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))
```

```
    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)"
```

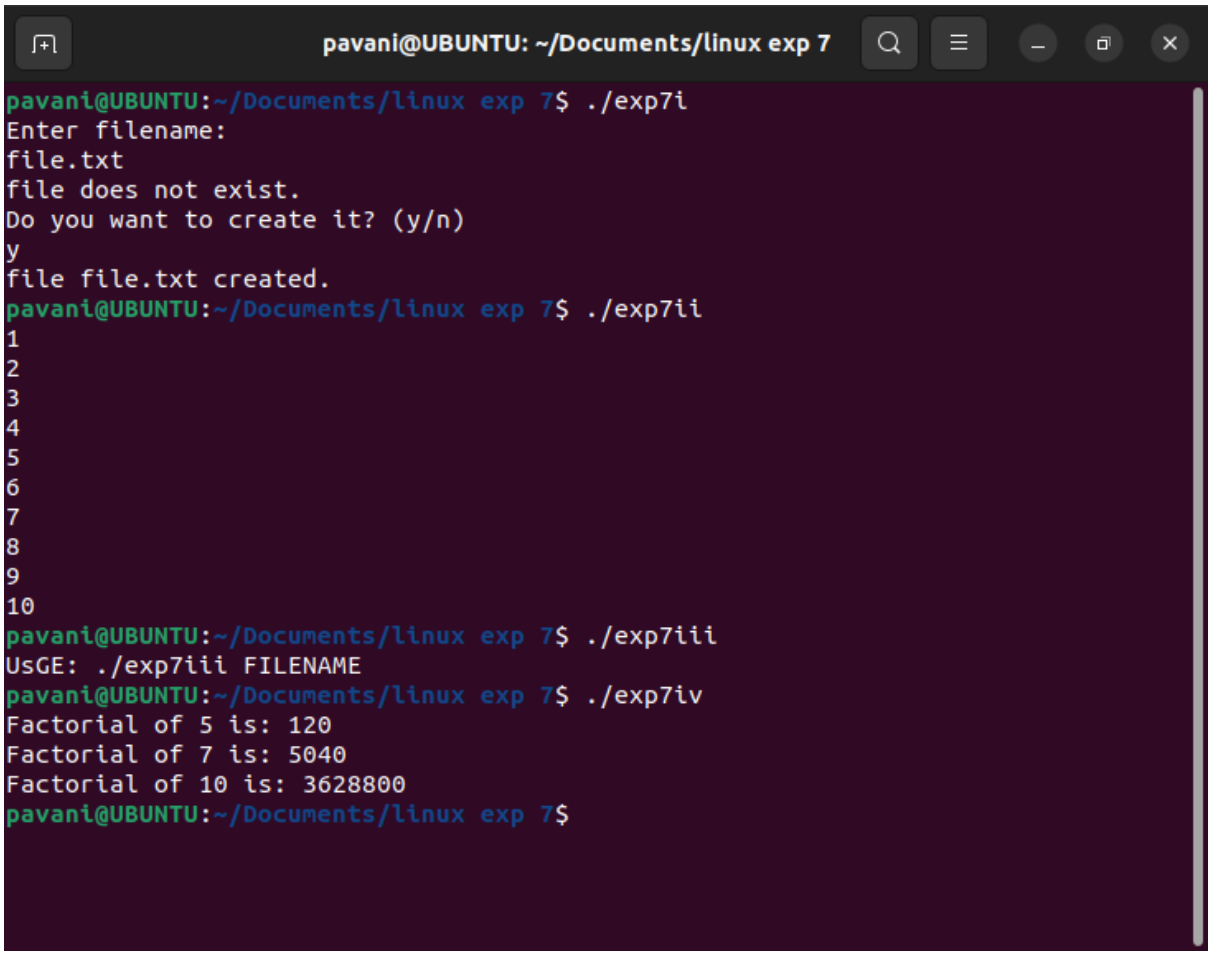
New Concepts:

