CS340 Test 2

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 s/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 }

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

Q2 (16 points)

Consider the Philosopher problem. Discuss the NO STARVATION and NO DEADLOCK conditions.

The used semaphores are binary semaphores with queues.

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

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)

2) P2, P(S5)

3) P3, P(S1)

4) P5, P(S4)

5) P2, V(S5)

6) P1, P(S6)

7) P2, P(S4)

8) P6, V(S3)

9) P4, V(S4)

10) P4, P(S6)

11) P1, V(S5)

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

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;

read x; write x;

write y; read z;

} }

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

(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);

}

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