**Process Control and Management [ ps, pstree, nice, renice, chrt, kill, top, htop, strace ]**

**Introduction**

Process control and management is all about how an operating system handles running programs, called processes. It ensures that processes are created, scheduled, monitored, and terminated smoothly, while also managing resources like CPU and memory so everything works efficiently. This is super important because it keeps your system responsive, avoids crashes, and ensures that all tasks—whether it's opening apps or running background updates—get the resources they need without interfering with each other.

**Process**

A process is basically a program that's running. It includes the program's code, what it's currently doing, and the resources it's using, like memory or CPU. Every process has a unique ID, called a Process ID (PID), and it runs separately from other processes. Processes are important because they let multiple programs run at the same time, while the operating system makes sure everything works smoothly without any conflicts.

**Steps involved in Process Creation**

1. When you open any process like google chrome in this example, fork( ) system call is called. The fork( ) creates a process (child process) identical to the current process (parent process) with the same cpu and memory resource as the current process (allocated by the Kernel).
2. The child process uses one of the functions in the exec( ) family, and when exec( ) is called the child process changes its state according to the process (say google chrome in this example), memory segments are reset to initial state.
3. PID is then assigned to the child process, the parent process is usually bash when the process is opened from terminal and if the process is open from GUI or desktop environment the gnome-shell becomes the parent process in most of the cases.

**Steps involved in Process Termination**

1. When a process is complete the exit( ) routine is called to notify the Kernel that it is ready to die.
2. The child process supplies the exit code (an integer) to the Kernel usually ‘0’ on normal termination.
3. Before the child process disappears completely the Kernel wants the Parent to acknowledge (wait( ) call) the death of the child process so it receives the copy of the exit code.
4. If the parent process dies before the termination of child process the child process becomes orphan process and systemd becomes the new parent of that orphan child process.

**Process States**

The various states that a process can take are

* Uninterruptible Sleep ( D )
* Interruptible Sleep ( S )
* Running ( R )
* Idle Kernel Thread ( I )
* Stopped by job control signal ( T )
* Stopped by debugger during tracing ( t )
* Dead Process ( X ) - can never be seen
* Zombie Process ( Z )

There are a few symbols that are displayed as well in the stat column they are

* < ( High Priority )
* N ( Low Priority )
* s ( Session Leader )
* L ( Pages locked into memory )
* l ( Multi Threaded )
* + ( Is in Foreground Process Group )

**Ps**

The ps command in Linux is used to display information about active processes running on the system. It provides details like process IDs (PIDs), process owners, CPU and memory usage, state of the process, CPU time and the command that started each process.

*ps <options> (or) ps -aux (or) ps -elF*

**Working**

1. The shell executes the ps binary.
2. ps reads process information from the /proc pseudo-filesystem, specifically files like /proc/[pid]/stat.
3. It parses the data (e.g., process ID, CPU usage, memory usage) and formats it for output.
4. The final output is printed to the terminal.

|  |  |
| --- | --- |
| **Field** | **Contents** |
| USER | Username of process owner |
| PID | Process ID |
| PPID | Parent process ID |
| %CPU | Percentage of CPU this process is using |
| %MEM | Percentage of memory this process is using |
| LWP | Light weight process ID |
| NLWP | Number of light weight process |
| VSZ | Virtual size of the process |
| RSS | Resident set size (number of pages in memory) |
| TTY | Control terminal ID |
| STAT | Status of current process |
| TIME | CPU time the process has consumed |
| COMMAND | Command name and arguments |

**Options**

a - To display all processes. (BSD format used with u, x, l).

e - To display all processes (standard format used with l, f, L, o).

u - To display in user-oriented format.

f - To do full-format listing. (used with L option to display LWP & NLWP column).

F - Extra full format.

l - Long format.

L - To Show threads, possibly with LWP and NLWP columns.

x - To display processes without controlling ttys.

o - To display in user defined or specified format.

w - To display wide (more info in command column) output.

p - To select the process using PID.

C - To select the process using the command name.

**Pstree**

The pstree command in Linux displays running processes in a tree-like structure, showing the parent-child relationship between them.

*pstree <options> (or) pstree -p (or) pstree -s*

**Working**

1. Like ps, pstree reads data from /proc.
2. It identifies parent-child relationships by checking each process’s ppid (parent process ID).
3. It builds a tree-like structure in memory, representing the hierarchy.
4. The tree is formatted and displayed on the terminal.

**Options**

p - To show PIDs.

s - To show parents of the selected process.

a - To show command line arguments.

c - Don't compact identical subtrees.

t - To show full thread names.

T - To hide threads, show only processes.

v - To display version information.

l - Don't truncate long lines.

n - To sort the output using PID.

**Nice**

The nice command in Linux is used to start a process with a specific priority, influencing how the operating system allocates CPU time to it. By default, processes start with a nice value of 0, but nice lets you adjust this value to make a process run with a higher or lower priority.

*nice -n <niceness> <command> (or) nice -n 10 ping 127.0.0.1*

**Working**

1. The shell finds the nice binary and executes it.
2. nice modifies the process’s "niceness" value by making a setpriority() system call to the kernel.
3. The kernel adjusts the process scheduling priority based on the niceness value.

**Options**

n - To adjust the nice value ( -19 to 20 ).

v - To display version information.

h - To display the help menu.

**Renice :**

The renice command in Linux is used to change the priority (nice value) of a running process. Unlike nice, which sets the priority when a process starts, renice allows you to adjust the priority of processes that are already running. This can be helpful for tuning the system’s responsiveness and ensuring that critical processes get the CPU time they need.