- Function definition in bash:

function_name() { commands }

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

OUTPUT OF EXERCISE 1,2,3,4

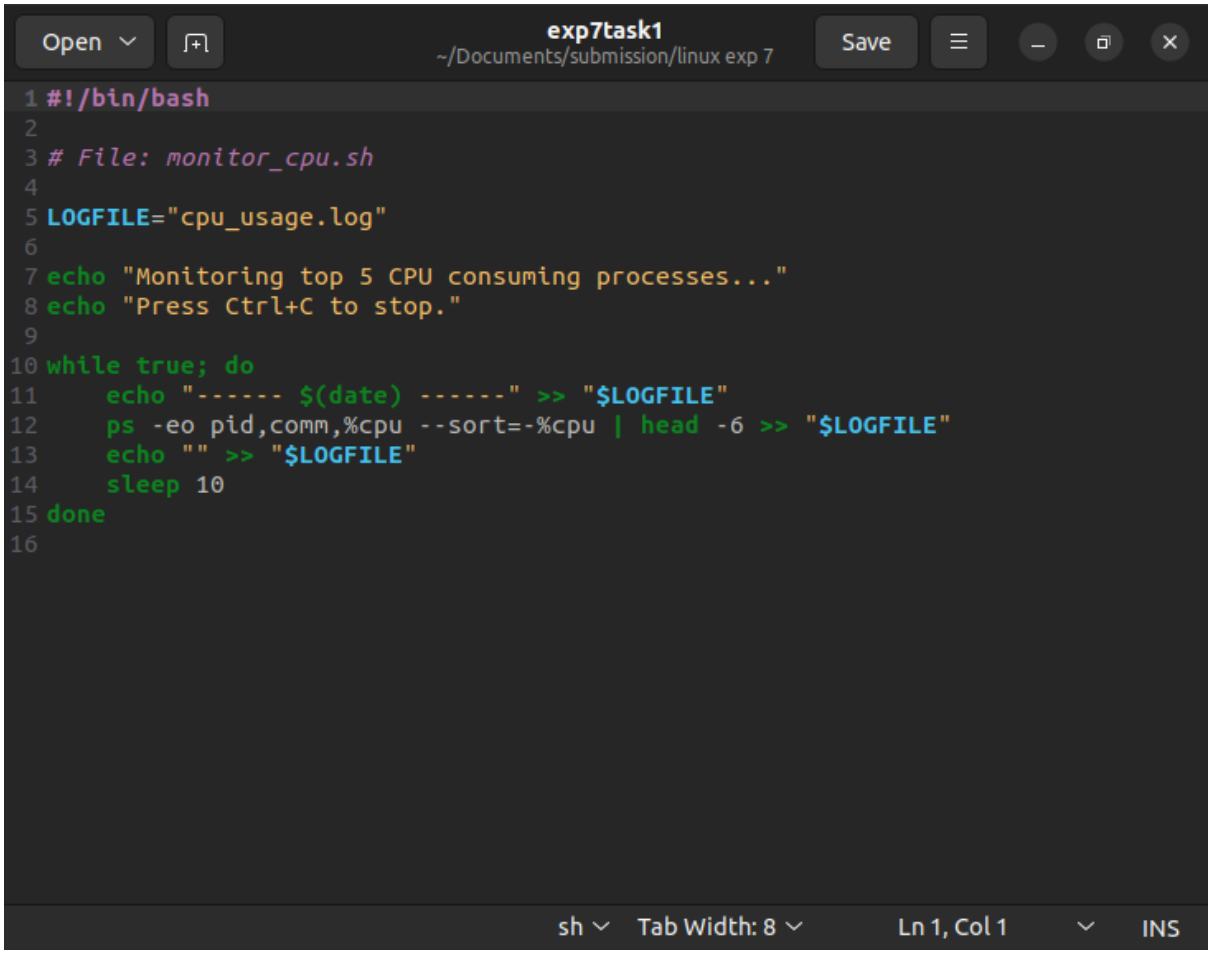


LAB TASKS

