

Lab Project - 3

Objective: Linux process management lab

Lab 1 : Process Exploration and Identification

Objective:

Understand how processes work in Linux, and how to identify and explore running processes.

Task:

1. List Running Processes:

Use ps, top, or htop to list all running processes on the system.

Understand the difference between ps, top, and htop, and experiment with their options (e.g., ps aux, top -u <username>).

top

%Cpu(s): 0.0 us, 0.1 sy, 0.0 ni, 99.8 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st										
MiB Mem : 2913.6 total, 2352.1 free, 393.2 used, 168.3 buff/cache										
MiB Swap: 1024.0 total, 1024.0 free, 0.0 used. 2369.2 avail Mem										
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
135	systemd+	20	0	89364	6540	5736	S	0.3	0.2	0:00.80 systemd-timesyn

ps

vinu@DESKTOP-5K616C3:~/backup\$ ps			
PID	TTY	TIME	CMD
350	pts/0	00:00:00	bash
997	pts/0	00:00:00	bash
1278	pts/0	00:00:00	top
1279	pts/0	00:00:00	ps

top -u vinu

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
350	vinu	20	0	6256	5168	3388	S	0.0	0.2	0:00.64	bash
441	vinu	20	0	16932	9184	7728	S	0.0	0.3	0:00.26	systemd
442	vinu	20	0	103448	3504	8	S	0.0	0.1	0:00.00	(sd-pam)
448	vinu	20	0	6104	4876	3320	S	0.0	0.2	0:00.06	bash
997	vinu	20	0	4916	3524	3128	T	0.0	0.1	0:00.01	bash
1278	vinu	20	0	7796	3660	3060	T	0.0	0.1	0:00.13	top

1. Using `ps` (Process Status)

The `ps` command provides a snapshot of the currently running processes.

Basic Usage:

```
bash
```

```
ps aux
```

- `a` → Shows processes from all users.
- `u` → Displays user-oriented format.
- `x` → Lists processes not attached to a terminal.

Detailed View:

```
bash
```

```
ps -eo pid,user,%cpu,%mem,command --sort=-%cpu
```

- Lists all processes (`-e`).
- Displays user, CPU usage, memory usage, and command.

```
vinu@DESKTOP-5K616C3:~/backup$  
vinu@DESKTOP-5K616C3:~/backup$ ps aux  
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root         1  0.0  0.3 165928 11384 ?        Ss   Feb23   0:06 /sbin/init  
root         2  0.0  0.0   2776  1920 ?        Sl   Feb23   0:00 /init  
root         6  0.0  0.0   2776   132 ?        Sl   Feb23   0:00 plan9 --control-socket 7 --log-level 4 --server-f  
root        86  0.0  0.5  47736 15308 ?        S<s  Feb23   0:03 /lib/systemd/systemd-journald  
root       114  0.0  0.1  21964   5864 ?        Ss   Feb23   0:01 /lib/systemd/systemd-udevd  
_rpc      133  0.0  0.1   8104   4064 ?        Ss   Feb23   0:00 /sbin/rpcbind -f -w  
systemd+  134  0.0  0.4  25672 12764 ?        Ss   Feb23   0:00 /lib/systemd/systemd-resolved  
systemd+  135  0.0  0.2  89364  6540 ?        Ssl  Feb23   0:00 /lib/systemd/systemd-timesyncd  
root      143  0.0  0.0   5128   176 ?        Ss   Feb23   0:00 /usr/sbin/blkmapd  
root      144  0.0  0.0   3108   2208 ?        Ss   Feb23   0:00 /usr/sbin/rpc.idmapd  
root      145  0.0  0.0   5464   2804 ?        Ss   Feb23   0:00 /usr/sbin/nfsdclld  
root      205  0.0  0.0   4308   2748 ?        Ss   Feb23   0:00 /usr/sbin/cron -f -P
```

Detailed View:

```
bash
```

```
ps -eo pid,user,%cpu,%mem,command --sort=-%cpu
```

- Lists all processes (`-e`).
- Displays user, CPU usage, memory usage, and command.
- Sorts by CPU usage in descending order.

2. Find a Specific Process:

```
# Use pgrep to find the PID (process ID) of a specific running process like apache2 or nginx.
```

```
# Use pstree to view a tree of processes and their parent-child relationships.
```

1. Finding a Specific Process with `pgrep`

The `pgrep` command is used to search for processes by name and return their PIDs.

Basic Usage:

```
bash
```

```
pgrep apache2
```

This command returns the process ID(s) of all running `apache2` processes.

