Victoria University of Bangladesh
Department of CSE
Sem:-Summer 2022
Program: BSc in CSIT

Course title:- Operating System Concepts
Course code:- CSI 231

Name:- Nusrat Jahan Tanshe
ID:- 2520200011
Batch:- 20

Term:- Final

Ans to the Q no -(1)-a

⊞ Valid bit:-

"Valid" indicates that the associated page is in the process' logical address space, and is thus a legal page.

Invalid bit:-

"Invalid" indicates that the page is not in the process' logical address space.

Ans to the Q no- 1(b)

⊞ Architecture of segmentation:-

A segment is a collection of logical unit such as-

- main program
- procedure
- function
- method
- object
- local variables,
- global variables
- Common block
- stack

symbol table
arrays

Segmentation Architecture:-

- Logical address consists of a two tuple:

  <segment-number, offset>,

- Segment table – maps two-dimensional physical addresses;

  ⓐ base:- contains the starting physical address where the segments reside in memory

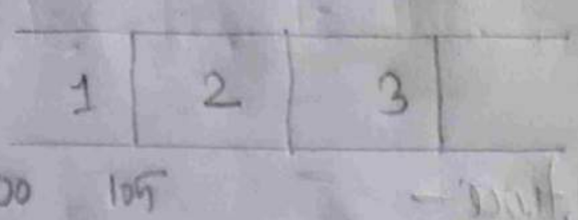
  ⓑ limit:- specifies the length of the segment

- Segment-table base register (STBR) points to the segment table's location in memory

- Segment table length register (STLR) indicates number of segments used by a program;

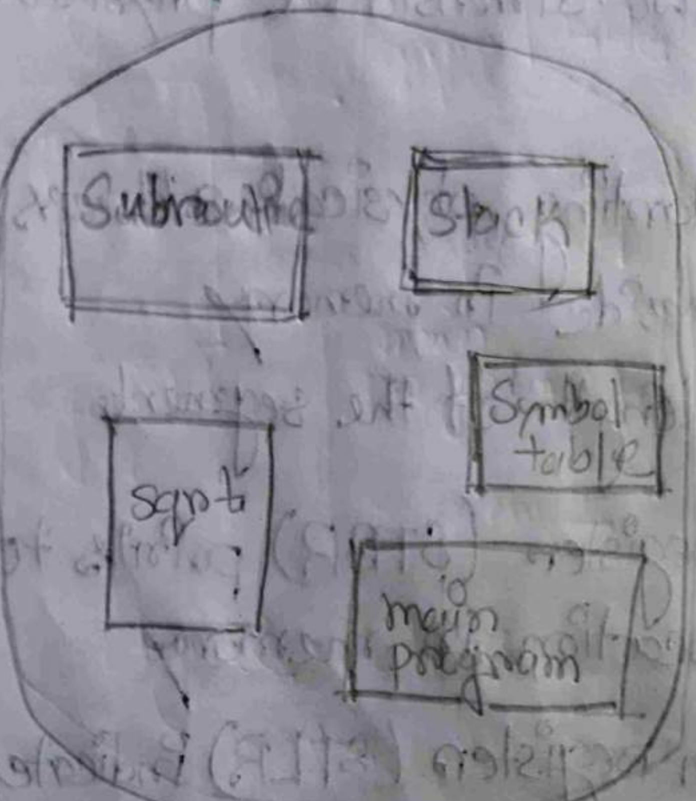
  segment number s is legal if

  $s < STLR$.



