CSCI 3323 (Principles of Operating Systems), Fall 2014

Homework X

Credit: Up to 30 extra-credit points.

1 General Instructions

Answer as many (or few) of the following questions as you like. (Notice, however, that you can receive at most 30 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.

2 Problems

For these problems, please submit hard copy (in my mailbox in the department office or under my door). (If that's a huge hassle, e-mail is okay, but I will print it to grade it.)

2.1 Problems from Chapter 9

Answer any or all of the following questions (from the textbook chapter on security).

1. (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?)
2. (Up to 2 points.) Answer question 27 on p. 709 of the textbook.
3. (Up to 2 points.) Answer question 34 on p. 709 of the textbook.
4. (Up to 2 points.) Answer question 37 on p. 709 of the textbook.
5. (Up to 2 points.) Answer question 46 on p. 710 of the textbook.

2.2 Essay Questions

Write a page or more of prose about one or more of the following questions, writing for an audience of fellow students. Include a short informal bibliography listing the source(s) of your information.

1. (Up to 10 points) We talked 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.)

2. (Up to 10 points) The computer industry has a history of guessing wrong about how much memory will be "enough for anyone" and therefore choosing a size for virtual/physical addresses that proves to be too restrictive — with the result that at some point a transition to larger addresses must be made, while still allowing (as far as possible) programs using the older/smaller addresses to execute. Currently there are several features that allow processors to make use of more memory than can be addressed with 32 bits (e.g., PAE ("Physical Address Extensions") and full 64-bit addressing). How do they work, and how do they (if they do) allow running programs that use 32-bit addressing?

3 Programming Problems

Do one or more of the following optional programming problems. Submit source code and other files by e-mail, as for previous assignments. (I.e., 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 3323 extra credit"). 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 extra-credit 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[1]
- Output[2]

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2014fall/Homeworks/HW0X/Problems/pagingsimulator.in`

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2014fall/Homeworks/HW0X/Problems/pagingsimulator.out`