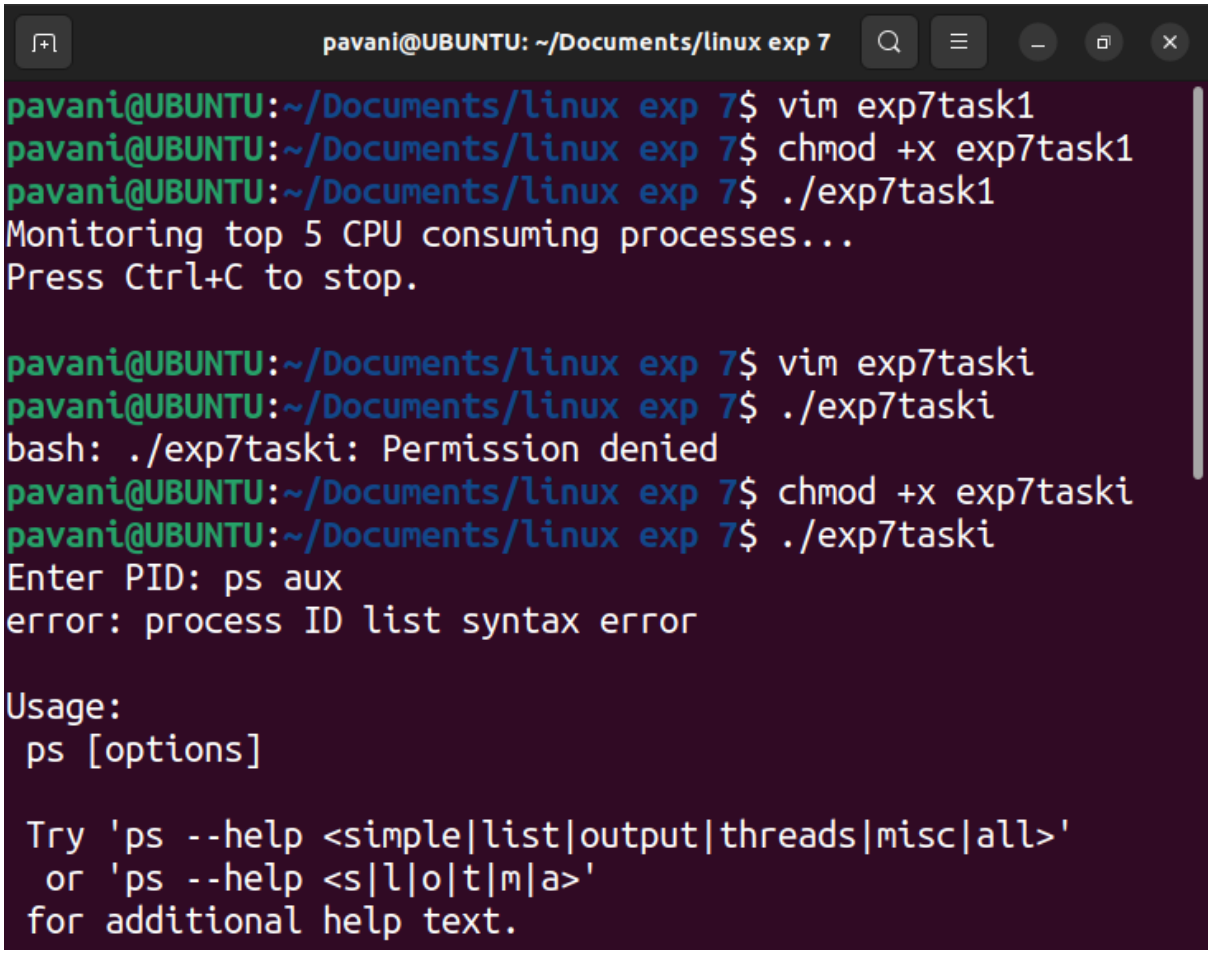
TASK l. Write a script that monitors the top 5 processes consuming the most CPU and logs them into a file every 10 seconds.

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.



