Instructions

Assalamoalaikum. Design the scripts below. Explain the code in your own words. Show output. Submit PDF. This is individual (to help everyone develop concepts). It is best to play around with your system first to generate some activity. Upload some data and do the basic Ubuntu commands.

ZUHA AQIB

26106

Task 1

Log Analyzer

Write a Bash that:

- Reads /var/log/syslog.
- Extracts all unique services that generated error messages.
- Counts how many times each service reported an error.
- Stores the top 5 most error-prone services into a CSV file (errors.csv) with the format: *Service,Occurrences*

I first generated some system activity so the error list wouldn't be empty, then wrote a Bash script that scans /var/log/syslog (or /var/log/messages on some systems). It finds lines with "error", extracts the service/program name, counts how often each one failed, and saves the Top-5 into errors.csv as Service,Occurrences.

So first lets generate some activity (errors) so that the CSV is not empty:

```
zaqib@bdacourse:~$ logger -p user.err "Test error A from demo_app"
zaqib@bdacourse:~$ logger -p user.err "Test error B from demo_app"
zaqib@bdacourse:~$ logger -p user.err "Another bad thing from cron"
zaqib@bdacourse:~$
```

Now lets write a script which is a log analyzer:

```
zaqib@bdacourse:~$ nano log_analyzer.sh
zaqib@bdacourse:~$
```

```
  GNU nano 7.2                                                log_analyzer.sh
#!/usr/bin/env bash
set -euo pipefail

LOGFILE="/var/log/syslog"
[ -f "$LOGFILE" ] || LOGFILE="/var/log/messages"

grep -i "error" "$LOGFILE" \
  | awk '{print $5}' \
  | sed 's/\[.*\]://g' \
  | sort \
  | uniq -c \
  | sort -nr \
  | awk '{print $2","$1}' \
  | head -n 5 \
  | { echo "Service,Occurrences"; cat; } > errors.csv

cat errors.csv
```

#!/usr/bin/env bash

set -euo pipefail

LOGFILE="/var/log/syslog"

[-f "$LOGFILE"] || LOGFILE="/var/log/messages"

**grep -i "error" "$LOGFILE" **

**| awk '{print $5}' **

**| sed 's/\[.*\]://g' **

**| sort **

**| uniq -c **

**| sort -nr **

**| awk '{print $2","$1}' **

**| head -n 5 **

| { echo "Service,Occurrences"; cat; } > errors.csv

cat errors.csv

I made a Bash file for the log analyzer and started with the shebang (#!/usr/bin/env bash) plus set -euo pipefail so the script stops on errors or undefined variables. Then I point LOGFILE to /var/log/syslog, and if that file doesn't exist on this machine, I fall back to /var/log/messages. From

there I scan the log for anything that looks like an error using grep -i "error" (case-insensitive), and pipe those lines into awk '{print $5}' to pull the service/program field. Some log lines include a PID like sshd[1234]:, so I clean that tag with sed 's/\[.*\]://g' to remove the [pid]: part and keep just the pure service name. After that I sort, then uniq -c to count how many times each service appears, then sort -nr to put the biggest counts first. I convert the "count then name" format into CSV as Service,Occurrences using awk '{print $2","$1}', grab only the top 5 with head -n 5, and prepend a CSV header by echoing Service,Occurrences before the data. I redirect the whole thing into errors.csv, and finally I cat errors.csv so I can see the result immediately in the terminal.

Lets make it executable:

```
zaqib@bdacourse:~$ chmod +x log_analyzer.sh
zaqib@bdacourse:~$
```

And then lets run it

```
zaqib@bdacourse:~$ ./log_analyzer.sh
grep: /var/log/syslog: Permission denied
zaqib@bdacourse:~$ sudo ./log_analyzer.sh
[sudo] password for zaqib:
Service,Occurrences
type=1400,6915
parser,408
clipboard_event_selection_request:,65
Cannot,64
xrdp_iso_send:,53
zaqib@bdacourse:~$
```

Task 2

Custom Backup System

Write a script that:

- Takes a directory name as input.
- Compresses it into a .tar.gz file with today's date in the filename.
- Before compressing, it must skip files larger than 1 MB and log their names into skipped_files.txt.
- At the end, it should print the total size of the backup.

I wrote backup.sh that takes a directory, skips files bigger than 1 MB (logs those in skipped_files.txt), compresses the rest into backup_YYYY-MM-DD.tar.gz, and prints the backup size at the end.

First lets make backup.sh

```
zaqib@bdacourse:~$ nano backup.sh
zaqib@bdacourse:~$
```

```
  GNU nano 7.2                                                backup.sh *
#!/usr/bin/env bash
set -euo pipefail

if [ $# -ne 1 ]; then
  echo "Usage: $0 <directory>"
  exit 1
fi

DIR="$1"
DATE=$(date +%F)
ARCHIVE="backup_${DATE}.tar.gz"

> skipped_files.txt

find "$DIR" -type f -size +1M > skipped_files.txt

tar --exclude-from=skipped_files.txt -czf "$ARCHIVE" "$DIR"

echo "Backup created: $ARCHIVE"
echo "Skipped files logged in skipped_files.txt"
echo "Total backup size:"
du -sh "$ARCHIVE"
```

