CS 332/532 – 1G- Systems Programming

Lab 8

Objectives

1. Review processes in a Linux environment
2. Monitor processes in Linux environment
3. Explore the /proc file system

1. Processes in a Linux environment

User processes in a Linux environment could be in one of the following three states: foreground, background, or suspended.

Most interactive applications that take input from the keyboard or command-line argument and display output in a terminal are considered as foreground processes. Till now, we have been executed all our programs as foreground processes by typing the name of the command in the bash shell.

Non-interactive processes that are typically not connected to a terminal and execute in the background are considered as background processes. You can execute a program in the background by typing the program name followed by the symbol & at the end. You will notice that the shell returns the command-prompt with the background process number and the corresponding process identifier (PID) of the process that was created. For example:

```
$ nano myprog.c &
[1] 16946
$
```

You can display various jobs that are currently running in the background using the *jobs* command. It will show the job number, the current state of the job, and the job name. For example:

```
$ jobs
[1]+  Stopped                 nano myprog.c
$
```

If you like to list additional information such as the PID of the job you can use the *-l* option. For example:

```
$ jobs -l
[1]+ 16946 Stopped (tty output)    nano myprog.c
$
```

In case of the above example, we have invoked an editor and it has been stopped since it requires terminal to display the output and continue. If we had created a non-interactive job in the background then it would be in the running state. For example:

```
$ sleep 20 &
[2] 17513
$ jobs
[1]+  Stopped                 nano myprog.c
[2]-  Running                 sleep 20 &
$
```

We can bring a job that is running in the background to foreground by using the *fg* command. For example, to switch the *sleep* process to foreground, we specify the job number after the % symbol:

```
$ fg %2
sleep 20
$ jobs
[1]+  Stopped                 nano myprog.c
$
```

You will notice that the sleep process will return the command prompt in the terminal when it completes execution and when you type *jobs* again, it will only show one process. You can use *fg* command to switch to the editor using: *fg %1*. You can continue with your edits, save the file, and exit the editor. After you exit, if you type *jobs* again, you will notice that it does not list any processes since the *nano* process is no longer executing.

If you like to suspend a foreground process then type *Control-Z* when the program is executing and that process will be suspended. For example, if we started the sleep process in foreground and would like to suspend it, then type *Control-Z* and you will see a message similar to what you saw when you started a process in background:

```
$
$ jobs
$ sleep 100
^Z
[1]+  Stopped                 sleep 100
$ jobs
[1]+  Stopped                 sleep 100
$
```

However, notice that the sleep process is stopped (it is not running) unlike the previous case when it was running in the background. If you like the sleep process to continue then you have to using the *bg* command as follows:

```
$ bg %1
[1]+ sleep 100 &
$ jobs
[1]+  Running                 sleep 100 &
$
[1]+  Done                    sleep 100
$ jobs
$
```

Now you notice that the sleep process is running and when it is done you will see the message *Done*in your terminal.

There are special background processes that are started at system startup and they continue to run till the system is shutdown. These special background processes are called ***daemons***. These processes typically end in “d” and some examples are: systemd, crond, ntpd, nfsd, sshd, httpd, named.

If you like to terminate a process that is executing in the foreground, you use Control-C to kill it. If you like to terminate a process in the background, you could bring it to foreground and then use Control-C or use the *kill* command to terminate the background process directly. For example:

```
$ jobs
$ sleep 100 &
[1] 1519
$ jobs
[1]+  Running                 sleep 100 &
$ kill %1
[1]+  Terminated              sleep 100
$
$ jobs
$
```

You can also provide the PID as the argument to *kill* command to terminate a process. We will discuss this in the next section.

2. Monitor processes in Linux environment

You can use the *ps* command to display information about various processes running on a Linux system. Login to one of CS Linux systems and enter the *ps* command, you will see the following information displayed:

```
$ ps
  PID TTY          TIME CMD
16096 pts/0    00:00:00 bash
17053 pts/0    00:00:00 ps
```

By default, *ps* lists processes for the current user that are associated with the terminal that invoked the command and the output is unsorted. The following information is shown above: the process ID (PID), the terminal associated with the process (TTY), the cumulative CPU time in hh:mm:ss format (TIME), and the executable name (CMD). You will notice that there are two processes currently executing – bash and ps (the command you just executed) along with their corresponding process ID (in the first column under PID).

The *ps* command has a large number of options that you can use to get more detailed information about the various processes currently running. We will look at some of these options below, you can find out more about these options using *man ps*.