Output



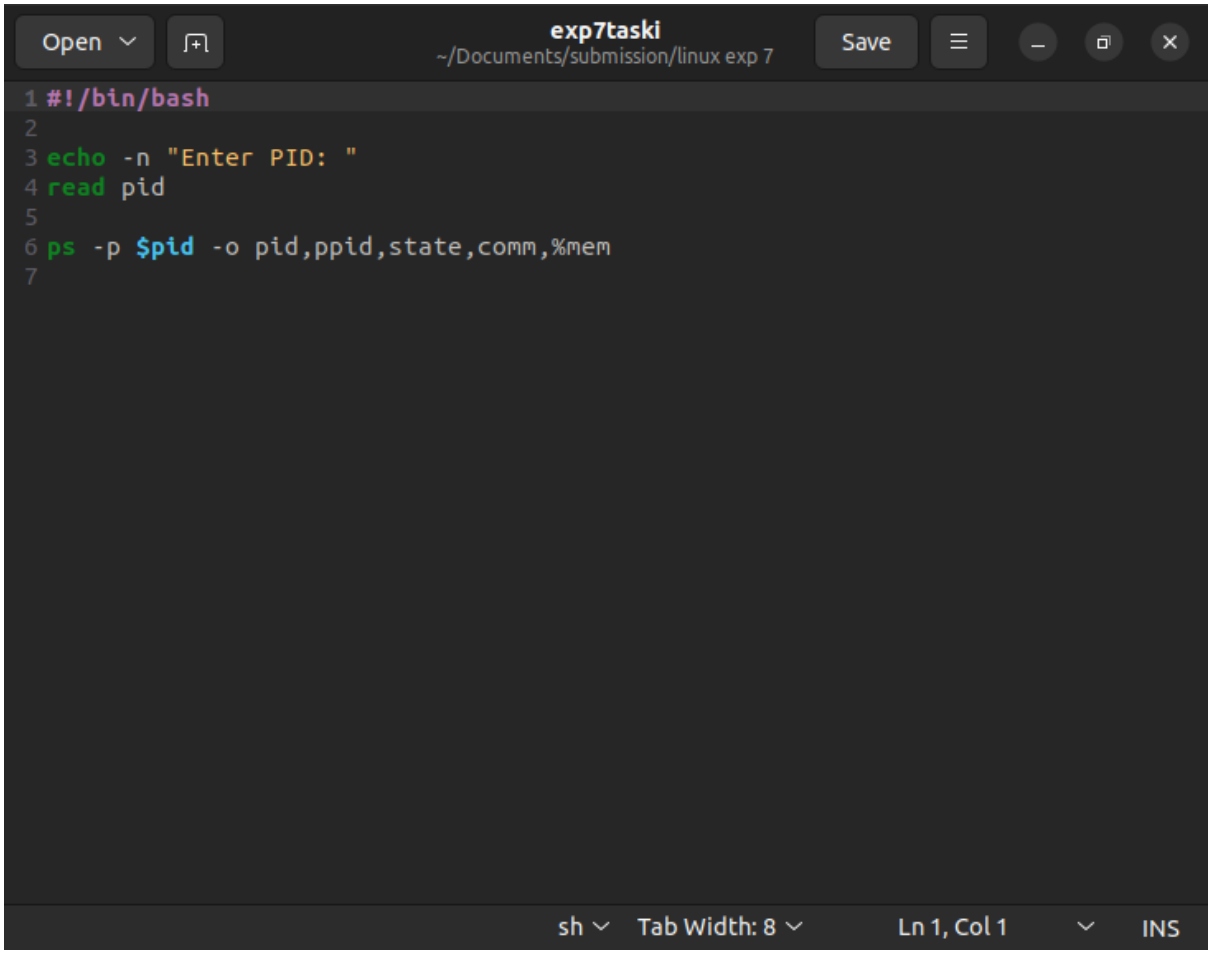
TASK 2. Write a script that accepts a PID from the user and displays its details (state, parent process, memory usage).

