CSCI 4320 (Principles of Operating Systems), Fall 2008
Homework 7

Assigned: December 3, 2008.

Due: December 8, 2008, at 5pm. (Accepted without penalty through December 10.)

Credit: 30 points.

1 Reading

Be sure you have read Chapters 5 and 6.

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. (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 each character printed requires an interrupt that takes a total of 50 microseconds (i.e., 50×10^{-6} seconds) to process.

   (b) A memory-mapped video terminal, 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 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) Consider a computer system that maintains date and time using a 32-bit unsigned integer whose value represents a number of seconds since January 1, 1970. (So, a value of 362 would represent 12:06:02 am, January 1, 1970.) In what year will this scheme become unworkable because the 32-bit integer is not big enough? What if instead the system uses a signed 32-bit integer, allowing negative values to represent dates and times before January 1, 1970? (Ignore leap-year complications and assume that the average year has 365.25 days.)

4. (5 points) Suppose at a given point in time a disk driver has in its queue requests to read cylinders 10, 22, 20, 2, 40, 6, and 38, received in that order. If a seek takes 5 milliseconds (i.e., 5×10^{-3} seconds) per cylinder moved, and the arm is initially at cylinder 20, how much seek time is needed to process these requests using each of the three scheduling algorithms discussed (FCFS, SSF, and elevator)? Assume that no other requests arrive while these are being processed and that for the elevator algorithm the initial direction of movement is outward (toward larger cylinder numbers).

5. (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.)

6. (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.

3 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu`, with each file as an attachment. Please use a subject line that mentions the course number and the assignment (e.g., "csci 4320 homework 7"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's

Fedora 9 Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (Up to 5 extra-credit points). The Linux lab machines have special files `/dev/random` and `/dev/urandom` that generate sequences of "random" bytes. (Read the man page for `urandom` for an explanation of the difference between them.) Write a program that compares the results of generating N integers using one of these special files to the results of generating N integers using function `rand()`. (It's up to you to decide how to compare them. A simple test might be to count how many are even and how many are odd. You may have a better idea!) Submit your source code and a text file containing output of one or more executions.

   Hints:

   - I recommend getting N as a command-line argument, so you can experiment with different values. If you don't remember how to access command-line arguments in C, here is a simple example:

     ```
     #include <stdio.h>
     #include <stdlib.h>
     int main(int argc, char *argv[]) {
         int n = -1;
         if ((argc < 2) || ((n = atoi(argv[1])) <= 0)) {
             fprintf(stderr, "usage:  %s n (positive number)\n", argv[0]);
             return EXIT_FAILURE;
         }
         printf("%d\n", n);
         return EXIT_SUCCESS;
     }
     ```

   - You will probably need to use `open` and `read` rather than `fopen` and `fscanf` to read from the special file. `man` pages for these two functions can be found via `man 2 open` and `man 2 read`.