

# Experiment No. 7

## Shell Programming – Processes

### Aim

To study processes, their states, hierarchy, and management in Linux.

### Requirements

Linux OS

Terminal access

### Concept

1. Processes: A running program with a unique PID.
2. Process States: Running, Ready, Waiting, Stopped, Terminated.
3. Process Hierarchy: Parent–child structure, starting from init/systemd.
4. Killing Processes: Ending unwanted processes to free resources.
5. Process Prioritisation: Adjusting CPU time using priority (nice values).
6. Scheduling Processes: Decides order of execution (FCFS, Round Robin, Priority).

### Procedure

Observe process IDs and states.

Check hierarchy of parent and child processes.

Practice killing, changing priority, and scheduling.

## LAB EXERCISES

### EXERCISE 1 – CHECK IF FILE EXISTS

```
#!/bin/bash

echo "Enter filename: "

read file

if [ -e "$file" ]
then

    echo "File exists. Contents are:"

    cat "$file"

else

    echo "File does not exist."

    echo "Do you want to create it? (y/n)"

    read choice

    if [ "$choice" = "y" ]; then

        touch "$file"

        echo "File $file created."

    fi

fi
```

### EXERCISE 2 - **Print Numbers from 1 to 10**

```
#!/bin/bash

for i in {1..10}

do

    echo $i

done
```

### **New Concept:**

- {1..10} → brace expansion to generate sequence

### EXERCISE 3 - **Count Lines, Words, and Characters**

```
fi#!/bin/bash

if [ $# -eq 0 ]

then
```

```

    echo "Usage: $0 filename"

    exit 1
fi

file=$1

if [ -e "$file" ]
then
    echo "Lines: $(wc -l < $file)"
    echo "Words: $(wc -w < $file)"
    echo "Characters: $(wc -m < $file)"
else
    echo "File not found!"
fi

```

## New Commands:

- \$# → number of command line arguments.
- \$1 → first argument.
- wc → word count utility:
  - wc -l → count lines.
  - wc -w → count words.
  - wc -m → count characters.

## EXERCISE 4 - Factorial Using Function

```

#!/bin/bash

factorial() {
    num=$1
    fact=1
    while [ $num -gt 1 ]
    do
        fact=$((fact * num))
    done
}

```

```
    num=$((num - 1))  
  
done  
  
echo $fact  
}
```

```
echo "Factorial of 5 is: ${factorial 5}"  
echo "Factorial of 7 is: ${factorial 7}"  
echo "Factorial of 10 is: ${factorial 10}"
```

### **New Concepts:**

- Function definition in bash:

**function\_name()** { commands }

- \$1 → function argument.
- \$(command) → command substitution (returns output).

