COMP 3413 – Operating Systems – Lab 2
Linux Familiarization and Introduction to Processes

Purpose

The goal of this lab is simply to introduce you to the labs themselves and to familiarize you with the creation and use of processes under Linux.

How to Prepare for the Lab

Your firststep in preparation for the lab is to, if necessary, review the basic Linux/Unix commands (including `ls, cd, pwd, less/more, gcc, ps,` etc.) and C program syntax. For those unfamiliar with Linux/C, there are links to online references that are available on the course homepage. Your third step is to read over the lab questions described below and make sure you understand them generally. The lab demonstrator will be available to answer specific questions during your actual lab. Your final steps in preparation for the lab should be to quickly review the notes covered in class, up to the end of the section on processes. You may find it useful to bring the notes with you to your lab.

What to do?

This lab will be done using one of the Linux machines, which you will access remotely or use your VM of an FCS machine. Follow the directions below providing answers to the questions asked on the remainder of this sheet as you do so. Be sure to hand in your completed lab hand–in sheet to D2L before the end of your lab day. This is how you will be graded!

The Linux /proc pseudo file system

Linux, like some other Unix variants, utilizes the `/proc` pseudo file system which provides the ability for users to inspect certain characteristics of the operating system and processes and the machine they are running on. (It is a "pseudo" file system because the "files" are stored in memory and don't actually exist on disk)

Questions

1. Login to a FCS Linux machine using SSH. Note the name of the machine you are logged into.

2a. All versions of the Unix operating system provide a version of the `ps` (process status) command. Type `ps` and list the PID (process ID) of your shell.

2b. List the contents of the `/proc` directory (using `ls /proc`). Each process running on the machine has a numerically named directory in the listing (the number is the same as the process ID). There will be a directory for your shell process. Examine the status of your shell process by examining the contents of its status file (for example, if your shell process has PID 9827, use `less /proc/9827/status`). What is the state of your shell? Can you explain why it is in this state?

2c. A process can examine its own status, without knowing its PID, by using the `self` directory, which is a link to the currently running process. Type the command `less /proc/self/status`. Which is the name of the currently running command, and what is its state?

3 On the lab 2 page of the course d2l web site you will find a program called `os_info.c`, which reads files in `/proc/sys/kernel` and prints information about the operating system.

3a. Download, compile and run the program on the CS Linux system.

There are several methods for downloading the program.

For a small file such as this, you can copy it in the browser window and paste it into an editor (not really a download, but it works fine).

You can download directly to the Linux system using the `wget` utility. From the command prompt, you simply supply the URL of the file you wish to download.

You can download the file onto the Windows machine and use ftp to transfer it to the Linux system. This method allows you to use a Windows GUI editor to make changes to the program. This method is clunky and **not recommended**

You can also use some sort of Integrated Development Environment (IDE) when working on programs, but I won't be discussing any in class.

Compile and run the program. What does the OS field say?

3b. Add new code to your program such that it will print the name of the computer (the host name), then compile it and run again. You can get the host name from another file located in `/proc/sys/kernel`. What is your hostname?

[10] 4. Using the programs `fork.c` and `myexec.c` on the lab page as a starting point, create a program `myshell.c` that will repeatedly read a line of user input (containing a Unix/Linux command to be executed) until the user enters `CTRL-D` (end of file). For each line entered, your program should `fork` a process that will execute the provided command using one of the `exec` commands. Your parent process should use `wait` (see `man 2 wait`) to await the completion of the child process. Your program should handle invalid commands by printing a suitable error message. This program is essentially a rudimentary command shell (like the one you used to enter your commands to compile and run your program). To do this question you will need to read a little more on the `exec` system call (see `man 3 exec`). You are free to use whatever resources you like. Don't expect the lab demonstrator to solve this problem for you during the lab!

Shown below is a pseudo–code implementation of the shell program.

```
char *cmd;
int return_code;

get cmd
while (cmd != NULL)
  fork a child process
  if(this process-id != 0)    // parent process after fork
    wait(NULL);  // parent waits for child process to finish
  else  // child process after fork
    exec cmd
    if(return-code != 0) // exec command failed
      exit(0); // this step is important (why?)
    end if
  end else
  get cmd
end while // shell terminates
```

When finished submit your completed program using the dropbox on the D2L by the end of the day – if you do demonstrate, then a dropbox submission is not necessary.