#!/usr/bin/env bash

```
set -euo pipefail

if [ $# -ne 1 ]; then
 echo "Usage: $0 <directory>"
 exit 1
fi

DIR="$1"
DATE=$(date +%F)
ARCHIVE="backup_${DATE}.tar.gz"

> skipped_files.txt

find "$DIR" -type f -size +1M > skipped_files.txt

tar --exclude-from=skipped_files.txt -czf "$ARCHIVE" "$DIR"

echo "Backup created: $ARCHIVE"
echo "Skipped files logged in skipped_files.txt"
echo "Total backup size:"
du -sh "$ARCHIVE"
```

For this backup system script, I first open the file with the shebang line and set -euo pipefail to make it safe and strict. Then I add a condition at the start that checks if exactly one argument is given. If not, it prints a usage message like “Usage: ./backup.sh <directory>” and exits. After that, I store the directory name in a variable called DIR, and I also take today’s date using date +%F so I can add it into the backup file name. That creates something like backup_2025-09-15.tar.gz.

Next, I clear or create an empty file named skipped_files.txt. Then I run find on the target directory, looking for files larger than 1 MB. Those file paths are written into skipped_files.txt so I know what got excluded. After that, the actual backup step happens with tar --exclude-from=skipped_files.txt -czf "$ARCHIVE" "$DIR". This means it compresses the directory into a .tar.gz archive but ignores the big files that were listed earlier.

Finally, the script prints out three things: the name of the backup archive it created, a reminder that skipped files are saved in skipped_files.txt, and then it runs du -sh on the archive to show the total size of the backup in a human-readable way. So when I run this script, I see a neat summary of what was backed up, what was skipped, and how big the backup turned out.

Make the file runnable

```
zaqib@bdacourse:~$ chmod +x backup.sh
zaqib@bdacourse:~$
```

Then execute it

```
zaqib@bdacourse:~$ ./backup.sh myfolder
find: ‘myfolder’: No such file or directory
zaqib@bdacourse:~$
```

So first lets make this folder and populate some files in it.

```
zaqib@bdacourse:~$ mkdir myfolder
zaqib@bdacourse:~$ echo "hello world" > myfolder/file1.txt
zaqib@bdacourse:~$ echo "another test" > myfolder/file2.txt
zaqib@bdacourse:~$ dd if=/dev/zero of=myfolder/largefile.bin bs=1M count=2
2+0 records in
2+0 records out
2097152 bytes (2.1 MB, 2.0 MiB) copied, 0.00203052 s, 1.0 GB/s
zaqib@bdacourse:~$
```

```
zaqib@bdacourse:~$ ls myfolder
file1.txt  file2.txt  largefile.bin
zaqib@bdacourse:~$
```

Now lets run the script

```
zaqib@bdacourse:~$ ./backup.sh myfolder
Backup created: backup_2025-09-15.tar.gz
Skipped files logged in skipped_files.txt
Total backup size:
4.0K    backup_2025-09-15.tar.gz
zaqib@bdacourse:~$
```

Lets all see what is in this backup:

```
zaqib@bdacourse:~$ tar -tzf backup_2025-09-15.tar.gz
myfolder/
myfolder/file2.txt
myfolder/file1.txt
zaqib@bdacourse:~$
```

This shows that the bigger file was not included.

Task 3

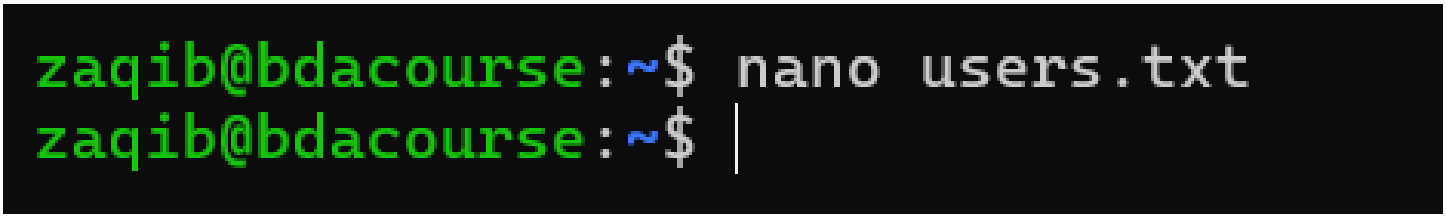
User Management with Constraints:

Write a script that:

- Reads usernames from a text file (users.txt).
- Creates each user with:
    - A home directory.
    - A default password (ChangeMe123).
- Then, disables login for users whose names start with “temp”.

I created a users.txt with a few names, including two starting with temp.... The script creates each user with a home dir and a default password ChangeMe123. If the name starts with temp, it disables login immediately.

