Process Management

OPS102 Week 5 Class 1

Tiayyba Riaz/John Sellens

February 4, 2025

Seneca Polytechnic

Outline

Monitoring Processes

Process Management in Operating Systems

- Process management is an important concept in all operating systems.
- All programs that are executing on an operating system are referred to as processes.
- During the lifetime of a process, it uses many system resources like CPU and memory.
- The OS keeps track of the processes and of the system resources so that it can manage all the processes in the system fairly.
  - There can be different scheduling strategies.
- We will look at process management for both Linux and Windows.

Monitoring Processes in Linux

- For system administrators it is crucial to be able to monitor
  - Which processes are running in the system
  - The current state of the processes
  - Resources these processes are taking
  - Which user started which process
- A number of tools are available for terminal to monitor the processes like:
  - `"ps"` offers a snapshot of processes
  - `"pstree"` offers a tree view of process, branching from parent process to child process
  - `"top"` offers a dynamic real time view of processes

Linux: the "ps" Command

- By default it shows only processes/scripts running via terminal.
- Command line options can be used to display other processes as well.
  - "ps x" – display all processes of current user
  - "ps –e" – display all processes currently running
  - "ps aux" – user-oriented variant, all processes
  - "ps –u username" to display process of a user named username
- Sample output:

```
tiayyba@ubuntu:~$ ps
   PID TTY          TIME CMD
  2506 pts/17   00:00:00 bash
  3579 pts/17   00:00:00 ps
tiayyba@ubuntu:~$
```

Linux: the "ps" Command cont'd

- Columns description for ps command
  - PID: Process ID
  - TTY: The terminal that controls the process. In this case it is pts (pseudo terminal slave)
  - Time: the number of hours, minutes and seconds the process has been running
  - CMD: the command line, the process was called with

```
tiayyba@ubuntu:~$ ps
   PID TTY          TIME CMD
  2506 pts/17   00:00:00 bash
  3579 pts/17   00:00:00 ps
tiayyba@ubuntu:~$
```

- Other options provide other information/columns

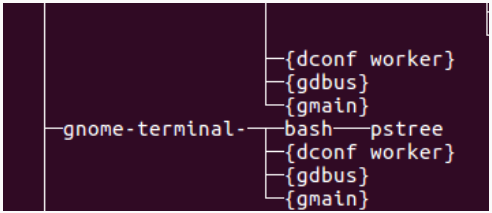
Process States in Linux

- Processes in Linux can exist in four different states
  - (R)unning: currently using the CPU
  - (S)leeping: waiting in queue to use the CPU
  - s(T)opped: stopped (but not terminated), either by user or other process
  - (Z)ombie: terminated but is waiting for its parent process to retrieve its exit code

```
tiayyba@ubuntu:~$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    2506    2395  0  80   0 -  5627 wait   pts/17   00:00:00 bash
0 S  1000    4198    2506  4  80   0 - 143975 poll_s pts/17   00:00:00 gedit
0 R  1000    4205    2506  0  80   0 -  7229 -      pts/17   00:00:00 ps
```

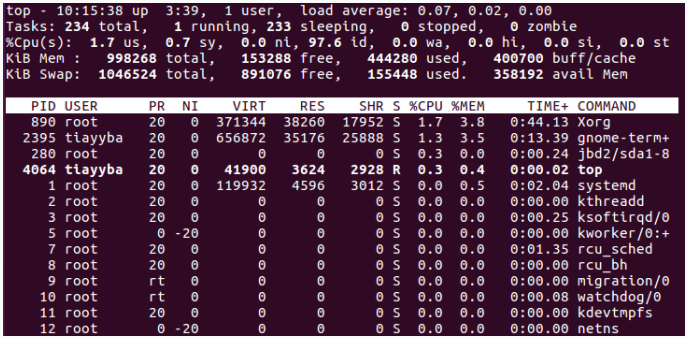
Linux: the "pstree" Command

- Processes are always instantiated by other processes
- Your system starts with the "systemd" process
- Parent processes start child processes
- "pstree" shows a tree view of all current processes
- In the image below, the terminal emulator instantiated "bash" that instantiated "pstree"



Linux: the "top" Command

- Provides a dynamic view of what's going on
- Shows process listed according to CPU usage
- Shows memory usage and status



Process Control

Process Control in Linux

- There are two types of processes in Linux:
  - Foreground: interactive, initialized by a user and controlled through terminal session.
  - Background: non interactive, not connected to a terminal, don't expect user input.
- User initiated processes run in foreground by default.
- Foreground processes block the command line until the process is finished.
- You can start a process in the background by appending `"&"` at the end of the command line.
  - Example: `"./script &"`
- System related process usually run in the background and are called daemons.

Starting a Process

- Once you run a command or program, it will start a process in the system.
  e.g.
  - `find -name "*.sh"`
  - `./sum.sh`
- It will be connected to the terminal and a user can send input to it.
- To start a process in the background (non-interactive), use the `"&"` symbol.
  e.g.
  - `firefox &`
- You can send a process to the background (while also stopping it) by pressing Ctrl+Z and then the `bg` command

Terminating (Signalling) a Process

- In order to terminate a process we can use the `"kill"` command.
- Syntax: `"kill PID"` (PID or %job)
- The kill command kills a single process at a time with the specified process id or job number.
- While the kill command is used to “kill” processes, its real purpose is to send signals to processes.
- Signals are intended to tell the process to (among other things) go away by gracefully terminating
- Many different signals are available
  - Ctrl+Z sends the TSTP (terminal stop) signal

Jobs – Foreground, Stopped, Background

- The shell starts processes and (with `kill`) signals processes
- And allows you to stop/re-start and foreground/background processes (jobs)
- Ctrl+Z stops the currently active foreground job and returns you to the shell prompt
  - Easy way to pause, look something up, and resume your task
- The `jobs` command shows stopped and background processes
- `bg` moves a job to the background, `fg` moves to foreground
  - Add a job number to affect a particular job e.g. `bg %2`
- Consider: edit, save, stop editor, compile, run, `fg` back into editor, and repeat

Some Available Signals

NAME	NUMBER	DESCRIPTION
SIGINT	**2**	Interrupt signal. Usually terminates the process. Note that this is equivalent of pressing **Crtl+C**
SIGKILL	**9**	Kill signal. Note that this signal, contrary to most other signals, cannot be caught or ignored. It will terminate the processes instantly, without waiting it to perform its normal termination procedures, such as cleaning up memory.
SIGTERM	**15**	Terminates a process, but first waits for it to perform its common termination procedures. This is the default signal sent by the **kill** and **killall** commands.
SIGCONT	**18**	Continue signal. This signal is used to restart a process that was previously stopped.
SIGSTOP	**19**	Stop signal. This signal is used to stop (pause) a process. This is equivalent of pressing **Crtl+Z**. Stopped processes can be later re-started using the **SIGCONT** signal.

Sending Signals to Processes

- You can send signals to process using the kill command:
- `kill -SIGNAL PID`
- `"kill -15 PID"` or `"kill -TERM PID"` sends a terminating signal to process PID
- `"kill -9 PID"` sends a KILL signal to terminate the process instantly
- If no option is specified kill command send a TERM signal

Summary

- Process management is an important component of every operating system.
- As users, we should monitor the processes for better system performance.
- Next class: Windows Process Management