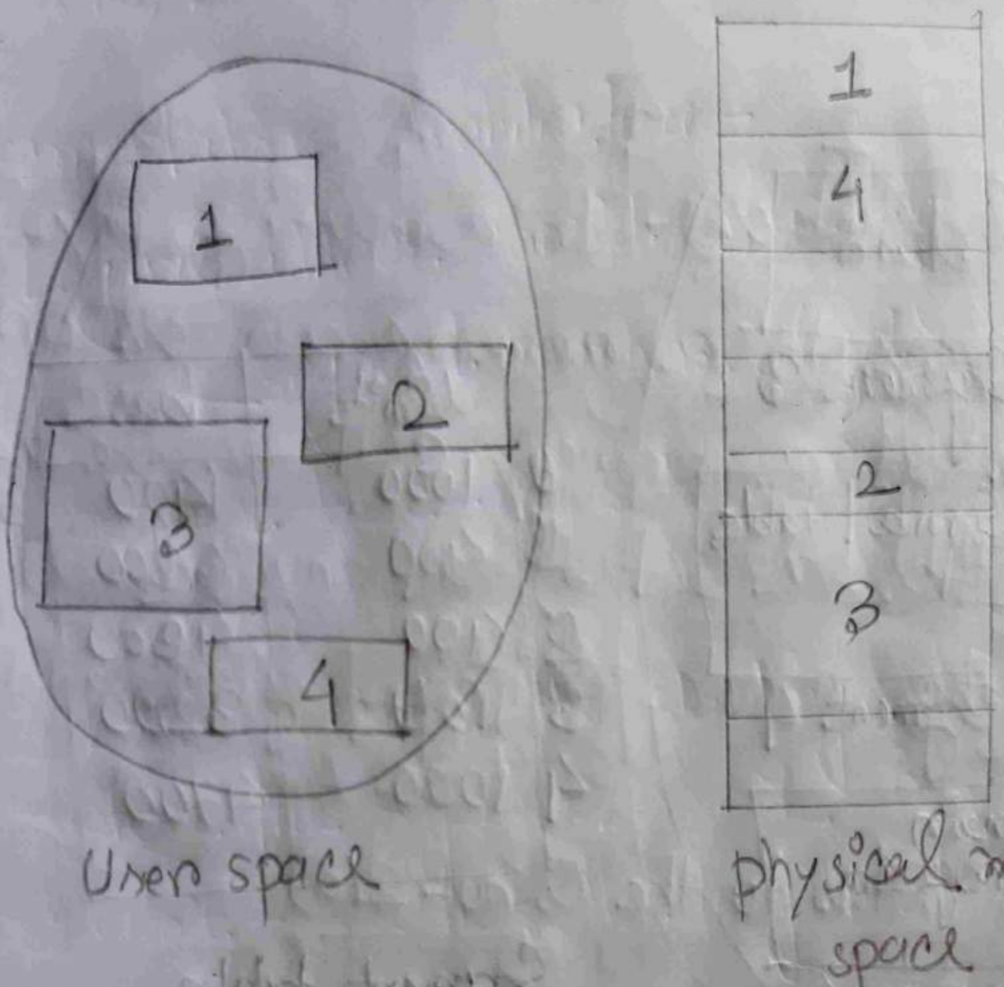
So this blocks range is 100 and its range is 105 so the limit is 5.

User view



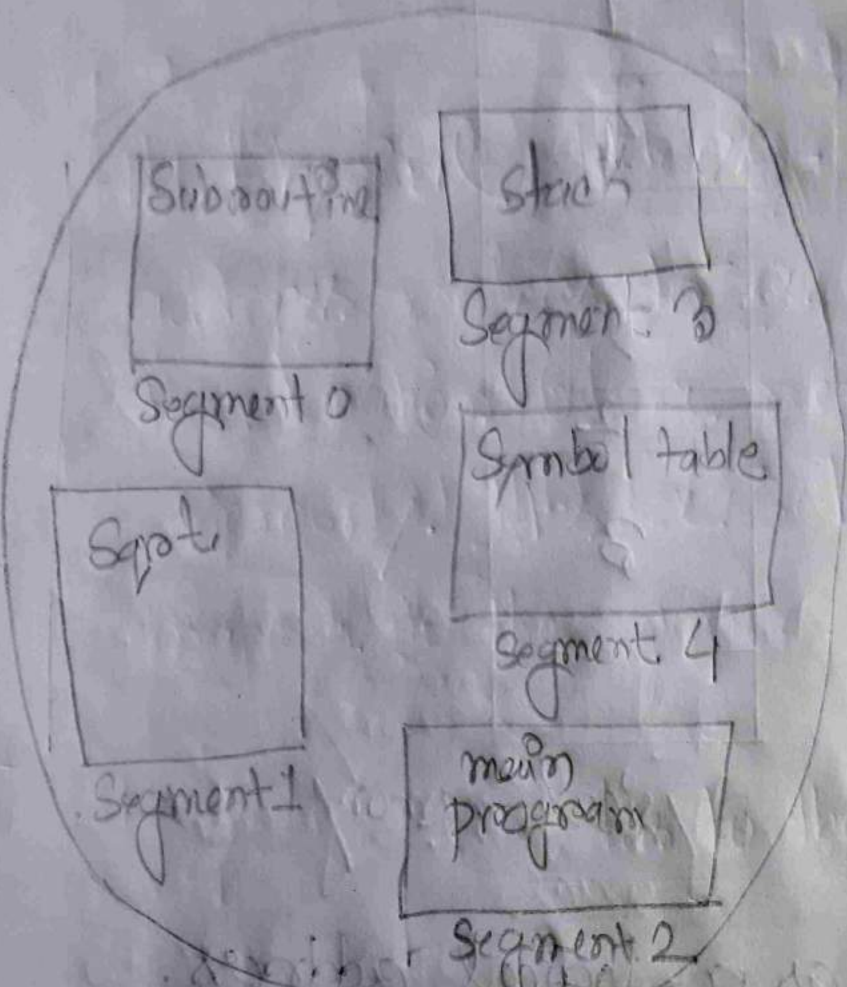
If we denonet sub-routine as 1, ~~Seg~~ <segment name, segment number> <subroutine, 1>. that's how to declare this.