*renice -n <niceness> -p <PID> (or) renice -n 20 -p 2525*

**Working**

1. The renice binary is executed by the shell.
2. It uses the setpriority() system call to modify the niceness value of a running process, identified by its PID.
3. The kernel immediately updates the process scheduling priority.

**Options**

n - To adjust the nice value ( -19 to 20 ).

u - To select all the processes of the specified user.

p - To select the process using PID.

v - To display the version information.

h - To display the help menu.

**Kill**

The kill command in Linux is used to terminate processes by sending signals to them. While its main use is to stop a process, kill can also be used to send other types of signals, such as pausing or continuing processes, depending on what you need.

*kill <option> <PID> (or) kill -9 2520*

**Working**

1. The shell executes the kill binary with the given process ID (PID).
2. kill sends a signal (e.g., SIGTERM, SIGKILL) to the target process using the kill() system call.
3. The kernel delivers the signal to the process, which handles it according to the signal type (e.g., terminate, stop).

|  |  |  |
| --- | --- | --- |
| **Signal** | **Number** | **Note** |
| SIGHUP (Hung Up) | 1 | Stop and restart a process with the same PID. |
| SIGINT (Interrupt) | 2 | Send an interrupt signal. |
| SIGQUIT (Core Dump) | 3 | Terminate a process and store info in a file ‘core’. |
| SIGKILL (Absolute Kill) | 9 | Force the process to stop and send it to a special device. |
| SIGBUS (Bus Error) | 10 | To Bring down a crashed program. |
| SIGSEGV(Segmentation Fault) | 11 | To Bring down a crashed program. |
| SIGTERM (Termination) | 15 | Default Kali command |
| SIGCONT (continue) | 18 | Resume stopped process |
| SIGTSTP (Keyboard Stop) | 19 | To stop a running process |

**Options**

s - To send a specified signal to the process.

l - To list the available signals.

**Top**

The top command in Linux is a powerful tool for monitoring system performance in real-time. It provides a dynamic, real-time view of the system's resource usage, including CPU, memory, and processes. It's particularly useful for tracking which processes are consuming the most resources and for troubleshooting performance issues.

*top <options> (or) top -d 5*

**Working**

1. The shell executes the top binary.
2. top periodically reads process information from /proc, similar to ps.
3. It calculates CPU and memory usage by comparing process states over time.
4. The terminal is updated dynamically using a curses-based interface.

**Options**

d - To change the refresh time (default is 3 seconds).

k - To kill a process using PID.

1 - To display individual CPU’s.

q - To exit the top program.

v - To display version information.

e - To display all processes.

H - To view thread information.

h - To display the help menu.

**Htop**

The htop is an interactive process viewer and system monitor for Unix-like systems, providing a more user-friendly and visually appealing alternative to top. It offers real-time monitoring of system processes, memory, CPU usage, and more, with the added benefit of an easy-to-navigate interface and additional features.

*htop*

**Working**

1. htop works similarly to top but uses the ncurses library for a more interactive and colorful interface.
2. It reads additional details from /proc to display graphs and more user-friendly metrics.

**Options**

F1 - To display the help menu.

F2 - To open the setup menu for customized display.

F3 - To search for a process.

F4 - To filter processes by search string.

F5 - To display in tree format.

F6 - To sort the process based on a specific column.

F7 - To decrease niceness of a process (increase process priority).

F8 - To increase niceness of a process (decrease process priority).

F9 - To kill a process.

F10 - To quit the htop program.

**Strace**

The strace command in Linux is used to trace system calls and signals made by a process. It is a powerful debugging tool that shows the interactions between a program and the kernel, making it useful for troubleshooting issues, debugging applications, and understanding program behavior at a low level.

*strace <command> (or) strace ls*

**Working**

1. When you run strace, it attaches to the target process using the ptrace() system call.
2. It intercepts every system call the process makes, logs the parameters passed, and records the return values.
3. The output is displayed in real time, showing the program's interaction with the kernel.

**Chrt**

The chrt command in Linux is used to set or modify the real-time scheduling policies and priority of a process. It allows you to start a new process or change the scheduling of an existing one. This is particularly useful for applications that require precise control over CPU scheduling, such as multimedia or real-time systems.

*chrt -<schedule policy> <priority> <command> (or) chrt -f 10 ping 127.0.0.1*

**Working**

1. The shell locates and executes the chrt binary, parsing the scheduling policy, priority, and target process ID (PID) from the input.
2. Queries: sched\_getscheduler() retrieves the current policy, and sched\_getparam() fetches the priority.
3. Modifies: sched\_setscheduler() sets the new policy (e.g., SCHED\_RR, SCHED\_FIFO) and priority.
4. The kernel verifies the policy, checks the priority range, and ensures the user has sufficient permissions (real-time policies often require root access).
5. The process’s task\_struct is updated with the new scheduling settings, and the kernel scheduler adjusts the process's CPU allocation accordingly.
6. Success updates the scheduling or displays the new policy and priority; errors return descriptive messages using errno.

**Options**

b - To set schedule policy to SCHED\_BATCH.

d - To set schedule policy to SCHED\_DEADLINE.

f - To set schedule policy to SCHED\_FIFO.

r - To set schedule policy to SCHED\_RR.

o - To set schedule policy to SCHED\_OTHER.

i - To set schedule policy to SCHED\_IDLE.

T - Runtime parameter for DEADLINE (nano seconds).

P - Period parameter for DEADLINE (nano seconds).

D - Deadline parameter for DEADLINE (nano seconds).

p - To operate on the existing PID.

m - To show maximum and minimum valid priority for each scheduling policy.

h - To display the help menu.

v - To display version information.