

Experiment 8 – Shell Programming

Aim

To learn about process handling in Shell Programming, including signals, monitoring, communication, synchronization, background processes, job control, and system logging.

Requirements

- A computer with Linux/Ubuntu installed
 - Basic knowledge of Linux commands
 - Terminal (command line) access
-

Theory

1. Process Control (Signals)

- Every running program is called a **process**.
 - We can **control a process** by sending it signals.
 - Example signals:
 - **Ctrl+C** → Stop the process (Interrupt)
 - **SIGKILL** → Force stop a process
 - **SIGSTOP / SIGCONT** → Pause and resume a process
-

2. Process Monitoring & Resource Usage

- We can check which processes are running using commands like ps, top, or htop.
 - Monitoring tells us how much **CPU, memory, and resources** each process is using.
 - This helps in finding and stopping programs that use too many resources.
-

3. Process Communication

- Sometimes processes need to **talk to each other**.
 - This can be done using:
 - **Pipes (|)** → Connects output of one program to another
 - **Signals** → Small notifications to processes
 - **Shared memory/message queues** (advanced)
-

4. Process Synchronization

- When two processes use the **same resource at the same time**, problems can happen.
 - Synchronization means making processes wait for each other in an **organized way**.
 - This prevents errors like **race conditions** (when two processes change data together).
-

5. Background Processes & Job Control

- A process can run in the **foreground** (directly in terminal) or in the **background** (without blocking terminal).
 - **&** → Runs process in background
 - **jobs** → Shows background jobs
 - **fg** → Brings a job to foreground
 - **bg** → Sends a job to background
-

6. System Monitoring & Logging

- The system keeps **logs** of activities and errors.
- Logs are stored in **/var/log/**.
- **dmesg** shows system messages.
- **logger** can add custom messages to logs.
- Logs are very helpful in **debugging problems**.

Lab Exercises

i. Check File Permissions

```
#!/bin/bash
```

```
echo "Enter filename:"
```

```
read file
```

```
if [ -e "$file" ]; then
```

```
    [ -r "$file" ] && echo "File is readable"
```

```
    [ -w "$file" ] && echo "File is writable"
```

```
    [ -x "$file" ] && echo "File is executable"
```

```
else
```

```
    echo "File does not exist."
```

```
fi
```

New Concepts:

- -r → check if readable.
- -w → check if writable.
- -x → check if executable.

ii. String Operations

```
#!/bin/bash
```

```
echo "Enter first string:"
```

```
read str1
```

```
echo "Enter second string:"
```

```
read str2
```

```
# String length
```

```
echo "Length of first string: ${#str1}"
```

```
echo "Length of second string: ${#str2}"
```

```
# Concatenation
```

```
concat="$str1$str2"
```

```
echo "Concatenated string: $concat"
```

```
# Comparison
```

```
if [ "$str1" = "$str2" ]; then
```

```
    echo "Strings are equal"
```

```
else
```

```
    echo "Strings are not equal"
```

```
fi
```

New Concepts:

- \${#var} → length of string.
- "\$str1\$str2" → string concatenation.
- = operator → string comparison.

iii. Search for a Pattern in a File

```
#!/bin/bash
```

```
echo "Enter filename:"
```

```
read file
```

```
echo "Enter pattern to search:"
```

```
read pattern
```

```
if [ -e "$file" ]; then
```

```
    echo "Matching lines:"
```

```
    grep "$pattern" "$file"
```

```
else
```

```
echo "File not found!"
```

```
fi
```

New Command:

- `grep pattern file` → searches for matching lines.

iv. Display System Information

```
#!/bin/bash
```

```
echo "System Information:"
```

```
echo "-----"
```

```
echo "Date and Time: $(date)"
```

```
echo "Logged in users: $(who)"
```

```
echo "System Uptime: $(uptime -p)"
```

```
echo "Memory Usage:"
```

```
free -h
```

