CSCI 3323 (Principles of Operating Systems), Fall 2015

Homework 5

Credit: 20 points.

1 Reading

Be sure you have read (or at least skimmed) Chapters 5, 6, and 9.

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 one of my mailboxes (outside my office or in the ASO).

1. (5 points) Consider the following two I/O devices. For each device, say whether you think programmed I/O or interrupt-driven I/O makes the most sense, and justify your answer, preferably with data such as a comparison of times involved. (*Hint:* Consider the time required for interrupt processing versus the time needed for the actual input/output operation.)

   (a) A printer that prints at a maximum rate of 400 characters per second, connected to a computer system in which writing to the printer's output register takes essentially no time, and using interrupt-driven I/O means that each character printed requires an interrupt that takes a total of 50 microseconds (i.e., 50×10^{-6} seconds) to process.

   (b) A simple memory-mapped video terminal (output only), connected to a system where interrupts take a minimum of 100 nsec to process and copying a byte into the terminal's video RAM takes 10 nsec.

2. (5 points) Student H. Hacker installs a new disk driver that its author claims improves performance by using the elevator algorithm and also processing requests for multiple sectors within a cylinder in sector order. Hacker, very impressed with this claim, writes a program to test the new driver's performance by reading 10,000 blocks spread randomly across the disk. The observed performance, however, is no better than what would be expected if the driver used a first-come first-served algorithm. Why? What would be a better test of whether the new driver is faster? (*Hint:* The test program reads the blocks one at a time. Think about how many requests will be on the disk driver's queue at any one time.)

3. (5 points) Suppose you are designing an electronic funds transfer system, in which there will be many identical processes that work as follows: Each process accepts as input an amount of money to transfer, the account to be credited, and the account to be debited. It then locks both accounts (one at a time), transfers the money, and releases the locks when done. Many of these processes could be running at the same time. Clearly a design goal for this system is that two transfers that affect the same account should not take place at the same time, since that might lead to race conditions. However, no problems should arise from doing a transfer from, say, account A to account B at the same time as a transfer from account C to account D, so another design goal is for this to be possible. The available locking mechanism

is fairly primitive: It acquires locks one at a time, and there is no provision for testing a lock to find out whether it is available (you must simply attempt to acquire it, and wait if it's not available). A friend proposes a simple scheme for locking the accounts: First lock the account to be credited; then lock the account to be debited. Can this scheme lead to deadlock? If you think it cannot, briefly explain why not. If you think it can, first give an example of a possible deadlock situation, and then design a scheme that avoids deadlocks, meets the stated design goals, *and uses only the locking mechanism just described.*

4. (5 points) Programs or program updates sometimes come packaged as "self-extracting archives", which combine the files that make up the archive with a program to extract them. Compare this with other ways of packaging programs and updates (e.g., as RPMs or tarballs) with regard to security and any other factors that seem relevant.