- Find the PID of Nginx:

bash

```
pgrep nginx
```

- To see more details, use:

bash

```
ps -fp $(pgrep apache2)
```

```

vinu@DESKTOP-5K616C3:~/backup$ pstree
systemd--2*[agetty]
        |
        +--blkmapd
        |
        +--cron
        |
        +--dbus-daemon
        |
        +--init-systemd(Ub)
            |
            +--SessionLeader--Relay(350)--bash
                |
                +--bash--sudo--sudo--apt
                |
                +--pstree
                |
                +--4*[top]
            |
            +--init--{init}
            |
            +--login--bash
            |
            +--{init-systemd(Ub)}
        |
        +--networkd-dispat
        |
        +--nfsdcld
        |
        +--rpc.idmapd
        |
        +--rpc.mountd
        |
        +--rpc.statd
        |
        +--rpcbind
        |
        +--rsyslogd--3*[{rsyslogd}]
        |
        +--squid--squid
            |
            +--log_file_daemon
            |
            +--pinger

```

3. Investigate Process Details:

Use lsof to identify files opened by a process.

```
vinu@DESKTOP-5K616C3:~/backup$ lsof
COMMAND    PID TID TASKCMD USER  FD   TYPE    DEVICE  SIZE/OFF      NODE NAME
systemd    1                                root   cwd    unknown                /proc/1/cwd (readli
k: Permission denied)
systemd    1                                root   rtd    unknown                /proc/1/root (readl
nk: Permission denied)
systemd    1                                root   txt    unknown                /proc/1/exe (readli
k: Permission denied)
systemd    1                                root   NOFD                   /proc/1/fd (opendir
Permission denied)
init       6                                root   cwd    unknown                /proc/6/cwd (readli
k: Permission denied)
init       6                                root   rtd    unknown                /proc/6/root (readl
nk: Permission denied)
init       6                                root   txt    unknown                /proc/6/exe (readli
k: Permission denied)
```

Check the memory usage and CPU time of a process using ps -eo pid,etime,%mem,%cpu,comm.

```
For more details see ps(1).
vinu@DESKTOP-5K616C3:~$ ps -eo pid,etime,%mem,%cpu,comm
  PID      ELAPSED %MEM %CPU COMMAND
    1      04:17:10  0.3  0.0 systemd
    2      04:17:09  0.0  0.0 init-systemd(Ub
    6      04:17:09  0.0  0.0 init
   86      04:17:06  0.5  0.0 systemd-journal
  114      04:17:06  0.1  0.0 systemd-udev
  133      04:17:05  0.1  0.0 rpcbind
  134      04:17:05  0.4  0.0 systemd-resolve
  135      04:17:05  0.2  0.0 systemd-timesyn
  143      04:17:05  0.0  0.0 blkmapd
  144      04:17:05  0.0  0.0 rpc.idmapd
  145      04:17:05  0.0  0.0 nfsdcld
  205      04:17:05  0.0  0.0 cron
  207      04:17:05  0.1  0.0 dbus-daemon
  212      04:17:05  0.6  0.0 networkd-dispat
  213      04:17:05  0.2  0.0 rsyslogd
  216      04:17:05  0.2  0.0 systemd-logind
  246      04:17:04  0.0  0.0 rpc.statd
  247      04:17:04  0.0  0.0 rpc.mountd
```

Lab 2 : Process Control and Termination

Objective:

Learn how to control, pause, resume, and terminate processes in Linux.

Task:

1. Send Signals to Processes:

Use kill to send signals to processes. Try sending a SIGTERM and SIGKILL to terminate a process by PID.

```
vinu@DESKTOP-5K616C3:~$ kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
vinu@DESKTOP-5K616C3:~$
```

Use kill -s STOP <PID> and kill -s CONT <PID> to stop and resume a process.

```
vinu@DESKTOP-5K616C3:~$ kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
vinu@DESKTOP-5K616C3:~$
```

Example Usage:

1. Find the PID of a process

```
perl

ps aux | grep <process_name>
```

This command will display the process ID (PID) of the running process.

2. Send a SIGTERM signal (graceful termination)

```
bash

kill -15 <PID>
```

OR

```
bash

kill <PID>    # Default signal is SIGTERM
```

```
vinu@DESKTOP-5K616C3:~$ ps aux | grep 1333
vinu      1335  0.0  0.0  4028  2060 pts/0    S+   03:04   0:00 grep --color=auto 1333
vinu@DESKTOP-5K616C3:~$
```

2. Send Custom Signals:

Send a SIGINT signal to a running process (e.g., when running a program in the terminal, use Ctrl+C or `kill -s SIGINT <PID>`).

```
bash

pkill -SIGINT process_name
```

Example:

```
bash

pkill -SIGINT python
```

```
vinu@DESKTOP-5K616C3:~$ kill -s SIGINT
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
vinu@DESKTOP-5K616C3:~$
```

3. Test Process Termination:

Start a process, for example, sleep 300, then find its PID and try to terminate it using kill or kill -9.