## OUTPUT OF EXERCISE 1,2,3,4

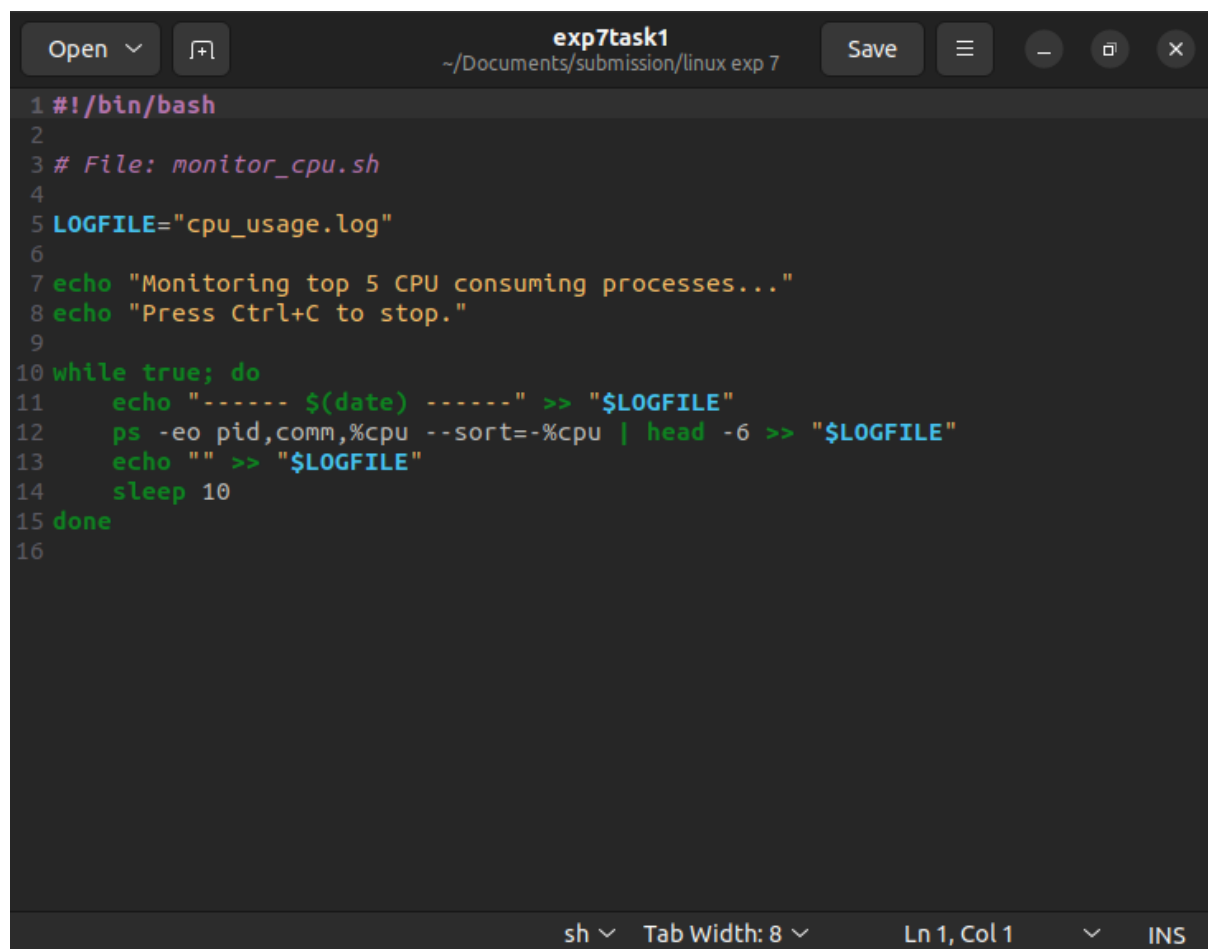
```
pavani@UBUNTU: ~/Documents/linux exp 7
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7i
Enter filename:
file.txt
file does not exist.
Do you want to create it? (y/n)
y
file file.txt created.
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7ii
1
2
3
4
5
6
7
8
9
10
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7iii
UsGE: ./exp7iii FILENAME
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7iv
Factorial of 5 is: 120
Factorial of 7 is: 5040
Factorial of 10 is: 3628800
pavani@UBUNTU:~/Documents/linux exp 7$
```

## LAB TASKS

TASK I. Write a script that monitors the top 5 processes consuming the most CPU and logs them into a file every 10 seconds.

Explanation:

This script continuously monitors the system and finds out which processes are consuming the highest amount of CPU resources. It lists the top 5 CPU-hungry processes and stores their details into a log file. The script repeats this action every 10 seconds. This task is important for system performance analysis and identifying programs that use too much CPU.



```
exp7task1
~/Documents/submission/linux exp 7
Open Save
1 #!/bin/bash
2
3 # File: monitor_cpu.sh
4
5 LOGFILE="cpu_usage.log"
6
7 echo "Monitoring top 5 CPU consuming processes..."
8 echo "Press Ctrl+C to stop."
9
10 while true; do
11     echo "----- $(date) -----" >> "$LOGFILE"
12     ps -eo pid,comm,%cpu --sort=-%cpu | head -6 >> "$LOGFILE"
13     echo "" >> "$LOGFILE"
14     sleep 10
15 done
16
sh Tab Width: 8 Ln 1, Col 1 INS
```

## Output

```
pavani@UBUNTU: ~/Documents/linux exp 7
pavani@UBUNTU:~/Documents/linux exp 7$ vim exp7task1
pavani@UBUNTU:~/Documents/linux exp 7$ chmod +x exp7task1
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7task1
Monitoring top 5 CPU consuming processes...
Press Ctrl+C to stop.

pavani@UBUNTU:~/Documents/linux exp 7$ vim exp7taski
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7taski
bash: ./exp7taski: Permission denied
pavani@UBUNTU:~/Documents/linux exp 7$ chmod +x exp7taski
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7taski
Enter PID: ps aux
error: process ID list syntax error

Usage:
ps [options]

Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.
```

