Linux Basics
Lesson 3

Agenda

Understanding Processes: The Lifeblood of a Linux System

The Lifeblood of a Linux System

Processes are the running instances of a program (an **executable**) in a Linux system

They are **the lifeblood of the system**, as they perform all the essential tasks required to keep the system running

Importance of Processes in Linux

- Processes allow the system to perform multiple tasks at the same time
- They allow programs to execute concurrently, improving system performance
- Processes help isolate applications, preventing them from interfering with each other

Process states & attributes

Process States

- Processes in Linux can be in one of five states: r**unning, sleeping, stopped, zombie, and traced**
- The state of a process determines its behavior and how it interacts with the system

Process Attributes

- Each process in Linux has a unique **process ID** (PID)
- Other important attributes of a process include i**ts parent process ID (PPID), memory usage, CPU usage, and priority**

Process Monitoring Tools

The "**ps**" command is used to view processes running in a Linux system. It shows a static snapshot of the processes at the time the command was executed. By default, it displays the processes that are owned by the current user in a tabular format, but it can be used with various options to display more detailed or specific information.

"**ps -aux**" displays all processes running in the system, along with their attributes.

The "**top**" command is a real-time process monitoring tool that displays information about the processes currently running on a Linux system. It displays the processes in a dynamic, interactive format that updates periodically, allowing the user to monitor the system in real-time and sort the processes based on various criteria

Other process monitoring tools include "**htop**" and "**glances**", which provide more advanced features for monitoring processes in a Linux system.

The Managing Processes in Linux

Lifecycle of a Process

- A process begins when a user or system starts a program
- The operating system creates a new process and allocates system resources to it
- The process then executes the program code, performs its tasks, and waits for further instructions
- Once the process completes its tasks, the operating system terminates it

Linux manages processes by allocating system resources to them. The operating system provides a unique process ID (PID) to each process, which allows the system to manage it effectively

Processes are managed using commands like "ps", "kill", and "nice". The "**kill**" command is used to terminate a process. The "**nice**" command is used to set the priority of a process, allowing the user to control how much CPU time a process is allocated

Diving into Types of Processes

Foreground & Background processes

Foreground processes are those that are currently executing in the shell and require user input.

Background processes are those that run independently of the shell and do not require user input.

Use cases:

- Foreground processes: executing commands that require user input, such as text editors or interactive programs.
- Background processes: executing long-running tasks that do not require user input, such as system backups or file transfers.

Command line tools: fg, bg, jobs, ctrl-z

Parent & Child processes

A child process is a process created by another process, known as the parent process. Child processes inherit certain characteristics from their parent process, such as environment variables and file descriptors.

Parent processes can spawn multiple child processes.

Use cases:

- Parent processes: managing multiple instances of a program, running scripts that launch other programs or processes.
- Child processes: executing a specific task or function, such as a child process created by a web server to handle incoming requests.

Command line tools: ps, top, kill

Daemon processes

Daemon processes are long-running background processes that perform system tasks or provide services.

They do not have a controlling terminal and run as system users.

Use cases:

- Providing network services such as web servers, email servers or DNS servers.
- Performing system tasks such as log rotation, backups or updates.

Command line tools: systemctl, service

Working with Processes: Essential Tools and Commands

ps

Purpose: shows information about running processes.

Basic usage: ps [options]

Options:

- -e: display information about all processes.
- -f: show detailed information about processes.

Tips:

- Use "ps aux" to display a full list of processes with detailed information.
- Use "ps -C <process name>" to display information about a specific process.

pstree

Purpose: pstree is a command-line utility used to display a tree-like structure of processes on the system.

Basic usage: pstree [OPTIONS]

Options:

- -p: Shows process PIDs.
- -u: Shows user names.
- -h: Highlights the current process and its ancestors.
- -s: Displays command-line arguments.

pgrep

Purpose: pgrep (process grep) is a command-line utility that is used to find processes based on their name, PID, or other attributes. Basic usage: pgrep [OPTIONS] PATTERN

Options:

- -a: Displays the entire command line for each process.
- -l: Displays the process name and the entire command line.
- -u: Searches for processes owned by a specific user.
- -f: Searches for processes based on the full command line.

Tips:

- pgrep can be combined with other commands, such as kill, to automate the process management process.
- By default, pgrep only returns the PID of matching processes. To display more information about the processes, you can use the -a or -l options.

top

Purpose: provides real-time monitoring of processes and system resources.

Basic usage: top

Options:

- Press "Shift+P" to sort processes by CPU usage.
- Press "Shift+M" to sort processes by memory usage.

