Programming Project: littlelast

Introduction

For this assignment you will write a program that implements some of the functions of the Unix **last** command. In doing so, you will have a chance to use the open(), read(), lseek(), and close() system calls, work with functions for managing the Unix representation of time, and experiment with file buffering . Much of the information you will need will come from the on-line manual.

Specific Details

Write a program called **llast** (for littlelast) that implements the part of the regular **last** command that handles a single ttyname as a command line argument and the part that handles the **-f** option. That is, your program should handle commands such as

```
% llast ttyr9
% llast -f wtmp.sample r9
```

Its output should be identical to that produced by the regular version of last. In the example just given, it might look like:

```
king      ttyr9        net-30103.dce.ha Fri Feb 02 17:04   still logged in
smith     ttyr9        aippp6.buffnet.n Fri Feb 02 14:02 - 17:02  (03:00)
williams  ttyr9        net-22806.hsa.ha Fri Feb 02 13:08 - 14:01  (00:53)
hoang     ttyr9        102.85.60.100    Fri Feb 02 12:03 - 13:07  (01:04)
henrik    ttyr9        roam89-168.stude Fri Feb 02 12:01 - 12:02  (00:00)
wolk      ttyr9        roam165-82.stude Fri Feb 02 11:56 - 11:56  (00:00)
wylupski  ttyr9        justice.medford. Fri Feb 02 08:23 - 08:55  (00:32)
vineleaf  ttyr9        cs04.marathon.co Thu Feb 01 19:45 - 20:06  (00:20)
```

Getting Started

Make a directory   In your ice account, create a subdirectory for this project (mkdir llast). Change into that directory (cd llast) and copy into it some tools from the course account (cp ˜COURSE/hw/llast/files/* **.**). Note: there is a space and a dot after the * in that cp command. These files are sample data for some special cases and some programs you can use to explore the wtmp file and extract subsets to test your progam on.

Read the manual   Read the manual page on the **last** command. Try the **last** command. Login and logout a few times and see what it does. Find what file it uses. Read the manual on that file. The login info is added to that file at the end. **last** lists info in reverse order; that is, it lists most recent events first, even though the most recent events are stored at the end of the file.

Reading the login info file   Write a short program that reads the login info file in reverse order, record by record, and prints the info to the screen. Use the lecture examples of **who** to get started. That will acquaint you with the open(), lseek(), and read() system calls. It will also acquaint you with the data structure used to store the login information. Add the **-f** option to your program so it can be told to read from a file other than the default login info file.

Look for activity   Next develop a system to read the file with an eye for a particular terminal. The terminal name will be given on the command line, and it may be abbreviated (the tty part can be omitted). Your program should print an error message if it does not get one terminal name as its argument. You still need to read the file in reverse order, but do not print information about the login session until you get to the login entry (this will make sense when you get into the project). You need to print the range of times as well as the elapsed time.

Testing your program Test your program by comparing its output to the output from the version of **last** on the system. These three commands will run both versions and compare the output:

```
% last ttyXX > output.last
% llast ttyXX > output.llast
% diff output.last output.llast
```

note: do not type "ttyXX". Use the name of a real terminal, one that has a history of logins. Type the command **tty** to see the tty you are using.

Getting Finished

Once you get the basic version working, watch for people who do not logout but who are caught in crashes or system shutdowns. What does ‘getting caught in a crash’ look like in the file? Use the dumputmp.c program and the three sample wtmp files to investigate.

After you have the basic version working and handling shutdowns and crashes, tune your program by buffering file input. Isolate your buffering into a separate file with a clearly defined set of functions. By doing so, you can modify how you read from the file without having to change the main program. You program must do some sort of input buffering.

Handing in Your Homework

Hand in source code to your program, a sample of it running, and a sample of it being tested on the three sample wtmp files. Be sure to show that it can accept an abbreviation for the tty.

Extra Credit

You can earn 10 points of extra credit if you expand your program to handle an arbitrary number of ttys named on the command line. The regular **last** does this. Try it.

You won’t get extra credit for adding logname arguments, but you might think through how you would implement such an feature. What sort of data structure would you need? What sort of data management functions would these data structures require? How can you make these efficient?

Warning

Be careful not to use SEEK_END with lseek for this project. The wtmp file might grow while your program is running. If that happens, the end of the file moves, so lseeks relative to the end are not likely to set the file position where you want.