The *-u username* option lists all processes that belong to the user *username*:

```
$ ps -u puri
  PID TTY          TIME CMD
16095 ?        00:00:00 sshd
16096 pts/0    00:00:00 bash
17147 pts/0    00:00:00 ps
```

Now you notice that there is an additional process (sshd) that belongs to you and there is no terminal associated with that process (hence the ? under the TTY column). This process was started by the OS when you connected to this computer using an SSH client. You can use the -f or -F option to get a full listing:

```
$ ps -fu puri
UID        PID  PPID  C STIME TTY          TIME CMD
puri     16095 16075  0 17:04 ?        00:00:00 sshd: puri@pts/0
puri     16096 16095  0 17:04 pts/0    00:00:00 -bash
puri     17520 16096  0 18:02 pts/0    00:00:00 ps -fu puri
$
$ ps -Fu puri
UID        PID  PPID  C    SZ   RSS PSR STIME TTY          TIME CMD
puri     16095 16075  0 49918  2784   2 17:04 ?        00:00:00 sshd: puri@pt
s/0
puri     16096 16095  0 31344  3816   0 17:04 pts/0    00:00:00 -bash
puri     17521 16096  0 40384  1876   2 18:02 pts/0    00:00:00 ps -Fu puri
```

Now we see several additional fields displayed such the user ID, parent PI, process with command-line options, etc.

Exercise: Review the man page for *ps* and find out what the other fields mean.

By looking at the PID and PPID we can identify that the *ps* command was created by the *bash* process and the *bash* process was in turn created by the *sshd* process. We can display the process tree with the ps command using the --forest option:

```
$ ps -fu puri --forest
UID        PID  PPID  C STIME TTY          TIME CMD
puri     16095 16075  0 17:04 ?        00:00:00 sshd: puri@pts/0
puri     16096 16095  0 17:04 pts/0    00:00:00  \_ -bash
puri     17716 16096  0 18:10 pts/0    00:00:00      \_ ps -fu puri --forest
```

Exercise: Determine the parent process of the sshd process above. Who is the owner of the parent process?

You can use *ps* to list every process currently running (not just processes that belong to you) using the -e option.

Exercise: Type one of the following commands and observe the output.

```
ps -e | more
ps -ef | more
ps -eF | more
ps -ely | more
```

You have to press the **spacebar** to scroll through the list. You can send the output to the *wc* command to determine the total number of processes currently running on the system, for example:

```
$ ps -e | wc -l
165
```

At this the above command was executed on one of the CS Linux system, we had 165 processes currently running on the system, even though there are only three processes that belong to you. What are all these processes? Who is running these processes?

You can also use the ***pstree*** command to display the process tree (you can find out more about the various options supported by *pstree* using *man pstree*). For example:

```
pstree -np | more
```

Similarly, you can use the ***top*** command (instead of *ps*) to display all processes currently running (you have to enter *q* to quit *top*). The *top* command provides a real-time update on the various processes running on the system. You can look at processes that belong to a specific user by typing *u* and the user name when top is running or start *top* with the *-u user* option.

To terminate a process, we can use the *kill* command. The *kill* command can take the PID or the command name and terminate a specific process with the given PID or terminate all processes with the specified command name. We can also specify the type of signal, either as a signal name (*e.g.,* KILL for kill) or signal number (9 for kill), to send to a process with the *kill* command. Here is an example that shows how to use the *kill* command to terminate a process using PID.

```
$ sleep 100 &
[1] 2510
$ kill -TERM 2510
$
[1]+  Terminated              sleep 100
$
```

You can find the complete list of signal names and numbers using the *-l* option of *kill* command. We will discuss more about signals in the later labs.

3. Explore the /proc file system

The proc file system (procfs) is a virtual file system that is created by the OS at system boot time to provide an interface between the kernel space and user space. It is commonly mounted at /proc. It provides information of processes currently running on the system and tools such as *ps* use this to display information about these processes. The table below provides a partial list of some of the useful files in /proc (for complete listing of all files and their corresponding description is available in the man pages, type: *man proc*).

File	Description
/proc/cpuinfo	information about the CPU architecture, used by *lscpu* command
/proc/loadavg	system load averages data, used by *uptime* command
/proc/meminfo	memory usage statistics on the system, used by *free* command
/proc/stat	kernel/system statistics
/proc/version	version of the kernel currently running on the system, used by *uname*command

/proc/[pid]	a subdirectory for each running process
/proc/[pid]/cmdline	command string of the process along with arguments separated by null (‘\0’) character
/proc/[pid]/cmd	a symbolic link to the current working directory of the process
/proc/[pid]/environ	environment variables and values used by the process separated by null (‘\0’) character (use strings /proc/[pid]/environ to display the environment variable and values)
/proc/[pid]/maps	information on memory mapped regions of the process
/proc/[pid]/mem	information to access pages of the process through I/O calls
/proc/[pid]/stat	status information about the process, used by *ps* command
/proc/[pid]/statm	status of memory used by the process, measured in pages

You can list the contents of /proc using the *ls* command and view files using the *cat* command. For files that contain strings separated with null characters you have use the *strings* command to display the contents of such files correctly. Here is an example to look at the environ file for the *bash* process:

```
$ strings /proc/$$/environ
```

Note: $$ in bash refers to the process ID of the current process, you can replace it with the actual PID of bash and test the above command.

Let us take a quick tour of the Linux Shell and Shell Programming
– `LinuxShellTutorial.pdf`

Homework

No submission is required for this homework. Please make sure that you execute all the commands shown above and understand what each command does. We will use them in the future labs.