Process management in Unix / Linux

When you run a program on a Unix system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program if no other program is running on the system.

When you run a program on a Unix system, the system creates a special environment for that program. This environment contains everything needed for the system to run the program if no other program is running on the system.

Whenever you report a Unix command, it creates or starts a new process. When you try to execute a ls command to list directory contents, you start a process. A process, in a simple way, is an example of a running program.

The operating system tracks processes through a 5-digit ID that is known as the pid ID or Process ID. Each process in the system has a unique **pid** .

Pid often repeats because all numbers can be used and the next pid repeats. At any time, it is impossible to have two processes with the same two pid coexist in the system because it is the pid that Unix uses to track each process.

Start a process in Unix / Linux

When you start a process (run a command), there are 2 ways for you to run it:

Foreground Process

Background Process

Foreground Process in Unix / Linux

By default, all processes that you start running are Foreground Process. It receives input from the keyboard and sends output to the screen.

You can observe this happening with the ls command. If you want to list all the files in the current directory, you can use the following command:

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

It will display all the files the name starts with and ends with .doc.

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

The process runs in Foreground, its results are directed on my screen and if the ls command wants any input, it waits from the keyboard.

While a program is running in the Foreground and it takes a long time, we cannot run any other commands (start another process) because the prompt is not available until the program is running. submit and exit.

Background process in Unix / Linux

Background Process runs without being connected to your keyboard. If the Background process requires any input from the keyboard, it waits.

The advantage of running a background program is that you can run other commands; You don't have to wait until it ends to start a new process!

The easiest way to start a Background process is to add a ampersand (&) at the end of the command.

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

This also displays all the files the name starts with and ends with .doc:

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

Here, if the **ls** command wants any input (which it does not), it enters the stop state until you move it into the Foreground and give it data from the keyboard.

The first line contains information about the Background Process - the job number and Process ID. You need to know about the Job number to manipulate it between Background and Foreground.

If you press the Enter key now, you see the following:

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

The first line tells you that the ls command in the Background Process has been successfully completed. The second line is a prompt for another command.

List processes running in Unix / Linux

It is easy to observe your processes by running **ps** (process status) as follows:

```
$ ps PID TTY TIME CMD 18358 ttyp3 00 : 00 : 00 sh 18361 ttyp3 00 : 01 : 31 abi
```

One of the flags is used for ps as -f (short for full), which provides a lot of information like the example below:

```
$ ps - f UID PID PPID C STIME TTY TIME CMD amrood 6738 3662 0 : 23 : 03 pts / 
```

Below is a description of all the files displayed by the ps -f command.

Column Description of the **UID of** the user ID that this process belongs to (who runs it). **PID** Process ID. The original **PPID** Process (process ID that starts it). **C** CPU usage of the process. **STIME Process** start time. **TTY** Terminal type associated with the process. **TIME** Time of CPU used by the process. **CMD** Order that begins this process.

There are other functions that can be used in parallel with ps:

Function Description **-a** Indicates information about all users. **-x** Indicates information about processes that do not have a terminal. **-u** Indicates additional information such as the -f function. **-e** Display information is expanded.

Stop the process in Unix / Linux

The end of a process can be done in several different ways. Normally, from a console-based command, sending CTRL + C with the keystroke (default is the interrupt character) will cancel the command. It works when the process is running in Foreground mode.

If a process is running in Background mode, you first need to get a job ID using the ps command and then you can use the kill command to remove the following process:

```
$ ps - f UID PID PPID C STIME TTY TIME CMD amrood 6738 3662 0 : 23 : 03 pts / 0
```

Here the kill command will end the first_one process. If a process often ignores a kill command, you can use kill -9 followed by the Process ID as follows:

```
$ kill - 9 6738 Terminated
```

Mother process and child process in Unix / Linux

Each Unix process has two IDs assigned to it: Process ID (pid) and Parent Process ID (ppid). Each process in the system has a Parent Process.

Most of the commands that you run have Shell as its mother. Check the ps -f example where this command lists both the Process ID and the original Process ID.

Process Zombie and Orphan

Normally, when a child process is dropped, the Parent Process is notified via the SIGCHLD symbol. After that, the original process may perform some other task or start the child process again if necessary. However, sometimes the Parent Process is removed before its child process is removed. In this case, the Parent Process of all processes, "init process", becomes the new PPID (Parent ID Process). Sometimes these processes are called Orphan processes.

When a process is removed, the ps list can still show progress with the Z state. This is the Zombie state, or the process does not exist. This process is removed and not used. These processes are different from the orphan process. These are processes that have been completed but rhyme has an entry in the process table.

Daemon Process in Unix / Linux

Daemons are background processes that relate to the system that usually run with root access permissions and the services required from another process.

A daemon process does not have a control terminal. It cannot open / dev / tty. If you make a ps-ef and look at the tty field, will all daemons have a mark? for tty.

For greater certainty, a daemon is just a process that runs in the background, often waiting for something to happen that is capable of working with, just like a deamon printer is waiting for print jobs.

If you have a program that needs a long process, then its value to create a daemon and run it in the Background.

Top command in Unix / Linux

The top command is a very useful tool for quickly displaying processes sorted by diverse standards.

It is an interactive diagnostic tool that updates regularly and displays information about physical memory and virtual memory, CPU usage .

Here is a simple syntax to run top command and observe the statistical results of CPU usage by different processes:

```
$ top
```

Job ID with Process ID in Unix / Linux

Background and Foreground processes are usually manipulated via Job ID. This number is different from Process ID and is used because it is shorter.

In addition, a job can include multiple processes running in the series or at the same time, in parallel, so using Job ID is easier to track individual processes.

According to Tutorialspoint

Previous article: Filter and Pipe in Unix / Linux

Next lesson: Network communication utilities in Unix / Linux