Process Management

PART 1.

When you execute a program on your Linux/Unix system, the system creates a special environment for that program.

This environment contains everything needed for the system to run the program as if no other program were running on the system.

Whenever you issue a command in Linux/Unix, it creates, or starts, a new process. When you tried out the ls command to list the directory contents, you started a process.

A process, in simple terms, is an instance of a running program.

PART 2

The operating system tracks processes through a five-digit ID number known as the pid or the process ID. Each process in the system has a unique pid.

Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

PART 3

Starting a Process When you start a process (run a command), there are two ways you can run it:

- Foreground
- Background

PART 4: Foreground Processes

By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen. You can see this happen with the ls command. If you wish to list all the files in your current directory, you can use the following command:

```
$ls ch*.doc
```

This would display all the files, the names of which start with ch and end with .doc

```
ch01-1.doc ch010.doc ch02.doc ch03-2.doc ch04-1.doc ch040.doc ch05.doc ch06-2.doc
```

The process runs in the foreground, the output is directed to your screen, and if the ls command wants any input (which it does not), it waits for it from the keyboard.

PART 5: Foreground Processes pt. 2

While a program is running in the foreground and is time-consuming, no other commands can be run (start any other processes) because the prompt would not be available until the program finishes processing and comes out.

EXAMPLE: try sleep 1000 as a command and you will see that your terminal will be taken as a hostage, until the whole process is either finished or stopped.

PART 6: Background Processes

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits. The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another! The simplest way to start a background process is to add an ampersand (&) at the end of the command.

```
$ls ch*.doc &
```

This displays all those files the names of which start with ch and end with .doc

```
ch01-1.doc ch010.doc ch02.doc ch03-2.doc
```

Here, if the ls command wants any input (which it does not), it goes into a stop state until we move it into the foreground and give it the data from the keyboard.

That first line contains information about the background process - the job number and the process ID. You need to know the job number to manipulate it between the background and the foreground.

PART 6: Background processes pt 2

Press the Enter key and you will see the following:

```
[1] + Done ls ch*.doc &
```

The first line tells you that the ls command background process finishes successfully. The second is a prompt for another command.

Part 7: Listing Running Processes

It is easy to see your own processes by running the ps (process status) command as follows:

```
$ps

PID     TTY     TIME     CMD

18358   ttyp3   00:00:00 sh

18361   ttyp3   00:01:31 abiword

18789   ttyp3   00:00:00 ps
```

Part 8: Listing Running Processes pt 2

One of the most commonly used flags for ps is the -f (f for full) option, which provides more information as shown in the following example:

```
$ps -f

UID     PID    PPID   C   STIME    TTY   TIME   CMD

amrood 6738    3662   0   10:23:03 pts/6 0:00 first_one

amrood 6739    3662   0   10:22:54 pts/6 0:00 second_one

amrood 3662    3657   0   08:10:53 pts/6 0:00 -ksh

amrood 6892    3662   4   10:51:50 pts/6 0:00 ps -f
```

Part 9: Listing Running Processes pt 3

Here is the description of all the fields displayed by ps -f command:

UID = User ID that this process belongs to (the person running it)

PID = Process ID

PPID = Parent process ID (the ID of the process that started it)

C = CPU utilization of process

STIME = Process start time

TTY = Terminal type associated with the process

TIME = CPU time taken by the process

CMD = The command that started this process

Part 10: Listing Running Processes pt 4

There are other options which can be used along with ps command:

-a =Shows information about all users

-x =Shows information about processes without terminals

-u =Shows additional information like -f option

-e =Displays extended information

Part 11: Stopping Processes

Ending a process can be done in several different ways. Often, from a console-based command, sending a CTRL + C keystroke (the default interrupt character) will exit the command. This works when the process is running in the foreground mode. If a process is running in the background, you should get its Job ID using the ps command. After that, you can use the kill command to kill the process as follows – >

Part 12: Stopping Processes pt2

```
$ps -f

UID PID PPID C STIME TTY TIME CMD

amrood 6738 3662 0 10:23:03 pts/6 0:00 first_one

amrood 6739 3662 0 10:22:54 pts/6 0:00 second_one

amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh

amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f

$kill 6738

Terminated
```

Here, the kill command terminates the first_one process. If a process ignores a regular kill command, you can use kill -9 followed by the process ID as follows – $kill -9 6738 Terminated

Part 13: Parent and Child Processes

Each linux/unix process has two ID numbers assigned to it: The Process ID (pid) and the Parent process ID (ppid). Each user process in the system has a parent process. Most of the commands that you run have the shell as their parent. Check the ps -f example where this command listed both the process ID and the parent process ID.

Part 14: Zombie and Orphan Processes

Normally, when a child process is killed, the parent process is updated via a SIGCHLD signal. Then the parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed before its child is killed. In this case, the "parent of all processes," the init process, becomes the new PPID (parent process ID). In some cases, these processes are called orphan processes. When a process is killed, a ps listing may still show the process with a Z state. This is a zombie or defunct process. The process is dead and not being used. These processes are different from the orphan processes. They have completed execution but still find an entry in the process table.

Part 15: Daemon Processes

Daemons are system-related background processes that often run with the permissions of root and services requests from other processes. A daemon has no controlling terminal. It cannot open /dev/tty. If you do a "ps -ef" and look at the tty field, all daemons will have a ? for the tty. To be precise, a daemon is a process that runs in the background, usually waiting for something to happen that it is capable of working with. For example, a printer daemon waiting for print commands. If you have a program that calls for lengthy processing, then it’s worth to make it a daemon and run it in the background.

Part 16: The top Command

The top command is a very useful tool for quickly showing processes sorted by various criteria. It is an interactive diagnostic tool that updates frequently and shows information about physical and virtual memory, CPU usage, load averages, and your busy processes.

Part 17: Job ID Versus Process ID

Background and suspended processes are usually manipulated via job number (job ID). This number is different from the process ID and is used because it is shorter. In addition, a job can consist of multiple processes running in a series or at the same time, in parallel. Using the job ID is easier than tracking individual processes.