EC605: Computer Engineering Fundamentals Fall 2016
Lab 6: Advanced Linux

Goals

- More in-depth familiarity with Linux features.

Overview

This lab will teach you some advanced techniques used with the Linux operating system including some unique file manipulation and Linux commands.

Tasks

Task 1: Hello World

The first task is one that you have likely done many times before: To write a simple "Hello World!" program in C, compile it with the Linux gnu compiler collection (GCC), and run it in a Linux shell. Your program should be named Lab6_hello.c and the output of the program should say "Hello EC605 from <your buid>!".

Example:

> *> gcc -o hello Lab6_hello.c*
> *> ./hello*
> *Hello EC605 from ruship!*

Task 2: I/O Redirection

Using your Hello World program, the next task is to learn how to redirect output from one executable into a file for later. Using the application *cowsay* and your new compiled c program, your task is to get the cow to say "Hello EC605 from <your buid>!", and to save this output into a file which can be opened later. This will involve using output redirection and piping. Please name the file *Lab6_cowsaid*. Save the command you called into a file called *Lab6_commands*.

To learn about I/O redirection: http://linuxcommand.org/lc3_lts0070.php
To learn about the cowsay application: https://en.wikipedia.org/wiki/Cowsay

Task 3: File Permissions

The */tmp* directory is a local directory, on your specific lab computer. It is not part of your user account, and its content is not saved after you log out of the computer. We will use this directory for this part of the lab. Create a new directory inside the */tmp* directory, called *permissions*. Copy your *Lab6_hello.c* program to the */tmp/permissions* directory.

Within this directory, list its content as follows:

>ls –l

Your output should look similar to the following:

-rwx------ 1 <user> <users> 0 Aug 10 10:58 Lab6_hello.c

In a multiuser system, such as Linux, the OS needs a method of preventing users from accessing other users' files.

In Unix/Linux, there are 3 categories of protection settings:
- User – owner, creator of the file
- Group – a predefined list of users (collaborators)
- Other – everybody else, rest of the world

There are 3 types of permissions for each category:
- Read
- Write or delete
- Execute (only applicable if the file is executable)

The *chmod* command allows you to change permissions on a file.
For example:
> chmod g+r Lab6_hello.c

will result in adding read permissions to the 'group' category.

Add read permission for the 'other' category of your *Lab6_hello.c* file. Verify this by typing

> ls –l Lab6_hello.c

Copy the result into the *Lab6_commands* file. On another line, include the command you typed to revoke the permissions.

Note that your user directory, along with other directories located on the server, have pre-defined permissions, and you do not have full control of changing their permissions.

Task 4: Fork

In this task you will use the fork command to write a C program that performs two different functions: one by the parent process, and the other by a child process.

- The parent process should count from 0 to 1000 and print the iteration number and process id of the parent when the iteration number is a multiple of 100.
- The child process should generate the Fibonacci sequence from 0 to 1,000,000 and print the current value if it is even, along with the process id of the child.

Save your program in a file named Lab6_fork.c.

Hint: To get the process id of the parent use *getppid()*.
For more information on fork, you may use *man fork* to view the manual page of this command.

Example Output:
Child pid 4571 2
Child pid 4571 8
Child pid 4571 34
Parent pid 4570 0
Parent pid 4570 100
Child pid 4571 2584
Parent pid 4570 200
…

Task 5: Cat and Grep

The shell command *cat* is used to concatenate files and print to the standard output. The shell command *grep* is used to grab and print lines matching a pattern. See the man pages for more information.

Run the program from the previous task 3 times, saving all outputs into a single file named *Lab6_fork_output*. Once the output is saved, using simple bash commands to read the file and separate the child output from the parent. The child output should be collected into a file called *Lab6_fork_child*, and the parent output into *Lab6_fork_parent*. Save all commands used into the same file from task 2, *Lab6_commands*.

Hint: Appending output redirection, in addition to cat and grep, may be useful here.

Deliverables

- Submit the following files:

  Lab6_hello.c
  Lab6_cowsaid
  Lab6_fork.c
  Lab6_fork_output
  Lab6_fork_child
  Lab6_fork_parent
  Lab6_commands