3. Send a SIGKILL signal (force termination)

```
bash
```

```
kill -9 <PID>
```

```
vinu@DESKTOP-5K616C3:~$ kill -9
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
vinu@DESKTOP-5K616C3:~$ kill -l
```

Lab 3 : Managing Background and Foreground Processes

Objective:

Learn how to run processes in the background and manage jobs effectively.

Task:

1. Run a Process in the Background:

Start a process in the background using &, e.g., sleep 100 &.

```
vinu@DESKTOP-5K616C3:~$ sleep 100
^Z
[7]+  Stopped                  sleep 100
vinu@DESKTOP-5K616C3:~$
```


Use jobs to see a list of background jobs.

```
vinu@DESKTOP-5K616C3:~$ job
Command 'job' not found, did you mean:
  command 'jo' from snap jo (1.9)
  command 'mob' from snap mob-sh (4.2.0)
  command 'joe' from deb joe (4.6-1build2)
  command 'joe' from deb joe-jupp (3.1.40-1)
  command 'jot' from deb athena-jot (9.0-8)
  command 'wob' from deb wob (0.12-1)
```

2. Bring a Process to the Foreground:

Use the fg command to bring a background process to the foreground.

```
vinu@DESKTOP-5K616C3:~$ fg
sleep 100
vinu@DESKTOP-5K616C3:~$
```

3. Pause and Resume a Process:

Pause a background process using Ctrl+Z and resume it in the background with the bg command.

```
vinu@DESKTOP-5K616C3:~$ sleep 100
^Z
[7]+  Stopped                  sleep 100
```

4. Control Multiple Jobs:

Start multiple jobs in the background and manage them with jobs, fg, and bg.

Jobs

```
vinu@DESKTOP-5K616C3:~$ jobs
[1]  Stopped                  bash auto_update.sh  (wd: ~/backup)
[2]  Stopped                  top  (wd: ~/backup)
[3]  Stopped                  top -u vinu  (wd: ~/backup)
[4]  Stopped                  top -u vinu  (wd: ~/backup)
[5]- Stopped                  top -u vinu  (wd: ~/backup)
[6]+ Stopped                  sleep 300
```

fg

```
vinu@DESKTOP-5K616C3:~$ fg
sleep 300
```

bg

```
vinu@DESKTOP-5K616C3:~$ bg
[5]+ top -u vinu &      (wd: ~/backup)
vinu@DESKTOP-5K616C3:~$
```

Lab 4 : Monitoring System Performance and Resource Usage

Objective:

Learn how to monitor system resources and analyze processes consuming system resources.

Task:

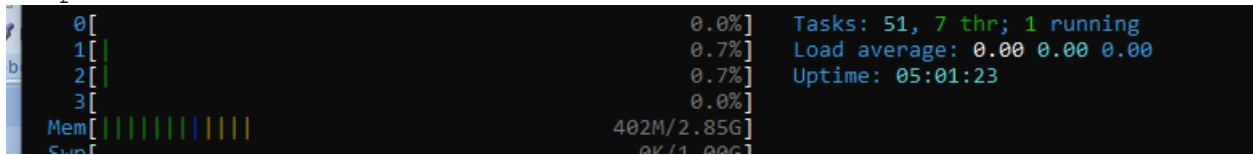
1. Monitor CPU Usage:

Use top or htop to monitor CPU usage in real-time.

top

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	165928	11384	8296	S	0.0	0.4	0:06.08	systemd
2	root	20	0	2776	1920	1796	S	0.0	0.1	0:00.23	init-systemd(Ub
6	root	20	0	2776	132	132	S	0.0	0.0	0:00.01	init
86	root	19	-1	47736	15320	14260	S	0.0	0.5	0:03.39	systemd-journal
114	root	20	0	21964	5864	4568	S	0.0	0.2	0:01.18	systemd-udevd
133	_rpc	20	0	8104	4064	3624	S	0.0	0.1	0:00.06	rpcbind
134	systemd+	20	0	25672	12764	8412	S	0.0	0.4	0:00.61	systemd-resolve
135	systemd+	20	0	89364	6540	5736	S	0.0	0.2	0:00.81	systemd-timesyn