Explanation:

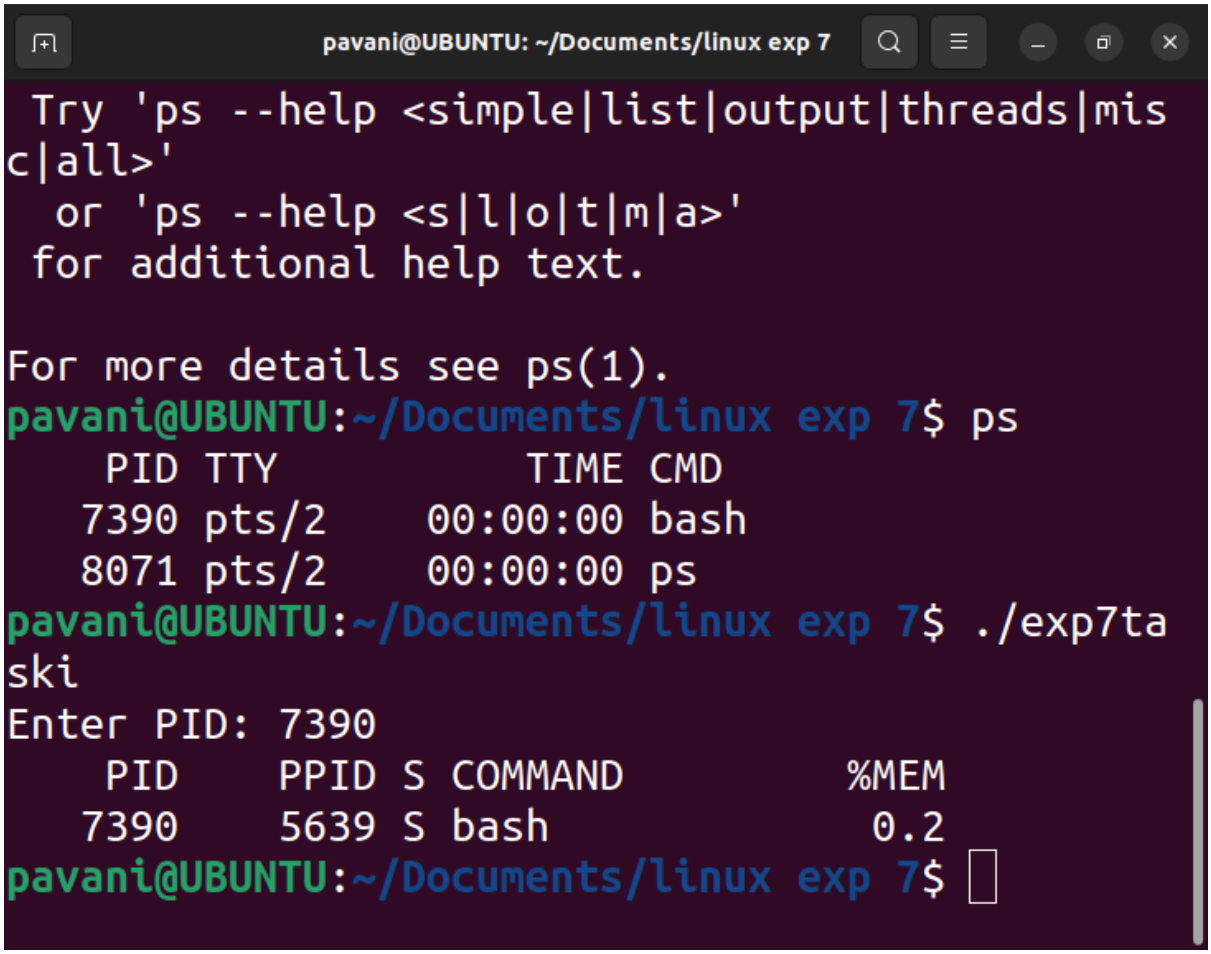
Every process in Linux has a unique Process ID (PID). This script asks the user to enter a PID and then displays information about that process, such as its current state (running, sleeping, stopped), its parent process ID (the process that started it), the command associated with it, and its memory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

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

COMMAND



OUTPUT



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.