Ok so first lets make this users.txt file with username in it



Wait lets make one or two with temp

zuha

zara

tempmahnoor

zehra

tempfarah

Now lets make a script to read each user name and make a account and remove temp ones

```
zaqib@bdacourse:~$ nano user_manage.sh
zaqib@bdacourse:~$
```

```
  GNU nano 7.2                                                  user_manage.sh *
#!/usr/bin/env bash
set -euo pipefail

FILE="users.txt"
if [ ! -f "$FILE" ]; then
  echo "users.txt not found"
  exit 1
fi

while IFS= read -r u; do
  [ -z "$u" ] && continue
  [[ "$u" =~ ^# ]] && continue
  if id -u "$u" >/dev/null 2>&1; then
    continue
  fi
  useradd -m -s /bin/bash "$u"
  echo "$u:ChangeMe123" | chpasswd
  if [[ "$u" == temp* ]]; then
    usermod -L "$u"
    usermod -s /usr/sbin/nologin "$u"
  fi
done < "$FILE"
```

#!/usr/bin/env bash

set -euo pipefail

FILE="users.txt"

if [! -f "$FILE"]; then

 echo "users.txt not found"

 exit 1

fi

while IFS= read -r u; do

 [-z "$u"] && continue

```
[[ "$u" =~ ^# ]] && continue
if id -u "$u" >/dev/null 2>&1; then
  continue
fi
useradd -m -s /bin/bash "$u"
echo "$u:ChangeMe123" | chpasswd
if [[ "$u" == temp* ]]; then
  usermod -L "$u"
  usermod -s /usr/sbin/nologin "$u"
fi
done < "$FILE"
```

For the user management script, I start the file with the shebang and set -euo pipefail for safety. Then I set a variable FILE="users.txt", because that's where the list of usernames will come from. Right after that I check if the file actually exists with [! -f "$FILE"]. If it doesn't, the script prints "users.txt not found" and exits.

The main part is a while loop that reads each line of the file into the variable u. Inside the loop, I skip empty lines ([-z "$u"] && continue) and also skip any lines that start with #, since those are comments. Then I check if the user already exists on the system with id -u "$u". If it does, I just continue without doing anything.

If the user doesn't exist, I create them with useradd -m -s /bin/bash "$u", which gives them a home directory and sets their shell to bash. After that, I immediately set a default password by piping "$u:ChangeMe123" into chpasswd. This way each account is usable right away.

There's also a condition for temporary users: if the username starts with "temp", then I run usermod -L to lock their account and also usermod -s /usr/sbin/nologin so they cannot log into the shell. This enforces the constraint given in the assignment.

At the end, the loop finishes after processing all the lines in users.txt. So the result is: real users get created with home directories and default passwords, while any temporary ones are immediately disabled.

Make it executable

```
zaqib@bdacourse:~$ chmod +x user_manage.sh
zaqib@bdacourse:~$
```

Run WITH SUDO

```
zaqib@bdacourse:~$ sudo ./user_manage.sh
[sudo] password for zaqib:
zaqib@bdacourse:~$
```

Lets check it

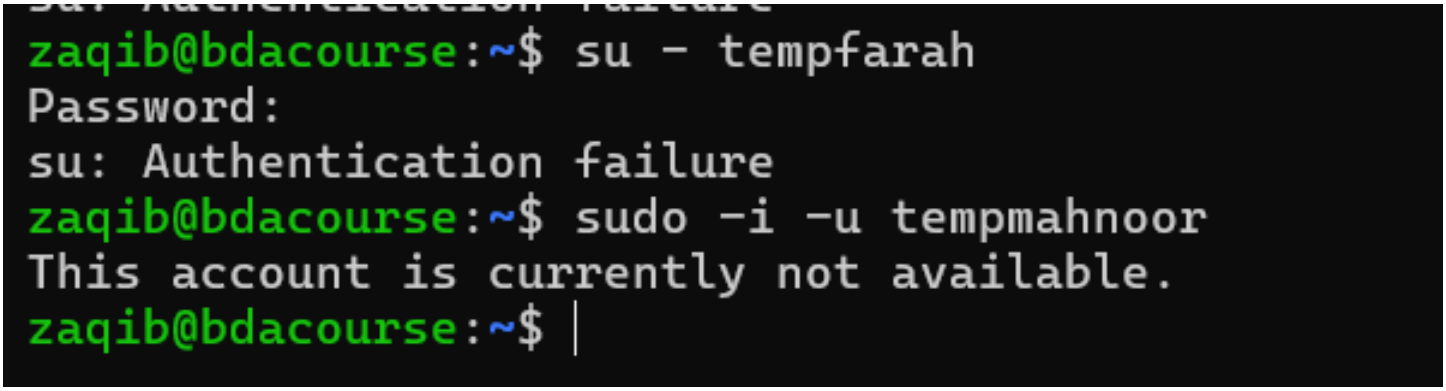
```
zaqib@bdacourse:~$ getent passwd zuha zara tempmahnoor zehra tempfarah
zuha:x:1008:1008::/home/zuha:/bin/bash
zara:x:1009:1009::/home/zara:/bin/bash
tempmahnoor:x:1010:1010::/home/tempmahnoor:/usr/sbin/nologin
zehra:x:1011:1011::/home/zehra:/bin/bash
tempfarah:x:1012:1012::/home/tempfarah:/usr/sbin/nologin
zaqib@bdacourse:~$
```

```
zaqib@bdacourse:~$ ls -ld /home/zuha /home/zara /home/tempmahnoor /home/zehra /home/tempfarah
drwxr-x--- 2 tempfarah   tempfarah   4096 Sep 15 00:43 /home/tempfarah
drwxr-x--- 2 tempmahnoor tempmahnoor 4096 Sep 15 00:43 /home/tempmahnoor
drwxr-x--- 2 zara        zara        4096 Sep 15 00:43 /home/zara
drwxr-x--- 2 zehra       zehra       4096 Sep 15 00:43 /home/zehra
drwxr-x--- 2 zuha        zuha        4096 Sep 15 00:43 /home/zuha
zaqib@bdacourse:~$
```

```
zaqib@bdacourse:~$ sudo passwd -S tempmahnoor
tempmahnoor L 2025-09-14 0 99999 7 -1
zaqib@bdacourse:~$ sudo passwd -S tempfarah
tempfarah L 2025-09-14 0 99999 7 -1
zaqib@bdacourse:~$
```

```
zaqib@bdacourse:~$ su - zuha
Password:
zuha@bdacourse:~$ whoami
zuha
zuha@bdacourse:~$ exit
logout
zaqib@bdacourse:~$
```

All passwords are set as ChangeMe123



This shows that the users zuha, zara, zehra were created successfully but the temp users tempfarah and tempmahnoor were not creared and are blocked.

