Tamanna Shabnam

ID: 1821801

sec: 01.

Answer to the que no: 01

(a) void atome-dec (int *val)

{ val = compare- and- swap (val * val *(*val) -1))

}

int * compare- and- swap (int *v * int old, int new)

{

ATome ()

int old_v = *v

if (old-v = = old) * v = new;

End- ATome ();

return v; }

value of variable after followin instruction,

atome- set (val, 20); val=20;

atome - add (10, & val); val = 30;

atom e - dec (& val); val = 29;

atome. sub(5 & val; val = 24

finally, value = 24

(b) In peterson's solution we have two shared variables

1) boolen_flag(i): intialized to false, initially no one ~~interes~~ interested in entering critical section.

ii) int turn: The process who turn is to enter critical section

```
do {  ~~flag[i] = True~~;
    flag[i] = TRUE;
    turn = j;
while (flag [j] && turn == j) ; // critical sec
flag[i] = FALSE; // Remaider sec
   }
while (TRUE);
```

Answer to the que no: 02

b)

(b)(i)

process	Arrival time	Burst time	priority
P1	0	11	2
P2	1	7	1
P3	2	4	3
P4	3	5	2

Preemptive SRJF

Gantt Chart

P1	P2	P3	P4	P2	P1

0 1 2 6 11 17 27

0 0

preemptive priority

Gantt chart with time ~~quantum 3~~: quantum: 3:

P1	P2	P1	P4	P1	P1	P1	P3

0 1 8 11 14 17 19 23 27

Round Robin

Gantt Chart with time quantum 3:-

P1	P2	P3	P4	P1	P2	P3	P4	P1	P4	P2

0 3 6 9 12 15 18 19 22 24 25 27

ii) preemptive SRJF :-

Av waiting time = $\{(7-1)+(11-2)+(4-2)+(6-3)\}/4$

$= \frac{28}{4} = 7$

Avg response time = $\{(0)+(1-1)+(2-2)+(6-3)\}/4$

$= 3/4 = 0.75$.

preemptive priority:

Avg waiting time $= \{12+0+21+17\}/4 = 50/4 = 12.5$

Avg response time $= \{0+(1-1)+(23-2)+(11-3)\}/4$

$= 29/4 = 7.25$

Round Robin

avg waiting time: $(16+15+13+16)/4 = 60/4 = 15$

avg response time $= \{0+(3-1)+(6-2)+(9-3)\}/4$

$= 12/4 = 3$

minimum average waiting time is

preemptive SRJF = 7

minimum average response time is

preemtive SRJF = 0.75.

Answer to the que no: 03

(a)

1. signal state :- (i) It ~~it~~ indicates that a resource is available for a process or thread.

ii) signaled state object will not cause any thread and will wait on the object to block.

iii) It has capacity to ~~not~~ relase the threads.

2. Non-signaled state :-

i) It indicates resources is in use.

ii) Non-signaled state will cause any thread that waits on that object to block untill the object becomes signaled

iii) will not release any thread.

(b) A low priority process blocks execution of high priority process by keeping of it's resources by a phenomenon known as priority inversion.

Scanned with CamScanner

Example:-

Let, P1, P2 and P3 are respectively highest in between highest and lowest priority.

a) P3 becomes ready and enters critical region, reserving shared resources.

a) P2 becomes ready and preempts P3.

a) P1 becomes ready and will preempts P2 and start to run only untill reaching critical section.

P1 will continue, P2 must be finished and allow P3 to resume and finish It's critical section. only P3 is finished then P1 can resume.

Overcoming priority Inversion:-

1) priority ceiling:

2) Disabling interrupts:

3) priority inheritance.

4) No blocking.

5) Random Bosting.

1 c

c) peterson's solution is not guranted to b work on modern computer due to vagaries of loads and stone operations petersons solution entry section for process.

flag[i]= true; //stone instruction
fork[j]; // stone instruction.
while (flag[j]= = True) // load instruction

Since flag[i] and flag[j] refer to different main address, their respective stone and load instruction can be reversed like,

turn= j;
while(flag[j]= true && turn= j);
flag[i]= True;

same for i

intially flag[i] and flag[j] were false and hence both will be able to execute this critical section at the same time. This violates the mutual exclusion requirment.

3

(d)

~~RS~~ POSIX semaphores can be named and unnamed.

Named semaphores are like process-shared semaphores except that named semaphores are referenced with a pathname rather than a pshared value.

On the other hand unnamed semaphores are allocated in process memory and initialized. It might be usable by more than one process, depending on how to semaphore is allocated and initialized.

(e)

I) There are 3 unique process will be created.

II) There are 5 unique threads will be created.

