Instructor: Ioannis Nikolaidis

ID Number:	**Mark:**
Last Name:	
First Name:	

CMPUT 379, Fall 2000, Midterm Examination (A3)

October 25th, 2000

This is an open book exam. All questions have equal weights and must be answered in whatever space is available on this form. No additional sheets are allowed. Copying random excerpts from your text book is not a good practice. The exam is marked out of 20 which is the percentage of its contribution to the final mark. All questions carry an equal weight. If you disagree with the mark received in this exam, you have two weeks from the day you are notified about your mark to contact the instructor and discuss your objection. Notification about your mark will be performed via e-mail to your ugrad.cs.ualberta.ca account.

[1] Twelve processes are scheduled for execution on a single CPU. According to one schedule, the processes, numbered P1 to P12 are executed in their ID order (that is P1,P2,P3,...,P12). If the time for the execution of each process is given in the following table (in msec), what is the average waiting time for the schedule which follows the numerical order? (The waiting time includes the execution time of the process.) Can you schedule the processes in a different order such that an average waiting time of (a) less than 12 msec is achieved? (b) less than 10 msec is achieved? Why? Explain.

P1	1
P2	2
P3	2
P4	3
P5	2
P6	1
P7	2
P8	3
P9	2
P10	1
P11	3
P12	1

[2] A sequence of page references produced by a process is:
5,9,4,5,3,2,3,3,3,2,1,0,1,7,8,9,1,9,9,9,1,3,5,5,5,0,0,1,1,1,0,0,3,4,5,5,2,2,0,0,1,0,1,0,0,3,3,3,1,4
If the page replacement policy used is LRU, what are the contents of the LRU stack at the end of the above sequence, assuming a total number of exactly 4 pages are allowed to be present in physical memory? What is the LRU stack contents if 5 pages were allowed for the process? (Note you need to describe the *stack.* Not just the pages that are present but also their relative order in the stack from top to bottom.)

[3] The following is the mutual exclusion solution proposed by Peterson.

```
P1 () {
  while (1) {
      . . .
      p1wantstoenter = 1;
      turn = 2;
      while (p2wantstoenter && turn == 2) { }
       CS for P1;
      p1wantstoenter = 0;
       . . .
  }
}
```

```
P2 () {
  while (1) {
      . . .
      p2wantstoenter = 1;
      turn = 1;
       while (p1wantstoenter && turn == 1) { }
       CS for P2;
      p2wantstoenter = 0;
       . . .
  }
}
```

If we were to change the statements assigning the value to *turn*, such that **P1** would execute *turn*=1; (instead of *turn*=2;) and **P2** would execute *turn*=2; (instead of *turn*=1;) instead, what would this modification cause:

1. No real change at all. The purpose of *turn* is to resolve which process can proceed to the critical section, when both processes wish to enter the critical section at the same time.
2. Deadlock. The mutual exclusion will be preserved, but the processes will eventually deadlock.
3. Lack of mutual exclusion. The processes will not reach deadlock, but they will be able to be both present in the critical section at the same time.
4. Both deadlock and lack of mutual exclusion. It is possible that the processes will deadlock and it is also possible that they will both be present in the critical section at the same time.

Explain.

[4] In the subsection about "Protection and Sharing" of the "Segmentation" section, the textbook claims that if we try to share the same segment (which, in our example, contains the instructions of a library) then all processes sharing the code (thus, this is a *shared* library) must define the shared code segment to have the same segment number across all processes. The need for the same segment number corresponding to the same (shared) segment across all sharing processes complicates the memory management. If, instead of segmentation we use paging, what is a similar problem we need to address and why? (Note that even in the paging environment, the code of the shared library may have to perform a conditional jump to a location within the library code itself. It should be able to perform this jump regardless of which process (from the ones using this shared library) is currently executing.)