LESSON NOTES

Intro to Linux

System Management

1.4.2 Process Management

Lesson Overview:

Students will:

- Understand how to manage processes on a system

Guiding Question: How are processes managed on a system?

Suggested Grade Levels: 9 - 12

Technology Needed: None

CompTIA Linux+ XK0-005 Objective:

1.4 - Given a scenario, configure and use the appropriate processes and services

- Process management
  - Kill signals
    - SIGTERM
    - SIGKILL
    - SIGHUP
  - Listing processes and open files
    - top
    - ps
    - lsof
    - Htop
  - Setting priorities
    - nice
    - renice
  - Process states
    - Zombie
    - Sleeping
    - Running
    - Stopped
  - Job control
    - bg
    - fg
    - jobs
    - Ctrl+Z
    - Ctrl+C
    - Ctrl+D
  - pgrep
  - pkill
  - pidof

This content is based upon work supported by the US Department of Homeland Security's Cybersecurity & Infrastructure Security Agency under the Cybersecurity Education Training and Assistance Program (CETAP).

Process Management

Process management is a critical aspect of efficiently handling and controlling the execution of programs on a computer system. In the realm of Unix-like operating systems, understanding and mastering process management tools and techniques are essential for system administrators and users alike. This involves the ability to send signals to processes for graceful termination or immediate termination, monitoring and listing processes and their associated files, adjusting process priorities, recognizing different process states, and employing job control mechanisms to manage foreground and background tasks effectively. The tools mentioned, such as top, ps, lsof, htop, nice, renice, and the concepts of job control and process states, empower users to navigate and optimize the execution of programs, ensuring a smoothly running system. Additionally, utilities like pgrep, pkill, and pidof provide efficient ways to identify and manage processes based on their attributes. This foundational knowledge is invaluable for anyone working in a Unix-like environment, enabling them to maintain system stability and responsiveness.

Kill Signals

Kill signals play a crucial role in managing processes by signaling them to perform specific actions. The following is an expanded explanation of each signal.

SIGTERM is a graceful way to request a process to terminate. When a process receives SIGTERM, it's expected to perform cleanup activities before exiting. This allows the process to release resources, close files, and generally ensure a smooth termination. It's often the default signal sent by commands like kill and is a way to politely ask a process to shut down. Consider a server process that needs to save its state or perform other cleanup tasks before shutting down. Sending SIGTERM gives the process an opportunity to do this before exiting.

SIGKILL is a more forceful signal that immediately terminates a process without giving it a chance to perform any cleanup. It's a powerful signal and should be used with caution because it doesn't allow the process to release resources or clean up files—it simply terminates the process abruptly. When a process is unresponsive or refuses to terminate gracefully after receiving SIGTERM, SIGKILL can be used as a last resort to forcefully end the process.

SIGHUP is historically associated with the hangup event on a terminal. When the terminal is closed or a connection is lost, a SIGHUP signal is sent to processes associated with that terminal. In the context of process management, SIGHUP is often used to instruct a process to reload its configuration or restart. Imagine a daemon process running in the background that reads a configuration file. Instead of restarting the entire process, sending SIGHUP to the process prompts it to reload its configuration, allowing for dynamic updates without a full restart.

Listing Processes and Open Files

The tools for listing processes and open files provide valuable insight into the activities happening on a system. The following is a more detailed explanation of each.

`top` (intentionally lowercase) is a dynamic, real-time process viewer that provides a continuously updated display of system processes. It offers a comprehensive overview of CPU usage, memory usage, running processes, and other system statistics. The display is interactive, allowing users to sort and filter processes based on various criteria. Administrators often use top to monitor system performance, identify resource-intensive processes, and get a real-time snapshot of the system's health. Its dynamic nature makes it especially useful for detecting sudden changes in resource usage.

`ps` (Process Status) is a command-line utility that displays information about processes running on the system. It provides a static snapshot of the current processes, showing details such as process IDs (PIDs), CPU and memory usage, and the command that started each process. ps is commonly used for troubleshooting and obtaining a quick overview of the processes running on the system. Different options can be used to customize the output and focus on specific information, making it a versatile tool for process inspection.

`lsof` (List Open Files) is a powerful command that lists open files and the processes that have them open. It provides detailed information about files, sockets, and devices that processes are currently using. This includes network connections, files being read or written, and more. lsof is invaluable for troubleshooting scenarios where you need to identify which processes are accessing specific files or ports. It aids in understanding the relationships between processes and the files they interact with, making it a crucial tool for system administrators and developers.

`htop` is an interactive and user-friendly process viewer, similar to top but with enhanced features. It provides a colorful and visually appealing representation of system processes, allowing users to scroll, search, and navigate through the process list more conveniently. htop is preferred by many users for its improved usability over top. It allows users to interactively manage processes, send signals, and easily identify resource-hungry processes. The visual design and keyboard shortcuts make it a popular choice for monitoring and managing processes in real-time.

Setting Priorities

Setting priorities for processes is a way to influence their execution in relation to other processes on a system. The nice command and renice utility are tools that allow users to manage the priority of processes. The following is a more detailed explanation of each.