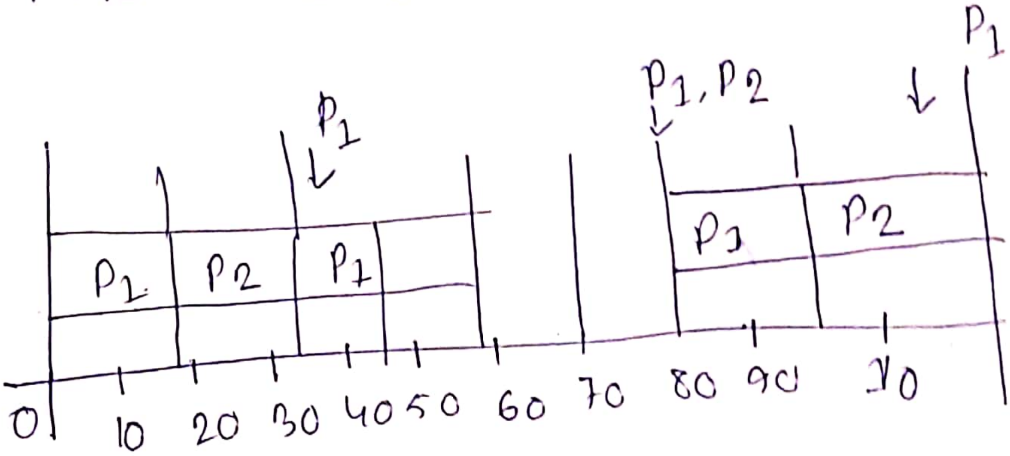
Answer to the que no : 04

a) i) Two processer are P1 and P2

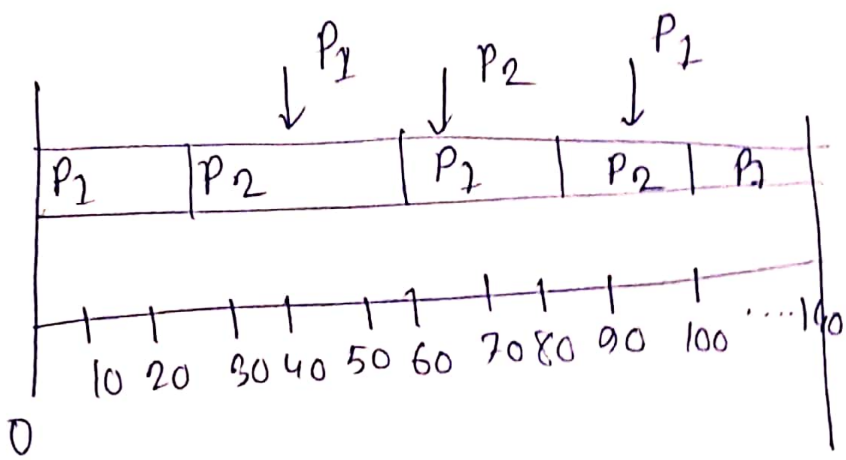
$P_1 = 60, t_1 = 30$

$P_2 = 80, t_2 = 35$

RMS → Gantt chart



EMS - Gantt chart



ii) since $P_1 < P_2$ priority of $P_1 > P_2$. P_1 runs first & it completes its cpu write a 25 time unit them P_2 start to run till the time unit 50. P_1 is available to run then pre emption is done and P_1 stards to run after it finish.

6) The dining philosopher problem states that there are 5 philosophers staring a circular table and they eat and think alternatively there is a bowl of rice for each of them may only eat if there are both chopsticks their right & o left chopsticks to eat. A hungry one may only eat if there are both. A solution of Dining ~~Phil~~ philosopher problem is to use a semaphore to represent a chopstick. A a chopstick can be picked by executing a unit option on the samaphore and released by executing a signal semaphore.

The structure of chopsticks is shown below,

The structure of the random philosopher is given as follows.

do { wait (chopsticks [i]);

wait (chopsticks (i+1) %5);

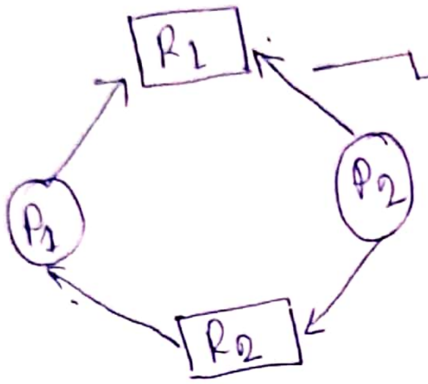
Eating the rice.

}

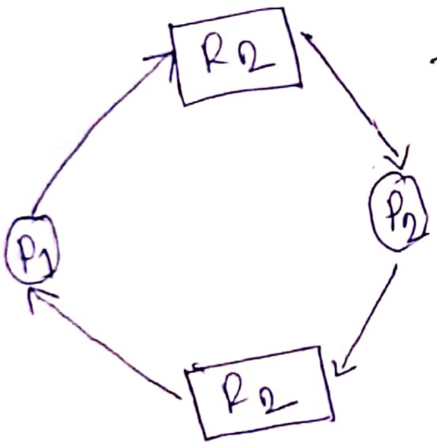
(e) Race condition, thus far we have paid a little attention to the problem concurrency A lockfree algorithm guaranted forward Progress in to finite. C++11 provides a new memory mode ordering. There is a very easy to miss race condition. that went unnoticed in our signal store. A race condition of an electronic software or other system. A race condition occurs when an input has two transition in less.

Answer to the que no: 05

a)



Now converting this clam edge to convert edge



now this will result into dedlock as P1 is holding R1 and writing for R2 and P2 is holding R2 and waiting for P1.

(b)

1) Banker's Algorithm:

	Allocation	Max	Need	Available
	ABCD	ABCD	ABCD	ABCD
				1533
P_0	0012	0012	0000	
				24 12 13
P_1	1000	1050	0650	2887
P_2	1354	2355	1001	2 14 11 9
P_3	0632	0652	0020	2 14 12 13
P_4	0014	0156	6142	

Need = max - Allocation.

II) Now to check system is in safe state or not first allocated resource and execute the process. first wants 0000 execute. Now availabe will be, The resource allocated to Po is free, the Available is 0012+1533 new Available 1593

Scanned with CamScanner

Next P_1 cannot be executed as

Needs > Available

0650 > 1533

Execute P_2 as 1001 < 1533

New available → 1533 + 1354

→ 2887

New available → 0632 + 2887

2 14 11 9

Now at the end execute P_4

0 7 4 2 < 24 11 9

New available 0 0 1 4 + 214 11 9

2 14 12 13

Now P_1 can be executed

Hence the system is in safe state.

iii) yes the request can be granted immediately as the

Need < Available

1910 < 1533 .