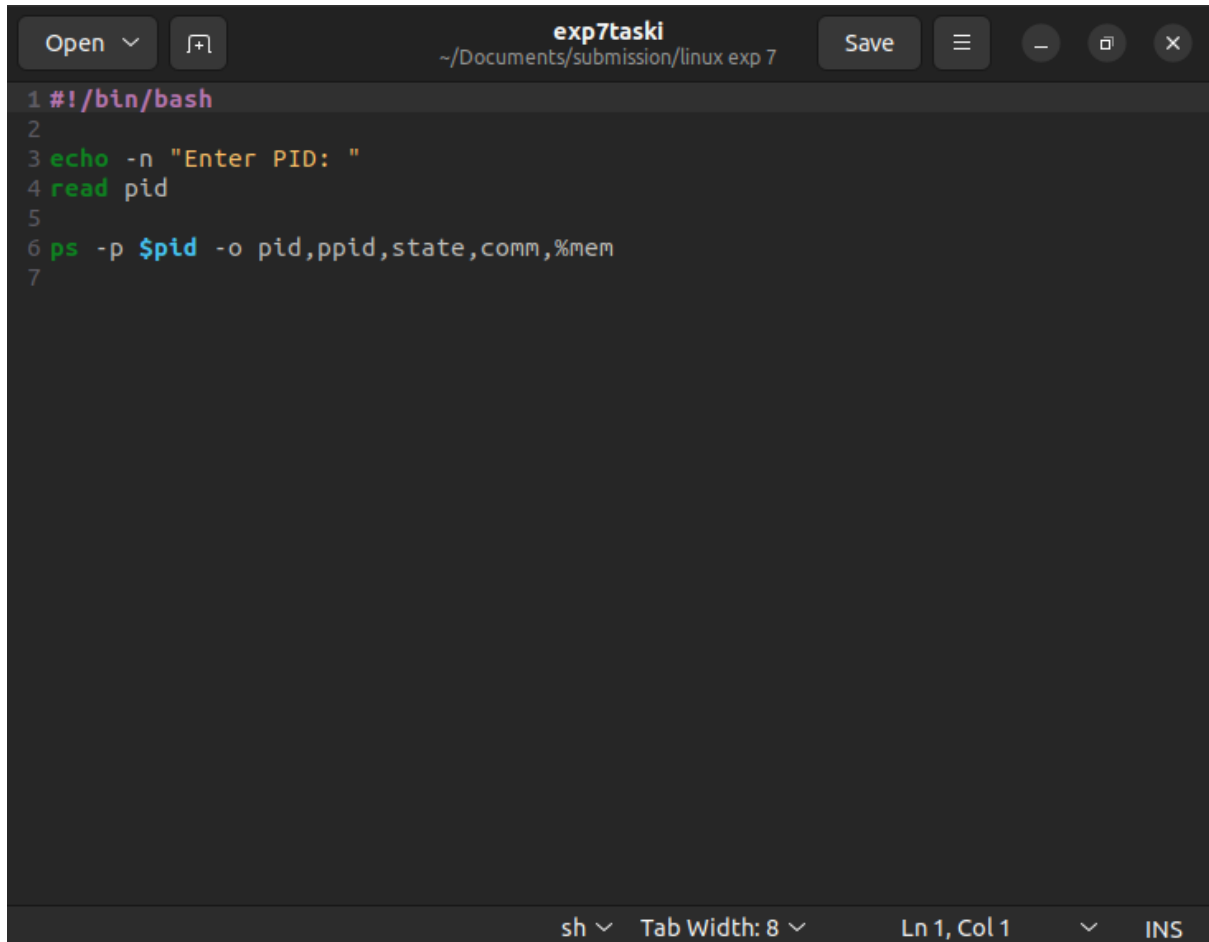
TASK 2. Write a script that accepts a PID from the user and displays its details (state, parent process, memory usage).

Explanation:

Every process in Linux has a unique Process ID (PID). This script asks the user to enter a PID and then displays information about that process, such as its current state (running, sleeping, stopped), its parent process ID (the process that started it), the command associated with it, and its memory usage. This

assignment is useful for process management and learning how to extract detailed process information.

