Project 1: The /PROC File System

There are several goals for this assignment.

1. Continue to familiarize yourself with the C programming language. Dealing with separate compilation is likely to be new to some of you.
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 topeek around one of them.
4. Get some insights as to how an important Linux utility works.
5. Learn about the /proc filesystem.
6. Get more experience with separate compilation, makefiles, and the Linux gdb debugger.

Task

Your assignment is to write a simple version of the `ps` command. Your version of `ps`, called `TUps` will 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. As an example of what your program will do, consider a command such as:

```
TUps -p 1234 -s -U -c
```

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

```
1234: R utime=150 [myprog -x -y file1 myoption]
```

UNIX/Linux Manual Pages and Other information Sources

As noted previously, a serious systems programmer working on Linux (or other UNIX-like systems) needs to be familiar with the manuals and the online "man" facility. if you wanted to know how the `ps` (list processes) command works, you would type:

```
man 1 ps
```

Or you can find the manual page on the web using a Google search. However, the versions installed on your particular Linux system, accessible via the `man` command, will be the definitive and accurate version for your system.

The UNIX manual is organized into many sections, and 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`.

For this project, 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`. You were introduced to the function in Project 0.

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
```

The /proc Filesystem

The /proc file system provides access to the state of each active process and thread in the system. The /proc file system contains a directory entry for each process running on the Linux system. The name of each directory in the filesystem is the process ID of the corresponding process. These directories appear and disappear dynamically as processes start and terminate on the system. Each directory contains several entries providing access to information about the running process. From these process directories the /proc file system gets its name.

These entries are subdirectories and the owner of each is determined by the user ID of the process. Access to the process state is provided by additional files contained within each subdirectory. Except where otherwise specified, the term /proc file is meant to refer to a non-directory file within the hierarchy rooted at /proc.

/proc is known as a "pseudo-filesystem", which means it is not a true filesystem that is consuming storage. The files and directories in /proc are entry points into kernel tables, such as the open file table or the process table.

Some of the directories in the /proc filesystem are:

directory	**description**
/proc/PID/cmdline	Command line arguments.
/proc/PID/cpu	Current and last cpu in which it was executed.
/proc/PID/cwd	Link to the current working directory.
/proc/PID/environ	Values of environment variables.
/proc/PID/exe	Link to the executable of this process.
/proc/PID/fd	Directory, which contains all file descriptors.
/proc/PID/maps	Memory maps to executables and library files.
/proc/PID/mem	Memory held by this process.
/proc/PID/root	Link to the root directory of this process.
/proc/PID/stat	Process status.
/proc/PID/statm	Process memory status information.
/proc/PID/status	Process status in human readable form.

You can learn about /proc from Web sources such as this chapter from "Advanced Linux Programming" (Mitchell, Oldham, Samuel) or this chapter from Redhat documentation. Directory access was used in Project 0, and here is a sample program that opens a directory and reads the directory entries using the readdir() system call.

Exploring /proc

Most of the files in /proc are in text so you can learn about /proc and explore a bit just by going to the /proc directory and using programs such as cat, more, or grep to look around. This project will require you to locate information from the/proc directories and files and then extract and display that information.

Project Features

Your program will implement the features triggered by the following options for the ps command.

-p <pid>
: Display process information only for the process whose number is pid. It does not matter if the specified process is owned by the current user. If this option is not present, then display information for all processes of the current user (and only ofthe current user). You only need to support a single -p option.

-s
: Display the single-character state information about the process. This information is found in the stat file in the process's directory, by looking at the third ("state") field. Note that the information that you read from the stat file is a character string.

-U
: DO NOT Display the amount of user time consumed by this process. This information is found in the stat file in the process's directory, by looking at the "utime" field. If this option is not present, then user time information is displayed.

-S
: Display the amount of system time consumed so far by this process. This information is found in the stat file in the process's directory, by looking at the "stime" field.

-v
: Display the amount of virtual memory currently used (in pages) by this program. This information is found in the statm file in the process's directory, by looking at the first ("size") field.

-c
