CSE321 Final

Name : Mahmudul Hasan Emon

ID : 19101098

Section : 04

Declaration : I did not follow any kind of unfair means while giving the exam. I have not shared answers with anyone, did not help anyone, and did not take help from anyone for this exam. The answers I have given here are done by myself only.

Answer to the Q. No 1

(a) (i)

Threads	Allocation matrix	Max Matrix	Available	Need matrix
	A B C D E	A B C D E	A B C D E	A B C D E
T_0	4 2 0 2 1	4 4 2 2 1	1 1 1 2 1	0 2 2 0 0
T_1	2 1 5 0 2	3 1 5 2 2		1 0 0 2 0
T_2	3 2 2 1 1	3 4 2 4 1		0 2 0 3 0
T_3	1 3 5 1 1	2 3 5 3 1		1 0 0 2 0
T_4	0 1 3 1 3	3 1 3 2 3		3 0 0 1 0

finish:

0	1	2	3	4
~~F~~	~~F~~	F	~~F~~	~~F~~

$T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_0 \rightarrow T_2$

Safe sequence

Work :

1 1 1 2 1

2 1 5 0 2

3 2 6 2 3

1 3 5 1 1

4 5 11 3 4

0 1 3 1 3

4 6 14 4 7

4 2 0 2 1

8 8 14 6 8

3 2 2 1 1

11 10 16 7 9

0 2 2 0 0 1 1 1 2 1 To wait, T7 wait,
1 0 0 2 0 3 0 0 1 0 4 5 1 1 3 4
0 2 0 3 0 3 2 6 2 3 x 0 2 2 0 0 4 6 1 4 4 7
1 0 0 2 0

The given state is unsafe. And the safe state should be $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_0 \rightarrow T_2$.

(ii)

Now T_1 arrives and request (0, 4, 2, 0, 2)

Given,

$T_1 = (1\ 0\ 0\ 2\ 0)$

So, (0, 4, 2, 0, 2) is not equal with (1, 0, 0, 2, 0)

As Here the new T_1 arrival (0, 4, 2, 0, 2) is not matched with T_1's Need matrix. So, request can't be granted.

1) b) i)

	Allocation				Request			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	1	0	0	0	0	1	0	0
P_2	0	1	0	0	0	0	1	0
P_3	0	0	1	0	0	0	0	1
P_4	0	1	0	1	1	0	0	0
P_5	0	0	0	1	0	0	0	0

Here available = (2 0 0 0)

(ii)

For detecting deadlock,

2 0 0 0
0 1 0 1 → P_4
---
2 1 0 1
0 0 0 1 → P_5
---
2 1 0 2
1 0 0 0 → P_1
---
3 1 0 2
0 1 0 0 → P_2
---
3 2 0 2
0 0 1 0 → P_3
---
3 2 1 2

∴ State: $P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3$

∴ There is no deadlock.

1 (c)

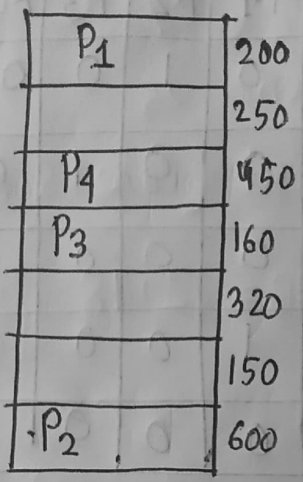
Hold and Wait and Circular Wait are conditions that are met when deadlock occur. This means that if these two condition are not met, we will not be in a deadlock. The hold and wait condition states that the process is holding onto a resources that may or may not be required by other process.

