CHAPTER-8 MONITOR AND MANAGE LINUX PROCESSES

Process States and Lifecycle

Definition of a Process

A process is a running instance of a launched, executable program. From the moment that a process is created, it consists of the following items:

- An address space of allocated memory
- Security properties including ownership credentials and privileges
- One or more execution threads of program code
- A process state

The environment of a process is a list of information that includes the following items:

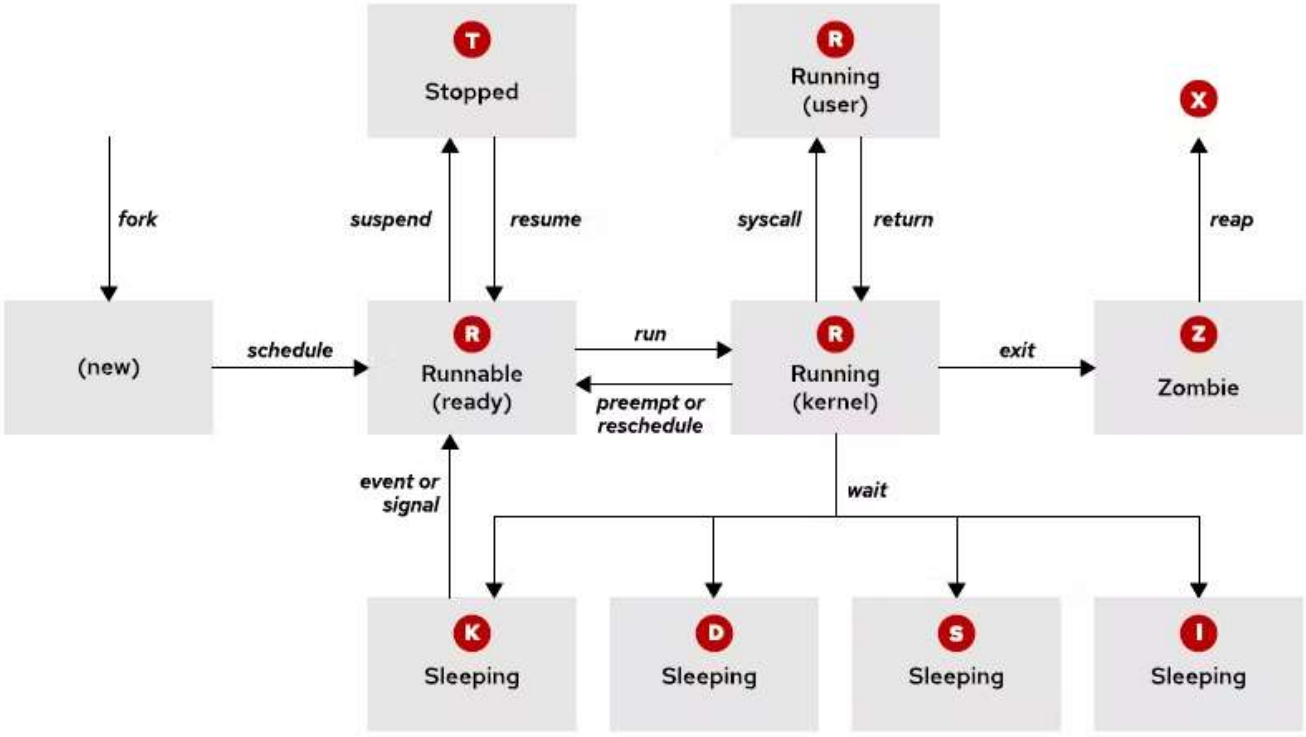
- Local and global variables
- A current scheduling context
- Allocated system resources, such as file descriptors and network ports

Describe Process States

In a multitasking operating system, each CPU (or CPU core) can be working on one process at a time. As a process runs, its immediate requirements for CPU time and resource allocation change. Processes are assigned a state, which changes as circumstances dictate.

The following diagram and table describe Linux process states in detail.