: DO NOT Display the command-line that started this program. This information is found in the cmdline file in the process's directory. Be careful with this one, because this file contains a list of null (zero byte) terminated strings. If this option is not present, then command line information is displayed.

If there are multiple options that are contradictory, such as -s followed later by -s-, the last occurring option determines the option setting.

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 perhaps importantly) you will get full points for the project.

Some basic coding standards are included at the end of this document. The readability and structure of the code is important to the review and grading of your project.

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. And, again of course, you will have a makefile that controls the building (compiling and linking) of your code.

Some suggestions for modules to develop and include in your design:

- *Options processing:* This module will process the command line options, setting state variables to record what the options specify.
- *Getting the process list:* If there is no -p option, then you will need to look through /proc to find the list of processes belonging to the user. This module will implement that functionality.
- *stat and statm parser:* This module will extract strings from the space-separated lists that are read

from the `stat` and `statm` files.

A library function that may be useful is getopt(). You can find information about the function at: gnu.org, geeksforgeeks, and getopt() man page

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. Make sure to try out an interesting variety of commands. The TAs will provide you with suggestions of what to test.

Deliverables

You will turn in your programs, including all .c and .h files and your makefile. Also include a 'personal README' file which describes your design and what you did to test this project and its implementation.

Note that you must run your programs on the Linux systems provided by the CIS Department. You can access these machines from the labs on the second floor of the SERC or from anywhere on the Internet using the ssh or putty remote login facility.

Handing in Your Assignment

You will be developing this project on your own system or a Temple workstation. You are to use your GitHub repository for the project, making commits often. The final project code and documents are to be submitted to the Project 1 assignment in Canvas. The due date for the project is the date by which your final submission is to be made to Canvas. The demo and evaluation of the project will be made from the code in Canvas, meeting the submission deadline.

You can make multiple submissions to Canvas and the weekly deliverables should be a Canvas submission by the week deadline.

Original Work

This assignment must be your original work. Unless you have explicit permission from your TA, you may not include code from any other source.

Use of unattributed code is considered plagiarism and will result in academic misconduct proceedings (an "F" in the course and a notation on your transcript).

Some Coding Standards

To help develop some professionalism in the code you write, and to make your code more readable (by you as well), follow at least this list of principles:

1. Check the return value of all system and library calls.
2. Always indent when statements are part of a control-flow statement such as "if" or "while". Choose a reasonable number of characters (such as 3 or 4) to indent and make sure that you indent consistently.
3. There should be no constants other than, perhaps, zero or one, in your code. Any constant values should be defined as using a "const" declaration.
4. Global variables are, in general, a really bad idea. Pass parameters to functions.

5. Comments:
   - Each file should have a comment block at the top with at least your name
   - Each function or method should have a comment at the beginning summarizing what it does and, if it is not obvious, how it does it. This comment should describe the input and output parameters.
   - For long functions or methods, you should include a comment at reasonable intervals. The intervals should break the function into logical chunks. The comment should summarize what the chunk does.
6. You are to use a "makefile" for your program. The default rule should build the whole program. You also need to have a "clean" rule that removes all .o and executable files. A brief Makefile tutorial is here. The GNU make manual is here
7. You can use (and are strongly encouraged to use) standard Linux library functions.
8. Including someone else's code in your program without citing the source of that code is plagiarism. Including someone else's code in your program also requires that the code is labeled with the permission to do so. Code with no copyright markings or permissions is not public or open source; it is poorly labeled private code to which you have no usage rights. For this course you are not permitted to include someone else's code in your program.
9. 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. 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. And, again of course, you will have a makefile that controls the building (compiling and linking) of your code. Be careful of going to the other extreme: one function per file is not good structure.

Deliverables:

Week 1:

Parsing the command line
Makefile as defined in the project

Week 2:

Code to get the process list
Design and implementation of parsers to extract from the various /proc files

Week 3:

Implementation of the specified options
Design and implementation of the display formats
Complete documentation of code and testing