This will restore in logical address and cpu will execute it and store it.



User space is working as a logical address. Cpu will convert or execute the programs and store them in the physical memory as a form of frame. ~~Tus~~ Thus logical address is getting convert to physical address.

Ⅱ Example:-

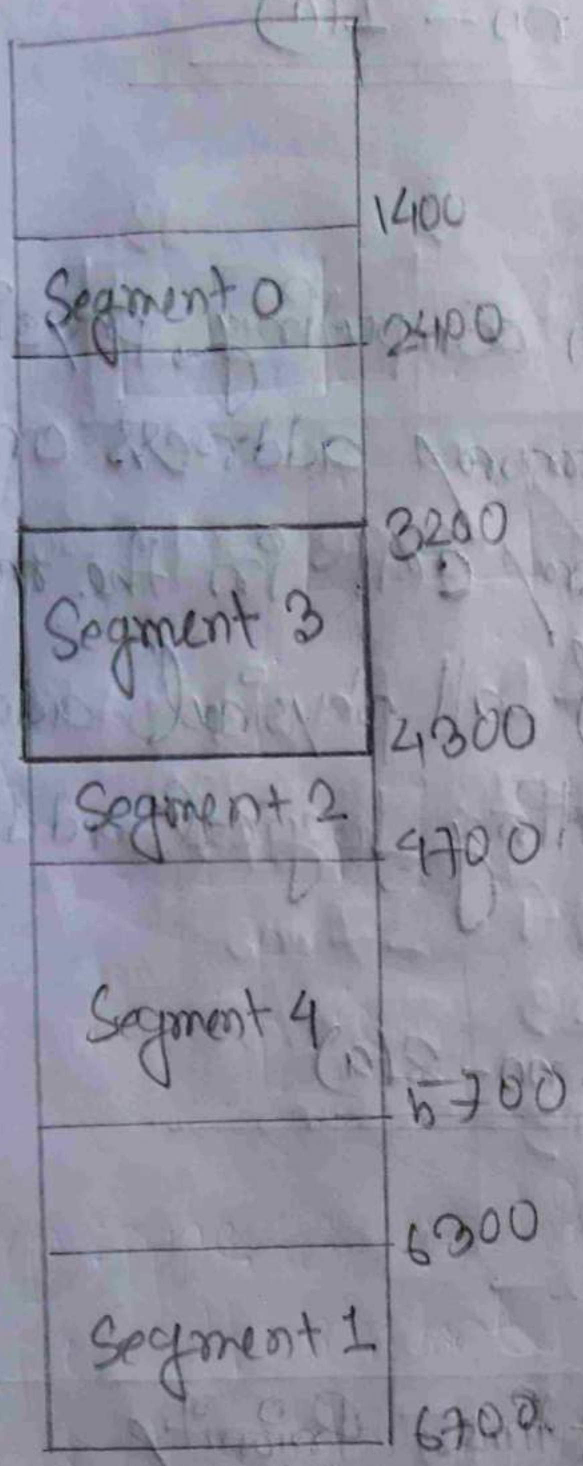


Logical Address space

	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Segment table

Here, limit + base = last address



Physical memory

For Segment 0,

Limit + base = 1000+1400

= 2400

So, 2400 is the last address of Segment 0. Same goes for the othe segments.