htop



Look for processes consuming high CPU and analyze them.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1	root	20	0	162M	11384	8296	S	0.0	0.4	0:06.08	/sbin/init
2	root	20	0	2776	1920	1796	S	0.0	0.1	0:00.23	/init
6	root	20	0	2776	132	132	S	0.0	0.0	0:00.01	plan9 --control-socket 7 --log-level 4 --server-fd 8
7	root	20	0	2776	132	132	S	0.0	0.0	0:00.00	plan9 --control-socket 7 --log-level 4 --server-fd 8
8	root	20	0	2776	1920	1796	S	0.0	0.1	0:00.00	/init
86	root	19	-1	47736	15320	14260	S	0.0	0.5	0:03.39	/lib/systemd/systemd-journald
114	root	20	0	21964	5864	4568	S	0.0	0.2	0:01.18	/lib/systemd/systemd-udev
133	_rpc	20	0	8104	4064	3624	S	0.0	0.1	0:00.07	/sbin/rpcbind -f -w
134	systemd-r	20	0	25672	12764	8412	S	0.0	0.4	0:00.61	/lib/systemd/systemd-resolved
135	systemd-t	20	0	89364	6540	5736	S	0.0	0.2	0:00.81	/lib/systemd/systemd-timesyncd
143	root	20	0	5128	176	4	S	0.0	0.0	0:00.00	/usr/sbin/blkmapd
144	root	20	0	3108	2208	2048	S	0.0	0.1	0:00.00	/usr/sbin/rpc.idmapd
145	root	20	0	5464	2804	2380	S	0.0	0.1	0:00.00	/usr/sbin/nfsdclld

2. Monitor Memory Usage:

Use free or vmstat to check system memory usage.

vmstat

```
vinu@DESKTOP-5K616C3:~$ vmstat
procs -----memory----- --swap-- -----io---- -system-- -----cpu-----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 0 0 0 2378468 2580 194424 0 0 14 4 2 11 0 0 100 0 0
vinu@DESKTOP-5K616C3:~$
```

Use ps aux --sort=-%mem to find processes using the most memory.

ps aux --sort=-%mem

```
vinu@DESKTOP-5K616C3:~$ ps aux --sort=-%mem
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        873  0.0  2.7 87924 82568 ?        T    00:20   0:01 apt upgrade -y
proxy     289  0.0  0.7 69312 23748 ?        S     Feb23   0:02 (squid-1) --kid squid-1 --foreground -sYC
root       259  0.0  0.7 107160 21244 ?        Ss1   Feb23   0:00 /usr/bin/python3 /usr/share/unattended-upgr
root       212  0.0  0.6 30084 19804 ?        Ss    Feb23   0:00 /usr/bin/python3 /usr/bin/networkd-dispatch
root       276  0.0  0.6 63772 18960 ?        Ss    Feb23   0:00 /usr/sbin/squid --foreground -sYC
root        86  0.0  0.5 47736 15320 ?        S<s   Feb23   0:03 /lib/systemd/systemd-journald
systemd+  134  0.0  0.4 25672 12764 ?        Ss    Feb23   0:00 /lib/systemd/systemd-resolved
root         1  0.0  0.3 165928 11384 ?        Ss    Feb23   0:06 /sbin/init
apt       911  0.0  0.3 19036 9372 ?        T    00:24   0:00 /usr/lib/apt/methods/http
apt       887  0.0  0.3 19036 9296 ?        T    00:20   0:00 /usr/lib/apt/methods/http
```

3. Disk Usage and I/O Monitoring:

Use iotop or dstat to monitor real-time disk I/O usage by processes.

dstat

```
vinu@DESKTOP-5K616C3:~$ sudo apt install dstat
Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontent. It is held by process 873 (apt)
Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontent. It is held by process 873 (apt)
Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontent. It is held by process 873 (apt)
Waiting for cache lock: Could not get lock /var/lib/dpkg/lock-frontent. It is held by process 873 (apt)..
```

4. Check Process Limits:

Use ulimit to check and modify user limits on processes (e.g., maximum number of open files).

```
vinu@DESKTOP-5K616C3:~$ ulimit
unlimited
vinu@DESKTOP-5K616C3:~$
```

```
limited: command not found
vinu@DESKTOP-5K616C3:~$ ulimit -a
real-time non-blocking time (microseconds, -R) unlimited
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
scheduling priority (-e) 0
file size (blocks, -f) unlimited
pending signals (-i) 11627
max locked memory (kbytes, -l) 65536
max memory size (kbytes, -m) unlimited
open files (-n) 1024
pipe size (512 bytes, -p) 8
POSIX message queues (bytes, -q) 819200
real-time priority (-r) 0
stack size (kbytes, -s) 8192
cpu time (seconds, -t) unlimited
max user processes (-u) 11627
virtual memory (kbytes, -v) unlimited
file locks (-x) unlimited
```

1. Viewing Current Limits

To check the current limits for a user, run:

```
bash

ulimit -a
```

This command displays all the limits for the current shell session, including:

- open files (nofile)
- max user processes (nproc)
- stack size (stack)
- memory usage (memlock)
- core dump size (core)
- CPU time (cpu)
- file size (fsize)

```
file locks (-x) unlimited
vinu@DESKTOP-5K616C3:~$ ulimit -n
1024
vinu@DESKTOP-5K616C3:~$ ulimit -1024
-bash: ulimit: -1: invalid option
ulimit: usage: ulimit [-SHabcdefiklmnpqrstuvxPT] [limit]
vinu@DESKTOP-5K616C3:~$
```

Lab 5 : Managing Daemons and Background Services

Objective:

Learn how to manage background services and daemons in Linux.