## COMMAND



```
1 #!/bin/bash
2
3 echo -n "Enter PID: "
4 read pid
5
6 ps -p $pid -o pid,ppid,state,comm,%mem
7
```

The screenshot shows a terminal window with a dark background. The title bar at the top reads 'exp7taski' and the path is '~/.Documents/submission/linux exp 7'. The terminal content consists of a shell script with seven lines. Line 1 is the shebang '#!/bin/bash'. Line 2 is empty. Line 3 is 'echo -n "Enter PID: "'. Line 4 is 'read pid'. Line 5 is empty. Line 6 is 'ps -p \$pid -o pid,ppid,state,comm,%mem'. Line 7 is empty. The bottom status bar shows 'sh', 'Tab Width: 8', 'Ln 1, Col 1', and 'INS'.

## OUTPUT



```
pavani@UBUNTU: ~/Documents/linux exp 7
Try 'ps --help <simple|list|output|threads|misc|all>'
or 'ps --help <s|l|o|t|m|a>'
for additional help text.

For more details see ps(1).
pavani@UBUNTU:~/Documents/linux exp 7$ ps
  PID TTY          TIME CMD
 7390 pts/2        00:00:00 bash
 8071 pts/2        00:00:00 ps
pavani@UBUNTU:~/Documents/linux exp 7$ ./exp7task1
Enter PID: 7390
  PID    PPID  S  COMMAND      %MEM
 7390    5639 S  bash         0.2
pavani@UBUNTU:~/Documents/linux exp 7$
```