Task 4

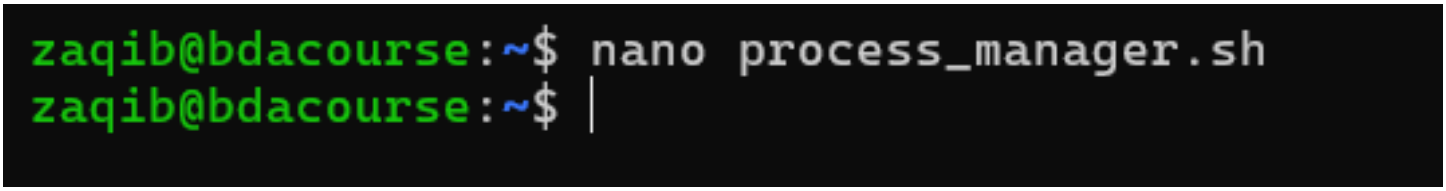
Process Monitor

Write a script that:

- Monitors processes every 5 seconds.
- If any process uses more than 5% CPU or 50 MB memory, log its PID, name, CPU%, and memory% into alerts.log.
- If the same process is logged 3 times consecutively, automatically kill it.

I built a monitor that checks every 5 seconds. If any process uses >5% CPU or >50 MB RAM, it logs a line to alert.log. If the same PID is flagged three checks in a row, it kills it (TERM then KILL if needed).

First task is to make the script



```
  GNU nano 7.2                                                process_manager.sh *
#!/usr/bin/env bash
# Monitors processes every 5 seconds.
# If any process uses >5% CPU OR >50 MB memory, log it to alert.log.
# If the SAME PID is logged 3 times consecutively, kill it automatically.

set -uo pipefail

# ---- Config ----
INTERVAL_SEC=5
CPU_THRESHOLD=5.0          # percent
MEM_THRESHOLD_MB=50        # megabytes
LOG_FILE="alert.log"

# Derived
RSS_KB_THRESHOLD=$(( MEM_THRESHOLD_MB * 1024 ))

# ---- State (per-PID consecutive hit counts) ----
declare -A HIT_COUNTS      # HIT_COUNTS[PID] = consecutive hits
declare -A FLAGGED_THIS_TICK

# ---- Helpers ----
timestamp() { date +"%Y-%m-%d %H:%M:%S"; }

log_line() {
  # $1=message
  printf "%s %s\n" "$(timestamp)" "$1" >> "$LOG_FILE"
}

# Header once per run (keeps appending safely)
if [[ ! -f "$LOG_FILE" ]]; then
  log_line "START $(timestamp) FORMAT=v1"
  log_line "FIELDS: ts PID=<pid> NAME=<comm> CPU%=<cpu> MEM%=<mem> RSS_KB=<rss>"
fi

# Graceful exit
trap 'echo; log_line "STOP $(timestamp)"; exit 0' INT TERM

while :; do
  # Clear "seen this tick" flags
  unset FLAGGED_THIS_TICK
  declare -A FLAGGED_THIS_TICK
```

```bash
  # Clear "seen this tick" flags
  unset FLAGGED_THIS_TICK
  declare -A FLAGGED_THIS_TICK

  # Grab current snapshot:
  # pid, command, %cpu, %mem, rss(KB)
  #
  # Filter in awk using thresholds so the loop only sees "violators".
  # Notes:
  # - Using "comm=" keeps just the executable name (no args), which is nice for logging.
  # - %mem is included because your spec asks to log memory%.
  while IFS= read -r line; do
    # Parse fields
    # shellcheck disable=SC2086
    read -r pid comm cpu mem rss <<<"$line"

    # Log the alert
    log_line "PID=$pid NAME=$comm CPU%=$cpu MEM%=$mem RSS_KB=$rss"

    # Update consecutive count
    current="${HIT_COUNTS[$pid]:-0}"
    next=$(( current + 1 ))
    HIT_COUNTS[$pid]=$next
    FLAGGED_THIS_TICK[$pid]=1

    # Kill if 3 consecutive hits
    if (( next >= 3 )); then
      # Attempt a graceful kill first; if it survives, SIGKILL
      if kill -TERM "$pid" 2>/dev/null; then
        sleep 0.5
      fi
      if kill -0 "$pid" 2>/dev/null; then
        kill -KILL "$pid" 2>/dev/null || true
      fi
      log_line "ACTION=KILLED PID=$pid NAME=$comm after $next consecutive alerts"
      # Reset counter so we don't keep trying to kill
      HIT_COUNTS[$pid]=0
    fi
  done < <(
    # ps output; fields: PID, COMM, %CPU, %MEM, RSS(KB)
    # Then awk filters by thresholds; pass bash vars via -v
```

```
    # ps output, fields: PID, COMM, %CPU, %MEM, RSS(KB)
    # Then awk filters by thresholds; pass bash vars via -v
    ps -eo pid=,comm=,%cpu=,%mem=,rss= \
    | awk -v c="$CPU_THRESHOLD" -v r="$RSS_KB_THRESHOLD" '
        {
          pid=$1; comm=$2; cpu=$3; mem=$4; rss=$5;
          # Filter: cpu > c OR rss > r
          if (cpu+0 > c || rss+0 > r) {
            # Print as a single space-separated line for the while-read above
            printf "%s %s %s %s %s\n", pid, comm, cpu, mem, rss
          }
        }'
  )

  # Reset consecutive counts for any PID that did NOT violate this tick
  for pid in "${!HIT_COUNTS[@]}"; do
    if [[ -z "${FLAGGED_THIS_TICK[$pid]+x}" ]]; then
      HIT_COUNTS[$pid]=0
    fi
  done

  sleep "$INTERVAL_SEC"
done
```