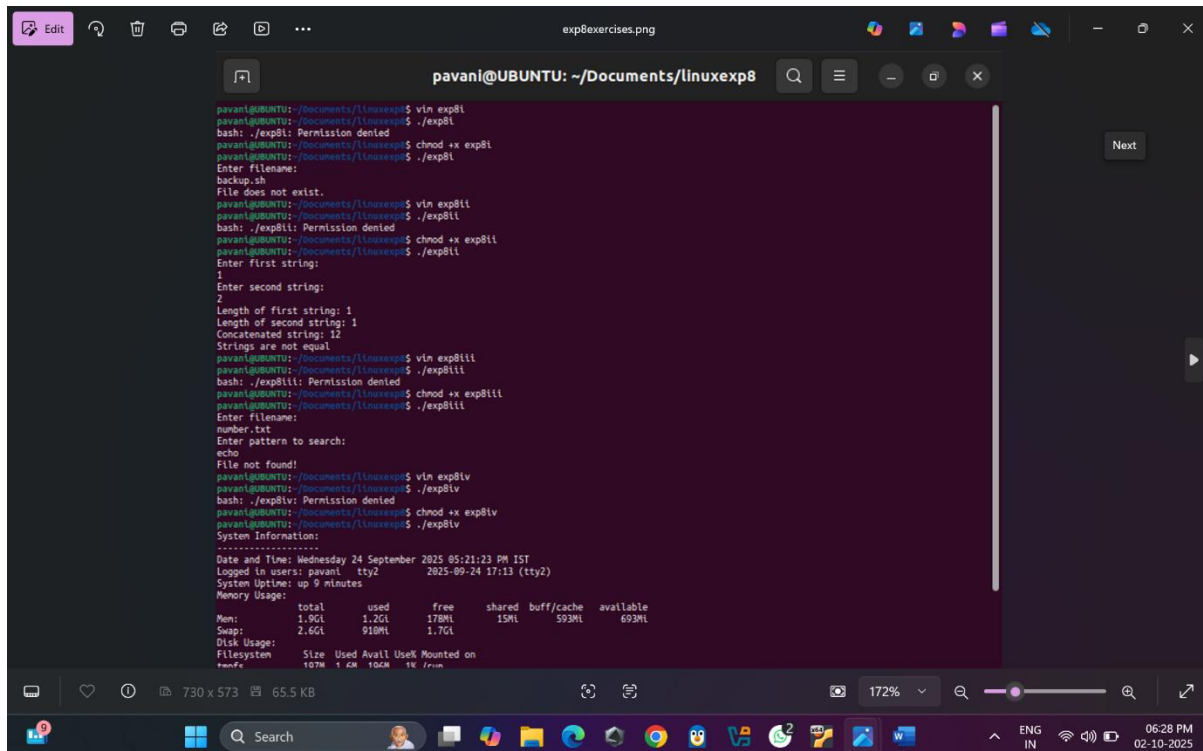
```
echo "Disk Usage:"
```

```
df -h
```

New Commands:

- `date` → current date and time.
- `who` → list logged-in users.
- `uptime -p` → pretty uptime format.
- `free -h` → memory usage in human-readable format.
- `df -h` → disk usage.

