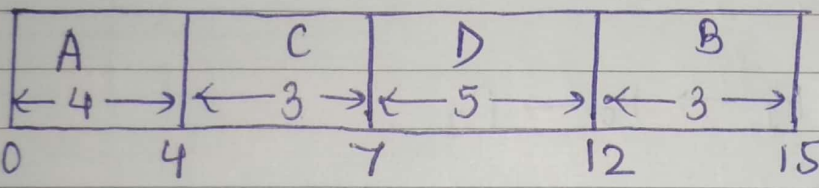
NOTES

ATAT = 9
AWT = 5.25

ii) Non - Preemptive

Process	B.T.	A.T.	Priority	TAT	WT	CT
A	4	0	3	4	0	4
B	3	1	4	14	11	15
C	3	2	6 (H)	5	2	7
D	5	3	5	9	4	12

Gantt chart :-



ATAT = 8

AWT = 9.5

Ques-2 Consider a system with 12 magnetic tape drives and 3 processes (P_0, P_1 and P_2). Process P_0 requires 10 tape drives, process P1 require 4, and process P2 may need 9 tape drives. Suppose that, at time t_0, process P_0 is holding 5 tape drives, process P1 is holding 2, and process P2 is also holding 2 tape drives. What is the process sequence?

NOTES

Solution → Total tape drives = 12

Process	Need	Allocated
P_0	10	5
P_1	4	2
P_2	9	2

Available = Total - Allocated
= 12 - (5+2+2)
= 3

Sequence → $P_1 \rightarrow P_0 \rightarrow P_2$

1) 2 of 3 available drives are give to P_1. After completion of P_1, 2 more drives get free.
2) Now there are 5 available drives which can be given to P_0. After completion of P_0 now total available will be 10.
3) Similarly process P_2 will be completed

Ques. 3 → Consider the following snapshot of a system
Answer the following ques using banker's algo.

i) What is the content of matrix need?
ii) Is the system in a safe state?
iii) If a request from process P2

arrive for (2,1,0). Can the request be granted immediately?

Solution →

Process	Allocation			Max			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8	1	4	5	4	3	10
P_2	2	0	3	4	3	3	2	3	0	6	~~3~~	~~13~~
P_3	1	2	4	3	4	4	2	2	0	~~7~~	~~5~~	~~17~~
										9	7	20

i) need = Max − Allocation

ii) Sequence ⇒ ~~$P_2 \rightarrow P_1 \rightarrow P_3$~~ $P_2 \rightarrow P_3 \rightarrow P_1$

Yes, the system is in safe state, as the sequence has all the processes.

iii) Now, $P_2 = \langle 2,1,0 \rangle$

Process	Allocation			Max			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8	1	4	5	~~4~~	~~3~~	~~10~~
P_2	2	1	0	4	3	3	2	2	3	4	2	13
P_3	1	2	4	3	4	4	2	2	0	6	3	13

Available = Total − Allocated

21 Wednesday May

Available = 9 12 15 − 5 5 7 = 4 7 8 ; 9 7 20 − 5 5 7 = 4 2 13

Sequence ⇒ ~~P_1~~ $\rightarrow P_2 \rightarrow P_3 \rightarrow P_1$

So, there is a safe sequence. Thus, the request for $P_2 < 2, 1, 0>$ can be granted immediately.

Ques 4 :- On the system using paging and segmentation, the virtual address space consists of up to 8 segments where each segment can be up to ~~200~~ 2^{29} bytes long. The hardware pages each segment into 256 - byte pages. Determine the bits needed in the virtual address to specify the : segment number, Page number, offset within page, Entire virtual address.

Ans :- 8 segments = 2^3 ~~bit~~ segments.

Thus, 3 bit size is required to hold the segment no.

NOTES

a) Segment no. bits = 3 bits.

Acc. to ques ⇒ each segment = 2^{29} byte long
So, 29-bit size is required for offset d.
In main memory, each segment = 256 bytes pages
= 2^8 bytes pages.
= 8-bit for each frame.

b) p = 29 − 8 = 21 bits size.

c) d' = 8 bit.

d) Entire virtual address size = s + d
= 3 + 29
= 32 bits.

Q5] Give the solution of Dining Philosphere problem by using the concept of semaphores?

Ans → Represent each chopstick with a semaphore. A philosopher tries to grab a chopstick by executing a wait() operation on that

semaphore ; she releases her chopsticks by executing the signal() operation on the appropriate semaphores.

Shared data → semaphore chopstick[5];

Initially → All value are 1.

The structure of philosopher i →

```
do
{
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
    ...
  // eat
    ...
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
    ...
  // think
    ...
}while(true);
```

The solution guarentees that no two neighbours are eating simultaneously.

NOTES