# CSCI 3323 (Principles of Operating Systems), Fall 2014

# Homework 5

**Credit:** 20 points.

## 1 Reading

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

## 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. (*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., $50 \times 10^{-6}$ 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) The textbook divides the many routines that make up an operating system's I/O software into four layers. In which of these layers should each of the following be done? Why? (Assume that in general functionality should be provided at the highest level at which it makes sense — e.g., in user-level software rather than device-independent software.)

   (a) Converting floating-point numbers to ASCII for printing.

   (b) Computing the track, sector, and head for a disk read operation.

   (c) Writing commands to initiate I/O to a printer controller's device registers.

   (d) Detecting that an application program is attempting to write data from an invalid buffer address. (Assume that detecting an invalid buffer address can only be done in supervisor mode.)

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.