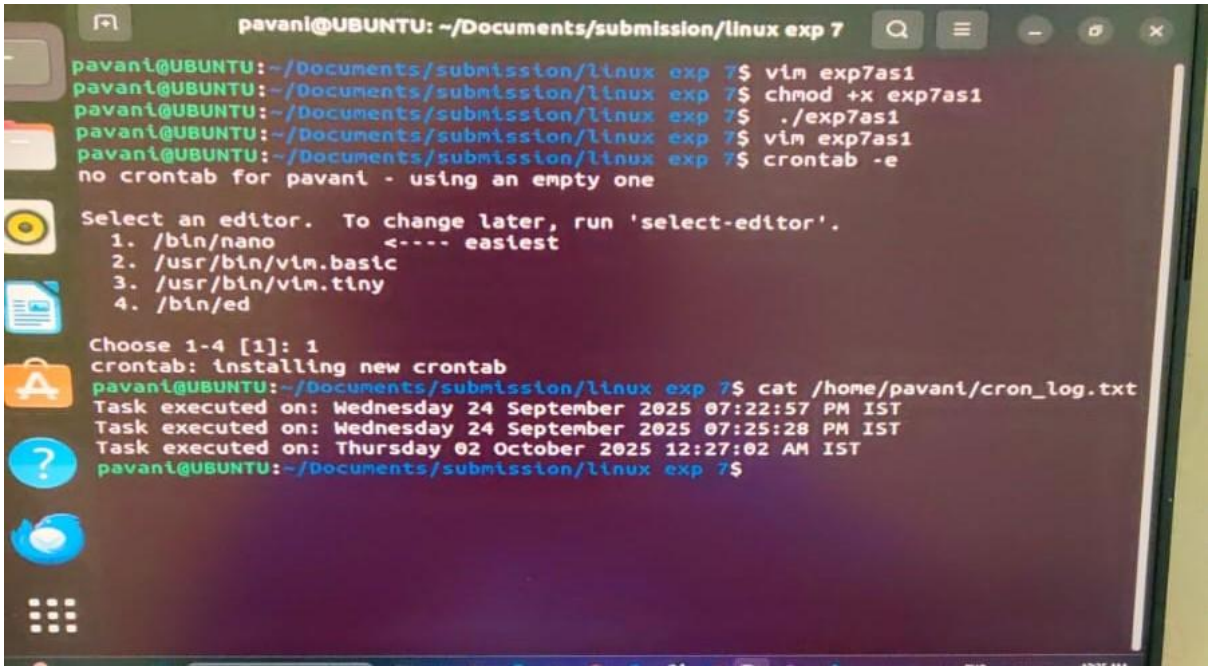
COMMAND

Open
exp7as1
~/submission/linux exp 7
Save

```
#!/bin/bash

# Define the path to your log file
LOG_FILE="/home/pavani/cron_log.txt"

# Get the current date and time
CURRENT_DATE=$(date)

# Append the date to the log file
echo "Task executed on: $CURRENT_DATE" >> "$LOG_FILE"
```

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

OUTPUT