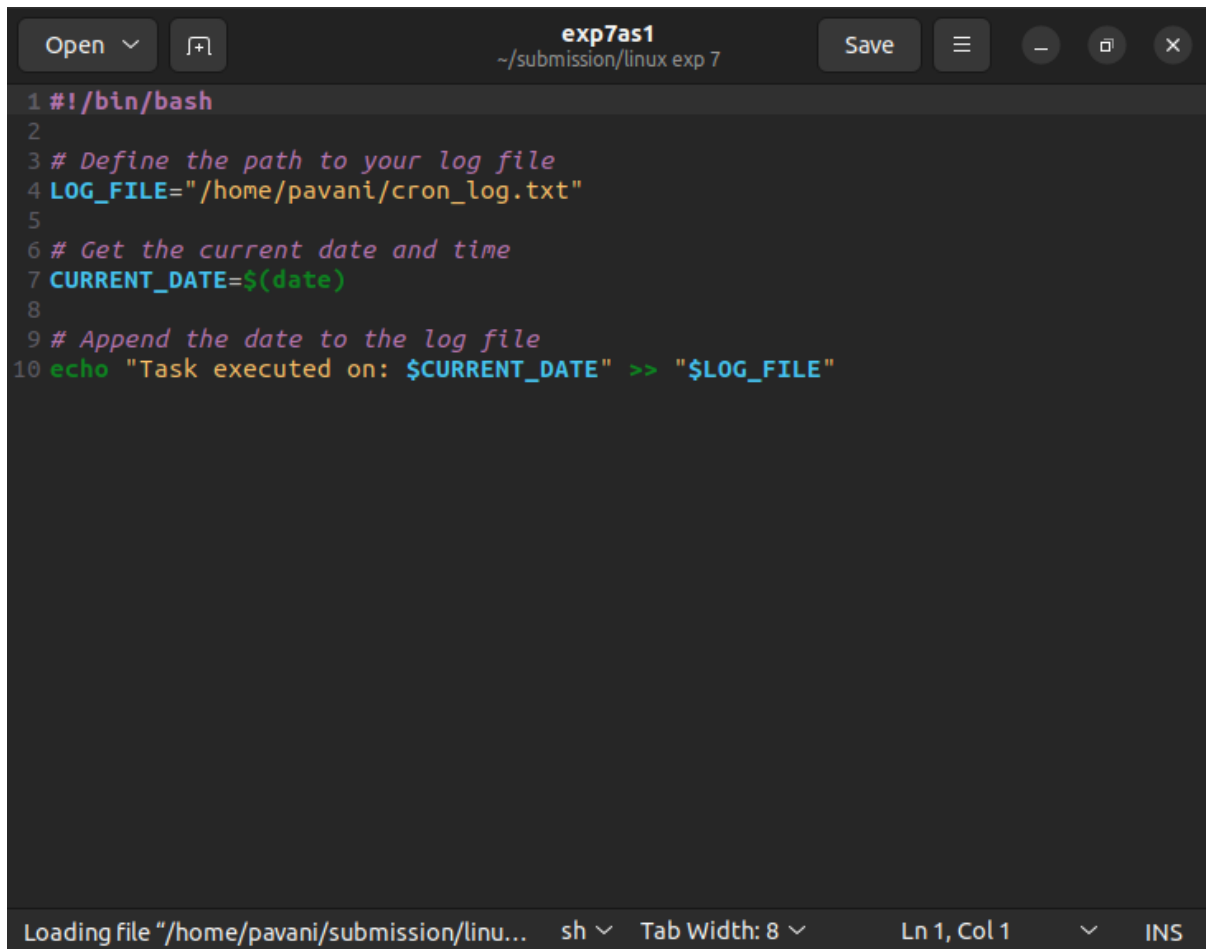
---

TASK3. Create a script that schedules a task to append the current date and time to a log file every minute using cron.

Explanation:

This assignment uses the cron scheduler to automate tasks in Linux. The script (or command) appends the system's current date and time into a log file, and cron is configured to run this task every minute. This way, a continuous log of timestamps is created automatically. It shows how cron jobs can be used for regular and repeated automation in system administration.

## COMMAND



A screenshot of a terminal window with a dark background. The window title is "exp7as1" and the path is "~/submission/linux exp 7". The terminal shows a script with 10 lines of code. Lines 3, 6, and 9 are comments. Lines 4, 7, and 10 are executable commands. The status bar at the bottom shows "Loading file '/home/pavani/submission/linu...", "sh", "Tab Width: 8", "Ln 1, Col 1", and "INS".

```
1 #!/bin/bash
2
3 # Define the path to your log file
4 LOG_FILE="/home/pavani/cron_log.txt"
5
6 # Get the current date and time
7 CURRENT_DATE=$(date)
8
9 # Append the date to the log file
10 echo "Task executed on: $CURRENT_DATE" >> "$LOG_FILE"
```