Task:

1. Start and Stop Services:

Use systemctl to start, stop, and restart system services

```
vinu@DESKTOP-5K616C3:~$ systemctl
UNIT
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-a4f30f41\x2d2314\x2d4ad2\x2d9c50\x2d26dbee134f82-pci2314
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-a4f30f41\x2d2314\x2d4ad2\x2d9c50\x2d26dbee134f82-pci2314
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-a4f30f41\x2d2314\x2d4ad2\x2d9c50\x2d26dbee134f82-pci2314
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-f80b1d83\x2dc44b\x2d4b95\x2da887\x2d1f77235f50c4-net-eth
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-fd1d2cbd\x2dce7c\x2d535c\x2d966b\x2deb5f811c95f0-host0-t
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-fd1d2cbd\x2dce7c\x2d535c\x2d966b\x2deb5f811c95f0-host0-t
sys-devices-LNXSYSTM:00-LNXSYBUS:00-ACPI0004:00-VMBUS:00-fd1d2cbd\x2dce7c\x2d535c\x2d966b\x2deb5f811c95f0-host0-t
```

(e.g., `systemctl start apache2`, `systemctl stop nginx`).

1. Checking Service Status

Before taking action on a service, you may want to check its current status.

```
bash
```

```
systemctl status <service_name>
```

Example:

```
bash
```

```
systemctl status apache2
```

This command provides information about the service's state, including whether it is (running) or inactive (stopped).

```
vinu@DESKTOP-5K616C3:~$ systemctl status
DESKTOP-5K616C3
State: running
Jobs: 0 queued
Failed: 0 units
Since: Sun 2025-02-23 18:33:27 IST; 9h ago
CGroup: /
├─user.slice
│   └─user-1000.slice
│       └─user@1000.service
│           └─init.scope
│               └─441 /lib/systemd/systemd --user
```

3. Stopping a Service

To stop a running service:

```
bash

systemctl stop <service_name>
```

Example:

```
bash

systemctl stop nginx
```

This will stop the Nginx web server.

```
vinu@DESKTOP-5K616C3:~$ systemctl stop nginx
Failed to stop nginx.service: Interactive authentication required.
```

4. Restarting a Service

Restarting is useful when changes have been made to a service's configuration and

```
bash

systemctl restart <service_name>
```

Example:

```
bash

systemctl restart sshd
```

This will restart the SSH service.

2. Enable/Disable Services on Boot:

Use systemctl enable and systemctl disable to manage whether a service starts on boot.

```
vinu@DESKTOP-5K616C3:~$ systemctl enable
Too few arguments.
vinu@DESKTOP-5K616C3:~$ systemctl disable
Too few arguments.
vinu@DESKTOP-5K616C3:~$ _
```

```
vinu@DESKTOP-5K616C3:~$ service starts on boot
starts: unrecognized service
vinu@DESKTOP-5K616C3:~$ _
```

3. Check Service Status:

Use systemctl status to check the status of a service (e.g., systemctl status apache2).

```
├─init.scope
│   ├──441 /lib/systemd/systemd --user
│   └──442 (sd-pam)
├─session-1.scope
│   ├──351 /bin/login -f
│   └──448 -bash
├─init.scope
│   └──1 /sbin/init
├─system.slice
│   ├──systemd-udevd.service
│   │   └──114 /lib/systemd/systemd-udevd
│   ├──cron.service
│   │   └──205 /usr/sbin/cron -f -P
│   ├──nfs-mountd.service
│   │   └──247 /usr/sbin/rpc.mountd
│   ├──polkit.service
│   │   └──1538 /usr/libexec/polkitd --no-debug
│   └─networkd-dispatcher.service
```

```
Unit apache2.service could not be found.
vinu@DESKTOP-5K616C3:~$ systemctl status apache2
Unit apache2.service could not be found.
vinu@DESKTOP-5K616C3:~$
```


4.Managing Logs for Services:

Use journalctl to check logs for systemd services.

```
vinu@DESKTOP-5K616C3:~$ systemd services
```

```
Excess arguments.
```

```
vinu@DESKTOP-5K616C3:~$
```