Linux Process States



Name	Flag	Kernel-defined state name and description
Running	R	TASK_RUNNING: The process is either executing on a CPU or waiting to run. The process can be executing user routines or kernel routines (system calls), or be queued and ready when in the *Running* (or *Runnable*) state.
Sleeping	S	TASK_INTERRUPTIBLE: The process is waiting for some condition: a hardware request, system resource access, or signal. When an event or signal satisfies the condition, the process returns to *Running*.
	D	TASK_UNINTERRUPTIBLE: This process is also sleeping, but unlike S state, does not respond to signals. Used only when process interruption might cause an unpredictable device state.
	K	TASK_KILLABLE: Identical to the uninterruptible D state, but modified to allow a waiting task to respond to the signal that it should be killed (exit completely). Utilities frequently display *Killable* processes as D state.
	I	TASK_REPORT_IDLE: A subset of state D. The kernel does not count these processes when calculating load average. Used for kernel threads. Flags TASK_UNINTERRUPTIBLE and TASK_NOLOAD are set. Similar to TASK_KILLABLE, also a subset of state D. It accepts fatal signals.

Name	Flag	Kernel-defined state name and description
Stopped	T	TASK_STOPPED: The process is stopped (suspended), usually by being signaled by a user or another process. The process can be continued (resumed) by another signal to return to running.
	T	TASK_TRACED: A process that is being debugged is also temporarily stopped and shares the same T state flag.
Zombie	Z	EXIT_ZOMBIE: A child process signals to its parent as it exits. All resources except for the process identity (PID) are released.
	X	EXIT_DEAD: When the parent cleans up (reaps) the remaining child process structure, the process is now released completely. This state cannot be observed in process-listing utilities.

Importance of Process States

When **troubleshooting a system**, it is important to **understand** how the **kernel communicates with processes** and how processes communicate with each other. The system assigns a state to every new process. The S column of the **top command** or the STAT column of the **ps command shows** the **state** of each **process**. On a single CPU system, only one process can run at a time.

It is possible to see several processes with an R state. However, not all processes are running consecutively; some of them are in waiting status.

DESCRIPTION	COMMANDS / OPTIONS
Process-related information including:	Process-related information including: **PID:** Process ID **USER:** Owner of the process **PR:** Priority **NI:** Nice value **VIRT:** Virtual memory usage **RES:** Resident set size (non-swapped physical memory used) **SHR:** Shared memory **S:** Process status (S: Sleeping, R: Running, I: Idle) **%CPU:** Percentage of CPU usage **%MEM:** Percentage of memory usage TIME+: Total CPU time **COMMAND:** Command or process name
To show the Linux processes (TOP command show a dynamic real-time view of the running system)	**Syntax:** top [OPTION] **Example:** [user@host ~]$top ↵ **-n** exit top command after N th time **Example:** [user@host ~]$top –n 10 ↵ **-u** Specific user process **Example:** [user@host ~]$top –u root ↵ **-d** It tells delay time between screen updates **-b** Send output from top to file or any other programs **-s** Use top in Secure mode **-c** starts top with last closed state

Fundamental Keystrokes in top Command

Key	Purpose
? *or* h	Help for interactive keystrokes.
l, t, m	Toggles for load, threads, and memory header lines.
1	Toggle for individual CPUs or a summary for all CPUs in the header.
s	Change the refresh (screen) rate, in decimal seconds (such as 0.5, 1, 5).
b	Toggle reverse highlighting for Running processes; default is bold only.
Shift+b	Enables bold use in display, in the header, and for *Running* processes.
Shift+h	Toggle threads; show process summary or individual threads.
u, Shift+u	Filter for any user name (effective, real).
Shift+m	Sort process listing by memory usage, in descending order.
Shift+p	Sort process listing by processor utilization, in descending order.
k	Kill a process. When prompted, enter PID, and then signal.
r	Renice a process. When prompted, enter PID, and then nice_value.
Shift+w	Write (save) the current display configuration for use at the next top restart.
q	Quit.
f	Manage the columns by enabling or disabling fields. You can also set the sort field for top.

