Lab 6 – Linux Process Management

Introduction to Key Concepts and Commands

- Processes: A process is an instance of a running program. In simpler terms, when you run a program (e.g., a web browser, a text editor, or even a command), the operating system creates a process to execute that program. For example, when you run a Python script, it becomes a process.

- CPU and Memory Usage: Processes consume CPU and memory. Use top to see a Python script using % CPU.

- Process Priority (nice/renice): Determines the order in which processes get CPU time. nice starts a process with a set priority. renice changes the priority of an existing process. E.g., renice 10 -p [PID].

  - **PR** (priority) is a dynamic value calculated by the system, influenced by the **nice value** (NI), CPU usage, etc.
  - A **lower PR** value indicates **higher priority**.
  - A **higher NI** (e.g., 10 or 19) means **lower priority**.

- Signals (kill command): is a mechanism used by the operating system to communicate with processes, allowing the kernel or other processes to send notifications to a running process. A signal can be used to indicate that a specific event has occurred, to notify a process of an error, or to request the process to perform some action (like termination or pause).
  - E.g. kill -9 [PID] : immediately stops a process.

- Process Monitoring (ps, top, htop): ps lists processes, top shows real-time process activity, and htop provides an advanced interface.
  - `ps` when you need a quick overview of processes in the current session or terminal.
  - `ps aux` when you need a more detailed and complete view of **all processes** running on the system, including system processes and processes from other users. E.g. ps aux | grep python : shows Python processes.
  - top provides a real-time, dynamic view of the system's running processes, including CPU usage, memory usage, and other system metrics.
  - `htop` is an interactive, user-friendly version of the top command. While `top` provides a text-based summary of system information, `htop` provides a more visually appealing and easier-to-read format with additional features.

- Unix allows you to send a signal to any process

`-1 = hangup`	**`kill -HUP 1234`**
`-2 = interrupt with ^C`	**`kill -2 1235`**
`no argument = terminate`	**`kill 1235`**
`-9 = kill`	**`kill -9 1236`**

- list your processes with `ps -u userid`

```
% kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46)
SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-
14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-
10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- ^C is 2 - SIGINT

Scenario: Web Server Process Management

You are managing a web server hosting several Python-based web applications. Each application runs as a separate process. Your tasks involve monitoring these processes, ensuring they don't consume excessive resources, and managing their priorities to ensure smooth operation of the server.

Lab Tasks

Task 1: Running and Monitoring Processes

1. Setup:
   a. Write three Python scripts: cpu_intensive.py, memory_intensive.py, and long_running.py.
   b. Run each script: python3 [script_name].py &
   c. Explain what does the command above do?
2. Monitoring:
   a. Use *ps aux* to list the processes.
   b. Observe CPU and memory usage with top or htop.
   c. Explain each attribute of the process associated with *ps aux*

Task 2: Managing Process Priority

3. Prioritization:
   a. Identify the PID of *cpu_intensive.py*.
   b. Lower its priority using renice.
   Priority range: -20 (highest prio) : 19 (lowest prio)

```
nice -n 10 python3 cpu_intensive_1.py &
nice -n -1 python3 cpu_intensive_2.py &

renice -n -10 -p  PID of cpu_intensive_1.py &
renice -n 5 -p PID cpu_intensive_2.py &
```

Task 3: Process Termination

4. Termination:
   a. Use kill to terminate long_running.py with SIGTERM and SIGKILL.
   b. Explain the differences between SIGTERM and SIGKILL.

Task 4: Resource Management

5. Resource Observation:
   a. Determine which script is most resource-intensive using top or htop.

Source Code for Python Scripts

1. CPU Intensive Script (cpu_intensive.py):

```
import time

def cpu_intensive_task():
    while True:
        sum([i**2 for i in range(10000)])

if __name__ == "__main__":
    while True:
        cpu_intensive_task()
        time.sleep(1)
```

2. Memory Intensive Script (memory_intensive.py):

```
import time

def memory_intensive_task():
    large_list = [0] * 10**7
    time.sleep(30)

if __name__ == "__main__":
    while True:
        memory_intensive_task()
```

3. Long Running Script (long_running.py):

```
import time

if __name__ == "__main__":
    while True:
        print("Running...")
        time.sleep(5)
```