Ans to the Q no- 1(c)

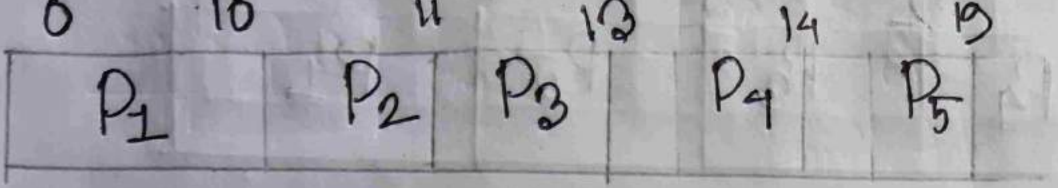
⊞ **Physical Address:-** In computing, physical address refers to a memory address or the location of a memory cell in the main memory. Its a set of all physical address mapped to the corresponding logical address.

Ans to the Q no- 2(a)

⊞

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

i) Gantt chart for "FCFS":-



(ii) Average waiting time:-

P1 = 0
P2 = 10
P3 = 11
P4 = 13
P5 = 14

(0+10+11+13+14)/5

= 9.6 ms

(iii) Average completion time

P1 = 10
P2 = 11
P3 = 13
P4 = 14
P5 = 19

(10+11+13+14+19)/5

= 13.4 ms

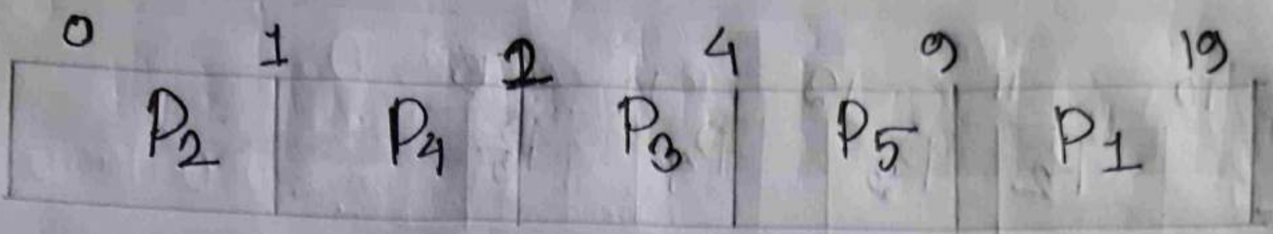
(iv) Average turnaround time

P1 = (10+0) = 10
P2 = (10+1) = 11
P3 = (11+2) = 13
P4 = (13+1) = 14
P5 = (14+5) = 19

(10+11+13+14+19)/5

= 13.4 ms

ii) SJF - nonpreemptive:-



(i) Average waiting time:-

P1 = 9
P2 = 0
P3 = 2
P4 = 1
P5 = 4

(9+0+2+1+4)/5
= 16/5
= 3.2ms

(ii) Average completion time:-

P2 = 1
P4 = 2
P3 = 4
P5 = 9
P1 = 19

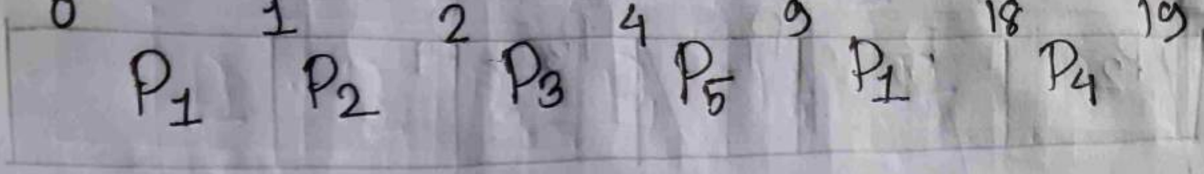
(1+2+4+9+19)/5
= 35/5
= 7 ms

(iii) Average turnaround time:-

P1 = (9+10) = 19
P2 = (0+1) = 1
P3 = (2+2) = 4
P4 = (1+1) = 2
P5 = (4+5) = 9

(19+1+4+2+9)/5
= 7 ms

iii) Priority Preemptive:-



⊙ Average waiting time:-

$P_1 = (9-1-0) = 8$
$P_2 = (1-0-1) = 0$
$P_3 = (2-0-2) = 0$
$P_4 = (18-0-3) = 15$
$P_5 = (4-0-4) = 0$

$(8+0+0+15+0)/5$
$= 23/5$
$= 4.6$ms

⊙ Average completion time:-

$P_1 = 18$
$P_2 = 2$
$P_3 = 4$
$P_{~~2~~4} = 19$
$P_5 = 9$