OUTPUT OF EXERCISE I,ii,iii.iv



```
pavanl@UBUNTU: ~/Documents/linuxexp8
pavanl@UBUNTU:~/Documents/linuxexp8$ vln exp8i
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8i
bash: ./exp8i: Permission denied
pavanl@UBUNTU:~/Documents/linuxexp8$ chmod +x exp8i
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8i
Enter filename:
backup.sh
File does not exist.
pavanl@UBUNTU:~/Documents/linuxexp8$ vln exp8ii
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8ii
bash: ./exp8ii: Permission denied
pavanl@UBUNTU:~/Documents/linuxexp8$ chmod +x exp8ii
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8ii
Enter first string:
1
Enter second string:
2
Length of first string: 1
Length of second string: 1
Concatenated string: 12
Strings are not equal
pavanl@UBUNTU:~/Documents/linuxexp8$ vln exp8iii
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8iii
bash: ./exp8iii: Permission denied
pavanl@UBUNTU:~/Documents/linuxexp8$ chmod +x exp8iii
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8iii
Enter filename:
number.txt
Enter pattern to search:
echo
File not found!
pavanl@UBUNTU:~/Documents/linuxexp8$ vln exp8iv
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8iv
bash: ./exp8iv: Permission denied
pavanl@UBUNTU:~/Documents/linuxexp8$ chmod +x exp8iv
pavanl@UBUNTU:~/Documents/linuxexp8$ ./exp8iv
System Information:
-----
Date and Time: Wednesday 24 September 2025 05:21:23 PM IST
Logged in users: pavanl  tty2  2025-09-24 17:13 (tty2)
System Uptime: up 9 minutes
Memory Usage:

```

	total	used	free	shared	buff/cache	available
Mem:	1.9Gi	1.2Gi	178Mi	15Mi	593Mi	693Mi
Swaps:	2.6Gi	910Mi	1.7Gi			

```

Disk Usage:
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           107M  1.4M  106M   1% /run

```

730 x 573 65.5 KB 172% 06:28 PM 02-10-2025

LAB TASKS

TASK 1. Write a script that starts a background job (e.g., sleep 60), lists all jobs, brings the job to the foreground, and then terminates it.

Explanation:

In this task, a command is started as a background job so that the terminal remains free while the job is running. The list of jobs can then be checked using a jobs command. After that,

the background job is brought to the foreground so that the user can interact with it directly. Finally, the job is terminated. This assignment demonstrates the difference between background and foreground processes and how to manage them.

COMMAND

```
#!/bin/bash
```

```
sleep 60 &
```

```
jobs
```

```
pid=$(jobs -p)
```

```
echo "PID of job:$pid"
```

```
kill $pid
```

```
echo "job killed"
```

Task 2. Create a script that compares two files and displays whether their contents are identical or different.

Explanation:

This script takes two files as input and compares their contents. If the contents are exactly the same, it displays that the files are identical. If there is any difference, it displays that the files are different. For this, Linux provides utilities like `cmp` and `diff` that check files line by line or byte by byte. This assignment helps in understanding file comparison operations in shell scripting.

COMMAND

```
#!/bin/bash
```

```
echo "Enter first filename:"
```

```
read file1
```

```
echo "Enter second filename:"
```

```
read file2
```

```
if cmp -s "$file1" "$file2"; then
```

```
    echo "Files are identical"
```

```
else
```

```
    echo "Files are different"
```

```
fi
```

Task 3. Write script that counts the number of processes currently being run by your user.

Explanation:

Every user has multiple processes running in the system, such as applications, background services, and scripts. This script counts how many processes are running under the current user account. The list of processes for the user can be checked with process listing commands, and the total count can be found using line counting tools. This task is useful for system monitoring and understanding how many processes are active.

COMMAND


```
#!/bin/bash
```

```
count=$(ps -u $USER | wc -l)
```

```
echo "Number of processes running by user $USER: $count"
```

TASK 4. Develop a script that monitors memory usage every 5 seconds and logs it into a file.

Explanation:

This script is designed for system monitoring. It repeatedly checks the system's memory usage at regular intervals (every 5 seconds) and records the output into a file. By running it in a loop, memory usage statistics are continuously stored for later analysis. This assignment demonstrates how to automate system monitoring tasks and maintain logs using shell scripting.

