CSCI 3323 (Principles of Operating Systems), Fall 2016

Homework X

Credit: Up to 40 extra-credit points.

1 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", plus one or more of the following about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For written assignments, it should go right after your name and the assignment number; for programming assignments, it should go in comments at the start of your program.

- This assignment is entirely my own work.
- This assignment is entirely my own work, except for portions I got from the assignment itself *(some programming assignments include "starter code")* or sample programs for the course *(from which you can borrow freely — that's what they're for).*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc.*
- I got significant help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. ("Significant" here means more than just a little assistance with tools — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided significant help to *names of students* on this assignment. *("Significant" here means more than just a little assistance with tools — you don't need to tell me about helping other students decipher compiler error messages, but beyond that, do tell me.)*

2 General Instructions

Answer as many (or few) of the following questions as you like. (Notice, however, that you can receive at most 40 extra-credit points.)

I am 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).

For this assignment, please work individually, without discussing the problems with other students. If you want to discuss problems with someone, talk to me.

[1] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

3 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, but please submit hard copy, either in class or in one of my mailboxes (outside my office or in the ASO).

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

   (a) (Up to 2 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.)

   (b) (Up to 8 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?

   (*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 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 46 on p. 710 of the textbook.

   (e) (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) We talked very briefly early in the semester about VM/370, an operating system that allows running multiple "guest" operating systems side by side. What

are some other ways of accomplishing similar things? How do they work? (The discussion of virtualization in Chapter 8 of the textbook looks promising as a source of information.)

(b) (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.)

4 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 (or should have written!) 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 10 points) Write a program that simulates execution of one or more of the following page replacement algorithms: FIFO, Optimal, Second Chance, Clock, NRU (Not Recently Used), LRU (Least Recently Used), NFU (Not Frequently Used), Aging (with a 16-bit counter), Working Set, WSClock. In writing your code, feel free to consult any descriptions of the algorithms, but do *not* look for code to copy/modify.

   How much credit you get will depend on how many algorithms you simulate and how correctly. You can use any language of your choice, as long as I can run/test your program on the department machines using an interface more or less like the following. (I'd prefer that you use exactly this interface for input — it makes my testing job easier — but if you use something else, put comments at the top of your code telling me how to run your code with my test data.)

   - Command-line arguments:
     - Required:
       - name of input file (format below)
       - number of page frames
     - Optional:

    - "–clockTickInterval N" to specify interval for "clock ticks", for algorithms that need this — the idea being to consider that a "clock tick" happens every N references)
    - "–tau N" to specify time interval for working set algorithms
- Input file format:
  - number of pages
  - one or more lines of the form "R n" or "W n", where R/W indicates whether this is a read or write reference, and n is the page number being referenced

Output should be the following information, for each page replacement algorithm implemented:

- name of algorithm
- total number of page references
- number of page references that changed the page ('W')
- number of page faults
- number of times a page had to be written out

Make the following assumptions:

- Initially memory is empty.
- All memory references are valid — if the page is not in memory, it can be read in from disk. (You don't have to simulate that part, just count how often it happens.)

Here are files containing some sample input and output:

- Command-line parameters `pagingsimulator.in 4 --clockTickInterval 10 --tau 20`
- Input file[2]
- Output[3]

3. (Up to 10 points) Do the optional programming problem for Homework 2 (implementation of a solution to the restroom problem). Refer to the assignment for more details and suggestions.

4. (Up to 10 points) Do the optional programming problem for Homework 3 (implementation of a monitor-based solution to the dining-philosophers program). Refer to the assignment for more details and suggestions.

5. (Up to 10 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 or both 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. (*Hint:* 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`.)

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW0X/Problems/pagingsimulator.in`

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW0X/Problems/pagingsimulator.out`