#!/usr/bin/env bash

Monitors processes every 5 seconds.

If any process uses >5% CPU OR >50 MB memory, log it to alert.log.

If the SAME PID is logged 3 times consecutively, kill it automatically.

set -uo pipefail

---- Config ----

INTERVAL_SEC=5

CPU_THRESHOLD=5.0 # percent

MEM_THRESHOLD_MB=50 # megabytes

LOG_FILE="alert.log"

Derived

RSS_KB_THRESHOLD=$((MEM_THRESHOLD_MB * 1024))

```
# ---- State (per-PID consecutive hit counts) ----
declare -A HIT_COUNTS     # HIT_COUNTS[PID] = consecutive hits
declare -A FLAGGED_THIS_TICK

# ---- Helpers ----
timestamp() { date +"%Y-%m-%d %H:%M:%S"; }

log_line() {
 # $1=message
 printf "%s %s\n" "$(timestamp)" "$1" >> "$LOG_FILE"
}

# Header once per run (keeps appending safely)
if [[ ! -f "$LOG_FILE" ]]; then
 log_line "START $(timestamp) FORMAT=v1"
 log_line "FIELDS: ts PID=<pid> NAME=<comm> CPU%=<cpu> MEM%=<mem> RSS_KB=<rss>"
fi

# Graceful exit
trap 'echo; log_line "STOP $(timestamp)"; exit 0' INT TERM

while :; do
 # Clear "seen this tick" flags
 unset FLAGGED_THIS_TICK
 declare -A FLAGGED_THIS_TICK

 # Grab current snapshot:
 # pid, command, %cpu, %mem, rss(KB)
 #
```

```
# Filter in awk using thresholds so the loop only sees "violators".
# Notes:
# - Using "comm=" keeps just the executable name (no args), which is nice for logging.
# - %mem is included because your spec asks to log memory%.
while IFS= read -r line; do
  # Parse fields
  # shellcheck disable=SC2086
  read -r pid comm cpu mem rss <<<"$line"

  # Log the alert
  log_line "PID=$pid NAME=$comm CPU%=$cpu MEM%=$mem RSS_KB=$rss"

  # Update consecutive count
  current="${HIT_COUNTS[$pid]:-0}"
  next=$(( current + 1 ))
  HIT_COUNTS[$pid]=$next
  FLAGGED_THIS_TICK[$pid]=1

  # Kill if 3 consecutive hits
  if (( next >= 3 )); then
    # Attempt a graceful kill first; if it survives, SIGKILL
    if kill -TERM "$pid" 2>/dev/null; then
      sleep 0.5
    fi
    if kill -0 "$pid" 2>/dev/null; then
      kill -KILL "$pid" 2>/dev/null || true
    fi
    log_line "ACTION=KILLED PID=$pid NAME=$comm after $next consecutive alerts"
    # Reset counter so we don't keep trying to kill
```

```
    HIT_COUNTS[$pid]=0
  fi
done < <(
  # ps output; fields: PID, COMM, %CPU, %MEM, RSS(KB)
  # Then awk filters by thresholds; pass bash vars via -v
  ps -eo pid=,comm=,%cpu=,%mem=,rss= \
  | awk -v c="$CPU_THRESHOLD" -v r="$RSS_KB_THRESHOLD" '
    {
      pid=$1; comm=$2; cpu=$3; mem=$4; rss=$5;
      # Filter: cpu > c OR rss > r
      if (cpu+0 > c || rss+0 > r) {
        # Print as a single space-separated line for the while-read above
        printf "%s %s %s %s %s\n", pid, comm, cpu, mem, rss
      }
    }'
)

# Reset consecutive counts for any PID that did NOT violate this tick
for pid in "${!HIT_COUNTS[@]}"; do
  if [[ -z "${FLAGGED_THIS_TICK[$pid]+x}" ]]; then
    HIT_COUNTS[$pid]=0
  fi
done

sleep "$INTERVAL_SEC"
done
```

I'm using set -uo pipefail for safer behavior. Then I defined a small config section: the check interval is 5 seconds, CPU threshold is 5% and memory threshold is 50 MB, and all alerts go into alert.log. I also convert the 50 MB into KB because ps reports RSS in kilobytes. Next, I set up two associative arrays to remember state across scans: one to track how many consecutive times a particular PID

has crossed the threshold, and one to mark which PIDs were flagged during the current tick. I made a tiny timestamp() function and a log_line() helper to keep the log format consistent, and on the very first run I write a header into alert.log. I also added a trap so if I stop the script with Ctrl-C, it prints a clean STOP line in the log.