Loading file "/home/pavani/submission/linu... sh Tab Width: 8 Ln 1, Col 1 INS

## OUTPUT

```
pavani@UBUNTU: ~/Documents/submission/linux exp 7
pavani@UBUNTU:~/Documents/submission/linux exp 7$ vim exp7as1
pavani@UBUNTU:~/Documents/submission/linux exp 7$ chmod +x exp7as1
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7as1
pavani@UBUNTU:~/Documents/submission/linux exp 7$ vim exp7as1
pavani@UBUNTU:~/Documents/submission/linux exp 7$ crontab -e
no crontab for pavani - using an empty one

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano          <---- easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny
 4. /bin/ed

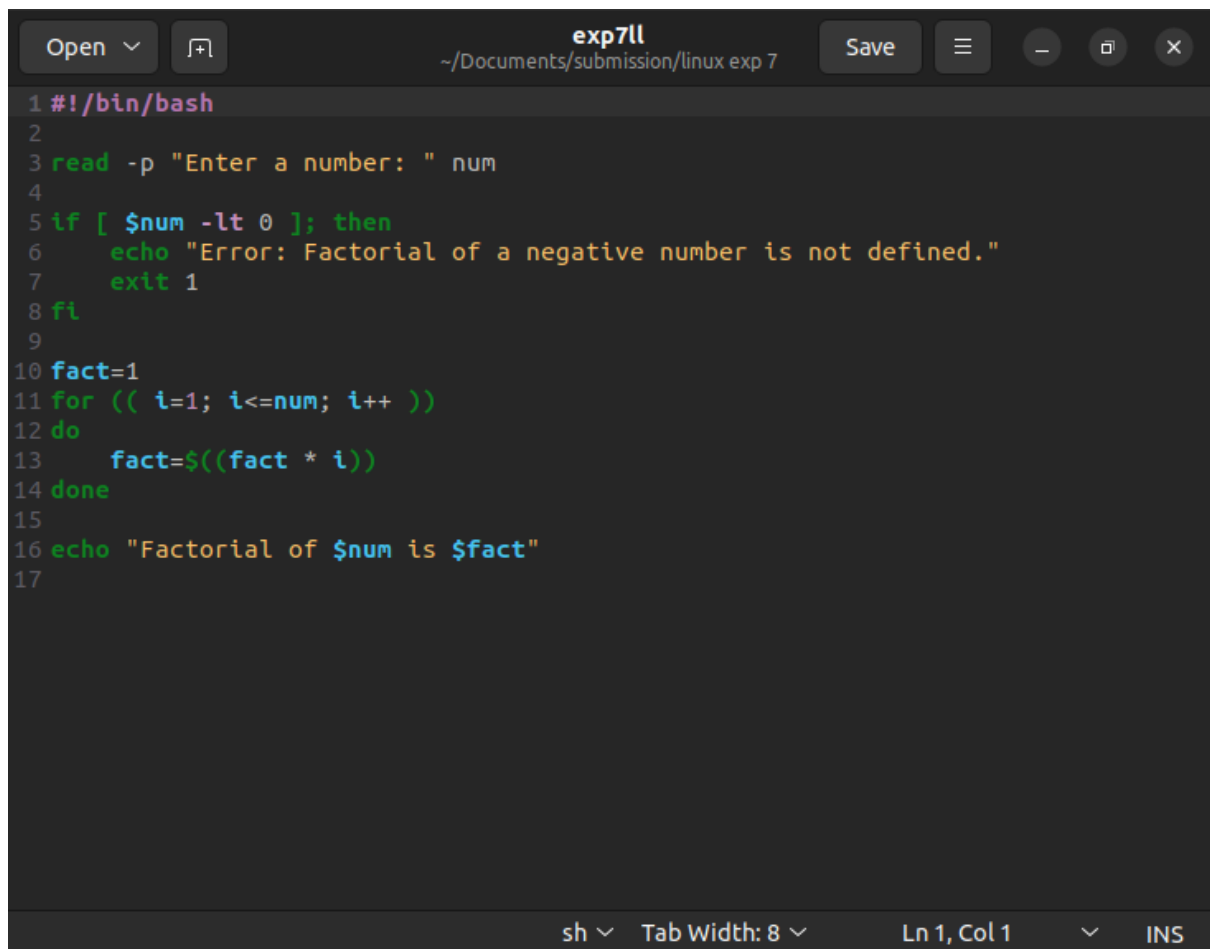
Choose 1-4 [1]: 1
crontab: installing new crontab
pavani@UBUNTU:~/Documents/submission/linux exp 7$ cat /home/pavani/cron_log.txt
Task executed on: Wednesday 24 September 2025 07:22:57 PM IST
Task executed on: Wednesday 24 September 2025 07:25:28 PM IST
Task executed on: Thursday 02 October 2025 12:27:02 AM IST
pavani@UBUNTU:~/Documents/submission/linux exp 7$
```

TASK 4. Modify the factorial function to check if input is negative. If yes, display an error message.

Explanation:

A factorial is defined only for non-negative integers. This modified script first checks if the given input number is negative. If it is negative, the script shows an error message instead of trying to calculate the factorial. This task highlights the importance of input validation and error handling in shell scripts.

COMMAND



The image shows a terminal window titled "exp7ll" with the path "~/Documents/submission/linux exp 7". The window contains a shell script for calculating the factorial of a number. The script starts with a shebang line, prompts the user for a number, checks if it's negative, and then uses a for loop to calculate the factorial. The status bar at the bottom indicates the shell is "sh", tab width is 8, and the cursor is at line 1, column 1 in insert mode.

```
1 #!/bin/bash
2
3 read -p "Enter a number: " num
4
5 if [ $num -lt 0 ]; then
6     echo "Error: Factorial of a negative number is not defined."
7     exit 1
8 fi
9
10 fact=1
11 for (( i=1; i<=num; i++ ))
12 do
13     fact=$((fact * i))
14 done
15
16 echo "Factorial of $num is $fact"
17
```

sh ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

OUTPUT

```
pavani@UBUNTU: ~/Documents/submission/linux exp 7
pavani@UBUNTU:~/Documents/submission/linux exp 7$ vim exp7ll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ chmod +x exp7ll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7ll
Enter a number: 23
Factorial of 23 is 8128291617894825984
pavani@UBUNTU:~/Documents/submission/linux exp 7$ vim exp7lll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ chmod +x exp7lll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7lll
Usage: ./exp7lll filename
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7lll exp7.sh
Number of lines starting with a vowel: 0
pavani@UBUNTU:~/Documents/submission/linux exp 7$
```