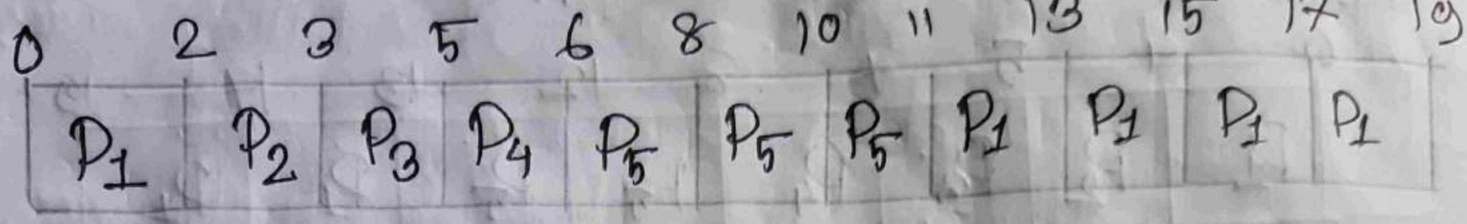
$(18+2+4+19+9)/5$
$= 52/5$
$= 10.4$ ms

⊙ Average turnaround time:-

$P_1 = (18-0) = 18$
$P_2 = (2-1) = 1$
$P_3 = (4-2) = 2$
$P_4 = (19-3) = 16$
$P_5 = (9-4) = 5$

$(18+1+2+16+5)/5$
$= 42/5$
$= 8.4$ms

i.v) Round Routine Quantam-2:-



④ Average waiting time:-

$P_1 = (11-2) = 9$
$P_2 = (2-0) = 2$
$P_3 = (3-0) = 3$
$P_4 = (5-0) = 5$
$P_5 = (10-6) = 4$

$(9+2+3+5+4)/5$
$= 23/5$
$= 4.6$ms

④ Average completion time:-

$P_1 = 19$
$P_2 = 3$
$P_3 = 5$
$P_4 = 6$
$P_5 = 11$

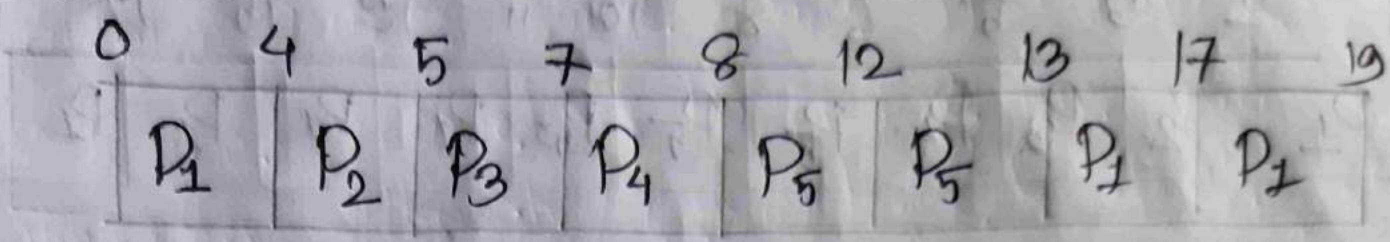
$(19+3+5+6+11)/5$
$= 44/5$
$= 8.8$ms

④ Average turnaround time:-

$P_1 = (9+10) = 19$
$P_2 = (2+1) = 3$
$P_3 = (3+2) = 5$
$P_4 = (5+1) = 6$
$P_5 = (4+5) = 9$

$(19+3+5+6+9)/5$
$= 42/5$
$= 8.4$ms

Round Robin Quantam-4:-



(i) Average waiting time:-

$P_1 = (13-4) = 9$
$P_2 = (~~5~~4 - 0) = ~~5~~4$
$P_3 = (5-0) = 5$
$P_4 = (7-0) = 7$
$P_5 = (12-8) = 4$

$(9+4+5+7+4)/5$
$= 5.8$ ms

(ii) Average completion time:-

$P_1 = 19$
$P_2 = 5$
$P_3 = 7$
$P_4 = 8$
$P_5 = 13$

$(19+5+7+8+13)/5$
$= 8.4$ ms

(iii) Average turnaround time:-

$P_1 = (9+10) = 19$
$P_2 = (4+1) = 5$
$P_3 = (5+2) = 7$
$P_4 = (7+1) = 8$
$P_5 = (4+5) = 9$

$(19+5+7+8+9)/5$
$= 48/5$
$= 9.6$ ms

Ans to the Q no - 2(b)

☐ Multilevel feedback queue schedular:-

In general, a multilevel feedback queue schedulers defined by the following parameters:

(i) The number of queues.

(ii) The scheduling algorithm for each queue.

