

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA**

**SCIENCE LAB MANUAL**

**CS23431 – OPERATING SYSTEMS**

**(REGULATION 2023)**

# RAJALAKSHMI ENGINEERING COLLEGE Thandalam, Chennai-602015

Name: Shruthi S

Register No: 231801165

Year / Branch / Section: 2nd / AI&DS / FB

Semester: IV

Academic Year: 2024 - 2025

**INDEX**

|  |  |  |
| --- | --- | --- |
| Exp.No | Title | Page No |
| 1a | Installation and Configuration of Linux. | 4 |
| 1b | Basic Linux Commands. | 9 |
| 2a | Shell script - Arithmetic Operation -using expr command Basic Calculator. | 27 |
| 2b | Shell script - Check leap year using if-else. | 31 |
| 3a | Shell script - Reverse the number using while loop. | 32 |
| 3b | Shell script - Fibonacci series using for loop. | 33 |
| 4a | Text processing using Awk script - Employee average pay. | 34 |
| 4b | Text processing using Awk script - Results of an examination. | 36 |
| 5 | System calls –fork(), exec(), getpid(),opendir(), readdir(). | 38 |
| 6a | FCFS | 41 |
| 6b | SJF | 43 |
| 6c | Priority | 46 |
| 6d | Round Robin | 48 |
| 7 | Inter-process Communication using Shared Memory. | 51 |
| 8 | Producer Consumer using Semaphores. | 55 |
| 9 | Bankers Deadlock Avoidance algorithms. | 58 |
| 10a | Best Fit | 61 |
| 10b | First Fit | 63 |
| 11a | FIFO | 65 |
| 11b | LRU | 67 |
| 11c | Optimal | 70 |
| 12 | File Organization Technique- single and Two level directory. | 73 |

**Ex No: 1a**

**Date:** 21/1/25

# INSTALLATION AND CONFIGURATION OF LINUX

**AIM:**

To install and configure Linux operating system in a Virtual Machine.

**INSTALLATION/CONFIGURATION STEPS:**

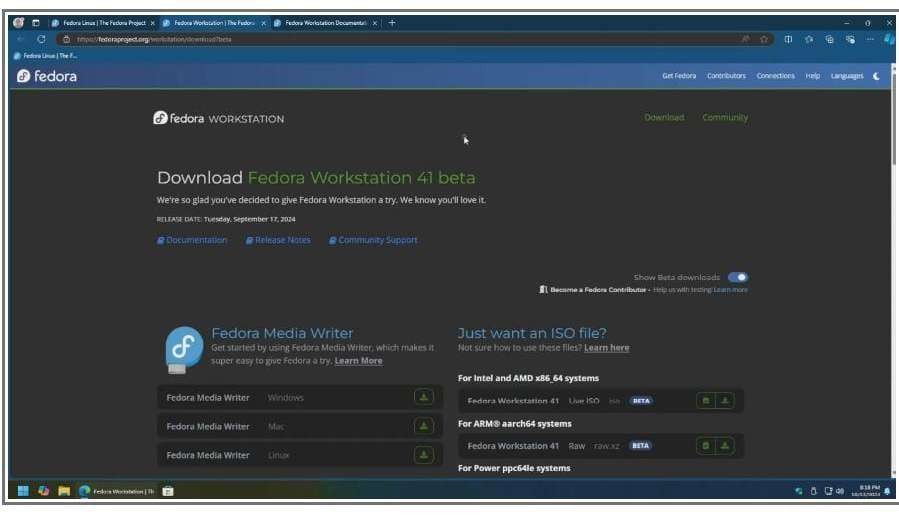
1. Install the required packages for virtualization dnf install xen virt-manager qemu libvirt 2. Configure xend to start up on boot systemctl enable virt-manager. service