Tips:

- Use "top -d <seconds>" to set the update interval of top.
- Use "top -u <username>" to only display processes of a specific user.

htop

Purpose: similar to top, but provides a more user-friendly interface.

Basic usage: htop

Options:

- Press "F6" to sort processes by different criteria.
- Press "F9" to send a signal to a selected process.

Tips:

- Use "htop -u <username>" to only display processes of a specific user.
- Use "htop -p <process ID>" to monitor a specific process.

kill

Purpose: sends a signal to a process to terminate it.

Basic usage: kill [options] <PID>

Options:

- -9: force a process to terminate immediately.

Tips:

- Use "killall <process name>" to terminate all processes with a specific name.

nice & renice commands

Purpose: adjust the priority of a process to control its CPU usage.

Basic usage: nice [options] <command>

Options:

- -n: set the priority level of a process.

Tips:

- A lower priority value means a higher priority for the process.
- Use "renice <priority> -p <PID>" to adjust the priority of a running process.

lsof

Purpose: lsof (list open files) is a command-line utility used to display information about files and processes that are currently open. Basic usage: lsof [OPTIONS]

Options:

- -p: Shows open files for a specific process.
- -u: Shows open files for a specific user.
- -i: Shows open files for a specific network socket.
- -c: Shows open files for a specific command name.

Tips:

- Use lsof to diagnose file access problems and to see which files are currently in use.
- The output of lsof can be quite verbose, so it is often helpful to pipe it through other commands like grep or awk to filter the output.

Interpreting Command and Script Completion Codes

Understanding Exit Statuses in Linux

In Linux, every command that is executed returns an exit status or completion code.

This exit status indicates the success or failure of the command execution.

Understanding exit statuses is important for troubleshooting and error handling in Linux.

Exit Status Convention

- In Linux, exit statuses are represented by numeric values.
- An exit status of 0 indicates success, while any non-zero value indicates a failure or error.
- Different non-zero values can indicate different types of errors, depending on the command.

Understanding Exit Statuses in Linux

Retrieving Exit Status

- To retrieve the exit status of the last executed command, use the special shell variable $?
- The value of $? will be the exit status of the last executed command.

Exit Statuses in Shell Scripting

In shell scripting, exit statuses are commonly used for error handling and control flow.

For example, you can use the exit status of a command to determine whether to continue or exit the script.

You can also set the exit status of a script using the "exit" command, which can be useful for signaling errors to other scripts or processes.

Examples of Commands With Exit Statuses

- "ls" command: exit status of 0 indicates success, non-zero indicates failure (such as permission denied or file not found)
- "grep" command: exit status of 0 indicates success (match found), non-zero indicates failure (no match found)
- "ping" command: exit status of 0 indicates success (target is reachable), non-zero indicates failure (target is unreachable)

Tips for Effective Use of Exit Statuses

- Always check the exit status of commands in your scripts to ensure proper error handling and control flow.
- Use descriptive non-zero exit status values to make it easier to identify the cause of errors.
- Combine commands with "&&" or "||" to execute them conditionally based on the exit status of the previous command.

Getting to Know Interrupt Signals

Signals in Linux

Signals are a way for the Linux kernel to communicate with processes and to notify them of specific events or actions. Signals are essential for managing and controlling processes in Linux.

Types of signals

There are many different types of signals, but some of the most commonly used signals are interrupt signals, which allow processes to be terminated or paused.

Handling signals

- Processes can handle signals in various ways, depending on how they are programmed.
- Processes can ignore signals, handle them in a specific way, or even generate their own signals.
- Signal handlers are used to specify how a process should handle a particular signal.

Interrupt signals

Interrupt signals

Interrupt signals are signals that are sent to a process in order to interrupt or terminate it. Two common interrupt signals are SIGINT (signal 2) and SIGTERM (signal 15).

SIGINT is typically sent by the user when they press Ctrl+C, while SIGTERM is usually sent by the system to request that a process terminate.

Sending signals with the kill command

The kill command is used to send signals to processes.

- The syntax for the kill command is "kill [signal] [process ID]".
- For example, "kill -SIGINT 1234" would send a SIGINT signal to the process with ID 1234.

Introduction to Serial Devices and Terminals

Serial devices & Terminals

Serial devices

Serial devices are hardware devices that transmit data in a serial format, one bit at a time.

Data transmission occurs through a serial port using a specific protocol and baud rate.

Terminals in Linux

A terminal is an interface used to interact with a computer system.

In Linux, terminals can be physical or virtual.

Terminals in Linux

Physical Terminals

- A physical terminal is a device that provides direct access to a computer system, such as a monitor and keyboard.
- They are commonly used in server rooms or other locations where direct access is necessary.

