Writing Your Own PS Program With Options

Fall 2020

There are several goals for this assignment.

1. (Re-)familiarize yourself with the C programming language and C calls
2. Learn how to use some of the I/O facilities and library functions provided by UNIX and C. There are likely to be some new functions that you have not seen yet.
3. Get experience with some of the system level interfaces provided in Linux. Operating system have many interesting dark corners, and you will get a chance to peek around one of them.
4. Get experience with separate compilation

Your assignment is to write a simple version of the `ps` command. Your version of `ps`, called `5ps` be executed from a command line. It will read a variety of information about one or more running programs (processes) on the computer, and then display that information. Since it takes up to 5 arguments it is called 5ps. Note that the arguments may appear in any order. When –p is used, it must be followed by a pid number.

As an example of what your program will do, consider a command such as:

```
5ps -p 1234 -s -t -c -v
```

This runs your 5`ps` command, and displays the status letter (in this case, running), amount of user and system time, and the command line for process 1234. You might expect the output to look like:

```
1234: R time=00:03:14 sz=123607 [myprog -x -y file1 myoption]
```

UNIX/Linux Manual Pages and Other information Sources

Any serious systems programmer working on Linux (or other UNIX-like systems) needs to be familiar with the manuals and the online "man" facility. This will be a great help in working on these assignments. For example, if you wanted to know how the `ps` (create process) command works, you would type:

```
man 1 ps
```

Or you can find the manual page on the web using Google. However, if the man pages are installed on your system, then the man pages are accessible via the `man` command, they will be the definitive and accurate version for your system.

The UNIX manual is organized into many sections. You will be mainly interested in the first three sections. Section 1 is for commands, like `ls`, `gcc`, or `cat`. Section 2 is for UNIX system

calls (calls directly to the UNIX kernel), such as `fork`, `open`, or `read`. You will typically not use Section 2. The UNIX library routines are in Section 3. These are calls such as `atof`, or `strcpy`.

Another function in Section 3 that will be useful for this assignment is the C library function that handles command-line argument parsing. The man page for that function is found by typing:

```
man 3 getopt
```

Note that you want to make sure that you get the `getopt` from Section 3, as the `getopt` in Section 1 is a command, not a library (and not what you need). The cool part about the `getopt` web page is that there is an "EXAMPLE" section that shows you how to use it. This example will help you to get started.

You will need to be able to read the contents of a Linux directory (in this case, `/proc`. For this purpose, you will use the `readdir` library function. The man page for that function is found by typing:

```
man 3 readdir
```

Here, the "3" is also essential, or you will get the man page for a lower-level function that is much harder to use. There are a lot of examples of code on the Web showing you how to use `readdir`.

There is another, less well-known section of the manual, which will be critical for this assignment. You will need to understand the format of the `/proc` file system (sometimes called the `procfs`), and for that use, you would type:

```
man 5 proc
```

You can also learn about `/proc` from Web sources.

Program Details

Program Features

Your program will implement the features triggered by the following options. If no options are present, only “5ps” is run, exit the program with no output.

`-p <pid>`
: Display process information only for the process whose number is `pid`. If this option is not present then use the pid number a 1 (init process). The user is expected to provide pid number when using –p option. If the user runs “5ps -p” with no value please give an error message and exit the program.

`-s`
: Display the single-character state information about the process. This information is found in the `stat` file in process's directory, looking at the third ("state") field. Note that the information that you read from the `stat` file is a character string. If this option is not present, do not display this information.

`-t`

Display the amount of time consumed by this process in hours:min:seconds format. This information is found in the `stat` file in process's directory, looking at the "utime" field. Add to this to the system time, “stime” field . This is the total time consumed by the process. This time is represented in clock ticks. You must divide my number of clock ticks per second (system constant) to obtain the number of seconds. It should then be displayed in hours:min:seconds format. If this option is not present, do not display this information.

`-v`

Display the amount of virtual memory currently being used (in pages) by this program. This information is found in the `statm` file in process's directory, looking at first ("size") field. If this option is not present, do not display this information.

`-c`

Display the command-line that started this program. This information is found in the `cmdline` file in process's directory. Be careful on this one, because this file contains a list of null (zero byte) terminated strings. If this option is not present, do not display this information.

All of your output must match with the output of “ps –ely” for the same pid.

Program Structure

Key to any good program, and a key to making your implementation life easier, is having a clean, modular design. With such a design, your code will be simpler, smaller, easier to debug, and (most important, perhaps) you will get full points.

Even though this is a simple program with few components, you will need to develop clean interfaces for each component. And, of course, each component will be in a separately compiled file, linked together as the final step.

Some suggestions for modules in your design: (at least two .c files and an .h file is required).

- *Options processing:* This module processes the command line options, setting state variables to record what the options specify.
- *`stat` and `statm` parser:* This module will extract strings from the space-separated lists that are read from the `stat` and `statm` files.

Testing Your Program

First test the separate parts of your program, such as processing command line options, listing files in `/proc`, and reading and parsing the individual files in a process's `/proc` directory.

Next, start assembling these pieces into a whole program, testing the options one at a time. You will want to learn to use shell scripts, so you can set up sequences of command lines to run over and again. You can navigate to the stat file of a specific processing by “cd”ing into /proc and then into specific process and then typing “more stat”. This way you can ensure that data is being read correctly from the proc file system.

Deliverables

You will turn in your programs, including all .c and .h files. Also include a README file and DEMO file as described in another document in Blackboard.

Original Work From The Team

This assignment must be the original work of you and one teammate (if any). Unless you have explicit permission, you may not include code from any other source or have anyone else write code for you.

Use of unattributed code is considered plagiarism and will result in academic misconduct proceedings as described in the Syllabus.