DESCRIPTION	COMMANDS / OPTIONS
To listing detailed information for current processes. (• User identification (UID), which determines process privileges. • Unique process identification (PID). • Amount of used CPU and real time. • Amount of allocated memory. • The process stdout location, which is known as the controlling terminal. • The current process state.)	**Syntax:** ps [options] **-a** List all ruining processes for all users **-u** Username **Example:** [root@host ~]# ps –u user ↵ **-x** View All Processes Owned By You **-e** Lists all processes on the entire system **-O** list-fields **-ao** list of fields to view **- lax** Long listing **To see every process on the system** -ef To display all processe -eF -ely **To see every process on the system using BSD** ax aux **To print process tree** -ejH -axjf

Control Jobs

With the job control shell feature, a single shell instance can run and manage multiple commands. A job is associated with each pipeline that is entered at a shell prompt. All processes in that pipeline are part of the job and are members of the same process group.

Only **one job at a time can read input** and keyboard-generated signals from a particular terminal window. Processes that are part of that job are foreground processes of that controlling terminal.

A background process of that controlling terminal is any other job that is associated with that terminal. Background processes of a terminal cannot read input or receive keyboard-generated interrupts from the terminal, but are able to write to the terminal. A background job might be stopped (suspended) or it might be running. If a running background job tries to read from the terminal, then it is automatically suspended.

Each terminal runs in its own session, and can have a foreground process and any number of background processes. A job is in only one session, which belongs to its controlling terminal.

DESCRIPTION	COMMANDS / OPTIONS
To display the list of jobs for the shell's session.	**Syntax:** job [options] **Example:** [root@host ~]# job ↵ **-l** Lists process IDs in addition to the normal information. **-n** List only processes that have changed status since the last notification. **-p** Lists process IDs only **-r** Restrict output to running jobs **-s** Restrict output to stopped jobs
To bring a background job to the foreground.	**Syntax:** fg [options][%job_number] **Example:** [root@host ~]# fg %1↵ **%n** Refer to job number n. **%str** Refer to a job which was started by a command beginning with str. **%?str** Refer to a job which was started by a command containing str. **%% or %+** Refer to the current job. fg and bg will operate on this job if no job_spec is given. **%-** Refer to the previous job.
To place foreground jobs in background	**Syntax:** bg[options][%job_number] **Example:** [root@host ~]# bg %1↵
The sleep command is running in the foreground on the controlling Terminal	**Syntax:** sleep [options][%job_number] **Example:** [root@host ~]# fg %1↵
To send a foreground process to the background, press the keyboard-generated suspend request (Ctrl+z) in the terminal. The job is placed in the background and suspended.	**To stop running process** Ctrl+z **To close running process** Ctrl+c

This command sends a process/script/command to the background. To trace running process	[commands] **&** **Syntax:** strace [options] command -c To count number of system calls **Example:** [root@host ~]# strace –c ls↵ -e To trace particular or specific system calls -r To print timestamp of each call -T To print time spent on system calls -t To print wall clock time of each system call -i To print instruction pointer -o To print output to a file

Kill Processes

A signal is a software interrupt that is delivered to a process. Signals report events to an executing program. Events that generate a signal can be an error, an external event (an I/O request or an expired timer), or by the explicit use of a signal-sending command or keyboard sequence.

The following table lists the fundamental signals that system administrators use for routine process management.

Signal	Name	Definition
1	HUP	`Hangup` : Reports termination of the controlling process of a terminal. Also requests process re-initialization (configuration reload) without termination.
2	INT	`Keyboard interrupt` : Causes program termination. It can be blocked or handled. Sent by pressing the INTR (Interrupt) key sequence (`Ctrl+c`).
3	QUIT	`Keyboard quit` : Similar to SIGINT; adds a process dump at termination. Sent by pressing the QUIT key sequence (`Ctrl+\`).
9	KILL	`Kill, unblockable` : Causes abrupt program termination. It cannot be blocked, ignored, or handled; consistently fatal.
15 *default*	TERM	`Terminate` : Causes program termination. Unlike SIGKILL, it can be blocked, ignored, or handled. The "clean" way to ask a program to terminate; it allows the program to complete essential operations and self-cleanup before termination.
18	CONT	`Continue` : Sent to a process to resume if stopped. It cannot be blocked. Even if handled, it always resumes the process.
19	STOP	`Stop, unblockable` : Suspends the process. It cannot be blocked or handled.
20	TSTP	`Keyboard stop` : Unlike SIGSTOP, it can be blocked, ignored, or handled. Sent by pressing the suspend key sequence (`Ctrl+z`).

