What is process in linux ?

A process is a running instance of a command. It is an executable file that is read from the disk, loaded into memory, and executed by the system.

What is accutally being executed ?

A C program
A shell script
A command executed in the terminal

- External commands
- Built-in shell commands

The Origin of All Processes in Linux

All processes in a Linux system originate from a single parent process. This process is known as systemd (in modern Linux distributions) or sysVinit (in older distributions). It is the first process that runs when the system boots up and is responsible for spawning all other processes.

- Systemd is assigned PID 1, and every subsequent process is created in an ordered manner.
- Even PID 1 has a parent process, which is the kernel itself, assigned PID 0.

Types of Processes ?

System processes
These processes are essential for the operating system and are initiated during system boot. The kernel is responsible for managing their execution.

User processes
These are processes started manually by users, remotely, or through scheduled tasks at specific times.

Services (Daemons)

In Linux, a service or daemon is a background process that runs independently of user interaction. These processes provide essential functionalities to the system or applications, ensuring continuous operation without requiring direct user input.

Managing Linux Processes

Managing processes in Linux is a fundamental aspect of system administration. Every running program or service in Linux is considered a process, whether it's a user application or a system daemon (background service). Proper process management ensures efficient resource utilization, system stability, and smooth multitasking.

Process Lifecycle in Linux

A process in Linux goes through multiple stages from creation to termination. Understanding the Process Lifecycle helps in managing and troubleshooting processes effectively. The main stages are:

1- Creation (Forking)

- A new process is created using the fork() system call, which duplicates an existing process (typically the parent process).
- The new process gets a unique Process ID (PID) and starts execution.

2- Ready (Runnable)

- The process is ready to run but waiting for the CPU to schedule it.
- The Linux scheduler decides which process gets CPU time based on priority and other factors.

3- Running (Executing)

- The process is actively running on the CPU, executing its instructions.
- If an interrupt occurs (like I/O request), it moves to the waiting state.

4- Waiting (Sleeping)

- The process is paused, waiting for an event (like user input, disk I/O, or a signal).
- This can be an interruptible sleep (wakes up on signals) or uninterruptible sleep (cannot be interrupted, usually due to kernel-level operations).

5- Terminated (Zombie or Exit)

- If a process finishes execution normally, it exits, freeing resources.
- If the parent doesn't acknowledge the exit, it becomes a zombie process (Z state), occupying space in the process table until the parent cleans it up.

6- Stopped (Suspended/Traced)

- A process can be paused manually (e.g., using CTRL+Z) or by a debugger (like gdb).
- It remains in memory but doesn't execute until resumed (fg or bg commands).

Commands/Topics related to managing linux processes

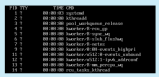
Listing processes commands

- ps — list all processes in this session terminal for this user
- ps -a — list processes from all users, providing a snapshot of active processes
- ps -e — list processes from all users and the whole system and "f" under tty means that this process running from the kernel itself
- ps -u — list processes with usernames , time the process started , memory and CPU usages , which TTY runs process , shows also VSE(virtual set size) the size of the image process , RSS (resident set size) the size of the program in memory

The VSE and RSS sizes may be different because VSE is the amount of memory allocated for the process, whereas RSS is the amount that is actually being used. RSS memory represents physical memory that cannot be swapped.

Note : the ps command and all its options gives a snapshot of one second , not a life show

- top — Gives a life display of running processes , the default is to display processes based on how much CPU time they are currently consuming. However, you can sort by other columns as well. All of this information is redisplayed every 5 seconds, by default.
  list includes actions that you can do with top to display information in different ways and modify running processes:
  - press h — to see help options
  - press M(capital) — to sort by memory usage instead of CPU, and then press T(capital) to return to sorting by CPU
  - press the number 1 — to toggle showing CPU usage of all your CPUs if you have more than one CPU in your system
  - press R (capital) — to reverse sort your output.
  - press u — and enter a username to display processes only for a particular user.
- htop — gives an advanced life display of running processes



In most times each system process or service executed through the kernel or even through the user itself but running in background as a system process , will usually ends with "d" letter and "d" for daemon

- pstree — Visualize tree of all system processes and all its child processes
- ps -l — give more details about the processes
- ps -f — shows the start time of each process (stime)
- ps ux — list all of the processes running on your Linux system for the current user
- ps aux — list all processes running for all users on your system
- ps -e --sort=pid — list all processes running with sort by a specific column
- ps -e --sort=-vsz — to see which processes are using the most memory

Killing processes commands

Kill Command and its uses

The kill command sends a specific signal to a process, identified by its Process ID (PID). By default, if no signal is specified, kill sends the SIGTERM signal, which requests the process to terminate gracefully.

Process ID

A process is identified on the system by what is referred to as a process ID (PID). That PID is unique for the current system. In other words, no other process can use that number as its process ID while that first process is still running. However, after a process has ended, another process can reuse that number.

Signals in Linux

Signals are software interrupts sent to a process to notify it of an event or request an action.

