CS 4/53201 # Homework #1 **Operating Systems**

Due in class on Wednesday 19 September 2001

1. **(Exercise 1.5(b) from OSC 6th edition, exercise 1.4(b) in 5th edition) In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems. Can we ensure that the same degree of security in a time-shared machine as we have in a dedicated machine? Explain your answer.**

   While it might be possible in theory to afford complete isolation for each user in a time-sharing environment, and practice might even approach such complete isolation, a design oversight in the OS or a glitch in the hardware or software could always cause problems. However, for the most part, the OS does provide such protection, in particular with hardware support for CPU scheduling, memory management, file management, etc.

2. **(Exercise 2.10 from OSC 6th and 5th edition) Writing an operating system that can operate without interference from malicious or undebugged user programs requires hardware assistance. Name three hardware aids for writing an operating system, and describe how they could be used together to protect the operating system.**

   Explain about user mode & kernel mode plus traps (for I/O), base & limit registers (for memory management), and a timer (for CPU scheduler), and how those architectural features support OS features…

3. **(Exercise 3.12 from OSC 6th edition, not in 5th edition) What is the main advantage of the microkernel approach to system design?**

   The kernel is a small program, making it easier to extend and to port to other platforms, and making both it and the non-kernel parts of the OS easier to design and debug. Additionally, since less of the OS runs in kernel mode, the OS itself is more secure.

4. **Exercise 4.5(a) and 4.5(e) from OSC 6th edition, not in 5th edition) What are the benefits and detriments of each of the following? Consider both the systems and the programmers' levels. (a) direct and indirect communication, (e) fixed-size and variable-sized messages. Explain your answer!**

   (a) Direct communication is easy to implement, but doesn't allow multiple "kinds" of messages to be sent and distinguished by the receiver. Indirect communication is more complex and thus harder to implement, but allows the receiver to have separate mailboxes for different kinds of data.

   (e) Fixed-size messages are easier to implement, but less efficient (multiples messages are required when the data is larger than the fixed size, and useless data must be sent when the data is smaller than the fixed size). Variable-length messages are harder to implement but more efficient since only one message is sent regardless of the length of the data.

   *Note that this question asked you to list the benefits and detriments of each, **not** to define each term or write down everything the book or I said on this topic in random fashion. Read the question carefully, and then clearly answer the question that was asked!*

5. (Exercise 5.3 from OSC 6th edition, exercise 4.7 in 5th edition) What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

See my slides in Lecture 08 for details. User level threads don't require OS involvement, while kernel level threads do. With user-level threads the user controls the thread scheduling, while with kernel-level threads the kernel's CPU scheduler schedules them. Switching between user-level threads is faster than switching between kernel-level threads because only a few registers need to be saved and a system call isn't involved. With user-level threads, if one thread blocks the entire process blocks, but with kernel-level threads, the CPU scheduler can simply switch to another thread in that same process.

For second part of the question, many answers are possible. For example, if a process implements a server with one dispatcher thread and many worker threads, it may be desirable to give that process more than one time slice and to allow each thread to block independently — in this case, kernel-level threads would be better than user-level threads.

Again, the first part of this question asked for the differences between the two, not a definition of each where I was expected to notice the difference between your two definitions. Read the question carefully, and then clearly answer the question that was asked! Also, notice that there were two parts to this question, not just one.