The main loop runs forever with while :; do ... done. At each tick I clear the "seen this tick" flags, then grab a live snapshot of processes using ps -eo pid=,comm=,%cpu=,%mem=,rss=. I pipe that into awk and only let through the rows where CPU is over 5% or RSS is over the 50 MB threshold. For every violating line, I parse out pid, the command name comm (just the executable, no args), CPU%, MEM%, and RSS(KB). I append a structured log line to alert.log, bump the consecutive hit counter for that PID, and if a process crosses the threshold three times in a row, I first try a graceful kill -TERM and, if it's still alive, I follow with a kill -KILL. I log that action and reset the counter so I don't spam kills. After processing the snapshot, I loop over any PIDs I'm tracking: if a PID wasn't flagged this tick, I reset its counter to zero—this way only truly consecutive breaches count. Finally I sleep for 5 seconds and repeat. Overall, when I run this file, it keeps writing timestamped alerts to alert.log and automatically handles any stubborn process that keeps hogging CPU or memory three intervals back-to-back.

Ok and then lets make the script runnable

```
zaqib@bdacourse:~$ chmod +x process_manager.sh
zaqib@bdacourse:~$
```

Now lets run it by starting a new process as well

```
zaqib@bdacourse:~$ nohup ./process_manager.sh >/dev/null 2>&1 &
[1] 1762979
```

And then lets see the alert.log

```
zaqib@bdacourse:~$ tail -f alert.log
2025-09-15 11:01:21 START 2025-09-15 11:01:21 FORMAT=v1
2025-09-15 11:01:21 FIELDS: ts PID=<pid> NAME=<comm> CPU%=<cpu> MEM%=<mem> RSS_KB=<rss>
2025-09-15 11:01:21 PID=730619 NAME=Xorg CPU%=0.0 MEM%=1.3 RSS_KB=106284
2025-09-15 11:01:21 PID=730647 NAME=unity-greeter CPU%=0.0 MEM%=1.3 RSS_KB=107752
2025-09-15 11:01:26 PID=730619 NAME=Xorg CPU%=0.0 MEM%=1.3 RSS_KB=106284
2025-09-15 11:01:26 PID=730647 NAME=unity-greeter CPU%=0.0 MEM%=1.3 RSS_KB=107752
2025-09-15 11:01:31 PID=730619 NAME=Xorg CPU%=0.0 MEM%=1.3 RSS_KB=106284
2025-09-15 11:01:31 ACTION=KILLED PID=730619 NAME=Xorg after 3 consecutive alerts
2025-09-15 11:01:31 PID=730647 NAME=unity-greeter CPU%=0.0 MEM%=1.3 RSS_KB=107752
2025-09-15 11:01:31 ACTION=KILLED PID=730647 NAME=unity-greeter after 3 consecutive alerts
2025-09-15 11:01:36 PID=730619 NAME=Xorg CPU%=0.0 MEM%=1.3 RSS_KB=106284
2025-09-15 11:01:36 PID=730647 NAME=unity-greeter CPU%=0.0 MEM%=1.3 RSS_KB=107752
2025-09-15 11:01:41 PID=730619 NAME=Xorg CPU%=0.0 MEM%=1.3 RSS_KB=106284
2025-09-15 11:01:41 PID=730647 NAME=unity-greeter CPU%=0.0 MEM%=1.3 RSS_KB=107752
2025-09-15 11:01:46 PID=730619 NAME=Xorg CPU%=0.0 MEM%=1.3 RSS_KB=106284
2025-09-15 11:01:46 ACTION=KILLED PID=730619 NAME=Xorg after 3 consecutive alerts
2025-09-15 11:01:46 PID=730647 NAME=unity-greeter CPU%=0.0 MEM%=1.3 RSS_KB=107752
2025-09-15 11:01:46 ACTION=KILLED PID=730647 NAME=unity-greeter after 3 consecutive alerts
2025-09-15 11:01:46 PID=1766711 NAME=ps CPU%=100 MEM%=0.0 RSS_KB=4244
^C
zaqib@bdacourse:~$
```

So we can see that its killed.

Xorg is the program that makes the Ubuntu desktop and graphics actually show up on screen.

Unity Greeter is the login screen you see before entering your username and password.

Task 5

Parallel Download Manager

Write a script that:

- Reads a list of URLs from urls.txt.
- Downloads them in parallel (max 3 at a time).
- Logs failed downloads separately.

I put some sample URLs in urls.txt. The script downloads with wget into a downloads/ folder, runs up to 3 downloads at the same time, and logs any failed URLs once into failed.log.