Types of Signals in Linux

- kill -SIGHUP (kill -1) — Hang up: restart the process with the same PID .
- kill -SIGINT (kill -2) — Interrupt: Sent when you press Ctrl+C to interrupt a process
- kill -SIGTERM (kill -15) — Terminate: Requests the process to terminate gracefully (default signal).
- kill -SIGKILL (kill -9) — Kill: Forces the process to terminate immediately (cannot be ignored). forcefull kill

When we say a process is terminated gracefully with the SIGTERM signal, it means the process is given an opportunity to clean up resources, save its state, and shut down in an orderly manner before it exits. This is in contrast to a forceful termination (e.g., with SIGKILL), where the process is immediately stopped without any chance to clean up.

what is meant by cleanup that process perform before termination ?
Closing open files. ✅
Releasing allocated memory. ✅
Saving data or state to disk. ✅
Terminating child processes. ✅
Cleaning up network connections. ✅

Imagine you have a text editor open with an unsaved document:

- If you send SIGTERM to the text editor, it can prompt you to save the document before exiting.
- If you send SIGKILL, the text editor will terminate immediately, and any unsaved changes will be lost.

Nicing and Renicing processes commands

Renicing : In Linux, renicing a process means changing its CPU priority level (or "nice value") in the system's scheduling algorithm. This determines how much CPU time the process gets relative to other processes.

Every process running on your system has a nice value between -20 and 19. By default, the nice value is set to 0.

facts about "nice values"

- The lower the nice value, the more access to the CPUs the process has.
- A regular user can set nice values only from 0 to 19. No negative values are allowed.
- A regular user can set the nice value higher, not lower, if a user sets the nice value to 10 and then later wants to set it back to 5, that action will fail
- A regular user can set the nice value only on the user's own processes.
- The root user can set the nice value on any process to any valid value, up or down.

renice -n -5 20284

You can use the **nice** command to run a command with a particular nice value. When a process is running, you can change the nice value using the **renice** command, along with the process ID of the process :

 # nice -n +5 updatedb &

by using the renice command. (Remember that a regular user can't reduce the nice value or ever set it to a negative number.) Here's how you would change the nice value for the updatedb command just run to -5

 # renice -n -5 20284

The Concept of Cgroups

cgroups (control groups) as a more powerful way to manage and limit system resources for processes, users, or applications

Limitations of "nice"

Nice Value: Adjusts the priority of a single process (e.g., giving it more or less CPU time).

Does Not Scale: It doesn't apply to child processes or related processes in a service.

No Resource Limits: It doesn't control the total resources (CPU, memory, disk I/O) that a user, application, or service can consume.

Why Cgroups ?!

In modern computing (e.g., cloud, virtualization, containers), systems are shared by many users or applications. To ensure fair and efficient resource allocation, Linux uses cgroups to:

Group Processes

Organize processes into tasks (control groups).

Set Resource Limits

Apply limits to CPU, memory, disk I/O, and more for entire groups of processes.

Inherit Limits

Child processes inherit the resource limits of their parent process.

How Cgroups Work

Hierarchy

Cgroups are organized in a hierarchy (tree structure).

Example: A parent group called daemons sets default limits for all server processes, and child groups (e.g., httpd, vsftpd) set specific limits for individual services.

Resource Control

Cgroups can limit:
- CPU: Restrict usage to a percentage of total CPU power.
- Memory: Limit RAM usage.
- Disk I/O: Control read/write access to storage.
- Network: Restrict bandwidth.

Inheritance

When a process in a cgroup spawns child processes, they inherit the same resource limits.

Example Use Case in action

Cloud/Hypervisor: On a Linux system acting as a hypervisor, cgroups can ensure that:
- Each virtual machine or container gets a fair share of resources.
- No single user or application consumes too much CPU, memory, or disk I/O.

Background and Foreground Processes

Although the bash shell doesn't include a GUI for running many programs at once, it does let you move active programs between the background and foreground. In this way, you can have lots of stuff running and selectively choose the one you want to deal with at the moment.

Placing a program in the background

- By adding an ampersand **(&)** to the end of a command line when you first run the command
- Use the **at** command to run commands in such a way that they are not connected to the shell.

Notice that the job number and process ID number are displayed when the command is launched in the background. To check which commands you have running in the background, use the **jobs** command

To put a running command in the background

press Ctrl+Z. After the command is stopped, you can either bring it back into the foreground to run (the **fg** command) or start it running in the background (the **bg** command). Keep in mind that any command running in the background might spew output during commands that you run subsequently from that shell

To see the process ID for the background job, add a -l (the lowercase letter L) option to the jobs command. If you type ps, you can use the process ID to figure out which command is for a particular background job.

Before you put a text processor, word processor, or similar program in the background, make sure that you save your file. It's easy to forget that you have a program in the background, and you will lose your data if you log out or the computer reboots

To refer to a background job (to cancel or bring it to the foreground) :

- Use a percent sign **(%)** followed by the job number
- The **%** command brings the most recent job to the foreground
- **%? string** Refers to a job where the command line contains a string at any point
- **%--** Refers to the job stopped before the one most recently stopped

Process States in Linux

In Linux, every process goes through multiple states during its lifecycle. Understanding these states is crucial for effective process management. The main process states include:

R (Running or Runnable)

The process is either actively running on the CPU or waiting in the queue to be scheduled.

S (Sleeping)

The process is waiting for an event or resource before it can continue execution.

Z (Zombie)

The process has completed execution but still has an entry in the process table, waiting for

T (Stopped or Traced)

The process is either paused (e.g., using CTRL+Z) or being debugged

D (Disk Sleep)

The process is in an uninterruptible sleep state, typically waiting for disk I/O.

Uninterruptible Sleep

Usually waiting for I/O operations and cannot be interrupted.

Interruptible Sleep

Can be woken up by signals.