COMMAND

```
#!/bin/bash
```

```
logfile="memory_log.txt"
```

```
echo "Logging memory usage to $logfile (press Ctrl+C to stop)"
```

```
while true
```

```
do
```

```
    free -m >> "$logfile"
```

```
    echo "-----" >> "$logfile"
```

```
    sleep 5
```

```
done
```

OUTPUT OF ABOVE TASKS

```
pavani@UBUNTU: ~/Documents/submission/linuxexp8
pavani@UBUNTU:~/Documents/submission/linuxexp8$ vim exp8taski
pavani@UBUNTU:~/Documents/submission/linuxexp8$ chmod +x exp8taski
pavani@UBUNTU:~/Documents/submission/linuxexp8$ ./exp8taski
[1]+  Running                  sleep 60 &
./exp8taski: line 9: fg: no job control
pavani@UBUNTU:~/Documents/submission/linuxexp8$ vim exp8taski
pavani@UBUNTU:~/Documents/submission/linuxexp8$ chmod +x exp8taski
pavani@UBUNTU:~/Documents/submission/linuxexp8$ ./exp8taski
[1]+  Running                  sleep 60 &
PID of job:6921
job killed
pavani@UBUNTU:~/Documents/submission/linuxexp8$ vim exp8taskii
pavani@UBUNTU:~/Documents/submission/linuxexp8$ chmod +x exp8taskii
pavani@UBUNTU:~/Documents/submission/linuxexp8$ ./exp8taskii
Enter first filename:
exp8i
Enter second filename:
exp8ii
Files are different
pavani@UBUNTU:~/Documents/submission/linuxexp8$ vim exp8task1ps
pavani@UBUNTU:~/Documents/submission/linuxexp8$ chmod +x exp8task1ps
pavani@UBUNTU:~/Documents/submission/linuxexp8$ ./exp8task1ps
Number of processes running by user pavani: 98
pavani@UBUNTU:~/Documents/submission/linuxexp8$ vim exp8task1free
pavani@UBUNTU:~/Documents/submission/linuxexp8$ chmod +x exp8task1free
pavani@UBUNTU:~/Documents/submission/linuxexp8$ ./exp8task1free
Logging memory usage to memory_log.txt (press Ctrl+C to stop)
^C
pavani@UBUNTU:~/Documents/submission/linuxexp8$ cat memory_log.txt
total      used      free      shared  buff/cache   available
Mem:      1968      1408       167        20       393        541
Swap:     2679       632     2047
-----
total      used      free      shared  buff/cache   available
Mem:      1968      1407       167        20       394        542
```

TASK 5. Write a script that prompts for a filename and a search pattern, then displays the count of matching lines.

Explanation:

This script asks the user to enter two things:

- 1. A filename to search within.**
- 2. A pattern (word or string) to look for.**

After that, it searches through the file and counts how many lines contain the given pattern. This is useful for quickly finding specific information in large files. Tools like grep are

used for searching, and an option can be used to count matches directly. This assignment shows how to combine user input with searching operations.

COMMAND

```
#!/bin/bash
```

```
echo "Enter filename:"
```

```
read filename
```

```
echo "Enter search pattern:"
```

```
read pattern
```

```
count=$(grep -c "$pattern" "$filename")
```

```
echo "Number of matching lines: $count"
```

OUTPUT

```
pavani@UBUNTU: ~/Documents/submission/linuxexp8
pavani@UBUNTU:~/Documents/submission/linuxexp8$ vim exp8task1grep
pavani@UBUNTU:~/Documents/submission/linuxexp8$ chmod +x exp8task1grep
pavani@UBUNTU:~/Documents/submission/linuxexp8$ ./exp8task1grep
Enter filename:
exp8i
Enter search pattern:
file
Number of matching lines: 6
pavani@UBUNTU:~/Documents/submission/linuxexp8$
```

Observations

- Processes can be controlled easily using signals.
- Monitoring tools show resource usage clearly.
- Background jobs help us multitask.
- Logs give important system information.

Conclusion

In this experiment, we learned how Shell Programming helps in process control and system management.

- **Signals** control processes.
- **Monitoring** checks system performance.

- **Communication and synchronization** make processes work together.
- **Job control** allows multitasking.
- **Logging** records system activities.

Thus, Shell Programming provides simple but powerful tools for managing processes in Linux.