Filter logs for specific services or time periods to troubleshoot issues.

```
vinu@DESKTOP-5K616C3:~$ journalctl -u nginx -f
```

```
^Z
```

```
[4]+  Stopped                  journalctl -u nginx -f
```

```
vinu@DESKTOP-5K616C3:~$ journalctl -u nginx
```

```
-- No entries --
```

```
vinu@DESKTOP-5K616C3:~$ journalctl -u nginx -f
```

```
^
```

2. Filtering Logs for a Specific Service

To check logs for a specific systemd service:

```
bash
```

```
journalctl -u <service-name>
```

Example:

```
bash
```

```
journalctl -u nginx
```

For real-time logs:

```
bash
```

```
journalctl -u nginx -f
```

3. Filtering Logs by Time

To check logs from the last hour:

```
bash
```

```
journalctl --since "1 hour ago"
```

```
vinu@DESKTOP-5K616C3:~$ journalctl --since "1 hour ago"
Feb 24 03:17:01 DESKTOP-5K616C3 CRON[1339]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Feb 24 03:17:01 DESKTOP-5K616C3 CRON[1340]: (root) CMD ( cd / && run-parts --report /etc/cron.hourly)
```

4. Filtering Logs by Priority (Severity)

Systemd logs have priority levels, where **0** is **emergency** and **7** is **debug**. To filter by

```
bash
```

```
journalctl -p 3 -u nginx
```

This will show only **errors** (priority 3 and above) for the **nginx** service.

Priority levels:

- 0 - Emergency
- 1 - Alert
- 2 - Critical
- 3 - Error
- 4 - Warning
- 5 - Notice
- 6 - Info
- 7 - Debug

6. Checking Logs for Previous Boots

List previous boot sessions:

```
bash
```

```
journalctl --list-boots
```

To view logs from the previous boot:

```
bash
```

```
journalctl -b -1
```

Or a specific boot (e.g., -2 for two boots ago):

```
bash
```

```
journalctl -b -2 -u nginx
```

7. Persisting Logs Across Reboots

By default, logs may be stored in memory only. To ensure logs persist after reboots:

```
bash

sudo mkdir -p /var/log/journal sudo systemctl restart systemd-journald
```

8. Exporting Logs

Save logs to a file:

```
bash

journalctl -u nginx --since "1 hour ago" > nginx_logs.txt
```

9. Clearing Logs

To clear logs:

```
bash

sudo journalctl --vacuum-time=7d
```

(This keeps logs for the last 7 days)

To limit the log size:

Lab 6 : Process Scheduling and Prioritization

Objective:

Learn how to control process priority and manage process scheduling.

Task:

1.Change Process Priority (Nice Value):

Use nice to start a new process with a custom priority level (e.g., nice -n 10 command).

```
or available locally via: info '(coreutils) nice invocation'
vinu@DESKTOP-5K616C3:~$ nice --version
nice (GNU coreutils) 8.32
Copyright (C) 2020 Free Software Foundation, Inc.
License: GPLv3+; GNU GPL version 3 or later (https://gnu.org/licenses/gpl.html)
```

Use renice to change the priority of an already running process by its PID (e.g., renice -n -5 <

```
vinu@DESKTOP-5K616C3:~$ renice -n -5 <
-bash: syntax error near unexpected token `newline'
vinu@DESKTOP-5K616C3:~$
```

2.Scheduling Processes:

Use at to schedule a one-time task (e.g., at 09:00 to run a script).

Crontab Examples

- Run a script every day at 9 AM:

```
bash
```

```
0 9 * * * /home/user/script.sh
```

- Run a script every Monday at 6 PM:

```
bash
```

```
0 18 * * 1 /home/user/script.sh
```

- Run a backup at midnight on the first day of each month:

```
bash
```

```
0 0 1 * * /home/user/backup.sh
```

- Run a script every 5 minutes:

```
bash
```

```
*/5 * * * * /home/user/script.sh
```

```
# Use cron to schedule recurring tasks by adding entries to
/etc/crontab or using crontab -e for user-specific jobs.
```

Crontab Examples

- Run a script every day at 9 AM:

```
bash

0 9 * * * /home/user/script.sh
```

- Run a script every Monday at 6 PM:

```
bash

0 18 * * 1 /home/user/script.sh
```

- Run a backup at midnight on the first day of each month:

```
bash

0 0 1 * * /home/user/backup.sh
```

- Run a script every 5 minutes:

```
bash

*/5 * * * * /home/user/script.sh
```

3. Monitor Process Execution Time:

```
# Use time to measure the execution time of a command or
script.
```

1. Using the `time` Command (Linux/macOS)

The `time` command is the simplest way to measure the execution time of a script

```
bash

time ./script.sh
```

Output Example:

```
sql

real    0m2.345s
user    0m1.456s
sys     0m0.234s
```

- **real**: Total elapsed time (wall clock time).
- **user**: CPU time spent in user mode.
- **sys**: CPU time spent in kernel mode.