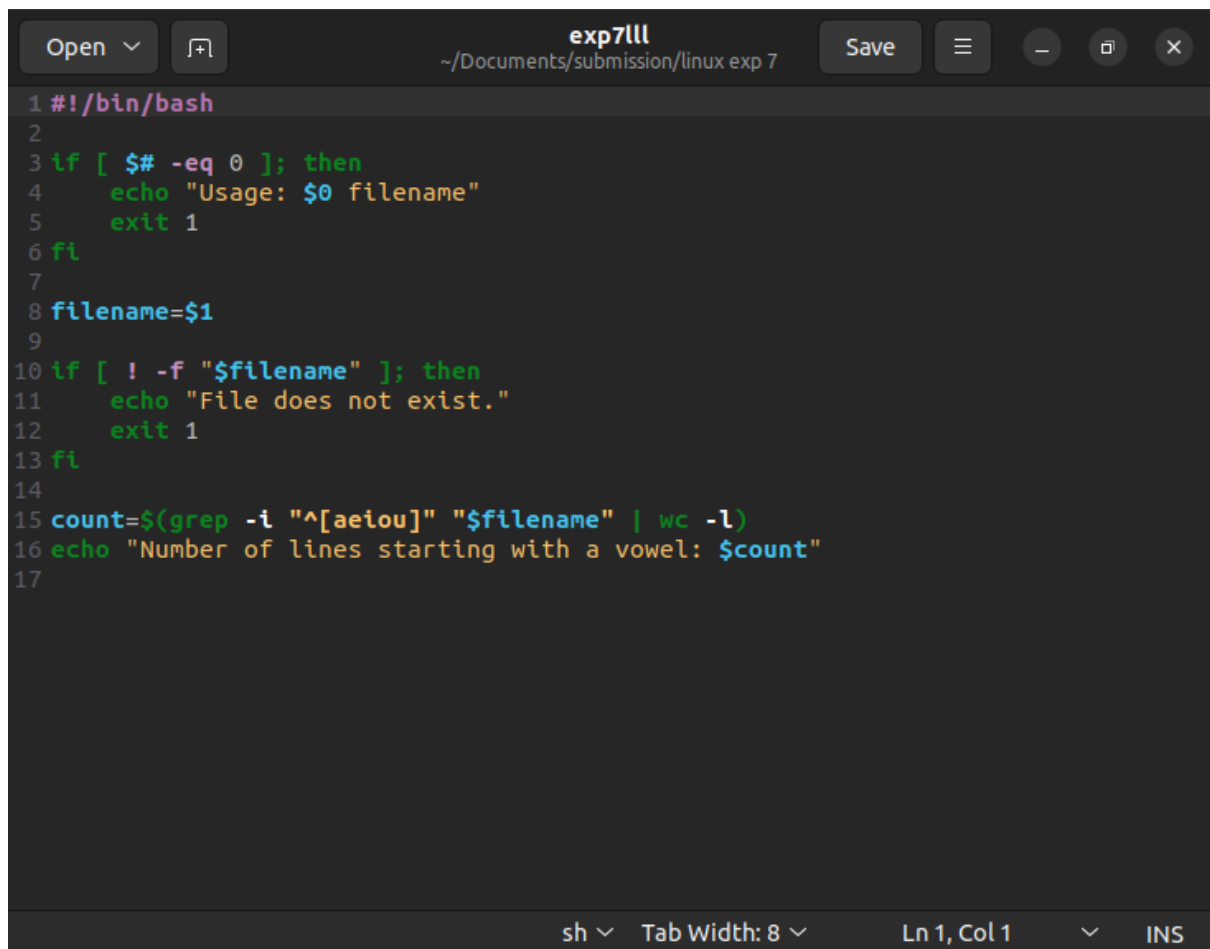
---

TASK 5. Write a script that accepts a filename as an argument. If the file exists, display the number of lines starting with a vowel.

Explanation:

This script takes a filename as input from the command line. First, it checks whether the file exists in the system. If it does, the script searches the file for lines beginning with a vowel (A, E, I, O, U – case-insensitive). It then counts how many such lines are present. This assignment demonstrates file handling, input checking, and pattern matching in shell scripting.