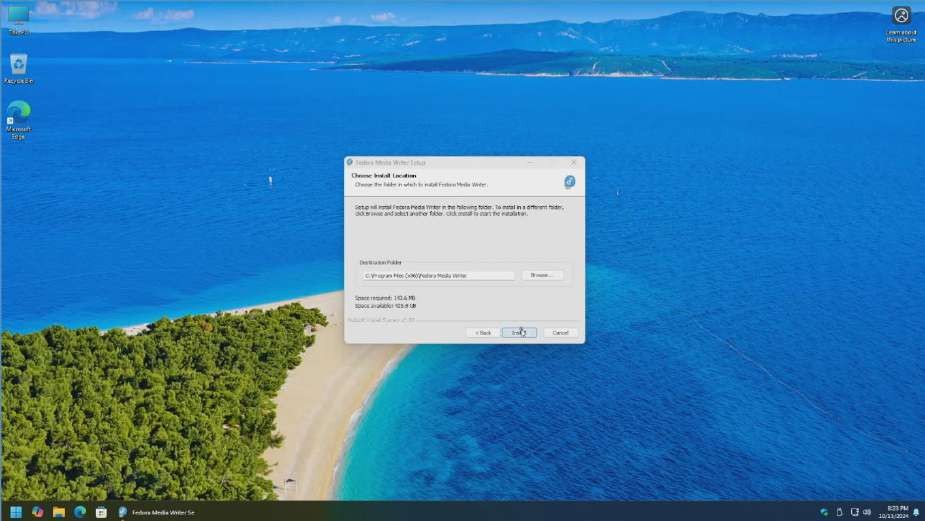
1. Reboot the machine

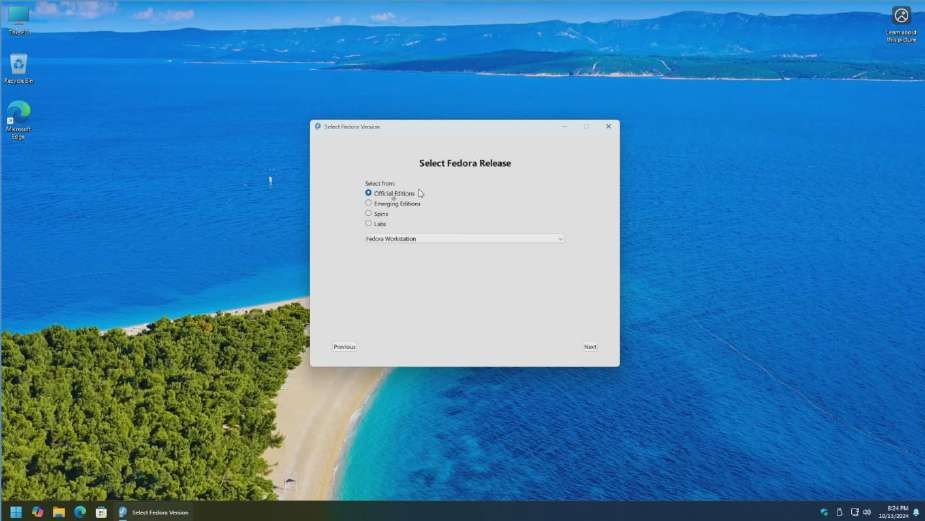
Reboot

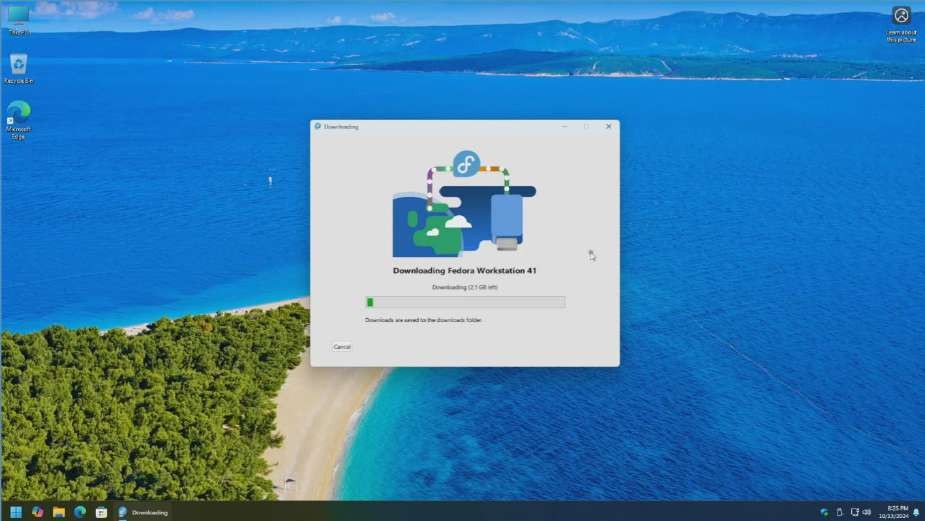
1. Create a Virtual machine by first running virt-manager virt-manager &
2. Click on File and then click to connect to localhost
3. In the base menu, right-click on the localhost (QEMU) to create a new VM 7. Select Linux ISO image
4. Choose puppy-linux.iso then the kernel version
5. Select CPU and RAM limits
6. Create default disk image to 8 GB
7. Click finish to create the new VM with PuppyLinux.