First the file directory, lets make urls.txt

```
zaqib@bdacourse:~$ cd assignments/assignment_01
zaqib@bdacourse:~/assignments/assignment_01$ mkdir -p parallel-dl && cd parallel-dl
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ cat > urls.txt <<'EOF'
https://example-files.online-convert.com/video/mp4/example.mp4
https://example-files.online-convert.com/image/jpeg/example.jpg
https://example-files.online-convert.com/document/pdf/example.pdf
# lines starting with # are ignored
EOF
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$
```

https://example-files.online-convert.com/video/mp4/example.mp4

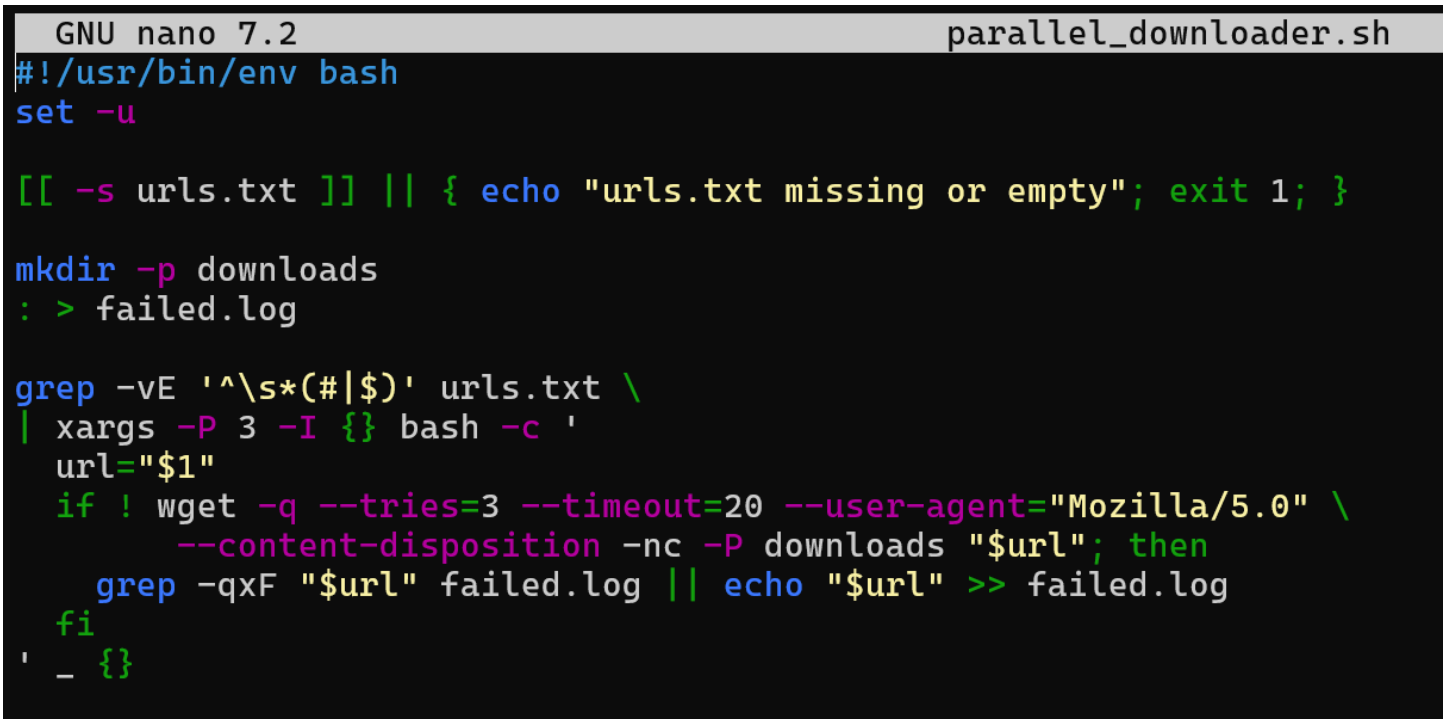
https://example-files.online-convert.com/image/jpeg/example.jpg

https://example-files.online-convert.com/document/pdf/example.pdf

 # lines starting with # are ignored

So then lets make the script

```
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ nano parallel_downloader.sh
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$
```



#!/usr/bin/env bash

set -u

[[-s urls.txt]] || { echo "urls.txt missing or empty"; exit 1; }

mkdir -p downloads

: > failed.log

grep -vE '^\s*(#|$)' urls.txt \

| xargs -P 3 -I {} bash -c '

url="$1"

if ! wget -q --tries=3 --timeout=20 --user-agent="Mozilla/5.0" \

--content-disposition -nc -P downloads "$url"; then

grep -qxF "$url" failed.log || echo "$url" >> failed.log

fi

' _ {}

I saved this as my parallel download script and kept it strict with set -u so missing variables throw errors. First thing I do is sanity-check urls.txt: [[-s urls.txt]] makes sure the file exists and isn't empty;

otherwise I print a friendly error and exit. Then I prep my workspace—mkdir -p downloads creates the output folder if it's not there already, and : > failed.log truncates/creates the failures log so I start fresh each run. Next, I clean the URL list by stripping out blank lines and comments using grep -vE '^\s*(#|$)' urls.txt. That cleaned stream is piped into xargs with -P 3 so at most three downloads run in parallel. For each URL, I run a tiny Bash snippet: I call wget quietly with 3 retries and a 20-second timeout, set a normal browser user agent, preserve the server's filename when possible (--content-disposition), don't overwrite existing files (-nc), and save to the downloads/ folder. If a download fails, I append the URL to failed.log—but only once—by checking with grep -qxF before writing. So the overall flow is: verify input, prep folders/logs, fan out up to three concurrent wgets, and keep a clean list of anything that didn't make it.

