CSCI 4320 (Principles of Operating Systems), Fall 2001

Homework 1

Assigned: September 27, 2001.

Due: October 4, 2001, at 5pm.

Credit: 40 points.

1 Reading

Be sure you have read chapter 1 and sections 2.1 through 2.4 of chapter 2.

2 Problems

Answer the following questions. You may write out your answers by hand or using a word processor or other program, but please submit hard copy, either in class or in my mailbox in the department office.

1. (4 points) For each of the following instructions, say whether it should be executed only in kernel (i.e., supervisor) mode and briefly explain why.

   (a) Set the time-of-day clock.

   (b) Disable all interrupts.

   (c) Read the time-of-day clock.

   (d) Change the base and limit registers (assuming the memory-management scheme described on pp. 26–27).

2. (4 points) Does a timesharing system need a process table? Why or why not? What about a personal-computer system in which only one process at a time can execute, that process taking over the whole machine until it is finished? Why or why not?

3. (4 points) When a computer is being designed, it is common to first simulate it using a program that runs one (simulated) instruction at a time. Even computers with more than one processor are simulated strictly sequentially like this. Is it possible for a race condition to occur when, as in this situation, there are no truly simultaneous events?

4. (4 points) Look again at the solution to the mutual-exclusion problem presented in Figure 2-20 in the textbook. If the two processes are running on a computer with two CPUs and a common memory, does this solution work? I.e., which if any of the criteria given on p. 102 does it satisfy? Briefly justify your answer.

5. (6 points) Consider a computer that does not have a test-and-set-lock (TSL) instruction, but does have an instruction to swap the contents of a register and a memory word in a single indivisible action. Use such an instruction (call it SWAP) to write a routine *enter_region* like the one found in Figure 2-22 in the textbook, or explain why this is impossible.

6. (6 points) Give a sketch (possibly pseudocode) of how you could implement semaphores on a single-CPU system on which the operating system can disable interrupts.

7. (6 points) In the solution to the dining philosophers problem shown in Figure 2-33 in the textbook, why is the state variable set to *HUNGRY* in the procedure *take_forks*?

8. (6 points) Consider the procedure *put_forks* in Figure 2-33 in the textbook. Suppose that the variable *state[i]* was set to *THINKING after* the two calls to *test* rather than before. How would this change affect the solution? (I.e., would it work as well as before? better? not as well?)