**COMP 4736 Lab 01: Introduction to Linux Tools**

**Notes:**

* You will need to be comfortable working in Linux environment, especially using command lines. For example, you will need perform basic operations such as creating, editing, copying, moving and viewing files using Linux commands, running commands from the terminal, and using command-line techniques like redirection, using pipes and searching for a string in the output. You will find many helpful tutorials online. Here is one from Ubuntu:  
  <https://ubuntu.com/tutorials/command-line-for-beginners#1-overview>
* For each question, provide **commands entered** and **output screen captures** to support your answers.

**Questions:**

1. In this question, we will understand the hardware configuration of your working machine using the **/proc** filesystem.
2. Run command more /proc/cpuinfo and explain the following terms: processor and cores. Use the command lscpu to verify your definitions. You may want to understand the concept of CPU hyperthreading at a high level before attempting this question.

A screenshot of a computer

Description automatically generated

Processor: The processor which is basically the CPU from that command represents the processor number so in this screenshot, the information displayed is for the first processor. For my machine, I have 16 processors.

Cores: The CPU cores represents the number of physical cores in my system. Each core is able to execute its own set of instructions independent from one and the other. In this case I have 8 cores for each processor.

A screenshot of a computer screen

Description automatically generated

We can verify this if we look at the CPU(s) line which says there are 16 processors, and we have 8 cores per socket.

1. How many cores does your machine have?

I have 8 cores.

1. How many processors does your machine have?

I have 16 processors.

1. What is the frequency of each processor?

A screen shot of a computer

Description automatically generated

1. What is the architecture of your CPU?

A black background with white text

Description automatically generated

1. How much physical memory does your system have?

A screen shot of a computer

Description automatically generated

Total is 7833924 kB, but 7023096 kB is available.

1. How much of this memory is free?

6305548 kB of memory is free.

1. What is total number of number of forks since the system booted up?

A screen shot of a computer

Description automatically generated

The total number of forks since system booted up is shown in processes which is 2361.

1. What is the number of context switches made by init?

A screen shot of a computer program

Description automatically generated

1. In this question, we will understand how to monitor the status of a running process using the **top** command. Compile the program cpu.c given to you and execute it in the bash (or any other shell of your choice) as follows.

$ gcc cpu.c -o cpu -Wall

$ ./cpu

This program runs in an infinite loop without terminating. Now open another terminal, run the top command and answer the following questions about the cpu process.

A screenshot of a computer

Description automatically generated

1. What is the PID of the process running the cpu command?

PID is 1724.

1. How much CPU and memory does this process consume?

From the screenshot, it seems like its consuming 100% of the CPU but 0% of the memory which makes sense because we are not actually storing any of the data and simply running an infinite loop.

1. What is the current state of the process? For example, is it running or in a blocked state or a zombie state?

Its in the running state because in the S column when entered the top command, we can see that it is labeled as R.

1. In this question, we will understand how the Linux shell (e.g., the bash shell) runs user commands by spawning new child processes to execute the various commands.
2. Compile the program cpu-print.c given to you and execute it in the bash or any other shell of your choice as follows. Open another terminal and use the **ps** command with suitable options to find out the pid of the cpu-print process.

$ gcc cpu-print.c -o cpu-print -Wall

$ ./cpu-print

This program runs in an infinite loop printing output to the screen. You may want to explore the ps command thoroughly to understand the various output fields it shows.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Find the PID of the parent of the cpu-print process, i.e., the shell process. Next, find the PIDs of all the ancestors, going back at least 5 generations (or until you reach the init process).



1. We will now understand how the shell performs output redirection. Run the following command.

./cpu-print > /tmp/tmp.txt &

Look at the proc file system information of the newly spawned process. Pay particular attention to where its file descriptors 0, 1, and 2 (standard input, output, and error) are pointing to. Using this information, describe how I/O redirection is being implemented by the shell.

A screen shot of a computer screen

Description automatically generated

We can see that standard input which is 0 is pointing to /dev/pts/0 while outputting to the /tmp/tmp.txt file by redirecting to that folder and finally any errors that occur will be shown in the /dev/pts/0 as seen on the last line.

(d) Next, we will understand how the shell implements pipes. Run the following command.

./cpu-print | grep hello &

Once again, identify the newly spawned processes, and find out where their standard input/output/error file descriptors are pointing to. Use this information to explain how pipes are implemented by the shell.

A screen shot of a computer

Description automatically generated

We can see that the pipe is created, and the standard output of the cpu-print is redirecting to the pipe 34898 which the same pipe that is being used by the grep’s standard input as seen in the screenshot. The standard input and error for cpu-print are both pointing to /dev/pts/0, similar thing can be seen with the grep’s standard output and error.

1. When you type in a command into the shell, the shell does one of two things. For some commands, executables that perform that functionality already come with your Linux kernel installation. For such commands, the shell simply invokes the executable like it runs the executables of your own programs. For other commands where the executable does not exist, the shell implements the command itself within its code. Consider the following commands that you can type in the bash shell: cd, ls, history, ps. Which of these commands already exist as executables, and which are implemented by the bash code itself?

cd: This is a built-in command in the shell, and it doesn’t exist as an executable since the shell itself handles the command.

ls: This is an external executable and is not built in the shell like ‘cd’.

history: This is also a built-in command in the shell since this command displays the history of commands the user entered, the shell itself would manage and keep track of those commands internally so it is not an executable.

ps: This is an external executable command since it provides information about all of the currently running processes it will exist in the system level and not the shell itself.

**— End of Lab 01 —**