1.

Students(){

//arrive at store

sleep();

//play if machine is available

P(machine)

//play

random time sleep()1:24 PM

V(supervisor)

P(mutex);

groupTotal++;

if(groupTotal%2!=0){V(mutex), P(group)}

else{ V(mutex), V(group)

}

for Supervisor the following code

Supervisor(){

while(true){

P(supervisor);

/\*assign machine to first student in queue\*/

V(machine)

}

group semaphore is counting

with value 0

init

and mutex is binary with value = 1

2.

No deadlock is not satisified.

If every philosopher takes the right chopstick, there will be no more chopsticks. There will be no left chopstick.

**alternate solution:**

binary semaphore mutex = 1;

while(true){

think();

P(mutex);

P(S\_leftch)

P(S\_rightch)

V(mutex)

eat(); //and so on}

No starvation is satisified for bin sem with queue.

This is because queue is FCFS. There is a specific order of execution, so the philosopher who is in the queue will be forced to pick up his chopstick and finish his execution.

No starvation is not satisfied with spin lock.

This is because no specific way for a processor (philosopher) to force other philosopher out of busy wait.

Other philosophers (processors) can use the chopstick faster than the other philosopher. CS.

3.

COUNTING SEMAPHORES: S1, S3, S6

BINARY SEMAPHORES: S2, S4, S5

S1=0, S2=1, S3=3, S4=0, S5=1, S6=0

1) P1, P(S3) S3 = 2

2) P2, P(S5) S5 = 0

3) P3, P(S1) S1 = -1 S1 Q(P3)

4) P5, P(S4) S4 = 0 S4 Q(P5)

5) P2, V(S5) S5 = 1

6) P1, P(S6) S6 = -1 S6 Q(P1)

7) P2, P(S4) S4 = 0 S4 Q(P5,P2)

8) P6, V(S3) S3 = 3

9) P4, V(S4) S4 = 0 S4 Q(P2) P5 has higher priority

10) P4, P(S6) S6 = -2 S6 Q(P1,P4)

11) P1 V(S5) S5 = 1

But alternatively worst case scenario she might ask in reverse i.e.

3.

Binary Semaphores: S1, S3, S6

Counting Semaphores: S2, S4, S5

S1=0, S2=1, S3=3, S4=0, S5=1, S6=0

(also change the values assigned to semaphores)

4.

a.

So on trying to serialize the variables, we come to know that there happens to be a deadlock when we do the following:

binary semaphore Sx = Sy = Sz = 0;

Thread A{ Thread B{

write z; P(Sy);

V(Sz); read y;

P(Sx); write x;

read x; V(Sx);

write y; P(Sz);

V(Sy); read z;

} }

P(Sx) will wait till it gets the signal whereas P(Sy) will wait till it gets the signal. Now, the signal won’t come because, according to the structure of the code, Sx gets the signal after Sy gets the signal and writes to y and similarly Sy get the signal after Sx get the signal and writes to x. Therefore we have seen then both the semaphores are waiting on each other to open up. So the complete serialization is impossible.

Partial Serialization:

b.

According to the code, when the reader exits the DB, it executes the P(mutex) command. On assumption that mutex is initialized to 1, the reader process will change it back to 0 and won’t release it if readerCount is not 0. So the next reader process that try to execute mutex and leave the DB, will enter the waiting queue thus building up the queue. The writer won’t be able to get into the DB since none of the readers are able to get out of the DB.