Each signal has a default action, which is usually one of the following actions:

- Term : Terminate a program (exit) at once.
- Core : Save a program's memory image (core dump), then terminate.
- Stop : Stop a running program (suspend) and wait to continue (resume).

Programs react to the expected event signals by implementing handler routines to ignore, replace, or extend a signal's default action.

You can signal the current foreground process by pressing a keyboard control sequence to suspend (Ctrl+z), kill (Ctrl+c), or core dump (Ctrl+\) the process. However, you might use signal-sending commands to send signals to background processes in a different session.

The **kill** command uses a **PID number** to send a signal to a process. Despite its name, you can use the kill command **to send any signal**, not just those signals for terminating programs. You can use the kill command -l option to list the names and numbers of all available signals.

DESCRIPTION	COMMANDS / OPTIONS
To terminate processes manually	**Syntax:** kill [options][signal]PID **-l** To display all the available signals **Example:** [root@host ~]# kill -l ↵ **Example:** [root@host ~]# kill 5194 ↵ **-s** To be sent to the process
To signal one or more processes that match selection criteria.	**Syntax:** pkill [options][signal] pattern **-f** To set the pattern is matched with the full command that started the process.
To find process ID (PID)	**Syntax:** pgrep[options] pattern **-l** To list the process names and IDs. **-u** specify user **Example:** [root@host ~]# pgrep -l -u Jon ↵
To view a process tree for the system or a single user.	**Syntax:** pstree [options] [pid or username] **-a** To include command line arguments in output **-p** To display PIDs **-c** To force pstree to expand identical subtrees in output **-n** To sort processes **-u** To see who is the owner/user of a process **-h** To highlight the current process or any other process **-g** To show process group IDs in output

Monitor Process Activity

Describe Load Average

Load average is a measurement that Linux kernel provides, to represent the perceived system load for a period of time. It can be used as a rough gauge of how many system resource requests are pending, to determine whether system load increases or decreases.

The **kernel collects** the **current load number every five seconds** based on the number of processes in runnable and uninterruptible states. This number is accumulated and reported as an exponential moving average over the **most recent 1, 5, and 15 minutes**.

Load Average Calculation

The load average represents the perceived system load for a period of time. Linux determines load average by reporting how many processes are ready to run on a CPU and how many processes are waiting for disk or network I/O to complete.

- The load number is a running average of the number of processes that are ready to run (in process state R) or are waiting for I/O to complete (in process state D).
- Some UNIX systems consider only CPU utilization or run queue length to indicate system load.
- Linux also includes disk or network utilization, because the high usage of these resources can significantly impact system performance as CPU load. For high load averages with minimal CPU activity, examine disk and network activity.

Interpret Load Average Values

The uptime command is one way to display the current load average. It prints the current time, how long the machine has been up, how many user sessions are running, and the current load average.

DESCRIPTION	COMMANDS / OPTIONS
To display the current load average.	**Syntax:** uptime [options] **-p** --pretty show uptime in pretty format **-h** --help display this help and exit **-s** --since system up since **Example:** [root@host ~]# uptime -p ↲ **Example:** [root@host ~]# uptime -s ↲
To getting detailed information about the CPU (Central Processing Unit) configuration of our system	**Syntax:** uptime [options] **-b** Generate machine-readable (raw) output. **-c** Display information about online CPUs. **-p** Format output in a parsable format. **-a** Display all information (equivalent to -bcp). **-x** Display extended information (shows flags and other details). **-y** Display NUMA (Non-Uniform Memory Access) information. **-s** Display a summary of the CPU architecture. **-t** Display the thread, socket, and core information. **-o** Display only the specified fields (use with -b or -p for customized output).
To change the priority of a processor command To changes the priority of a running program, called by its PID	**Syntax:** nice[options] **-n** To set the priority(n) of a process **Syntax:** renice[options] [priority] PID **-n** To set the priority(n) of a process **-p** Changing priority of the running process **-g** To change the priority of all programs of a specific group **-u** To change the priority of all programs of a specific user