ELEC 377- Operating Systems
Lab 1 - Introduction to Linux Programming
Lab Date: September 12/18, 19/25, 2011. Due: Sept 23/30, 2018

Objectives

- Familiarize the student with C programming, Linux basics, file locations, terminals, basic commands
- List running processes using */proc* interface

Linux Basics

As you know by now, Linux is an open-source, free operating system which will be used to better understand concepts and techniques in this operating systems course. Linux is primarily written in C, thus C is the language used in this course. We are using Slackware 10, which is a slightly older version of linux. The kernel is somewhat simpler (making these assignment easier) and it takes less space on the lab disks.

The *shell* - A "shell" is an interactive command interpreter. Once the prompt appears (eg `student@vmname:~>`) you can start entering commands. The default shell on our systems is *bash*. One difference between Linux and Windows is that you must always put a space between the command and the parameters. For example “cd /proc” instead of “cd/proc”.

Here are some useful commands standard to linux/unix systems:

ls - LiSt - lists the files in the current directory.
cd <new directory> - Change Directory. Just like Windows, this command changes the current working directory to the specified new directory. Unlike Windows, executing the command without a parameter does not print the current working directory. It takes you back to your home directory.
pwd - Print Working Directory. Display the full path to your current directory.
ps - Display running programs which you own on this terminal.
ps a - Display running programs which you own on all terminals.
ps aux - display all running programs on the system in extended format
startx - Start a new X windows session.
cat <file(s)> - concatenate files to the screen. Use to view the contents of a file.
more <file(s)> - like cat but stops every time you fill a window. Use the space bar to go to the next screen.
startx - Start a new X windows session (home vms only)
man - Displays the *man*ual (help) page for something. There are man pages for most linux commands and kernel functions. A very good source of information.
gcc - This is the open-source GNU C compiler. It generates *a.out* as the executable by default. To view the full list of options available for gcc, use *man gcc*.
make - takes a program description from *makefile* and runs the compiler for you

The /proc Pseudo-file system

As you found out in the introductory lab, the */proc* directory in the linux file system is not a "normal" type of directory. Usually a directory holds files and possibly other directories. */proc* is actually a file-based interface to the kernel. The "files" are basically variables, counters and other system information.

For example, the */proc/uptime* file:

```
student@slacktest:/proc> cat uptime
7149.72 6724.80
student@slacktest:/proc> cat uptime
7151.72 6725.90
```

The first number is the number of seconds the system has been running, the second number is the number of seconds it has been running idle. This "file" changes and will be different each time you read it.

Thus, the contents of /proc and the files in /proc are dynamically generated whenever needed. We will take advantage of this in the lab to generate a listing of currently running process numbers, names and status.

Each process has a 2 byte process id (0-65536). In the /proc directory, each process is represented as a directory. The name of the directory is the process id. So the first process created when the machine boots, named *init*, has a process id of 1 and is represented by the directory named “1” in /proc. The directory contains several files. The two files we are interested in are *status* and *cmdline*. The first file, *status*, contains the status information of the process. Each line contains a separate piece of information such as the name of the process, the process status and the numeric user id.

Objective:

- *Write a C program to list information about the currently running processes, giving the id , name, status, and symbolic user id.*
- *Become familiar with using structs, pointers and the /proc file system.*

Write a program to display a list of the running processes, giving the process id, the name of the process, the status, the user id and the command line to the screen. Your list will be similar to the PID list given by the "ps -A" command.

- The library function *scandir()* creates an array of pointers to structs of type *dirent*, which contains the file names. It takes 4 parameters. The first parameter is the directory to read. The second is the *address* of a variable used to return the directory listing. The other two parameters are function parameters used to handle filtering and sorting. We will only use the first one, the *select* parameter which is the third parameter to *scandir*.

```
#include <dirent.h>

int isProcessDir(constr struct dirent*){
  ... return 1 if a process, 0 if not a process...
  ... i.e. all characters of the name are digits...
}

int main(){
    struct dirent **namelist;
    int n;
    …
    n = scandir("/proc", &namelist, isProcessDir, NULL);
    …
}
```

The variable `namelist` now points to an array of pointers to `dirents`

- The property of the struct *dirent* you will need is *d_name*, the name of a file or directory (i.e. namlist[i]->d_name)
- True and False boolean types are not defined in C. Generally, 0 is false, non-zero is true.
- *isdigit()* can be used to check if a file name is a number or not. *You **must** check that **all** characters in the name are digits*.
- Your isProcessDir function will be called by the *scandir* routine for each entry in the directory. You should return non-zero if the entry is a process and zero if the entry is not a process. While you could obtain all of the entries and check the name later, one of the purposes of this assignment is to understand function pointers.
- *sprintf(string1,"%s%s",string2, string3)* where string1 is an array of characters, and string2 and string3 are C strings can be used to concatenate strings.
- Use appropriate string library routines for manipulating the contents you read from files

So essentially, this lab involves calling *scandir* and using a loop to look at each entry in the array. For each entry, you have to open the status file inside of the directory and read the name, status, user id and group id from the file. You must print the name of the directory and the appropriate parts of the lines read from file.

Sample output:

```
The currently running processes are:
#   Name         Status         User   Group
1   init         S (sleeping)    0     0
2   keventd      S (sleeping)    0     0
3   kapmd        S (sleeping)    0     0
4   ksoftirqd    S (sleeping)    0     0
5   kswapd       S (sleeping)    0     0
...
27  vi           S (sleeping)    1000  100
```

What to Hand in

At the end of the first Wednesday lab, save your current to your svn repository account as a backup.

After the end of the second lab period (Sept. 19/25) save your program and your test results to your svn account. You have until 9:00 am, Sept 23/30 to write up your lab. Your write-up must contain a description of your program. You must also include a description of your testing. Make sure your description of the testing is in a separate document from the description of your program. Both the program and the testing write-ups are to be checked into your svn account. They must be plain text or pdf (Adobe Acrobat) files. Make sure that both your name and your partner's name are in the documentation.

Before coming to the lab, think about how you will test your program. How will you show the input to your program (Hint, what system command tells you about processes)?