Virtual Terminals

- A virtual terminal is a software-based terminal that emulates a physical terminal.
- They can be accessed through a graphical interface or by using keyboard shortcuts.

Terminal Emulators

- A terminal emulator is a program that allows users to access a virtual terminal within a graphical user interface.
- Examples of terminal emulators in Linux include GNOME Terminal, Konsole, and xterm.

Communicating with Named Pipe Channels

Introduction to Named Pipes in Linux

Named pipes, also known as FIFOs, are a type of inter-process communication (IPC) mechanism used in Linux. They provide a way for two or more processes to communicate with each other by sharing data through a pipe.

Unlike unnamed pipes, named pipes have a name associated with them and can be accessed by multiple processes at the same time.

How Named Pipes Work

- Named pipes are similar to regular files, but they have no contents until data is written to them.
- When a process writes data to a named pipe, it is stored in a buffer until another process reads it.
- Multiple processes can read from the same named pipe, but the data is read in a first in, first out (FIFO) order.

Advantages of Named Pipes

Named pipes have several advantages over other IPC mechanisms, such as sockets and message queues:

- They are simple to create and use, require no special permissions or privileges, and can be accessed by any process with the appropriate permissions.
- Named pipes are also very efficient, as they do not require copying data between processes or kernel space.

Creating and Using Named Pipes

To create a named pipe, use the mkfifo command followed by the name of the pipe.

- For example: mkfifo mypipe

To write data to a named pipe, use the echo command followed by the data and the name of the pipe.

- For example: echo "Hello, world!" > mypipe

To read data from a named pipe, use the cat command followed by the name of the pipe.

- For example: cat mypipe

Named pipes can also be used in shell scripts to pass data between processes.

Automating Tasks with the System Scheduler: Cron

Introduction to Cron

Cron is a time-based job scheduler in Unix-like operating systems. It allows users to schedule tasks to run automatically at specified intervals.

Structure of a Cron Job

- A cron job consists of a schedule and a command.
- The schedule defines when the command should be executed.
- The command is the task to be executed.

Scheduling Tasks with Cron

- Cron uses a special syntax to define the schedule for a job.
- The syntax consists of five fields: minute, hour, day of the month, month, and day of the week.
- Each field can contain a range of values, a list of values, or a wildcard (*) to match any value.

Introduction to Cron

Using Cron

- To create a new cron job, users can edit the crontab file using the crontab command.
- The crontab command allows users to add, remove, and modify cron jobs.
- Users can also view their current cron jobs using the crontab -l command.

Importance of Cron

- Cron is a powerful tool for automating system tasks and maintaining system health.
- It can be used to perform a wide range of tasks, from simple file backups to complex system maintenance routines.
- Properly configured cron jobs can help prevent system downtime and ensure that critical tasks are performed on time.

Keeping an Eye on Your System: System Monitoring Basics

Linux system monitoring

System monitoring is essential for maintaining a healthy system.

Various aspects of a system should be monitored regularly to ensure optimal performance.

Key aspects to monitor:

- System load: Monitor the CPU and memory usage of the system.
- Memory usage: Monitor the amount of memory used by the system and applications.
- Disk usage: Monitor the space used by the file system and storage devices.
- Network activity: Monitor network connections and traffic.

Why do we need monitoring?

- Detect potential issues before they become problems.
- Optimize system performance and resource utilization.
- Improve system availability and reliability.

Linux system monitoring tools

top: A command-line tool that displays system information and resource usage.

- Basic usage: $ top

htop: A more advanced version of top with additional features and a user-friendly interface.

- Basic usage: $ htop

vmstat: A command-line tool that reports virtual memory statistics.

- Basic usage: $ vmstat

iostat: A command-line tool that reports I/O statistics for storage devices.

- Basic usage: $ iostat

netstat: A command-line tool that displays network connections and statistics.

- Basic usage: $ netstat

Understanding System Logs: Messages, Syslog, and More"

Understanding System Logs with Syslog in Linux

System logs are crucial for understanding the behavior of a Linux system and troubleshooting issues that may arise.

Syslog is the default logging mechanism on Linux systems and is responsible for collecting and managing logs.

Syslog is a daemon that runs in the background and collects logs from various sources such as system services and applications.

Logs are stored in various files located in the **/var/log directory**.

Viewing and Interpreting Logs

The most common tool for viewing logs is the command-line tool "**tail**", which displays the last few lines of a file.

Other useful tools include "**grep**" to search for specific keywords, "**less**" to navigate through large log files, and "**journalctl**" to view logs collected by systemd.