(iii) The method used to determine when to upgrade a process to a higher-priority queue.

(iv) The method used to determine when to denote a process to a lower-priority queue.

(v) The method used to determine which queue a process will enter when that process needs service.

The definition of a multilevel feedback queue scheduler makes it the most general cpu-scheduling algorithm. It can be configured to match a specific system under design. Unfortunately, it also requires some means of selecting values for all the parameters to define the best scheduler. Although a multilevel feedback queue is the most general scheme, it is also the most complex.

Ans to the Q no - 3(a)

[i] Deadlock characterization:-

Deadlock can arise if four conditions hold stimultaneously.

(i) Mutual exclusion:- At a time only one process can use the resource. Ex:- If some people try to use ~~the~~ an ATM machine only one person can cashout at a time and the others have to wait.

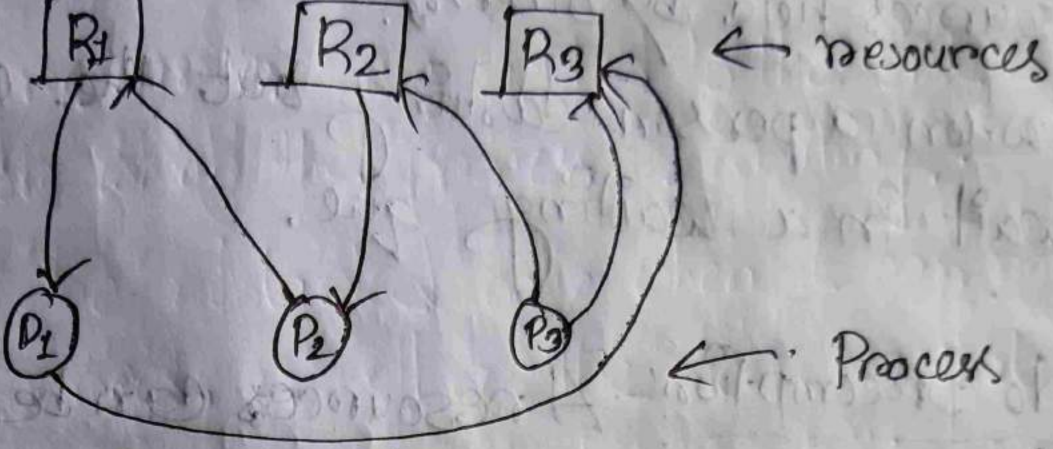
(iv) Hold and wait:- A process holding at least one resource is waiting to acquire additional resources held by others's.

So, when a person cashing out the others has to wait in a waiting que.

(v) No preemption:- A resources can be released only voluntarily by the process holding it, after that process has completed its tast.

That means, only one person can cash out and ~~the~~ if the person volantarity stops it only then the resource can start other process.

