Processes

Lesson on Linux Process Management

In Linux, **process management** is crucial for understanding how to monitor, control, and optimize running applications. A process in Linux is simply an instance of a running program, and each process is assigned a unique Process ID (PID). This lesson will cover essential commands and concepts for managing processes in Linux.

---

1. Process Basics

A **process** is an instance of a program in execution. It contains the program code and its current activity. Each process is assigned a **PID** (Process ID) and is handled by the Linux kernel. The life cycle of a process includes creation, execution, and termination.

2. Key Commands for Process Management

1. `ps`: View Running Processes

The `ps` command shows a snapshot of the currently running processes.

```
ps
```

This gives a simple list of processes running in the current shell session.

```
ps aux
```

- `a`: Displays processes of all users.
- `u`: Shows user-oriented format.
- `x`: Includes processes without a terminal.

Example:

```
ps aux | grep "apache2"
```

This will show all Apache processes.

2. `top`: Real-time Process Monitoring

The `top` command provides a real-time view of running processes and system resource usage (CPU, memory).

```
top
```

- **Key metrics**: CPU usage, memory usage, process PID, user, time, and command.
- Press `q` to quit.
- Press `k` and enter the PID to kill a process from within `top`.

For a more enhanced interface:

```
htop
```

(htop may need to be installed: `sudo apt install htop`)

3. `kill`: Terminate a Process

To terminate (kill) a process, use the `kill` command with its PID.

```
kill PID
```

Example:

```
kill 1234
```

This sends a **SIGTERM** (soft termination) signal to process `1234`.

To forcefully terminate (kill) a process:

```
kill -9 PID
```

- `-9` sends a **SIGKILL** signal, which forcefully kills the process.

4. `pkill`: Kill Processes by Name

The `pkill` command kills processes by their name.

```
pkill process_name
```

Example:

```
pkill firefox
```

This will kill all processes with the name "firefox".

5. `pgrep`: Search for Processes by Name

The `pgrep` command returns the PID(s) of processes matching a given name.

```
pgrep process_name
```

Example:

```
pgrep apache2
```

This will show all PIDs of Apache processes.

6. `nice` and `renice`: Adjust Process Priority

Each process in Linux has a **priority** that determines how much CPU time it gets. The priority is controlled by a "niceness" value (-20 to 19, where -20 is the highest priority).

- **nice**: Starts a process with a given priority.

```
nice -n 10 command
```

This starts the command with a niceness value of 10 (lower priority).

- **renice**: Change the priority of an already running process.

```
renice -n 5 -p PID
```

This changes the niceness of process `PID` to 5.

7. `nohup`: Run a Process in the Background

To run a process in the background, immune to hangups (disconnections), use the `nohup` command.

```
nohup command &
```

Example:

```
nohup ./script.sh &
```

This runs `script.sh` in the background and logs its output to `nohup.out`.

8. `jobs`: List Background Processes

When running processes in the background using `&`, the `jobs` command lists all background jobs for the current terminal session.

```
jobs
```

9. `bg` and `fg`: Manage Background and Foreground Processes

- **bg**: Resumes a background job.

  ```
  bg %job_number
  ```

- **fg**: Brings a background job to the foreground.

  ```
  fg %job_number
  ```

Example:

```
bg %1   # Resumes job 1 in the background
fg %1   # Brings job 1 to the foreground
```

10. `at` and `cron`: Schedule Processes

- **at**: Schedules a one-time task.

```
at 2pm
at> command
```

- **cron**: Schedules recurring tasks. Cron jobs are defined in a file called **crontab**.

```
crontab -e
```

Crontab syntax:

```
* * * * * /path/to/command
# m h dom mon dow command
```

Example:

```
0 2 * * * /usr/bin/backup.sh   # Run every day at 2 AM
```

11. `strace`: Trace System Calls of a Process

`strace` is used to trace the system calls made by a running process. It's useful for debugging.

```
strace -p PID
```

Example:

```
strace -p 1234
```

This will display the system calls made by process `1234`.

---

3. Process States in Linux

A process in Linux can be in one of several states:

1. **Running (R)**: The process is currently executing on the CPU.
2. **Sleeping (S)**: The process is waiting for an event (e.g., waiting for I/O).
3. **Stopped (T)**: The process has been stopped, usually by a signal or by the user.
4. **Zombie (Z)**: The process has completed execution but is waiting for its parent to retrieve its exit status.

To find zombie processes:

```
ps aux | grep 'Z'
```

---

4. Signals in Linux

Signals are used by Linux to communicate with processes. Some common signals include:

1. **SIGTERM** (15): Gracefully stop a process.
2. **SIGKILL** (9): Forcefully stop a process.
3. **SIGHUP** (1): Hangup signal, often used to reload configurations.
4. **SIGINT** (2): Interrupt a process (like Ctrl+C).
5. **SIGSTOP** (19): Stop a process (pause it).
6. **SIGCONT** (18): Continue a stopped process.

To send a signal:

```
kill -SIGNAL PID
```

Example:

```
kill -SIGTERM 1234   # Gracefully terminate process 1234
kill -SIGKILL 1234   # Forcefully kill process 1234
```

---

5. Monitoring Resource Usage

1. `vmstat`: Monitor System Performance

The `vmstat` command provides information about processes, memory, paging, block IO, traps, and CPU usage.

```
vmstat 1
```

This displays system resource usage every second.

2. `iostat`: Monitor CPU and Disk I/O

The `iostat` command reports CPU and I/O statistics.

```
iostat
```

3. `free`: Display Memory Usage

The `free` command displays the system's memory usage.

```
free -h
```

- `-h`: Shows memory in human-readable format (MB, GB).

4. `lsof`: List Open Files

`lsof` lists open files and the processes that opened them.

```
lsof -p PID
```

This shows the files opened by the process with PID.

---

6. Practical Use Cases

1. **Checking for High CPU Usage Processes**

   ```
   ps aux --sort=-%cpu | head
   ```

   This will display the top processes by CPU usage.

2. **Finding and Killing a Hung Process**

   ```
   pgrep "process_name"
   kill -9 PID
   ```

3. **Running a Script in the Background**

   ```
   nohup ./backup.sh &
   ```

   This runs `backup.sh` in the background and continues running after logging out.

4. **Scheduling a Task Using Cron**

   ```
   crontab -e
   ```

   Add a job to run every day at midnight:

   ```
   0 0 * * * /path/to/script.sh
   ```

5. **Resuming a Background Process**

   ```
   jobs
   bg %1   # Resumes the first background job
   ```

Conclusion

Process management is a fundamental skill in Linux that allows you to monitor, control, and optimize system processes. Mastering the key commands like `ps`, `top`, `kill`, `nice`, and others will help you maintain efficient system operations, handle issues like hung processes, and schedule tasks effectively.

Would you like to dive deeper into any of these topics or explore more specific commands?