A Deeper Dive into Specific Logs

Key Log Files

/var/log/messages: contains system-wide messages, including kernel messages, service messages, and other system messages.

/var/log/syslog: contains system messages generated by syslog.

/var/log/auth.log: contains logs related to authentication, such as user logins and system authentication events.

/var/log/dpkg.log: contains logs related to package management, such as installation and removal of packages.

/var/log/kern.log: contains kernel logs, including hardware errors and driver issues.

/var/log/boot.log: contains logs generated during the boot process, including hardware initialization and service startup.

auth.log

- Located at /var/log/auth.log
- Captures authentication-related events such as login attempts, su attempts, and sudo usage
- Contains the date, time, and hostname of the event along with the username and the action taken
- Useful for detecting unauthorized access attempts and understanding user activity on the system

dpkg.log

- Located at /var/log/dpkg.log
- Captures installation and removal events of Debian packages using the dpkg tool
- Contains the date, time, and hostname of the event along with the package name, version, and action taken
- Useful for tracking package changes, troubleshooting installation issues, and understanding system upgrades and changes

syslog

- Located at /var/log/syslog
- Captures a wide range of system events including kernel messages, daemon messages, and more
- Contains detailed information about the system activity and errors
- Useful for debugging system issues, analyzing system performance, and understanding system behavior

messages

- Located at /var/log/messages
- Captures a wide range of system events including kernel messages, daemon messages, and more
- Contains detailed information about the system activity and errors
- Useful for debugging system issues, analyzing system performance, and understanding system behavior.

kern.log

- Located at /var/log/kern.log
- Captures kernel-related messages and events, such as hardware errors, system crashes, and kernel module loading and unloading.
- Contains detailed information about the system's kernel-level activity and errors.
- Useful for debugging system issues, analyzing system performance, and understanding system behavior, especially related to hardware and kernel-level processes.

boot.log

- Located at /var/log/boot.log
- Contains information about the system boot process, including messages from the bootloader, kernel, and boot scripts
- Useful for diagnosing boot problems and understanding the boot sequence of the system.

Log management

- It is important to manage logs properly to avoid filling up disk space and to make it easier to locate and analyze logs
- Tools like **logrotate** can help manage logs by rotating and compressing log files on a regular basis
- Log management is crucial for maintaining system health, diagnosing issues, and improving system performance

Managing Logs Effectively with Log Rotate Scripts

Log rotation

Log rotation is the process of archiving and deleting log files on a regular basis to manage disk space and prevent logs from becoming too large.

In Linux, log rotation is typically managed using the logrotate utility

Importance of Log Rotation

- Log files can grow very large and take up a lot of disk space.
- Large log files can also make it difficult to find important information.
- Regular log rotation helps keep log files at a manageable size and makes it easier to find important information.
- Log rotation is also important for security and compliance purposes, as it ensures that logs are not lost or tampered with.

Logrotate utility

- logrotate is a command-line utility that is used to manage log files in Linux.
- It allows you to specify which log files to rotate, how often to rotate them, and where to store the archived logs.
- Logrotate can also compress archived logs to save disk space.

Basic Configuration and Usage of Logrotate

- The logrotate configuration file is located at /etc/logrotate.conf.
- To configure log rotation for a specific log file, create a new configuration file in the /etc/logrotate.d/ directory.
- In the configuration file, specify the log file to rotate, how often to rotate it, and any other options.
- To manually run logrotate, use the command "logrotate <config-file>".
- By default, logrotate is run daily via a cron job.

Introducing Auditd: Monitoring Events at a Granular Level

Introduction to Auditd

Auditd is a user-space component of the Linux Auditing System that monitors and records system activity.

It provides a granular level of control over the system's behavior and tracks user actions, system calls, and other events.

Auditd is a crucial component for system security, compliance, and debugging purposes.

Role of Auditd

- Auditd provides a comprehensive logging system that tracks all system activity.
- It monitors user activity, system calls, and file access, among others.
- Auditd logs the data into a binary file format, which can be easily searched and analyzed.

Importance of Auditd

- Auditd is essential for system security. It helps detect and respond to security threats by identifying unauthorized access attempts or suspicious activities.
- Auditd is useful for compliance purposes, as it can help organizations meet regulatory requirements and ensure that users follow company policies.
- Auditd is valuable for debugging purposes, as it provides detailed logs of system activity that can help identify system issues and resolve them quickly.

Resources to explore additionally:

“Linux Administration Handbook" by Evi Nemeth, Garth Snyder, Trent R. Hein, and Ben Whaley (in LMS system)

Q&A