DevOps Shack

Process Management in Linux

[Click Here To Enrol To Batch-6 | DevOps & Cloud DevOps]

Process management in Linux involves creating, scheduling, and terminating processes. Understanding how to manage processes is essential for system administration and ensuring the efficient operation of a Linux system.

Key Concepts in Process Management

Process

A process is an instance of a running program. Each process has a unique Process ID (PID).

Types of Processes

1. **Foreground Process:** Runs in the shell and takes input from the user. The shell waits for the process to complete before giving a prompt.
2. **Background Process:** Runs independently of the shell. The shell gives a prompt immediately after starting the process.

Process States

1. **Running:** The process is either currently executing or ready to execute.
2. **Waiting:** The process is waiting for some event to occur (e.g., I/O operation).
3. **Stopped:** The process has been stopped, usually by receiving a signal.
4. **Zombie:** The process has completed execution but still has an entry in the process table.

Commands for Process Management

Viewing Processes

1. **ps:** Displays information about active processes.

```
ps aux   # Detailed list of all running processes
```

2. **top:** Provides a dynamic, real-time view of running processes.

```
Top
```

3. **htop:** An enhanced version of top with a user-friendly interface (requires installation).

```
sudo apt install htop
htop
```

4. **pgrep:** Searches for processes by name.

```
pgrep -l bash   # List processes with the name "bash"
```

Controlling Processes

1. **&:** Run a process in the background.

```
command &
```

2. **fg:** Bring a background process to the foreground.

```
fg %1   # Bring the first background job to the foreground
```

3. **bg:** Resume a stopped job in the background.

```
bg %1   # Resume the first stopped job in the background
```

4. **kill:** Send a signal to a process, typically to terminate it.

```
kill PID   # Terminate the process with the specified PID
kill -9 PID   # Forcefully terminate the process
```

5. **pkill:** Send a signal to processes by name.

pkill firefox # Terminate all processes with the name "firefox"

6. **killall:** Send a signal to all processes matching a name.

killall firefox # Terminate all processes with the name "firefox"

7. **nice:** Start a process with a specified priority.

nice -n 10 command # Start a process with a nice value of 10

8. **renice:** Change the priority of an existing process.

renice -n 10 -p PID # Change the nice value of a process

Detailed Examples

Example 1: Viewing Processes

1. **ps Command:**

ps aux

Output:

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.1 166400 1044 ? Ss 10:00 0:01 /sbin/init

john 123 0.1 0.2 158784 2048 ? S 10:05 0:00 /usr/bin/bash

2. **top Command:**

top

Output:

top - 10:15:33 up 1 day, 1:15, 2 users, load average: 0.01, 0.05, 0.07

Tasks: 125 total, 1 running, 124 sleeping, 0 stopped, 0 zombie

%Cpu(s): 1.0 us, 0.5 sy, 0.0 ni, 98.5 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 2048000 total, 102400 free, 102400 used, 1843200 buff/cache

KiB Swap: 4096000 total, 4096000 free, 0 used. 1945600 avail Mem

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0  166400   1044    840 S   0.0  0.1   0:01.10 init
  123 john      20   0  158784   2048   1024 S   0.1  0.2   0:00.50 bash
```

Example 2: Controlling Processes

1. **Running a Process in the Background:**

```
sleep 100 &
```

Output:

```
[1] 4567
```

2. **Bringing a Background Process to the Foreground:**

```
fg %1
```

Output:

```
sleep 100
```

3. **Killing a Process:**

```
kill 4567   # Terminate the process with PID 4567
```

4. **Changing Process Priority:**

```
nice -n 10 sleep 100 &
renice -n 5 -p 4567   # Change the nice value of process 4567 to 5
```

Advanced Process Management

Process Scheduling

Linux uses a time-sharing scheduling algorithm to manage process execution. The scheduler assigns CPU time slices to processes based on their priority.

1. **Nice Value:** The nice value ranges from -20 (highest priority) to 19 (lowest priority). Lower nice values indicate higher priority.

```
nice -n -5 command   # Start a command with high priority
```

2. **Real-time Scheduling:** Real-time scheduling policies like SCHED_FIFO and SCHED_RR are used for processes requiring real-time execution.

```
chrt -f 10 command   # Start a command with real-time FIFO scheduling
```

Monitoring and Debugging Processes

1. **strace:** Traces system calls and signals.

```
strace -p PID   # Trace the system calls of a running process
```

2. **lsof:** Lists open files and the processes that opened them.

```
lsof -p PID   # List open files for a process
```

3. **pstack:** Prints a stack trace of a running process.

```
sudo pstack PID   # Print the stack trace of a process
```

4. **gdb:** The GNU Debugger can be used to debug running processes.

```
gdb -p PID   # Attach gdb to a running process
```

Example Scenario: Managing a Web Server Process

Starting the Web Server

1. **Run the web server in the background:**

sudo apache2ctl start &

2. **Check the status of the web server:**

ps aux | grep apache2

Monitoring Resource Usage

1. **Use top or htop to monitor resource usage:**

top -p $(pgrep -d',' apache2)

Adjusting Priority

1. **Increase the priority of the web server:**

sudo renice -n -10 -p $(pgrep apache2)

Debugging Issues

1. **Use strace to trace system calls:**

sudo strace -p $(pgrep -o apache2)

2. **Use lsof to check open files:**

sudo lsof -p $(pgrep -o apache2)

Summary

Process management in Linux involves a range of commands and tools to control and monitor processes. Understanding these tools is crucial for system administration, as they allow for efficient process control, resource management, and troubleshooting. Mastering process management ensures that a Linux system runs smoothly and efficiently, with optimal resource utilization and minimal downtime.