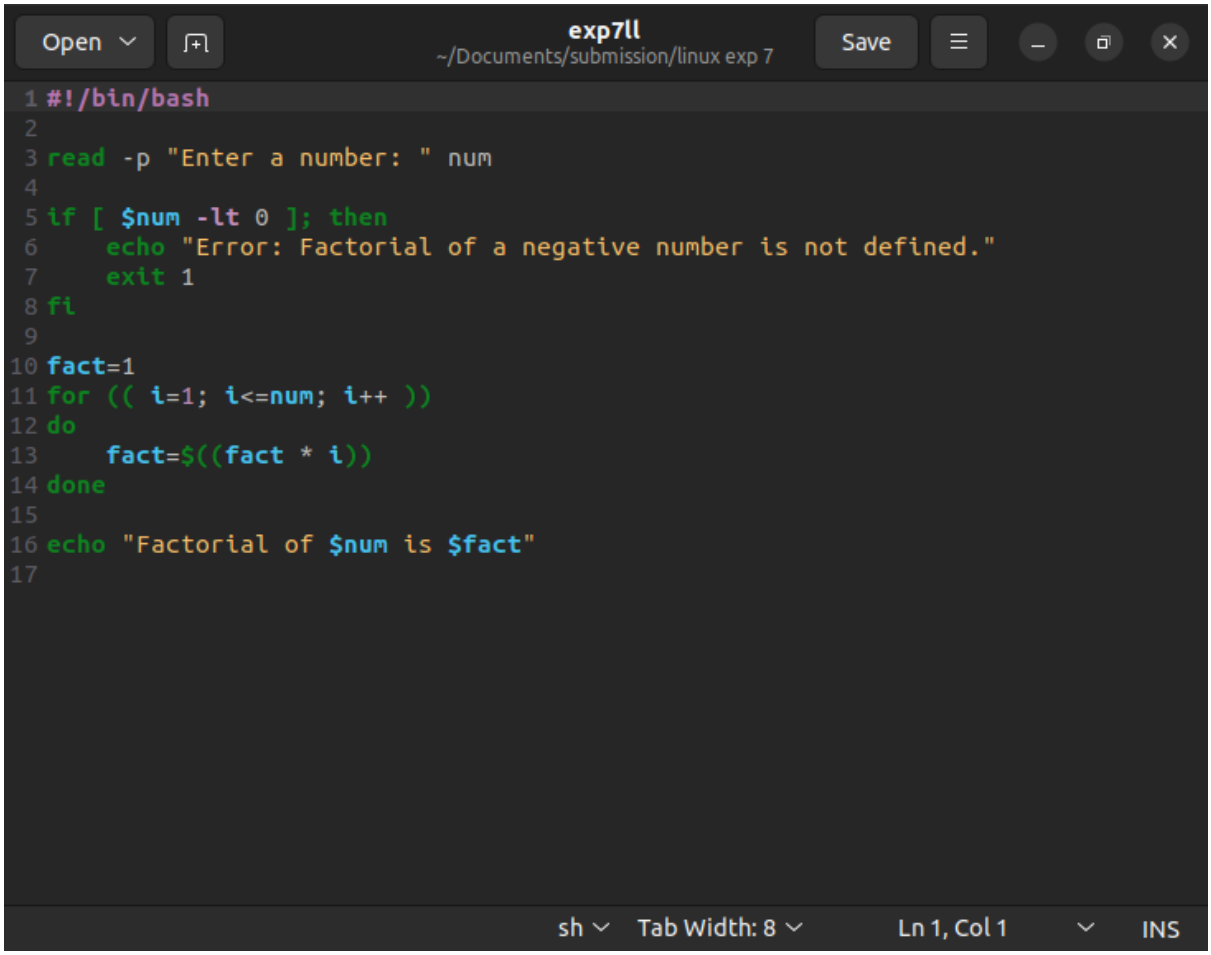


TASK 4. Modify the factorial function to check if input is negative. If yes, display an error message.

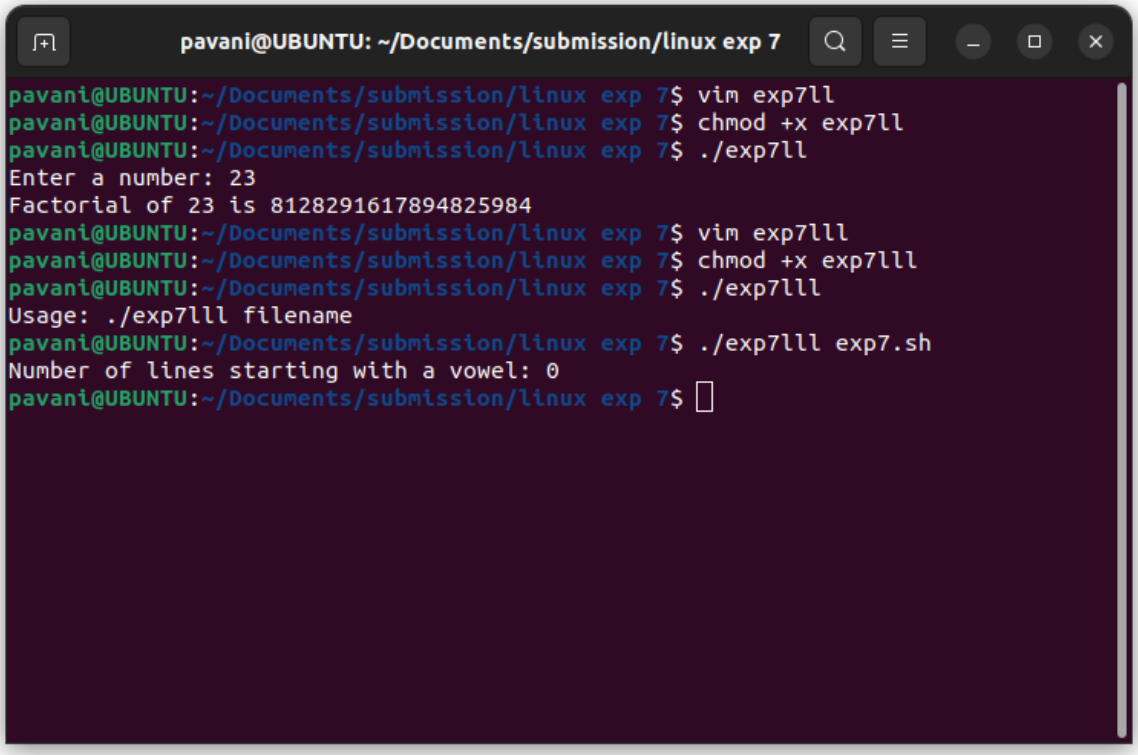
Explanation:

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


Open
exp7ll
~/Documents/submission/linux exp 7
Save
#!/bin/bash
read -p "Enter a number: " num
if [$num -lt 0]; then
echo "Error: Factorial of a negative number is not defined."
