Q1 (28 points)

Students go to a store to play video games. If there's no free game machine, student will wait until the supervisor assigns a machine for him/her, otherwise she/he will just take one of the free machines. When any machine becomes free, the supervisor will let the first student on line use the machine. After a student finishes, s/he releases the computer and waits until another one is done. When the group (of two) is formed, they leave.

Initially all the game machines are available. The number of machines is numMachines=5. Using semaphores and operations on semaphores, synchronize the 2 types of threads (student and supervisor). Roughly, before synchronization, a possible execution in pseudo-code might be:

Student: Supervisor:

arrive at store //napping while(true)

play //if machine is available {

form group assign game machine

leave }

Student: Supervisor:

arrive at store //napping while(true) {

if(numMachines > 0) {

P(Mutex)

numMachines --;

V(Mutex)

P(Mutex)

P(NumMachines)

play //if machine is available P(Mutex)

if(numMachines <1) {

P(NumMachines)

assign game machine

}

V(Mutex)

}

V(NumMachines)

}

else P(NumMachines)

P(Mutex2)

group++

If(group %2 ! = 0) P(Group)

else {

Form group;

Leave

}

V(Mutex2)

Binary Semaphore Mutex = 1;

Binary Semaphore Mutex2 = 1;

Binary Semaphore Group = 0;

Counting Semaphore NumMachines = 5;

Int group = 0;

Q2 (16 points)

Consider the Philosopher problem. Discuss the NO STARVATION and NO DEADLOCK conditions. The used semaphores are binary semaphores with queues.

No-Deadlock

1. The maximum number of Philosopher who pick up fork at the same time is 4.
2. The neighbor of the Philosopher who picks up 2 forks can’t pick up fork.
3. The P1.P3.P5 pick up their left side fork and then pick the right side if they can do it. After P1.P3.p5 pick up fork, P2.P4 pick up their right side fork and then pick their left side fork if they can.

(semaphore can’t sure that there is NO STARVATION and NO DEADLOCK)

Q3 (16 points)

After each step, give the value of the updated semaphore and the content of the updated

semaphore queue.

COUNTING SEMAPHORES: S1, S3, S6

BINARY SEMAPHORES: S2, S4, S5

Semaphore queues use Priority scheduling algorithms where low PID means low priority.

Semaphore initial values:

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 S1Q(P3)

4) P5, P(S4) S4 = 0 S4Q(P5)

5) P2, V(S5) S5 = 0 S4Q(P2)

6) P1, P(S6) S6 = -1 S6Q(P1)

7) P2, P(S4) S4 = 0 S4Q (P5, P2)

8) P6, V(S3) S3 = 3

9) P4, V(S4) S4 = 0

10) P4, P(S6) S6 = -1 S6Q(P4)

11) P1, V(S5) S5 = 1

Q4 (30 points)

(A) If it is possible, implement, with a minimum number of semaphores, a complete

serialization (for all variables) between READ and WRITE, such that the READ operation is always done after the WRITE operation. If a complete serialization isn't possible for all the variables, give the solution for a partial serialization and specify for which variables the serialization is impossible. For a partial serialization, consider as more important:

1) higher concurrency (vs minimum number of semaphores)

2) minimum number of semaphores (vs higher concurrency)

Specify the number, type, and initial values of the necessary semaphores.

ThreadA ThreadB

{ {

write z; read y;

P(X)

read x; write x;

V(X)

write y; read z;

} }

Reader Writer issue, there are two strings trading data through a settled size cradle.

.........................................................................................................................................

(B) Reader-Writer problem (correct code)

reader() writer()

{ {

while(true) while(true)

{ {

P(mutex); P(OKaccessDB);

readerCount++; accessDB;

if(readerCount==1) V(OKaccessDB);

P(OKaccessDB); }

V(mutex); }

accessDB;

P(mutex);

readerCount--;

if(readerCount==0)

V(OKaccessDB);

V(mutex);

}

}

What would be the outcome of replacing (in reader)

FROM:

if(readerCount==0)

V(OKaccessDB);

V(mutex);

TO:

if(readerCount==0)

{

V(OKaccessDB);

V(mutex);

}

DeadLock, Or Mutual Exclusion is violated.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

EXTRA CREDIT (6 points)

Processes 0,1,2,3,4,5,6 are executing concurrently.

Process 3, 4, 5, and 6 must start their execution after the execution of Process 0, 1,

and 2 ends.

There is no enforcement of order between the executions of Process 3, 4, 5, and 6.

There is no enforcement of order between the executions of Process 0, 1, and 2.

Implement the required synchronization using a minimum number of binary semaphores. Give the initial values of each semaphore.

ILLUSTRATION:

P0, P1, P2 point to a node.

coming out of the node are P3, P4, P5, P6.



The minimum number of counting semaphore (P3.P4.P5.P6)

(any integer)

The minimum number of binary semaphore (P0.P1.P2)

(1,0)