Advanced Linux administration marathon program Day - 5

Topics :-

1. **Process states**
2. **Shell scripting**

🌀Process in Operating Systems

A process is a program in execution. It is an active entity, as opposed to a program, which is a passive entity. A process goes through different states and transitions, depending on system resources and process execution.

🌀Types of Processes

1. **User Processes:** These are processes initiated by the user or applications, such as running a text editor or a web browser.
    - **Example:** `gedit`, `firefox`
2. **Daemon Processes:** These are background processes typically started during system boot and do not have direct interaction with the user. They handle system tasks and services.

    - **Example:** `sshd`, `cron`
3. **Kernel Processes:** These processes manage low-level system operations, interacting closely with hardware. These are essential processes managed by the operating system kernel.
    - **Example:** `kworker`, `ksoftirqd`

🌀Process States

A process can be in various states depending on its execution stage. The most common states are:

1. **New**: The process is being created.
2. **Ready**: The process is ready to run but is waiting for CPU time.
3. **Running**: The process is currently being executed on the CPU.
4. **Waiting (Blocked):** The process is waiting for an event or resource (like I/O completion).
5. **Terminated (Zombie):** The process has finished execution but is waiting for the parent process to acknowledge its termination.
6. **Stopped:** The process has been paused by a signal or the user (e.g., by pressing Ctrl+Z).

🌀Process Cycle

A process goes through various states during its lifecycle:

1. **Creation:** A new process is created from an existing process (parent process).
2. **Execution:** The process is dispatched to the CPU for execution.
3. **Waiting:** If a process requires an unavailable resource, it moves to the waiting state.
4. **Completion:** Once the process finishes, it moves to the terminated state.
5. **Zombie (Termination waiting)**: A process may enter a zombie state after it finishes but before its parent collects the exit status.

🌀`ps` Command and Options

The `ps` command is used to display information about the running processes.

Common options:

- **`ps`:** Default command shows processes for the current shell session.
- **`ps aux`:** Shows all processes for all users.

- `ps -e`: Lists all processes.
- `ps -f`: Shows processes in full format.
- `ps -l`: Displays processes in long format with extra details.
- `ps -u <username>`: Displays processes for a specific user.
- `ps -p <pid>`: Displays information about a specific process using the process ID (PID).
- `ps -ef | grep <process>`: Filters processes by a keyword.

🌀Zombie Process

A **zombie process** is a process that has completed execution but still has an entry in the process table. This happens when the parent process hasn't read the exit status of the terminated child process.

How to Kill a Zombie Process:

- A zombie process cannot be killed directly because it’s already "dead."
- To remove a zombie, the parent process must be killed or handled correctly to clean up the zombie process. Use the `kill` command:

➡️**kill -9 <parent-pid>**

- **To identify zombies:**

  ➡️**ps aux | grep Z**

🌀`top` Command

The `top` command displays real-time system statistics about processes. It shows details such as CPU usage, memory usage, running processes, and system load.

Key information displayed by `top`:

- **PID:** Process ID.
- **USER:** The user running the process.
- **PR:** Process priority.
- **NI:** The nice value.
- **%CPU:** CPU usage.
- **%MEM:** Memory usage.
- **TIME+:** Total CPU time the process has used.
- **COMMAND:** Name of the command or process.

🌀Priority and Nice Value

1. **Priority (PR):** The priority determines the order in which processes are scheduled. Lower numerical priority means higher priority.
2. **Nice (NI):** This is the value assigned to a process to adjust its priority. The range is from `-20` (highest priority) to `19` (lowest priority). A process with a lower nice value has higher priority.

Changing Priority:

- To change the nice value of a process:

  ➡️**renice <new_nice_value> -p <pid>**

- Example:

  ➡️**renice 10 -p 1234** **# Increases nice value, lowering process priority.**

🌀Notes on Commands and Process Management

Useful ps command examples:

- **To list all running processes:**

  ➡️**ps -e**

- **To show processes with user and memory details:**

➡️**ps aux**

- **To find a specific process by name:**

  ➡️**ps aux | grep <process_name>**

Killing a Process:

- **You can kill a process by its PID:**

  ➡️**kill <pid>**

- **If the process doesn't respond, use a forceful kill:**

  ➡️**kill -9 <pid>**

`top` Command in Detail:

The `top` command can be customized while running:

- **Press `k`:** To kill a process.
- **Press `r`:** To renice (change the priority) of a process.
- **Press `M`:** To sort processes by memory usage.
- **Press `P`:** To sort processes by CPU usage.
- **Press `q`:** To quit `top`.

Shell scripting

🌀Shell Scripting: An Introduction

Shell scripting is a way to automate tasks in a UNIX/Linux environment using command-line interpreters called shells. A shell script is essentially a text file containing commands that the shell executes sequentially.

🌀What is a Shell?

A **shell** is a command-line interpreter that provides a user interface for the Unix/Linux operating system. It interprets the commands entered by the user or the commands in a shell script and converts them into a form the operating system kernel can understand.