exit 1
fi
fact=1
for ((i=1; i<=num; i++))
do
fact=$((fact * i))
done
echo "Factorial of $num is $fact"
sh
Tab Width: 8
Ln 1, Col 1
INS

OUTPUT



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

Open

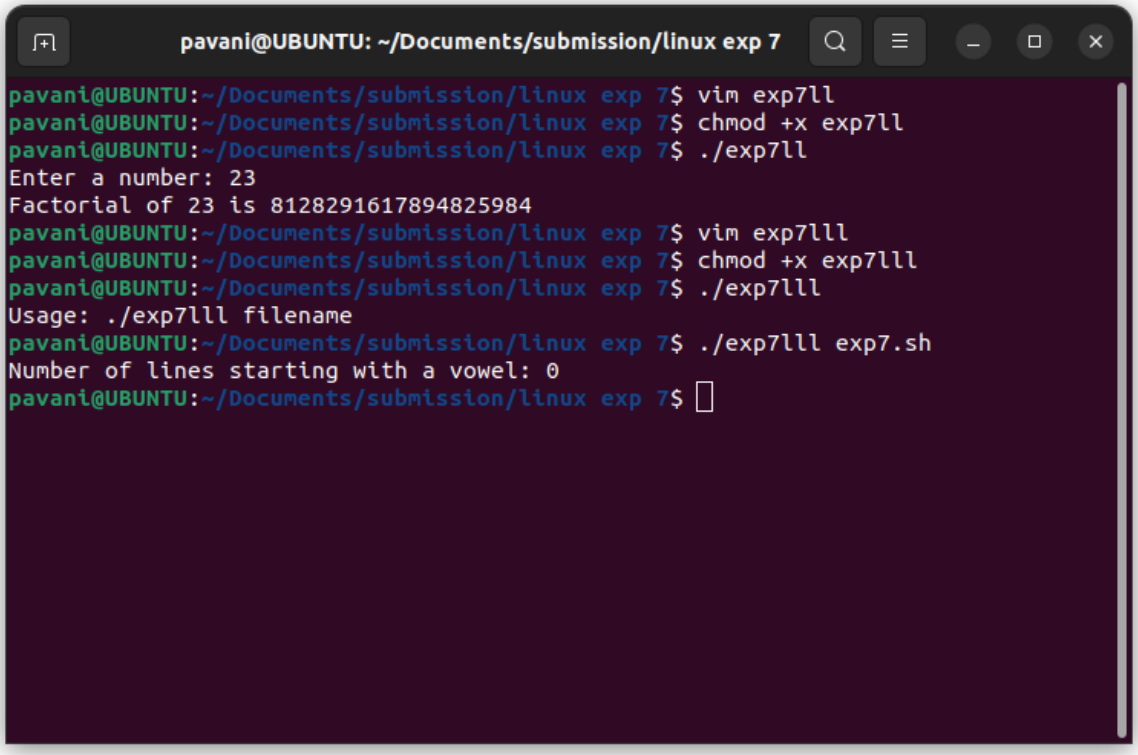
exp7lll
~/Documents/submission/linux exp 7

Save

```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usage: $0 filename"
    exit 1
fi

filename=$1

if [ ! -f "$filename" ]; then
    echo "File does not exist."
    exit 1
fi

count=$(grep -i "^[aeiou]" "$filename" | wc -l)
echo "Number of lines starting with a vowel: $count"

```

sh Tab Width: 8 Ln 1, Col 1 INS

OUTPUT



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.