Use `\time` to ensure you're using the built-in shell version, or `/usr/bin/time`

Lab 7 : Investigating and Debugging Stuck Processes

Objective:

Learn how to identify and debug processes that are stuck or unresponsive.

Task:

1. Check for Stuck Processes:

Use `ps` or `top` to identify processes that are stuck in a specific state, like `D` (uninterruptible sleep).

1. Identify Stuck Processes Using `ps`

The `ps` command can be used to find processes in a **D (uninterruptible sleep)** state, indicates I/O issues, such as waiting for disk responses.

```
bash
```

```
ps aux | awk '$8 ~ /D/ { print $0 }'
```

- `$8` represents the process state.
- **D** means uninterruptible sleep (often caused by disk I/O).

To display only essential information:

```
bash
```

```
ps -eo pid,ppid,user,state,%cpu,%mem,cmd | grep ' D'
```

This shows:

- **PID** (Process ID)
- **PPID** (Parent Process ID)
- **User** running the process
- **State**
- **CPU & Memory Usage**

2. Use `top` or `htop` to Check for Stuck Processes

Using `top`

Run:

```
bash
```

```
top
```

Press:

- **Shift + M** to sort by memory usage.

-
- **Shift + P** to sort by CPU usage.
 - Look for processes in **D** state.

Using `htop` (if installed)

```
bash
```

```
htop
```

- Press **F4** and type **D** to filter for stuck processes.
-

3. Checking Kernel Logs for Clues

If processes are stuck, check kernel logs:

```
bash
```

```
dmesg | tail -50
```

or

```
bash
```

```
journalctl -k -n 50
```

- Look for disk errors or messages related to **I/O wait**.

4. Find the File/Resource Holding the Process

To see which files a stuck process is accessing, use:

```
bash  
  
lsof -p <PID>
```

Replace <PID> with the stuck process ID.

Check for blocked I/O:

```
bash  
  
strace -p <PID>
```

- This shows system calls the process is waiting on.

5. Force Kill the Process (if necessary)

If the process is stuck and unresponsive:

```
bash  
  
kill -9 <PID>
```

If that doesn't work:

```
bash  
  
echo 1 > /proc/<PID>/oom_score_adj kill -9 <PID>
```

OR forcefully terminate the parent:

```
bash  
  
kill -9 $(ps -o ppid= -p <PID>)
```

Note: If processes keep getting stuck in **D state**, it could indicate:

- A failing disk (`smartctl -a /dev/sdX` to check).
- A bad NFS mount.
- Kernel bugs.

2.Trace Process Execution:

```
# Use strace to trace the system calls made by a process (e.g.,  
strace -p <PID>).
```

1. Basic Usage of strace

1.1 Attaching strace to a Running Process

To trace a running process, use:

```
bash  
  
strace -p <PID>
```

Example:

```
bash  
  
strace -p 1234
```

This attaches `strace` to process ID `1234` and starts logging its system calls.

1.2 Running a Program with strace

Instead of attaching to an existing process, you can start a program with `strace` :

```
bash  
  
strace <command>
```

3. Analyze Process Core Dumps:

Set up core dumps for processes by configuring /etc/security/limits.conf.

```
root@DESKTOP-5K616C3:~# nano /etc/security/limits.conf
root@DESKTOP-5K616C3:~# cat /proc/sys/kernel/core_pattern
|/wsl-capture-crash %t %E %p %s
root@DESKTOP-5K616C3:~#
```

1. Enable Core Dumps in /etc/security/limits.conf

By default, core dumps may be restricted. You need to enable them by configuring the limits.

Modify /etc/security/limits.conf

Add or modify the following lines to allow core dumps:

markdown



```
* soft core unlimited
* hard core unlimited
```

This ensures that all users can generate core dumps without size restrictions.

Check and Modify Other Configurations

- Ensure the core pattern is properly set in /proc/sys/kernel/core_pattern :

bash



```
cat /proc/sys/kernel/core_pattern
```

To store core dumps in a specific directory, modify it:

```
bash

echo "/var/dumps/core.%e.%p" | sudo tee /proc/sys/kernel/core_pattern
```

This stores core dumps as `core.<executable>.<pid>` in `/var/dumps`.

- Enable core dumps for systemd services by setting in `/etc/systemd/system.conf`:

```
ini

DefaultLimitCORE=infinity
```

- Restart services to apply changes.

Use `gdb` to analyze the core dump of a crashed process.

2. Use `gdb` to Analyze Core Dumps

Once a core dump is generated, use `gdb` to analyze it.

