CSCI 3323 (Principles of Operating Systems), Fall 2018
Homework X

Credit: Up to 50 extra-credit points.

1 General Instructions

You can do as many of the following problems as you like, but you can only receive a total of 50 extra points.

NOTE that the usual rules for collaboration do not apply to this assignment. Please work individually, without discussing problems with other students.

I'm also open to the possibility of giving extra credit for other work — other problems from the textbook, a report on something course-related, etc. If you have an idea for such a project, let's negotiate (by e-mail or in person).

2 Problems

Answer as many of the following questions as you like. You may write out your answers by hand or using a word processor or other program; I prefer hardcopy but will accept e-mail (PDF preferred).

1. Problems related to chapter 5 (I/O):

   (a) (Up to 6 points) Consider a system that uses its local area network as follows: An application program makes a system call to write data packets (each 1024 bytes, ignoring headers) to the network. The operating system first copies the data to be sent to a kernel buffer. Working on one packet at a time, it then copies the data to the network controller. When all 1024 bytes have been copied to the network controller, it sends them over the network at a rate of 10 megabits (10×10^6 bits) per second. The receiving controller receives each bit a microsecond after it is sent. When the last bit in the packet is received, the destination CPU is interrupted, and its operating system copies the packet into a kernel buffer, inspects it, and copies it into a buffer owned by the application program that should receive it. It then sends back an acknowledgment (assume one bit) to the sending computer, which interrupts the sending CPU, and work can begin on the next packet. How long does it take to send each packet, if it takes one millisecond to process an interrupt (on either CPU) and one microsecond to copy a byte? Assume that the time taken for the receiving CPU to inspect the packet is negligible. What is the effective transfer rate (in bits per second) over this connection? (If you don't remember your prefixes: A microsecond is 10^{-6} seconds; a millisecond is 10^{-3} seconds.)
   (*Hints:* Notice that some times are per bit and some are per byte. If you think you need to make additional assumptions, do so and explain them. If you show your calculations and briefly explain what you are doing, your odds of getting at least partial credit are better.)

2. Problems from chapter 9 (security):

   (a) (Up to 2 points) Answer question 26 on p. 708 of the textbook. (*Hint:* What are the odds of being able to guess the password if you know its length? if you don't?)

(b) (Up to 2 points) Answer question 34 on p. 709 of the textbook.

(c) (Up to 2 points) Answer question 37 on p. 709 of the textbook.

(d) (Up to 2 points) Answer question 48 on p. 710 of the textbook.

3. Essay problems (please include in your answer an informal bibliography listing sources on which it is based — Web sites, books, etc.):

   (a) (Up to 10 points) Most of the memory-management schemes discussed in the textbook are based on the idea that each process has its own "address space", each of which uses the same range of virtual addresses ranging from 0 to some large number (often the maximum possible based on the number of bits in an address). However, some of the older mainframe operating systems instead defined a single address space shared by all processes, with each process having a different range of virtual addresses. There have been indications in the not-so-dim past that this idea might be considered again. Speculate on how it might be done, what advantages there might be, what disadvantages there might be, and so forth. (In particular it might revive the program-relocation problem.)

3 Programming Problems

Do as many of the following programming problems as you like. 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 and the assignment (e.g., "csci 3323 hw X" or "O/S hw X"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (Up to 10 points) Add one or more features to the simple shell program you wrote for Homework 1. How much credit you get will depend on the level of difficulty involved. A not-too-difficult choice involves adding a command history; the `man` page for `readline` and associated reading is a good starting point.

2. (Up to 5 points each) Do any or all of the extra-credit programming problems from Homework 6 and Homework 7.

3. (Credit as described below) In class I briefly showed implementations of solutions to some of the classical IPC problems we discussed in class, plus programs to test them, using POSIX threads. (These implementations are available via "sample programs" on the course Web site.) For this assignment I'm providing test programs for additional problems, missing only the parts that do the actual synchronization. The test setup is somewhat elaborate, involving multiple code files, but I've organized them in a way that I think reduces duplication and allows semaphore- and monitor-based solutions to coexist. More about this below.

   (a) Dining philosophers (up to 10 extra-credit points): The textbook gives pseudocode for a semaphore-based solution to this problem, and the sample solution for Homework 3 gives pseudocode for a monitor-based solution. Write code for one or both of these solutions.

(b) Readers/writers with priority (up to 20 extra-credit points): In class we talked about the version of the readers/writers problem in which readers have priority (i.e., writers may wait indefinitely if new readers continue to arrive). A second version of the problem gives priority to writers: New readers cannot start reading if there are writers waiting. (The https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem refers to these two versions as the first and second readers-writers problems.) This second problem can be solved using either semaphores or monitors. Write code for one or both solutions. (The Wikipedia article has pseudocode for a semaphore-based solution. For a monitor-based solution, you may be able to adapt the code shown in class and in the sample program for the first readers-writers problem, or you may be able to find a solution online. If you do the latter, please cite your source.)

To start work on one or more of these problems, get a copy of the ZIP file http://www.cs.trinity.edu/~bmas and unzip it. The result will have two directories, one for each of the problems (dining philosophers and readers-writers). Within each of these directories there are two subdirectories `semaphore` and `monitor`. Within each subdirectory there are several symlinks to the test framework, plus a single `.c` file you will modify. Comments in the file indicate where you need to add code. Compile using the provided `Makefile`s; just typing `make` will do what's needed.

4 Honor Code Statement

Include in each part of the assignment (written and programming problems) the Honor Code pledge or just the word "pledged", plus one of the following statements, whichever applies:

- "This assignment is entirely my own work".
- "This assignment is entirely my own work, except that I also consulted *outside course — a book other than the textbook (give title and author), a Web site (give its URL), etc.*."

(As before, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site.)

5 Essay

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what about the assignment you found interesting, difficult, or otherwise noteworthy. For programming assignments, it should go in the body of the e-mail or in a plain-text file `essay.txt` (no word-processor files please).