CSCI 3323 (Principles of Operating Systems), Fall 2016

Homework 5

Credit: 45 points.

1 Reading

Be sure you have read, or at least skimmed, Chapter 3.

2 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.)*

3 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]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.

1. (5 points) The operating system designers at Acme Computer Company have been asked to think of a way of reducing the amount of disk space needed for paging. One person proposes never saving pages that only contain program code, but simply paging them in directly from the file containing the executable. Will this work always, never, or sometimes? If "sometimes", when will it work and when will it not? (*Hint:* Search your recollections of CSCI 2321 — or another source — for a definition of "self-modifying code".)

2. (5 points) How long it takes to access all elements of a large data structure can depend on whether they're accessed in contiguous order (i.e., one after another in the order in which they're stored in memory), or in some other order. The classic example is a 2D array, in which performance of nested loops such as

```
for (int r = 0; r < ROWS; ++r)
  for (int c = 0; c < COLS; ++c)
    array[r][c] = foo(r,c);
```

can change drastically for a large array if the order of the loops is reversed. Give two explanations for this phenomenon based on what you have learned from our discussion of memory management. (*Hint:* One possible explanation is based on a topic we discussed extensively but that on current systems is less likely than it was before huge amounts of RAM became common. The currently-more-likely explanation is one we touched on but did not discuss extensively.)

3. (10 points) Consider (imagine?) a very small computer system with only four page frames. Suppose you have implemented the aging algorithm for page replacement, using 4-bit counters and updating the counters after every clock tick, and suppose the R bits for the four pages are as follows after the first four clock ticks.

Time	R bit (page 0)	R bit (page 1)	R bit (page 2)	R bit (page 3)
after tick 1	0	1	1	1
after tick 2	1	0	1	1
after tick 3	1	0	1	0
after tick 4	1	1	0	1

What are the values of the counters (in binary) for all pages after these four clock ticks? If a page needed to be removed at that point, which page would be chosen for removal?

4. (10 points) A computer at Acme Company used as a compute server (i.e., to run non-interactive jobs) is observed to be running slowly (turnaround times longer than expected). The system uses demand paging, and there is a separate disk used exclusively for paging. The sysadmins are puzzled by the poor performance, so they decide to monitor the system. It is discovered that the CPU is in use about 20% of the time, the paging disk is in use about 98% of the time, and other disks are in use about 5% of the time. They are particularly puzzled by the CPU utilization (percentage of time the CPU is in use), since they believe most of the jobs are compute-bound (i.e., much more computation than I/O). First give your best explanation of why CPU utilization is so low, and then for each of the following, say whether it would be likely to increase it and why.

   (a) Installing a faster CPU.

(b) Installing a larger paging disk.

(c) Increasing the number of processes ("degree of multiprogramming").

(d) Decreasing the number of processes ("degree of multiprogramming").

(e) Installing more main memory.

(f) Installing a faster paging disk.

4 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 and the assignment (e.g., "csci 3323 hw 5" or "O/S hw 5"). 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. (15 points) Write a program or programs to demonstrate the phenomenon described in problem 2. Turn in your program(s) and output showing differences in execution time. (It's probably simplest to just put this output in a text file and send that together with your source code file(s).) Try to do this in a way that shows a non-trivial difference in execution time (so you will likely need to make the arrays or other data structures large). I *strongly* recommend that you write your programs in C or C++, or some other language where timing results are more predictable than they're apt to be in, for example, a JVM-based language such as Java or Scala (because "just-in-time" compilation makes attempts to collect meaningful performance data difficult). But anything that can be compiled and executed on one of the Linux lab machines is acceptable, as long as you tell me how to compile and execute what you turn in, if it's not C or C++. You don't have to develop and run your programs on one of the lab machines, but if you don't, (1) tell me what system you used instead, and (2) be sure your programs at least compile and run on one of the lab machines, even if they don't necessarily give the same timing results as on the system you used.

   Possibly-helpful hints:

   - An easy way to measure how long program `mypgm` takes on a Linux system is to run it by typing `time mypgm`. Another way is to run it with `/usr/bin/time mypgm`. (This gives more/different information — try it.) If you'd rather put something in the program itself to collect and print timing information, for C/C++ programs you could use the function in timer.h[2] to obtain starting and ending times for the section of the code you want to time.
   - Your program doesn't have to use a 2D array (you might be able to think of some other data structure that produces the same result). If you do use a 2D array, though, keep in mind the following:
     - To the best of my knowledge, most C and C++ implementations allocate local variables on "the stack", which may be limited in size. Dynamically allocated variables (i.e., those allocated with `malloc` or `new`) aren't subject to this limit.

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW05/Problems/timer.h`

- Dynamic allocation of 2D arrays in C is full of pitfalls. It may be easier to just allocate a 1D array and fake accessing it as a 2D array (e.g., the element in `x[i][j]`, if `x` is a 2D array, is at offset `i*ncols+j`).