Load the Core Dump

```
bash

gdb /path/to/executable /path/to/core
```

For example:

```
bash

gdb /usr/bin/myapp /var/dumps/core.myapp.1234
```



```

root@DESKTOP-5K616C3:~# gdb /path/to/executable /path/to/core
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/path/to/executable: No such file or directory.
/path/to/core: No such file or directory.
(gdb) _

```

4. Terminate or Kill a Stuck Process:

Use kill -9 to forcefully terminate a stuck process.

```

vinu@DESKTOP-5K616C3:~$ kill -9
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
vinu@DESKTOP-5K616C3:~$ kill -l

```

Investigate logs (e.g., /var/log/syslog) for additional clues.

```

vinu@DESKTOP-5K616C3:~$ tail -f /var/log/syslog
Feb 24 08:38:45 DESKTOP-5K616C3 systemd[1]: Condition check resulted in Landscape client daemons being skipped.
Feb 24 08:38:45 DESKTOP-5K616C3 systemd[1]: Starting Cleanup of Temporary Directories...
Feb 24 08:38:45 DESKTOP-5K616C3 systemd[1]: systemd-tmpfiles-clean.service: Deactivated successfully.
Feb 24 08:38:45 DESKTOP-5K616C3 systemd[1]: Finished Cleanup of Temporary Directories.
Feb 24 08:48:43 DESKTOP-5K616C3 kernel: [ 1500.908095] mini_init (117): drop_caches: 1
Feb 24 09:09:14 DESKTOP-5K616C3 squid[272]: Logfile: opening log stdio:/var/spool/squid/netdb.state
Feb 24 09:09:14 DESKTOP-5K616C3 squid[272]: Logfile: closing log stdio:/var/spool/squid/netdb.state
Feb 24 09:09:14 DESKTOP-5K616C3 squid[272]: NETDB state saved; 0 entries, 14 msec
Feb 24 09:17:01 DESKTOP-5K616C3 CRON[1548]: (root) CMD ( cd / && run-parts --report /etc/cron.hourly)
Feb 24 09:40:43 DESKTOP-5K616C3 kernel: [ 4621.266485] mini_init (117): drop_caches: 1

```

Lab 8 : Process Resource Usage and Optimization

Objective:

Optimize processes to improve system performance and reduce resource usage.

Task:

1. Analyze Resource Usage:

Use `ps aux --sort=-%mem` or `top` to find the processes consuming the most memory and CPU.

```
vinu@DESKTOP-5K616C3:~$ ps aux --sort=-%mem
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1886  0.0  1.1 382404 34888 pts/0    Tl   09:39   0:00 gdb /path/to/executable /path/to/core
root      1892  0.0  1.0 380860 30448 pts/0    Tl   09:41   0:00 gdb /path/to/executable /path/to/core
```

2. Optimize Memory Usage:

Identify memory leaks or inefficient memory usage with `valgrind` or `smem`.

1. Identify Memory Leaks with Valgrind

Valgrind's **memcheck** tool helps detect memory leaks and improper memory usage programs.

Install Valgrind

```
bash
```

```
sudo apt install valgrind # Debian/Ubuntu sudo dnf install valgrind # Fedo  
-S valgrind # Arch
```

Run Valgrind

```
bash
```

```
valgrind --leak-check=full --show-leak-kinds=all --track-origins=yes ./you
```

- `--leak-check=full` : Provides detailed memory leak information.
- `--show-leak-kinds=all` : Displays all types of memory leaks.
- `--track-origins=yes` : Helps track uninitialized memory usage.

2. Analyze Memory Usage with smem

`smem` provides detailed per-process memory usage, including shared memory.

Install smem

```
bash
```

```
sudo apt install smem # Debian/Ubuntu sudo yum install smem # Fedora/CentOS
```

View Memory Usage per Process

```
bash
```

```
smem -t
```

- `-t` : Shows total memory usage.

Sort by Resident Set Size (RSS)

```
bash

smem -r -s rss
```

- `-r` : Reverse sort.
- `-s rss` : Sort by RSS, the actual memory used.

Per-User Memory Usage

```
bash

smem -u
```

Additional Tips for Optimization

- Use `top`, `htop`, or `free -m` to monitor real-time memory usage.
- Optimize data structures (use efficient containers in C++ like `std::vector` instead of `std::list`).
- Free memory properly using `free()` (C) or `delete` / `delete[]` (C++).
- Use `mtrace` (glibc tool) to track memory allocations.
- Avoid memory fragmentation by reducing frequent allocations/deallocations.

Would you like help analyzing a specific program's memory usage? 🚀

```
root@DESKTOP-5K616C3:~# smem -r -s rss
PID User      Command                               Swap    USS    PSS    RSS
1886 root      gdb /path/to/executable /pa         0      12448   20921   35344
```

*****THANK YOU*****