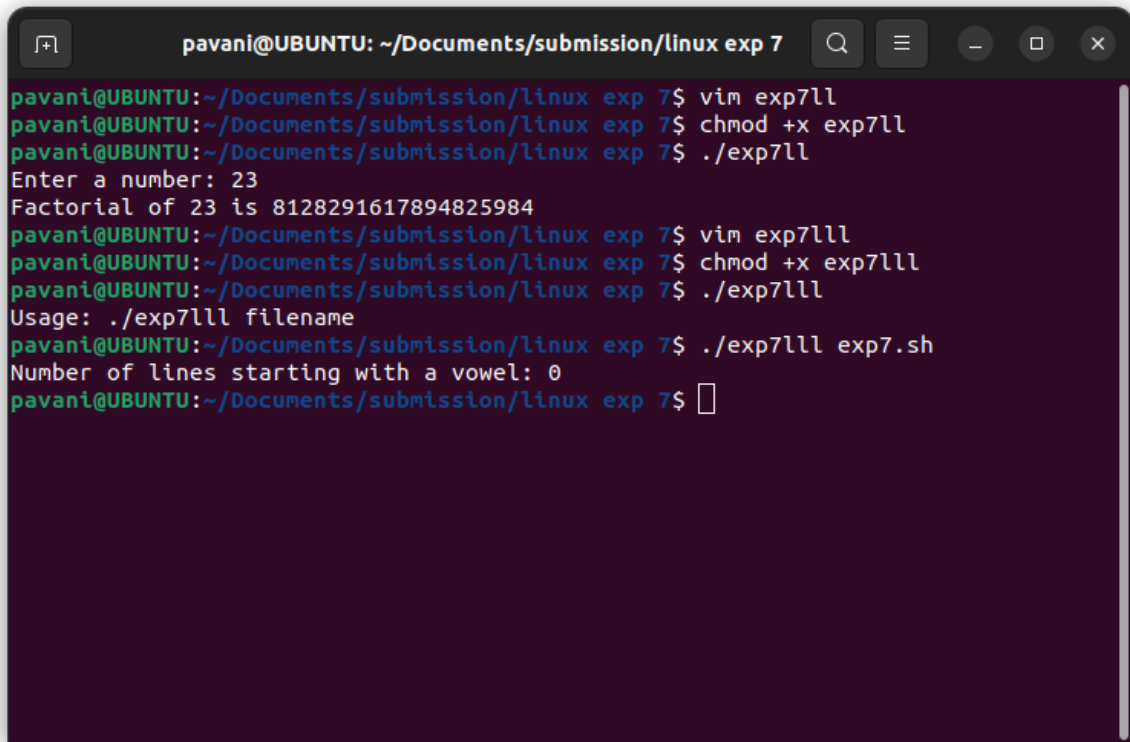
COMMAND



```
1 #!/bin/bash
2
3 if [ $# -eq 0 ]; then
4     echo "Usage: $0 filename"
5     exit 1
6 fi
7
8 filename=$1
9
10 if [ ! -f "$filename" ]; then
11     echo "File does not exist."
12     exit 1
13 fi
14
15 count=$(grep -i "^[aeiou]" "$filename" | wc -l)
16 echo "Number of lines starting with a vowel: $count"
17
```

sh Tab Width: 8 Ln 1, Col 1 INS

OUTPUT

A terminal window titled 'pavani@UBUNTU: ~/Documents/submission/linux exp 7' with search, menu, and window control icons. The terminal shows the following commands and output:

```
pavani@UBUNTU:~/Documents/submission/linux exp 7$ vim exp7ll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ chmod +x exp7ll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7ll
Enter a number: 23
Factorial of 23 is 8128291617894825984
pavani@UBUNTU:~/Documents/submission/linux exp 7$ vim exp7lll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ chmod +x exp7lll
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7lll
Usage: ./exp7lll filename
pavani@UBUNTU:~/Documents/submission/linux exp 7$ ./exp7lll exp7.sh
Number of lines starting with a vowel: 0
pavani@UBUNTU:~/Documents/submission/linux exp 7$
```

## Observation

Each process had a PID and belonged to a hierarchy.

Processes changed states depending on execution.

Priorities and scheduling affected CPU usage.

## Conclusion

Processes are basic execution units. They pass through states, form hierarchies, and can be managed by killing, prioritising, or scheduling.