Answer to the Q. No 2

(a)

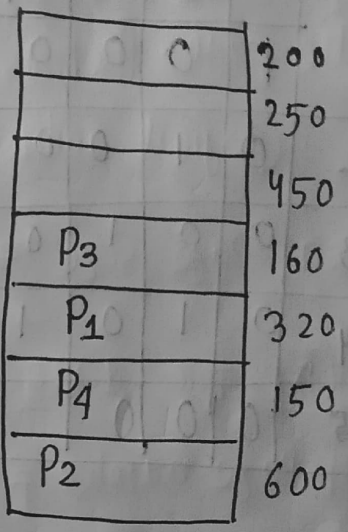
$P_1 \rightarrow 146KB$
$P_2 \rightarrow 425KB$
$P_3 \rightarrow 240KB$
$P_4 \rightarrow 89KB$
$P_5 \rightarrow 450KB$

First Fit



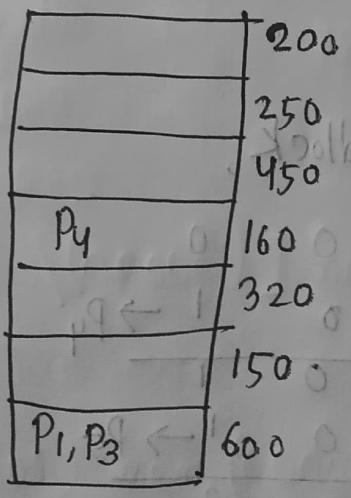
Here P_5 waiting

Best Fit



Here P_5 waiting

Worst Fit



Here P_2, $P_5 \rightarrow$ waiting

(ii)

Since, only 'best fit' can allocate all processes in the memory, it is the best algorithm to make the most efficient use of memory.

(2) (b)

The page number is given as

floor [(Logical address) / (Page size)]

The page-offset is given as

(Logical address) mod (Page size)

Given page size = 2KB = 2048

(i) for 1085,

Page number = floor [(1085) / (2048)]

= 0

Page offset = (1085) mod (2048)

= 1085

(ii) for 30000,

Page number = floor [(30000) / (2048)]

= 14

Page-offset = (30000) mod (2048)

= 1328

(iii) for 19467,

Page number = floor [(19467) / (2048)]

= 9

Page offset = (19467) mod (2048)

= 1035

(iv) For 15385,

Page-number = floor [(15385) / (2048)]

= 7

Page offset = (15385) mod (2048)

= 1049

(c)

Advantages of paging over segmentaion —

i) Does not cause external fragmentation

(ii) Less memory usage

(iii) More flexibility on page size.

(iv) Adds additional level of data access security.

3 (a)

writer:

```
do {
    wait (rw_mutex)
        /* writing is performed */
    signal (rw_mutex);
} while (true);
```

reader:

```
wait (mutex);
read_count++;
if (read_count == 1)
    wait (rw_mutex);
signal (mutex);
    /* reading is performed */
wait (mutex);
    read_count--;
if (read_count == 0)
        signal (rw_mutex);
    signal (mutex);
} while (true)
```

(3)

(b)

Time	Process 0	Proces 1
0-3	flag [0] = true	—
3-6	turn = 1	—
6-9	while() context switch	—
9-12	—	flag [1] = true
12-15	—	turn = 0
15-18	—	while() context switch
18-21	CS1	—
21-24	CS2	—
24-27	flag [0] = false, context switch	—
27-30	—	while( )
30-33	—	CS1
33-36	—	CS2 ; context switch
36-39	RS1	—
39-42	flag [0] = true	—
42-45	turn = 1 context switch	—

3(c)

We need to malify the producer consumer problem. In the problem waiters are the producers. All the tables combined is the buffer and the guests are consumers. From the given scenario, all the tables are need to be filled before the guest can start eating. So all the buffer need to contain item first. Then, all the guest must complete eating i.e. all the items in the buffers are needed to be ~~consum~~ consumed first and only then the food can be served again on all the table. Then the producer can fill the whole buffer agin. This process continuous.

3(d)

Three conditions any solution for critical section must meet.

① Mutual exclusion: If a process is executing in it's critical section, no other process can execute in their critical section.

② Progress: If no process is executing in it's critical section and some process wants to enter critical section, then only the processes not executing in remainder section can participate in deciding who will enter critical section next and it can't be postponed.

③ Bounded waiting: There must exist a limit or bound, on the ~~no~~ number of times that other process are allowed to enter their critical sections after a process has made a request to enter it's critical section and before the request is granted.