(vi) Circalar wait:- There exists a set $\{P_0, P_1 .. P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by $_1$, P_1 is ~~waith~~ waiting for a resource that is held by P_1, .. waiting for a resourse that is held by P_n, P_{n-1} is waiting for a resourse that is held by P_n, and P_0 is waiting for a resource that is hold by P_{0+1}.



Here is visual figure of how hold and circular wait works.

Ans to the Q no-(3) b

Dedlock preventation:-

- Restrain the ways request can be made.

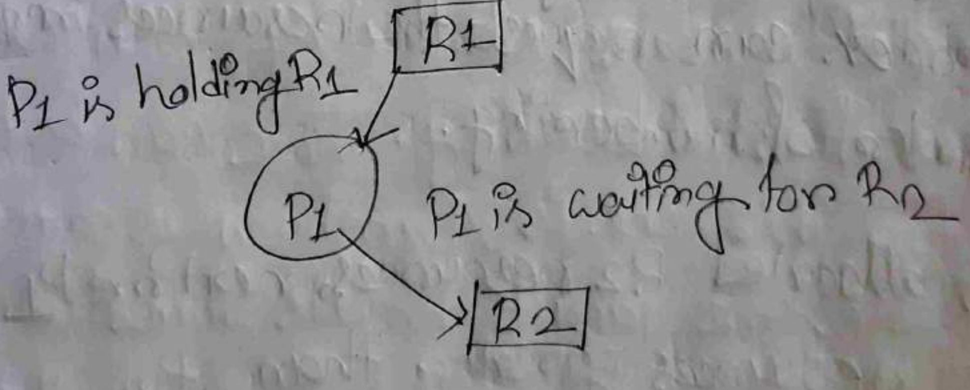
(i) Mutual Exclusion:-

Eliminate mutual exclusion. It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printers, are inherently non-shareable.

③ Hold and wait:-

Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated. Low resource utilization is gonna happen then.

This process will make anew request for resources after releasing the current set of of resources and starvation can happen



④ No preemption:-

If a process that is holding some resources requests another resources that cannot be immediately allocated to it, then all resources currently being hold are released.

Preempted resources are added to the list of resources for which the process is waiting.

Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

(iv) Circular wait:-

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

Ex:- If P_1 is allocated R2 resources, nextime P_1 must allacate resources greater than R_2.

Ans to the Q. no - 3(c)

□ A safe state:-

A state is safe if the system can allacate resources to each process (up to its maximum requirement) in some orders and still avoid a deadlock.

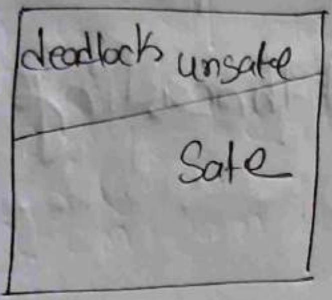


Fig:- Safe state.

Ans to the Q no-5(a)

☐ Multithreading issues:-

Multithreading allow the execution of multiple parts of a program at the same time. This parts are known as threads and are light weights processes available within the process.

Some of the issues with multithreading are:-

Thread cancellation:-

Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is "Asynchronous cancellation", which terminates the target threads immediately. The other is "Deffered cancellation" allows the target thread to periodically check if it should be cancelled.

Signal handling:- Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

Fork() System call:- fork() is a system call executed in the Kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

Security issues:- There can be security issues because of extensive sharing of resources between multiple threads.

There are many issues with multithreading but there are appropriate solutions available for them.

Ans to the Q no- 5(b)

$\boxed{1}$ Semaphore:-

It's a synchronization tool that does not require busy waiting. It's a S- integer variable. Two standard operations modify S: wait () and signal (), originally called P() and V().
It's less complicated and only be accessed ~~by~~ via two indivisible (atomic) operations,

```
(i) wait(S){
      while S<=0
        ;//no-op
      S--;
    }
(ii) signal(S){
      S++;
    }
```

Properties of Semaphore:-

- Its simple and always have a non-negative integer value.
- Works with many processes
- Can have many different critical sections with different semaphores.
- Each critical section has unique access semaphores.
- Can permit multiple processes into the critical section at once, if desirable.

Ans to the Q no – 5 ⓒ

[□] Worst fit:-

Worst fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.