Programming Project: almost ac

Introduction

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

ac is an accounting command that computes login time for users. For Linux, the command is included in the RPM package called psacct.

Specific Details

Write a program called **aac** (for **a**lmost-ac) that implements two parts of the regular **ac** command: the part that handles a single logname as a command line argument and the part that handles alternate login info files (the -w option). That is, your program should handle commands such as

```
% aac smith
% aac -w sample.wtmp smith
% aac smith -w sample.wtmp
```

The output of your program should be identical to that produced by the version of ac on the systems here. In the example just given, it might look like:

```
total     2.67
```

Getting Started

Make a subdirectory for this project. Change into that directory and link to the sample files and reference material. Type

```
mkdir aac
cd aac
ln   -s   ~COURSENAME/hw/aac/files/*   .        <- that's a dot at the end
```

Read the manual page for **ac**. Try the **ac** command. Find what file **ac** uses to get its data. Read the manual page for that file. Figure out to use that information to compute total connect time for a user.

The files include a program, **utmpcopy.c** , that you can use to make a subset of the real wtmp file. A sample file does not change; the real one does, and it is easier to test your software on a fixed set of data. You can use a sample subset to experiment with the real ac and with your program. To tell **ac** to use this file, use the -w option.

The sample files also include a program, **dumputmp.c** that prints out the contents of utmp files. Read the program, study the header file, make sense of data structures. To use **dumputmp** to examine the sample file, compile the program then type: `dumputmp sample.wtmp`

A wtmp file contains records of users logging in and logging out. A USER process means a user has logged in and is running user commands at the terminal, and a DEAD process means the user process has died. Examine the output of dumputmp to see what it contains and think how to process those records.

The purpose of your program is to compute total login time for a user specified on the command line. You need to devise a system to read the file with an eye for a specific logname. Your program should print an error message if it does not get one logname as an argument. (Note: the real ac program prints a total of all login session times if no argument is given; your program does not have to do this.) Read the file looking for records that represent logins for that user. Then look for the corresponding logout on that terminal. When your program finds the corresponding logout, the program can compute the connect time for that session at a terminal.

Special Conditions

Once you handle simple logins and logouts, enhance your logic to handle fancier conditions. First, the same user can be logged in at several terminals at the same time. All these connect sessions are counted separately. For example, if you login at one terminal from noon to 2pm, and at another terminal from 1pm to 4pm, you accrue 5 hours of connect time.

The next thing to worry about is that not every login has a matching logout. For example, what happens if a user is still logged in when you run **ac**? In that case, you have to use the current time as the ‘logout’ time. Second, what happens when the system crashes? In this case there will be a ‘reboot’ entry in the login data file. It is also possible for a user to type ‘login’ to the prompt and login as another user without explicitly logging out. Doing so creates a new login entry in the table without creating a matching logout process. Finally, the system administrator may change the system time. For example, how does setting the clock back an hour affect the time between login and logout entries?

Getting Finished

When you have the basic version working and handling special conditions, tune your program by buffering file input. The buffering code must be in 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 may use the buffering functions in **utmplib.c**, or you may write your own. Your program must do some sort of input buffering.

Testing Your Program

Your program is supposed to be an exact subset of the regular **ac**, so you can test your program by comparing its output to the output of **ac**. Here’s how:

```
/usr/sbin/ac $USER > output.ac
aac $USER > output.aac
diff output.ac output.aac
```

You can read the manual about the **diff** command.

What to Hand in

Submit your code for this solution, a Makefile, a project plan, and a README file.

Extra Credit

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

Note for Other Systems

You can develop and test your program on another Unix machine. Note that the size and format of wtmp files varies from one version of Unix to another. The BSD version is much simpler than the SYSV style many systems, including Linux, use. Your program should be portable, the data may not be portable. If you use GNU/Linux and do not find **ac** on your system, you can install it from the package called **psacct**.