15/20 good

Concordia University
Department of Computer Science and Software Engineering

Comp 346, Winter 2009
Instructor: Aiman Hanna

Mid-term Exam

Nar
ID.

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

Question # 1 (5 marks)

(A) What are the major differences between *batch* system and *timesharing* system? Is it possible to design a single system that is capable of behaving as both? If yes, explain how this can be achieved. If no, explain why such configuration is infeasible.

Batch executes one after another

In a batch system a set of processes are executed one at a time in sequence. Time sharing allows for processes to access & run at the same time (& multiple users). Therefore, by definition it is not possible to have both a batch system & time sharing in place.

(B) Assume a system that implements only processes (no threads), and that only one single copy of each process is to be executed at a time (no two or more of the same process can run at the same time). Under this specific configuration, is it possible to have deadlock in this system? If *yes*, give at least one scenario that illustrates how deadlock may occur. If *no*, explain clearly why deadlock cannot occur under the given conditions.

No, it is not possible to have deadlock under these conditions. Each process is unique and has it's own set of resources, therefore, there will never be a time where a process is waiting on another process (or code section) to finish using a given resource. Also, the processes can only run one at a time (there is only one single copy of each process) and even if they were to run together, deadlock wouldn't be an issue as deadlock occurs when two or more instances of a same process or Threads are waiting for a resource in a circular fashion. (eg. T_1 waits on T_2 and T_2 waits on T_3 and T_3 waits on T_1)

DMA → Direct Memory Access.

Question # 2 (3 marks)
Briefly explain what is the difference between interrupts and polling? Now assume a system with a DMA controller. Will such system still be able to utilize polling? If *yes*, give a detailed scenario (i.e. step-by-step example) that shows how polling and DMA can be used together. If *no*, explain clearly the main reasons behind such infeasibility.

The difference between interupts and polling is that in polling the CPU keeps checking if a device/process is ready where as in utilizing interupts a signal is sent to the CPU when the device/process is ready. The system if using a DMA controller will not be able to still provide polling, the DMA controls direct memory access

Question # 3 (2 marks)
What is the difference between the *many-to-one* and the *one-to-one* thread models. Which of the two models is a better; explain why? Are there any disadvantages with the better model? If *no*, explain briefly why this model is ideal. If *yes*, indicate what these disadvantages are.

The difference between the many-to-one and one-to-one thread models is that in the many-to-one, a single thread can branch out in to many threads, while in a one-to-one model a process can break in to threads but the threads are individuals & do not branch futher into sub threads. The one-to-one model is better as it would be easier to provide and track mutual exclusion & avoid deadlock

Question # 4 (5 marks)

A) Assume a single-CPU system that runs extreme time-sensitive processes (processes that must execute as fast as possible), and that all critical sections in these processes can be protected by binary semaphores. Further, the system does not provide Test-and-Set instruction. As the implementer of semaphores on that system, would you still keep the P() and V() operations as atomic? If *yes*, provide clear scenarios to show that the system will fail if the operations are not atomic. If no, explain the motives behind your decision and why it is still safe to have the operations as non-atomic under the current specification. Your answer must be clear for both P() and V().

In this case, as the implementer, I would keep P() & V() atomic.

The semaphores are binary, assuming and each critical section will have its own semaphore ∴ mutual exclusion will be a given, however with out test & set there could be deadlock.

B) There are various techniques with which messages can be communicated between two *user* processes. Describe two such methods. One of these two methods must however be capable of enhancing performance, in terms of both time and space.

1) Message passing
2) Interupts

Method 2 is capable of enhancing time & space.

↳ Method 1 implements a concept of a mailbox which is located in the kernal

↳ Method 2 sends a signal to the CPU

Question # 5 (4 marks)

Sometimes it is necessary to synchronize two or more processes in a group so that every process must finish its first phase of computation before any other process is allowed to start its second phase. This is called barrier synchronization; For example, for two processes P1 and P2, we can write

Semaphores: s1 = 0, s2 = 0;

process P1	process P2
"phase I" V (s1) P (s2) "phase II"	"phase I" V (s2) P (s1) "phase II"

For this question, you are required to synchronize four processes P1, P2, P3, and P4. The new rules are: **1)** Process P1 must finish Phase I before any other process starts that phase, **2)** all phase I's must finish before any phase II starts, and **3)** all phase II's must proceed in the order "1342" (that is P1 must finish Phase II before P3, and P3 must finish Phase II before P4, and finally P4 must finish Phase II before P2). You are allowed to use any number of semaphores.

Answer: (hint: you really do not need this much space available below)

Semaphores: S1=0; S2=0; S3=0; S4=0; int count = 4;

Process P1	Process P2	Process P3	Process P4

« Phase I >>
V(S1);
count--;
while (count != 0)
yeild();
<<Phase II >>
V(S2)
	P(S1);
	«Phase I >>
	V(S1);
	count--;
	while(count!=0)
	yeild();
	P(S4)
	<< Phase II >>
		P(S1);
		<<Phase I >>
		V(S1);
		count--;
		while(count
		!=0) yeild();
		P(S2)
		«Phase II >>
		V(S3)
			P(S1);
			<<Phase I >>
			V(S1);
			count--;
			while(count!=0)
			yeild();
			P(S3);
			<<Phase II >>
			V(S4);

2.75

Question # 6 (3 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)	P(s3)	P(s23)	P(s12)
C.S.1	C.S.3	C.S.1	C.S.1
V(s23)	V(s1)	V(s12)	C.S.2
P(s3)	P(s23)	V(s3)	V(s1)
C.S.2	C.S.2	V(s1)	P(s3)
V(s3)	C.S.3		C.S.3
V(s23)	V(s23)		V(s3)

CS1: yes, there will be mutual exclusion for C.S.1, P_0 is the only process that can run CS1 right away, all other processes have to wait, P_2 will run CS1 after P_1 & then give P_3 the ability to access CS1 only after it is done with it. In no case will CS1 be accessed by more than one process at a time.

2.75

CS2: There no mutual exclusion for CS2, It is possible that CS2 will be accessed at the same time. eg. P_1 & P_0 can access CS2 at the same time.

CS3: Yes, CS3 does have mutual exclusion. If P_1 will get access to CS3 then P2 will allow P3 access go. If P3 has access P1 →