CSCI 3323 (Principles of Operating Systems), Fall 2013

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 17 on p. 715 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 18 on p. 715 of the textbook.
3. (Up to 2 points.) Answer question 26 on p. 716 of the textbook.
4. (Up to 2 points.) Answer question 27 on p. 716 of the textbook.
5. (Up to 2 points.) Answer question 36 on p. 717 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]

2. (Optional — up to 5 extra-credit points) Write a program that given a directory D, blocksize B, and maximum number of blocks M as command-line arguments prints out how many files in D and its subdirectories are of size B or less, how many are of size between B and $2B$, etc., up to size MB. (This might be useful in getting an idea of what size files are typical, so if you had a choice of blocksize you would know what choice might make the most sense.) Include directories and symbolic links (but count the size of the link and not the file/directory it links to). Also turn in output of running this program on your home directory in `/users` with B and M as below.

   Here is sample output for running the program with $D =$ `/lib`, $B = 4096$, and $M = 20$, on one of the DIAS machines:

```
Results for directory /lib with blocksize 4096:

                910 files of size          1 blocks
               1927 files of size          2 blocks
               1707 files of size          3 blocks
               1495 files of size          4 blocks
               1106 files of size          5 blocks
                906 files of size          6 blocks
                576 files of size          7 blocks
                473 files of size          8 blocks
                389 files of size          9 blocks
                379 files of size         10 blocks
                263 files of size         11 blocks
                270 files of size         12 blocks
                282 files of size         13 blocks
                190 files of size         14 blocks
                175 files of size         15 blocks
```

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/pagingsimulator.in`

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/pagingsimulator.out`

```
  153 files of size         16 blocks
  174 files of size         17 blocks
  168 files of size         18 blocks
   99 files of size         19 blocks
  109 files of size         20 blocks
 1550 files of size         21 blocks or more
```

(Of course, you won't be able to examine files in directories you don't have access to. Just print error messages for files/directories you can't access.)

To get maximum points, your program should be in C or C++ and make no use of system commands such as `ls`. (You can use another language, or even write a shell script, but you will get fewer points.) Library functions `opendir`, `readdir`, and `lstat` will probably be helpful. You might also be interested in functions `chdir` and `strerror`. These functions are described by man pages. (Remember also that `man -a foo` gives all man pages for `foo`. This can be helpful if there is both a command `foo` and a functionfoo.)

Here is some starter code[3] that parses/checks the command-line arguments.

3. (Optional — up to 5 extra-credit points) Write a program that given a directory D as a command-line argument prints all the "broken" symbolic links in D or any of its subdirectories — that is, symbolic links that point to a file that doesn't exist. Here is sample output for running the program with D = `/users/bmassing/Local/HTML-Documents/CS3323/Homeworks/HW0X/Problems`:

```
Broken symbolic links in /users/bmassing/Local/HTML-Documents/CS3323/Homeworks/HW0X/Problems:

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/barfoo
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/foobar
```

(Again, you won't be able to examine files in directories you don't have access to, so just print error messages. You should be able to access everything in the above directory, however. If you want to create some test data of your own, remember that to make a symbolic link called `sym` pointing to `foo`, you type `ln -s foo sym`.)

To get maximum points, your program should be in C or C++ and make no use of system commands such as `ls`. (You can use another language, or even write a shell script, but you will get fewer points.) The library routines mentioned for the previous problem may be helpful. The starter code may also be helpful, in reminding you how to access command-line arguments in C.

4. (Optional — up to 5 extra-credit points) Write a program that given a directory D as a command-line argument finds all the files in D or any of its subdirectories to which there are two or more hard links and prints, for each of them, all the paths within D that point to that file. Here is sample output for running the program with D = `/users/bmassing/Local/HTML-Documents/CS3323/Homeworks/HW0X/Problems`:

```
Files with multiple hard links in /users/bmassing/Local/HTML-Documents/CS3323/Homeworks/HW0X/Problems:

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/homework/index.html
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/homework/homework.html

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/bbb
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/bbbb
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/bb
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/b

/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/dd
/users/bmassing/Local/HTML-Documents/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/TestData/d
```

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2013fall/Homeworks/HW0X/Problems/filesizes.c`

This output means that the two pathnames in the first group reference the same file, the four pathnames in the second group reference the same file, etc. Output can be in any order as long as paths that reference the same file are grouped together. (Again, you won't be able to examine files in directories you don't have access to, so just print error messages. You should be able to access everything in the above directory, however. If you want to create some test data of your own, remember that to make a hard link called `sym` pointing to `foo`, you type `ln foo sym`.)

To get maximum points, your program should be in C or C++ and make no use of system commands such as `ls`. (You can use another language, or even write a shell script, but you will get fewer points.) The library routines mentioned for the previous problems may be helpful. The starter code may also be helpful, in reminding you how to access command-line arguments in C.

5. (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 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`.)