Concordia University
Department of Computer Science

18.5/20 V. Good

Comp 346, Winter 2008
Instructor: Aiman Hanna
Time: 60 minutes

Mid-term Exam

Name

ID. #:

- **Keep your answer very organized & clean.**
- **Exam will be marked out of 20. Exam has 5 pages.**

Question # 1 (3 marks)

A) Which of the following operations should be allowed under user mode, and which must only be allowed under supervisor mode? Clearly explain your reasons:

i) Context switch

☐ User Mode ☑ Supervisor Mode

Reason: because then each process would want to make another process start context switch so they can control the CPU all to themselves.

ii) Reading memory

☑ User Mode ☐ Supervisor Mode

Reason:

Because each process/thread must be able to read memory of it's own to be able to execute the binary program.

3/3

iii) Acquiring a lock

☑ User Mode ☐ Supervisor Mode

Reason:

the lock is implemented as a class/object which is software and different processes/threads can lock if the lock is taken or not through if a while statement

iv) Releasing a lock

☑ User Mode ☐ Supervisor Mode

Reason:

the lock is implemented as a class/object so when you want to release the lock, you just make the lock = f. The lock is just a boolean.

this technique for mutal exclusion works because it create a new child thread so that the parent & child are me

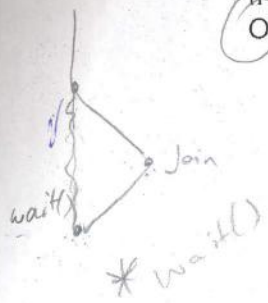
only problem ↳ kill parent ∴ kill child

Question # 2 (3 marks)

Indicate whether each of the following statements is *true* or *false*. **Explain** your answer.

(A) Using *Join-fork* technique around critical sections will provide mutual exclusion even if the system is a multi-processor system.

OTrue ●False ↳ each process has its own thread & they are me.



because you cannot use joint-fork operations when there is two different processors. Using it on 1 processor will only affect that 1 processor and not affect the other.

The parent says join -> a parent will wait for it's child to finish, otherwise the parent may finish before the child & exit.

(B) Using interrupts may result in more system overhead (in terms of time) compared to polling.

OTrue ●False

2/3

Because polling would waste a lot of computer time because the CPU keeps checking if the device is busy while in interrupts, the CPU gets to do other things and the device will let the CPU know when it has finished.

(D) It is not possible to design a single system that can act as both timesharing and batch systems.

OTrue ●False

Timesharing refers to having more than 1 user at a time. By having many users and each user using a batch system then timesharing and batch systems would be possible.

Question # 4 (6 marks)

A) Assume a system only uses binary semaphores. Should the V() operation be implemented as atomic? If *yes*, give a scenario that shows how a failure may occur otherwise. If *no*, explain clearly why this operation need not be atomic. Would your answer change if you have counting (general) semaphores? Why or why not? Explain clearly.

6/6

No, For binary semaphores, you don't need to implement atomicity for V() but for general semaphores, it needs V() to be atomic.

Let S be a semaphore: V() { S=S+1; }

V() will only execute when S=0, in a binary Semaphore so if it gets interrupted anywhere in the instruction S=S+1; then it's not a problem because nobody else (process/thread) can go since S=0 so the control will come back to the function to make S=1.

For general semaphores, it must be atomic because let's say S=3, process 1, 2 exists. P1 does P() then gets the V(), it does S=2+1; then gets interrupted before S becomes 3, P2 goes and does P(S) so S=1 then gets interrupted, control goes back to P1 and P1 makes S=2+1=3; which is wrong.

B) Using semaphores, give the code for each of processes P0, P1 & P2. The processes must always work in one of these two orders: P2, P0 then P1, or P2, P1 then P0. Any other order is not allowed. Assume that whatever the processes need to perform is coded inside a code segment called <*Phase I*>.

→ **Declare & Initialize your semaphores here:** S = 0

P0	P1	P2
P(S); <phaseI> V(S);	P(S); <phaseI> V(S);	<phaseI> V(S);

Question # 5 (4 marks)

Here is the code of 4 processes {P0, P1, P2 & P3}, which share a set of semaphores and some critical sections. Assume that at any point of time there will never be any duplicates of the same process in the system (for example, no 2 *P3s* can be there at the same time). Additionally, each process will run only one time. Does the code provide mutual exclusion to each of C.S.1, C.S.2 and C.S.3? Explain your answer very clearly for each of the critical sections.

Note: We are not interested in looking at any other issues beside mutual exclusion.

Semaphore s1 = 1, s12 = 0, s23 = 0, s3 = 0;

P0	P1	P2	P3
P(s1) C.S.1 V(s23) P(s3) C.S.2 V(s23) V(s3)	P(s3) C.S.3 V(s1) P(s23) C.S.2 C.S.3 V(s23)	P(s23) C.S.1 V(s12) V(s3)	P(s12) C.S.1 C.S.2 V(s1) P(s3) C.S.3 V(s3)

4/4

CS1 : Mutual exclusion is provided for C.S.2 because P0 will start and it must get out of C.S.1 before any other process thinks of starting. P3 is waiting for s12 to = 1 but only P2 provides V(s12) which is often getting out of C.S.1. So the only way mutual exclusion gets violated for C.S.1 is if P1 does V(s23) while P0 is in C.S.1 which is impossible because P1 is waiting for P0 to do V(s3) or V(s23) (so P2 can go do V(s3)) but this only happens when P0 left C.S.1

CS2 : Mutual exclusion is NOT provided for C.S.2

Scenario: P0 starts, executes P(s1), C.S.1, V(s23) gets interrupted.
P2 starts, executes P(s23), C.S.1, V(s12) V(s3)
P3 starts, executes P(s12), C.S.1, goes in C.S.2 and gets interrupted.
P0 comes back, does P(s3) and goes in C.S.2
Both P0 and P3 are in C.S.2 => Failure

CS3 : Please look at the back of this page →

C.S.3: Mutual exclusion is provided for C.S.3 because when P0 s/ents and does P(s1), C.S.1 then V(s23), this will allow P2 to run until it finishes so s3=1, P3 runs the whole way but gets interrupted right before P(s3). So now, the scenario is P2 is done. P0 is at P(s3) P1 didn't s/ent (is at P(s3)) and P3 is at P(s3) as well. This means that only one of P3 or P1 will go to C.S.3 because the only way they are both in C.S.3 at the same time is if V(s3) gets executed by P0 without P0 using P(s3) first which is not the case since P0 needs to do P(s3) to get to V(s3) first so only one of P1 or P3 will be in C.S.3 at any one time.