🌀Types of Shells

1. **Bourne Shell (sh):**
    - The original Unix shell, often used for scripting because of its simplicity and efficiency.
    - Path: `/bin/sh`

2. **Bash (Bourne Again Shell):**
    - The most popular shell in Linux, widely used due to its ease of use and extended features.
    - Path: `/bin/bash`
3. **C Shell (csh):**
    - A shell with C-like syntax, used primarily for scripting that requires C-like constructs.
    - Path: `/bin/csh`
4. **Korn Shell (ksh):**
    - Combines features from both the Bourne and C shells.
    - Path: `/bin/ksh`
5. **Z Shell (zsh):**
    - An extended version of `bash`, with improvements like themes and plugin support.
    - Path: `/bin/zsh`

🌀 Why Use Shell Scripting?

- **Automation:** Shell scripts can automate repetitive tasks (e.g., backups, updates).
- **System Administration:** Tasks such as user management, process management, and file manipulation can be automated with scripts.

- **Custom Commands:** Users can write shell scripts to create their own custom commands and utilities.
- **Rapid Prototyping:** Shell scripting allows for quick testing and debugging of smaller tasks.

🌀How to Check the Type of Shell on Your System

You can check what type of shell you are currently using with the following commands:

- `echo $SHELL`: Displays the current user's default shell.
- `echo $0`: Shows the name of the shell being used.
- Example:

  ➡️**echo $SHELL**

  ➡️**/bin/bash**

🌀Common Shell Scripting Basics

1. **Shebang (`#!`):** A script typically starts with the shebang line that specifies the shell to interpret the script.

   ➡️**#!/bin/bash**

2. **Comments:** Any text after # is considered a comment and ignored by the shell.

   ➡️**# This is a comment**

3. **Commands:** You can write any valid shell command inside a script, just like typing them in the terminal.
4. **Execution Permission:** To run a shell script, it needs execution permission. This can be set using:

   ➡️**chmod +x script.sh**

5. **Running a Script:** Once the script has execution permissions, you can run it using:

   ➡️**./script.sh**

🌀Variables in Shell Scripting

In shell scripting, variables store data that can be used throughout the script. Variables in shell scripting do not have a data type (like in most programming languages); everything is treated as a string unless used in a different context.

1. **Declaring a Variable:**

   ➡️**MY_VAR="Hello, World!"**

2. **Accessing a Variable:**

   ➡️**echo $MY_VAR**

3. **Rules for Naming Variables:**
   a. Can contain letters (A-Z, a-z), numbers (0-9), and underscores (`_`).
   b. Cannot begin with a number.
   c. Variables are case-sensitive (`MY_VAR` and `my_var` are different).
4. **Environment Variables:**
   a. Shell variables that are available to any child processes started from the shell are called **environment variables**.
   b. Example: `$PATH`, `$HOME`, `$USER`

- **To view all environment variables:**

  ➡️**printenv**

- **To export a variable as an environment variable:**

  ➡️**export MY_VAR="Hello, World!"**

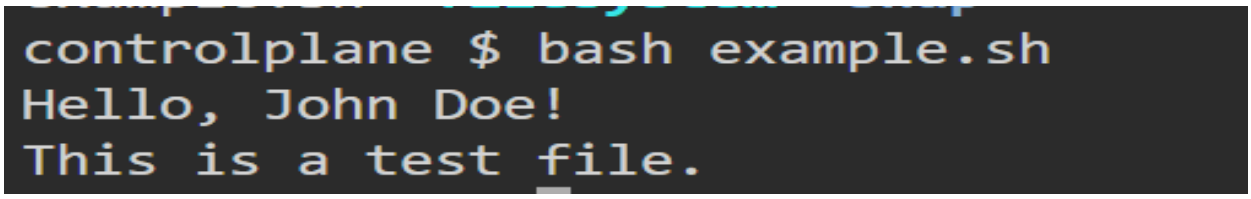
🌀Example of a Simple Shell Script

Here's a basic shell script that demonstrates the use of variables and simple commands:

➡️ Vim example.sh

```
#!/bin/bash

# Declare a variable
NAME="John Doe"

# Print a greeting
echo "Hello, $NAME!"

# Create a directory
mkdir my_directory

# Navigate into the directory
cd my_directory

# Create a file
touch my_file.txt

# Write content to the file
echo "This is a test file." > my_file.txt

# Display file contents
cat my_file.txt

# Exit script with success status
exit 0
```

Output of this script :-



🌀Conclusion

Processes are the backbone of modern operating systems. Managing processes effectively involves understanding different process types, process states, and using system tools like `ps`, `top`, and `kill`. Understanding the role of priority and nice values in scheduling also helps in optimizing system performance.

Shell scripting is an essential skill for anyone working in Linux or UNIX environments. It allows for automation of repetitive tasks, efficient system administration, and customization of command-line tools. By understanding shell types, variables, and basic scripting structures, you can start building powerful and flexible scripts to simplify your work.

Thank you..!

Author :- Sidhant bote

Mentor :- Pavan Wankhade sir