Make it runnable

```
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ chmod +x parallel_downloader.sh
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$
```

Run

```
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ ./parallel_downloader.sh
xargs: warning: options --max-args and --replace/-I/-i are mutually exclusive, ignoring previous --max-args value
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$
```

Test it

```
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ ls -lh downloads
total 13M
-rw-rw-r-- 1 zaqib zaqib  13M Jan 25  2022 example.mp4
-rw-rw-r-- 1 zaqib zaqib 150K Jan 25  2022 example.pdf
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ cat failed.log
https://example-files.online-convert.com/image/jpeg/example.jpg
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$
```

So we see that example.jpg has failed – so this is done using docker curl image I

```
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ curl -I https://example-files.online-convert.com/imag
e/jpeg/example.jpg
HTTP/2 404
date: Mon, 15 Sep 2025 16:40:30 GMT
content-type: text/html; charset=UTF-8
content-length: 146
cache-control: public
server: cloudflare
x-server: web5
cf-cache-status: HIT
strict-transport-security: max-age=15552000; includeSubDomains
vary: accept-encoding
cf-ray: 97f990f40bb4ff8f-SIN
```

done

Task 6

Customized ls command:

Write a script named my_ls.sh that:

- Prints files in a directory with:
    - File name
    - Size in KB
    - Owner
    - Last modified time

and sorts it by size (descending).

I wrote my_ls.sh which prints a neat table: file name, size in KB, owner, and last modified time. It sorts by size (largest first). I tested it both with no args (current dir) and on a specific path.

Wrote the script

```
zaqib@bdacourse:~/assignments/assignment_01/parallel-dl$ cd ~/assignments/assignment_01
zaqib@bdacourse:~/assignments/assignment_01$ nano my_ls.sh
zaqib@bdacourse:~/assignments/assignment_01$
```

```
  GNU nano 7.2                                          my_ls.sh *
#!/usr/bin/env bash
set -euo pipefail

dir="${1:-.}"

printf "%-40s %10s  %-10s  %s\n" "File" "Size(KB)" "Owner" "Modified"

# list files (top level), get bytes|owner|time|path, sort by bytes desc, then format
while IFS= read -r -d '' f; do
  stat -c "%s|%U|%y|%n" "$f"
done < <(find "$dir" -maxdepth 1 -type f -print0) \
| sort -t'|' -nrk1,1 \
| awk -F'|' '{
    kb=$1/1024;
    owner=$2;
    split($3, t, " ");
    gsub(/\..*/, "", t[2]);
    fname=$4; sub(".*/","",fname);
    printf "%-40s %10.1f KB  %-10s  %s %s\n", fname, kb, owner, t[1], t[2];
}'
```

#!/usr/bin/env bash

```
set -euo pipefail

dir="${1:-.}"

printf "%-40s %10s  %-10s  %s\n" "File" "Size(KB)" "Owner" "Modified"

# list files (top level), get bytes|owner|time|path, sort by bytes desc, then format
while IFS= read -r -d '' f; do
  stat -c "%s|%U|%y|%n" "$f"
done < <(find "$dir" -maxdepth 1 -type f -print0) \
| sort -t'|' -nrk1,1 \
| awk -F'|' '{
   kb=$1/1024;
   owner=$2;
   split($3, t, " ");
   gsub(/\..*/, "", t[2]);
   fname=$4; sub(".*/","",fname);
   printf "%-40s %10.1f KB  %-10s  %s %s\n", fname, kb, owner, t[1], t[2];
}'
```

I made a custom ls called my_ls.sh that prints a neat, sorted table. I start with the shebang and set -euo pipefail for safety, then accept an optional directory argument (defaults to the current folder with dir="${1:-.}"). I print a fixed header using printf with column widths so everything lines up (File, Size(KB), Owner, Modified). To list files, I use find "$dir" -maxdepth 1 -type f -print0 so it only looks at the top level and also handles filenames with spaces; I read those paths safely with while IFS= read -r -d '' f. For each file I call stat -c "%s|%U|%y|%n" to output a single pipe-separated line containing bytes, owner, full timestamp, and the path. I then sort the stream by the first field (bytes) in descending order with sort -t'|' -nrk1,1 so the largest files show first. Finally, I run an awk formatter that converts bytes to KB with one decimal, strips the milliseconds from the timestamp, extracts just the filename (not the full path), and prints each row with aligned columns via printf. Net result: when I run ./my_ls.sh or ./my_ls.sh <some/dir>, I get a clean table of files sorted by size (largest on top) with size in KB, the owner, and the last modified date/time.

Made it runnable

```
zaqib@bdacourse:~/assignments/assignment_01$ chmod +x my_ls.sh
zaqib@bdacourse:~/assignments/assignment_01$
```

Run it

```
zaqib@bdacourse:~/assignments/assignment_01$ ./my_ls.sh
File                                    Size(KB)  Owner       Modified
my_ls.sh                                     0.5 KB  zaqib       2025-09-15 21:54:49
zaqib@bdacourse:~/assignments/assignment_01$
```

```
zaqib@bdacourse:~/assignments/assignment_01$ ./my_ls.sh ~/assignments/assignment_01/parallel-dl/downloads
File                                   Size(KB)  Owner       Modified
example.mp4                             12645.4 KB  zaqib       2022-01-25 17:16:29
example.pdf                               149.8 KB  zaqib       2022-01-25 17:16:15
zaqib@bdacourse:~/assignments/assignment_01$
```

As we can see it has listed files from largest to smallest