Concordia University
Department of Computer Science & Software Engineering

17.5 / 20 good

Comp 346, Fall 2012
Instructor: Aiman Hanna
Time: 60 minutes

Mid-term Exam

Name: Mela Setijoso

ID. #: 2732491

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

Question # 1 (3 marks)
Indicate whether each of the following statements is *true* or *false*. **Explain** your answer.

i) If a system is guaranteed to be a single-user, single-CPU system there is no real need for the *mode bit*.

OTrue [✓ ●False]

Mode bit is required to differentiate what user and system can to such as user mode can't touch OS code, but supervisor mode can. Even though it's single user, the user need supervisor mode to do contol calls on OS.

1

ii) Assume a *time-sharing* system does not allow multitasking; that is, for each user it is simply a uni-programming environment. Under these particular conditions, there is no possibility of deadlock in that system.

OTrue [✓ ●False]

deadlock can still occur because processes are still in these system and there's a possibility a lock can't be unlocked. so the next process can't function.

1

iii) In a uni-programming system, there is no possibility of Deadlock.

OTrue ✗ ●False

~~Same answer as above~~

Question # 2 (5 marks)

As a team member of a system designers, you investigated the use of system calls in the system and concluded that they are actually prone to produce security problems (improper privilege escalation to system-mode) when calls are made to 8 particular routines that allow different controls (manipulations, use, etc.) to some hardware components of the system. Since the fixing of these routines is estimated to take longer time that what can be accommodated by the clients, another team member recommended a temporary solution to the problem, where these 8 routines are actually permitted to be executed under user-mode instead of system-mode, hence eliminating the privilege escalation problem. Does this proposal have any validity from your point of view? If *yes*, indicate clearly the reasons behind your support to the proposal. If *no*, explain clearly why such proposal is incorrect.

No because changing permission from system to user mode can create ~~more~~ problems than eliminating problems. For example, user will be able to change OS code of those controls. This can make the controls do something else than its original purpose or the controls will not work anymore (bug is found). ok

[illegible]

$\bigcirc{5/5}$

- user has control of it, can destroy what [illegible] manipulate hardware → dangerous
- don't say "use message passing" → shows you memorise stuff not know what to do

Scenario: Code is read only then user stays user mode

3 CS
6 P share CS

Question # 3 (5 marks)
Assume the following particular conditions/circumstances: A system has only 3 critical sections (C.S.), and only 6 processes actually share these critical sections. There are exactly 6 routines (methods) where one or more of these critical sections are being accessed. If more than one C.S. appears in a method, then the order of appearance is undetermined (that is, they can appear on any order). Further, we are only interested in achieving mutual exclusion (M.E.) and deadlock-free. We are *not* interested in good progress. Is it possible under these very particular conditions to implement a monitor as follows:

- All 6 routines are implemented as part of the monitor;
- 3 hidden semaphores (say *mutex*$_1$, *mutex*$_2$ and *mutex*$_3$) are used, one for each of the critical sections;
- At the start of each method, *P(mutex*$_i$*)* is automatically injected if the C.S. for that semaphore appears in the method (which is easy to detect). At the end of the method, *V(mutex*$_i$*)* is automatically injected to release the semaphores obtained at the start of the method.

Under these very particular conditions, is it possible to implement such a monitor to allow for high-level synchronization. If *yes*, explain clearly why this implementation is solid (reliable) in achieving what is needed. If *no*, show clearly how this implementation will fail.

```
P(mutex1)      P(mutex2)      P(mutex3)
   CS1            CS2            CS
V(mutex1)      V(mutex2)      V(mutex3)
```

Yes it is possible because mutex1, mutex2 and mutex3 provide a single lock for each CS. So if there was multiple CS1, V(mutex1) will signal to any P(mutex1) when lock is free. Process can only enter after that signal, which will avoid multiple entries. Since the lock will eventually open at one period, deadlock isn't possible since that only occurs ~~if the~~ key becomes unavailable. !!

what about M.E. !!

3.5/5

lock = F

Question # 4 (4 marks)

The following code is designed so that mutual exclusion over critical sections (C.S.) is achieved. A process calls *getLock()* prior to accessing the C.S. and calls *releaseLock()* after the C.S. execution is finished. *lock* is defined as a Boolean variable and initially assigned a *false* value (i.e. *Boolean lock = false;*). *D.I.* and *E.I.* denote Disable Interrupt and Enable Interrupt respectively.

Is the solution capable of providing the needed mutual exclusion? If *yes*, explain why the solution holds. If *no*, give a example/scenario that shows how the solution fails.

```
getLock()                                releaseLock()
{                                        {
while(true) {                              D.I. ;
   D.I.;                                   lock = false;
   if(lock == false)                       E.I.;
   { E.I.;                               }
     break;
   }
  yield;
  } // end while(true)
lock = true;
} // end getLock()
```

interrupt is enable so smtg can go wrong

getlock
CS1
releaseLock

~~Yes~~, there is mutual exclusion because the lock will prevent any other process going into the CS. The lock will only release after CS is done and process are put into ~~yield~~ yield while it's trying to get lock.

2/4

The boolean type also makes it possible to only have 1 lock, so there's no way for duplicate locks, which can let other processes to in the CS.

what is the case if interrupt happens before lock = true

Question # 5 (5 marks)

Here is the code of 4 processes {P0, P1, P2 & P3}, which share a set of binary 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, <u>each process will run only one time</u>. <u>Does the code provide</u> mutual exclusion to each of C.S.1, C.S.2 and C.S.3? <u>Explain your answer very clearly</u> for **<u>each</u>** of the critical sections.
<u>Note</u>: We are <u>not</u> interested in looking at any other issues beside mutual exclusion.

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

$S_1 = \cancel{1}, \cancel{0}, \cancel{1}, 0$
$S_{12} = \cancel{0}, \cancel{1}, \cancel{0}, 1$
$S_{23} = 1,$
$S_3 = \cancel{0}, \cancel{1}, \cancel{0}, 1, 0$

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

Scenario 1:

P_3 : $S_1 = 0$, CS3, $S_{12} = 1$, $S_3 = 1$
P_0 : $S_{12} = 0$, CS3, CS1 ✓ interrupt
P_2 : $S_3 = 0$, CS2, $S_1 = 1$, $S_{23} = 0$, CS1

CS2 is ME cos it's protected by P(S3) first so no way ~~to~~ for ME

Scenario 2:

P_3 : $S_1 = 0$, CS3, $S_{12} = 1$, $S_3 = 1$
P_0 : $S_{12} = 0$, CS3 ✓ interrupt
P_2 : $S_3 = 0$, CS2, $S_1 = 1$
P_1 : $S_1 = 0$, CS3 ✓

5/5

No ME for CS1 due to scenario 1, No ME for CS3 due to scenario 2, and ME ✓ for CS2