COMP 3430 – Operating Systems – Lab 1 Winter 2013
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.

Introduction

You have been given a computer account that will work on the machines in both E2–420 EITC (where you will physically sit to do this lab) and in E2–430 EITC (where you can "drop in" to do work when necessary). Your instructor will give the combo to E2–430 EITC in class. You have also been given a separate computer account that will work on the Computer Science Linux machines. The programming questions from the labs and assignments will be performed using the CS Linux machines. For both the Linux and Windows accounts, your userid will be the same as your UMnetID and your initial password will be your student number unless you already had an account for these labs in which case your old password should have been maintained. You will use the Linux machines (named `blackbird`, `cormorant`, `crow`, `eagle`, `falcon`, `finch`, `goose`, `grebe`, `grouse`, `hawk`, `heron`, `killdeer`, `kingfisher`, `loon`, `nuthatch`, `oriole`, `osprey`, `owl`, `pelican`, `robin`, `sandpiper`, `sparrow`, `swan`, `woodpecker` and `wren`) via remote login (please use ssh). Note that `aviary` connects you to a random machine.

How to Prepare for the Lab

Your first step in preparation for the lab should be to visit E2–430 EITC and login to make sure that your accounts work and to change your passwords (for Linux you can use the `yppasswd` command to change your password). Do this before your first lab since it may not be possible to solve userid/password problems during the lab. Your second step 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. **YOU ARE STRONGLY RECOMMENDED TO LEARN A LINUX EDITOR such as vi or pico, and work directly on the linux machine instead of copying files back and forth.** 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. Be sure to print a copy of this lab document and the hand–in sheet (if you have not already done so) to bring with you to the lab. In addition, note that there is an examples webpage posted: `http://cs.umanitoba.ca/~young/Files/3430/`

What to do?

This lab will be done using one of the Linux machines, which you will access remotely from the machines in the lab in E2–420 EITC. 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 the demonstrator before you leave the lab. 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.)

SSH

There are a couple of SSH clients you can use, I recommend putty for windows. Google will provide many methods for you. To use SSH in the Mac lab:

1. Login to a Mac using your CS Unix userid.
2. Launch the Terminal application, found in: Macintosh HD > Applications > Utilities
3. Then open a session to the Linux Lab: ssh aviary.

Questions

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

[1] 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.

[1] 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?

[2] 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 Examples page of the course 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.

[3] 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 (e.g. `wget http://cs.umanitoba.ca/~young/Files/3430/os_info.c`). This is probably the easiest choice for a file that needs little or no changes.

You can download the file onto the Windows machine and use ftp to transfer it to the Linux system (SSH has an ftp client built–in). This method allows you to use a Windows GUI editor to make changes to the program.

You can also use some sort of Integrated Development Environment (IDE) when working on programs, but I won't be discussing any in class. The CompSci Windows machines have Bloodshed Dev-C++ installed.

To compile the program, type `gcc os_info.c` at the command prompt. If it is successful, there will be no messages displayed, and an executable called `a.out` is created. To run the program, simply type `a.out` at the command prompt.

[2] 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`.

There are several ways to edit your program; you'll have to decide which way(s) you are comfortable with. To make a small change you can use a text-based editor directly in Linux (e.g. pico or vi). You can also make changes in a GUI editor in Windows, and ftp the changes to the Linux system.

[10] 4. Using the programs fork.c and myexec.c on the Examples 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! Demonstrate your program for the lab demonstrator. The demonstrator will initial and make any comments on your hand–in sheet.

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
```

If you are not able to demonstrate a working program during lab time then notify your TA and 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.