Instructor: Ioanis Nikolaidis

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

CMPUT 379, Fall 2000, Final Examination (A3)

December 11th, 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 38 which is the percentage of its contribution to the final mark. All eight 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.ualbert.ca account.

[1] The banker's algorithm requires a process to specify in advance its maximum resource demands. This may be inefficient and too restrictive to the system, e.g., in a situation when the process needed many resources in the past, but is not going to need them in the future.One can consider a modification of the banker's algorithm in which a process is allowed to re-declare its maximum needs dynamically. What rules would you impose on the dynamic re-declaration of the maximum needs such that the modified algorithm would still avoid deadlock? How can the system encourage processes to reduce their maximum needs if they can be reduced?

[2] Some processors include a special set of registers for floating point arithmetic. However, very few processes are expected to use the floating point register set. Because the floating point registers are lengthy, when a process is context switched, by default, the state of the floating point register set is *not* saved (nevertheless, other registers are saved as usual). In such processors, if an attempt is made to modify any of the floating point registers, an exception is raised. The exception is not raised if a specific bit in the status flags of the CPU (let us call it FPM, i.e., "floating point modify") is set. Two instructions provide the means to clear and set the specific bit (for example CLRFPM, SETFPM for "clearing" and "setting" respectively). How can you ensure that even though you are not saving the floating point registers by default, when the need arises you can save them, in order to ensure that the floating point operations of one process do not interfere with the floating point operations of another process? Should the CLRFPM instruction be privileged? Should the SETFPM instruction be privileged? Your answer should be generalizable to any number of processes. State any necessary and reasonable assumptions and provide pseudocode if necessary.

[3] A system with a single disk uses paging. The CPU is scheduled according to Round Robin. We measure the CPU utilization, the disk utilization and the number of processes in the system and find that they are 3 %, 96 % and 10 respectively. Which of the following statements make sense and which ones not.

1. The performance of the CPU scheduling algorithm is poor.
2. The CPU utilization may improve if SJF is used (instead of the current round robin).
3. All the processes in the system are I/O bound and there are no more processes to run.
4. A processes does not have its working set in memory and is thrashing.
5. There are too many processes admitted for execution, and they are all thrashing.
6. There is not enough memory in the system.
7. The disk is too small.

[4] Imagine that a *multiprogramming* system using an approximation of the working set model admitted a number of programs for execution. Initially, these programs were CPU-bound and they were all generating page faults at a rate comfortably close to the optimal rate. Suppose that suddenly one of those programs changes its behavior pattern to strongly I/O bound, i.e., whenever it gets the CPU it almost immediately issues a disk I/O request. You may assume that this I/O request is addressed to a different disk than the one used for paging (although it doesn't affect the answer). What is the most likely action to be performed by the working set monitor regarding the amount of memory allocated to the I/O bound process?

1. The amount of memory allocated to the process will increase (to reduce the impact of page faults on its heavily I/O-bound behavior).
2. The amount of memory allocated to the process will decrease (the rate of page faults generated by the process is now very low).
3. Nothing will change because the rate of page faults issued by a process does not depend on whether the process is CPU-bound or I/O-bound.
4. The system will swap out the process until its behavior changes back to CPU-bound.

Select the right answer and explain it.

[5] Assume that you are writing a tool to check the integrity of a Unix file system that uses the allocation scheme depicted in Figure 11.7 of the book and also utilizes a bit-vector for free-space management. Assume that it also uses linear list implementation for directories. What operations will this file system integrity tool perform and why?

[6] The *telnet* application sends user names and passwords in cleartext to a remote host. Thus, it allows anyone who intercepts the packets to obtain the passwords and break into the remote host. A new secure alternative to *telnet* is being designed. The new scheme calls for a Diffie-Helman key exchange to take place before anything is sent between the user and the remote host. Through the key exchange algorithm, the user and the remote host posses the same key. They subsequently use this key to encrypt all the traffic between them using conventional (symmetric) encryption (e.g., DES). Using this encrypted connection, they can now send the username and password to continue with the login procedure. Which of the following statements makes sense:

1. The Unix login procedure is now redundant because the user and the host have already exchanged keys and are encrypting whatever they send.
2. A man-in-the-middle attack could render the scheme ineffective since authentication is compromised.
3. Authentication is not compromised under any circumstances because the exchanged key is never exchanged in the clear, however it is still possible to mount a denial of service attack by requesting many connections to the host until the host's load reaches discomforting levels for all users.
4. The scheme is still insecure since we can factor the password into its prime factors easily.

[*Diffie-Hellman key exchange between Alice and Bob:* Alice and Bob both know a prime p and a primitive root of p, a. Alice picks a random number x and Bob picks another random number y. Alice sends $x' = a^x \bmod p$ to Bob and Bob sends $y' = a^y \bmod p$ to Alice. Alice calculates as key $k = (y')^x \bmod p$ and Bob calculates $k = (x')^y \bmod p$. The k is the shared key. *Man-in-the-middle attack:* Messages from A to B are captured by C and messages from B to A are also captured by C. C decides what to send to either A or B, and it can send any information it wishes.]

[7] We mentioned in class that if the *referenced* bit is not defined explicitly by the memory mapping hardware, it can be simulated in software (e.g., this has been done for the VAX). Imagine a hypothetical computer, we will call it ZAD, on which the page table entry includes two bits: *invalid* and *write-protected,* but there is neither *referenced* nor *written-into* (dirty) bit. Your favourite page replacement strategy that you would like to implement on ZAD requires both the *referenced* and *written-into* bits. How would you emulate the two bits on ZAD in a reasonably efficient way?

[8] Give a sample sequence of page references that produce the same number of page faults for FIFO and Second Chance and less for LRU. Assume that you have exactly three frames. Explain how you came up with the particular sequence.