The **`nice`** command is used to launch a new process with a specified priority level. In Unix-like systems, processes are assigned a priority value that determines how much CPU time they receive. The priority values typically range from -20 to 19, where lower value indicates higher priority. The nice command allows a user to start a new process with a specific priority level. The syntax is nice **`-n <priority> <command>`**. Users can use nice to launch a process with adjusted priority, ensuring that it gets more, or less, CPU time compared to other processes. For example, to start a process with a lower priority, you could use nice -n 10 command.

The **`renice`** command is used to change the priority of an already running process. It allows users to dynamically adjust the priority of a process while it's running. Similar to nice, the priority values range from -20 to 19. The syntax is **`renice <priority> -p <PID>`** Administrators or users may need to adjust the priority of a running process based on the system's workload. For example, to decrease the priority of a process with PID 1234, you could use **`renice +5 -p 1234`** to give it a lower priority.

Process States

Process states represent the various stages a process goes through during its lifecycle, indicating its current status and activity.

A ***Zombie*** process is one that has completed its execution but still has an entry in the process table. This entry is retained until the parent process retrieves the exit status of the terminated child process. Zombies consume minimal system resources but indicate a flaw in process management if they persist for an extended period without being reaped by the parent. Zombie processes often occur when a parent process fails to use the wait system call to retrieve the exit status of its terminated child processes. Properly managing child processes and collecting their exit status helps prevent the accumulation of zombie processes.

A ***Sleeping*** process is in a state of dormancy, waiting for a specific event to occur. This event could be the completion of an I/O operation, a timer reaching its expiry, or the reception of a signal. While a process is sleeping, it is not actively consuming CPU resources, allowing the system to efficiently use available resources. Many processes spend a significant amount of time in the Sleeping state, especially those waiting for user input, file operations, or network communications. This state is essential for optimizing resource utilization and responsiveness.

A ***Running*** process is actively executing instructions on the CPU. This is the state where a process is using system resources to perform its designated tasks. In a multitasking environment, multiple processes may be in the Running state, with the operating system rapidly switching between them to give the illusion of simultaneous execution. The Running state is the active phase of a process, where it is performing computations, responding to user input, or executing other instructions. Processes transition in and out of the Running state as the operating system schedules their execution.

A ***Stopped*** process is one that has been halted or suspended, usually by the receipt of a signal. This could be a user-initiated action or the result of a system event. Stopped processes do not consume CPU resources, and they can be resumed later by sending an appropriate signal. The Stopped state is commonly encountered when a user presses Ctrl+Z to suspend a foreground process. The process can be resumed with the fg command to bring it back to the foreground or bg to continue its execution in the background.

Job Control

Job control refers to the management of processes in the foreground and background, allowing users to interact with and control the execution of tasks. Here's a detailed explanation of each aspect of job control:

The **`bg`** (background) command is used to put a currently suspended or stopped process in the background, allowing it to continue its execution independently of the terminal. This is particularly useful when a process needs to run in the background without blocking the terminal for other tasks. Suppose you have suspended a process using Ctrl+Z and want it to continue running in the background while you perform other tasks in the terminal. You can use bg followed by the job ID or % sign and job number to achieve this.

The **fg** (foreground) command brings a background process to the foreground, making it the active task in the terminal. This is useful when you want to interact with a background process or monitor its output directly. If a process is running in the background, you can use fg followed by the job ID or % sign and job number to bring it back to the foreground, allowing you to interact with it.

The **jobs** (list jobs) command lists the jobs that are currently running or stopped in the background. It provides information about the job ID, status, and the command associated with each job. When multiple processes are running in the background, you can use jobs to get an overview of their status. This is helpful for identifying the job IDs needed for bg or fg commands.

Pressing *Ctrl+Z* suspends the currently running foreground process, putting it in a stopped state. The process is moved to the background, and you can use bg or fg to manage it further. If you want to temporarily stop a foreground process and move it to the background, you can use Ctrl+Z to suspend it.

Pressing *Ctrl+C* sends an interrupt signal to the currently running process in the foreground, typically causing it to terminate. It's a way to forcefully stop a process. When a process is unresponsive or needs to be terminated abruptly, Ctrl+C is often used to send an interrupt signal and stop the process.

Ctrl+D is used to send an end-of-file (EOF) signal. In the context of a terminal, it can be used to indicate the end of input or to logout from a session. In some contexts, such as when entering input in a terminal, Ctrl+D can be used to signal the end of the input stream.

Pgrep, Pkill, and Pidof

The commands **pgrep**, **pkill**, and **pidof** are essential for identifying and managing processes based on various criteria.

pgrep (process grep) is a command-line utility that searches for processes based on their name or other attributes and prints the process IDs (PIDs) of matching processes. It provides a simple and convenient way to find processes without the need for complex commands. If you want to find the PID of a process by specifying its name, you can use pgrep followed by the process name. For example, **pgrep firefox** would return the PID of the Firefox process.

pkill (process kill) is a command that sends a signal to processes based on their name. Instead of manually finding and specifying PIDs, pkill simplifies the process by allowing users to target processes using their names. By default, pkill sends the SIGTERM signal. To gracefully terminate a process by name, you can use pkill followed by the process name. For example, **pkill firefox** would send the SIGTERM signal to terminate all processes with the name "firefox."

pidof is a command that obtains the process ID (PID) of a running program based on its name. It returns the PID of the first process found with the specified name. If you need to quickly obtain the PID of a running process, you can use pidof followed by the process name. For instance, **pidof firefox** would return the PID of the running Firefox process.