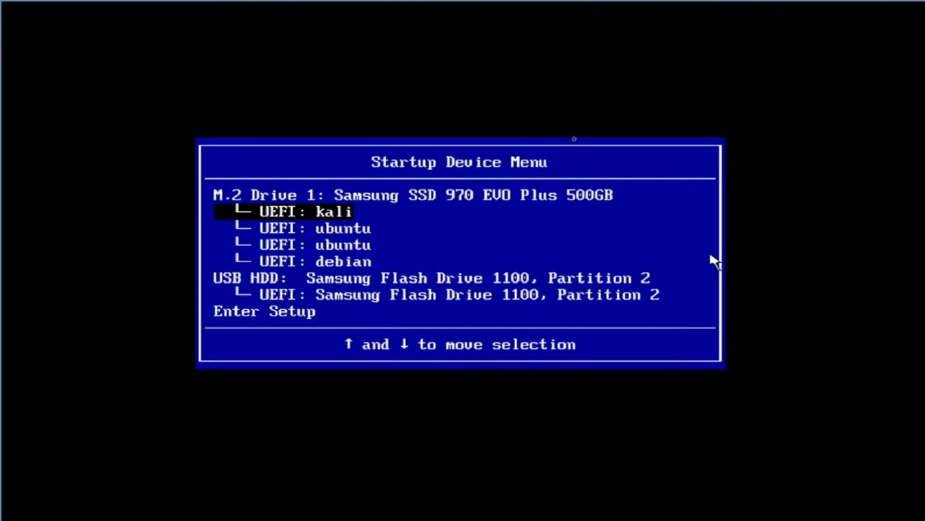
**OUTPUT:**

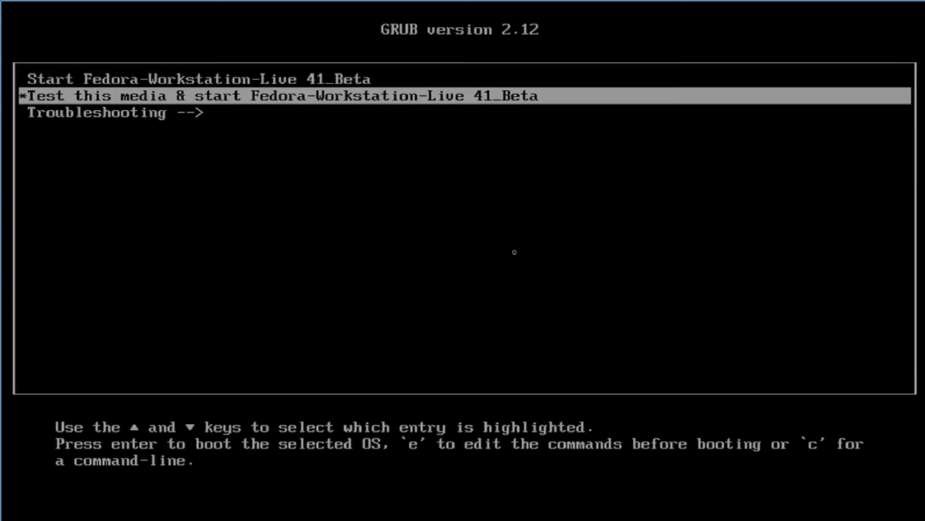


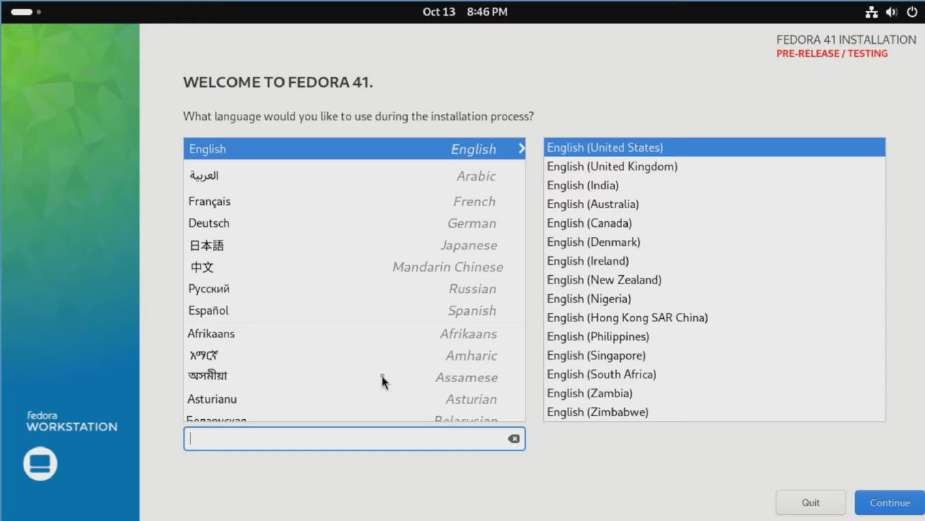


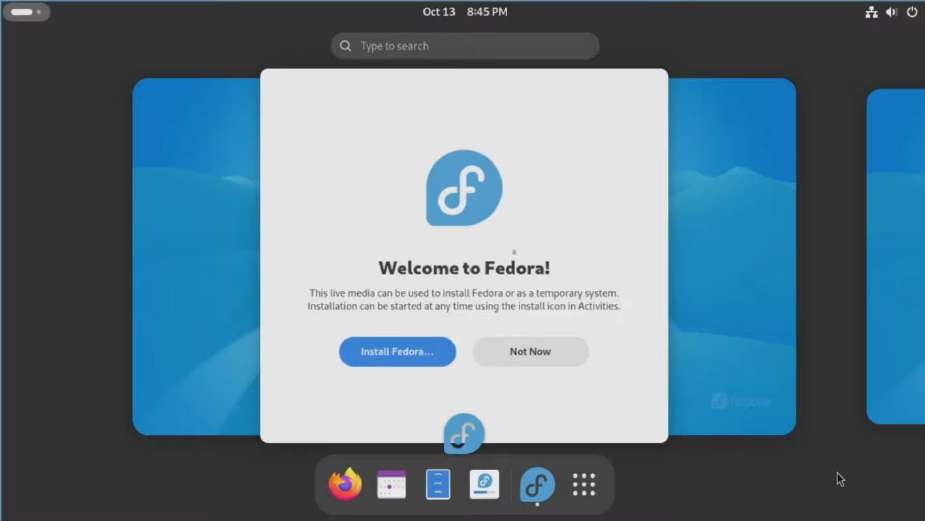












**RESULT:**

The Linux OS is Installed and Configured.

**Ex No: 1b**

**Date: 21/1/2025**

# BASIC LINUX COMMANDS

1.1 GENERAL PURPOSE COMMANDS

1. The ‘date’ command:

The date command displays the current date with day of week, month, day, time (24 hours clock) and the year.

SYNTAX: $ date

The date command can also be used with following format.

|  |  |  |
| --- | --- | --- |
| Format | Purpose | Example |
| + %m | To display only month | $ date + %m |
| + %h | To display month name | $ date + %h |
| + %d | To display day of month | $ date + %d |
| + %y | To display last two digits of the year | $ date + %y |
| + %H | To display Hours | $ date + %H |
| + %M | To display Minutes | $ date + %M |
| + %S | To display Seconds | $ date + %S |

1. The echo’command:

The echo command is used to print the message on the screen.

SYNTAX: $ echo

EXAMPLE: $ echo “God is Great” 3. The ‘cal’ command:

The cal command displays the specified month or year calendar.

SYNTAX: $ cal [month] [year] EXAMPLE: $ cal Jan 2012

1. The ‘bc’ command:

Unix offers an online calculator and can be invoked by the command bc.

SYNTAX: $ bc

EXAMPLE: bc –l

16/4

5/2

1. The ‘who’ command

The who command is used to display the data about all the users who are currently logged into the system. SYNTAX: $ who

1. The ‘who am i’ command

The who am i command displays data about login details of the user.

SYNTAX: $ who am i

1. The ‘id’ command

The id command displays the numerical value corresponding to your login.

SYNTAX: $ id

1. The ‘tty’ command

The tty (teletype) command is used to know the terminal name that we are using.

SYNTAX: $ tty

1. The ‘clear’ command

The clear command is used to clear the screen of your terminal.

SYNTAX: $ clear

1. The ‘man’ command

The man command gives you complete access to the Unix commands.

SYNTAX: $ man [command]

1. The ‘ps’ command

The ps command is used to the process currently alive in the machine with the 'ps' (process status) command, which displays information about process that are alive when you run the command. 'ps;' produces a snapshot of machine activity.

SYNTAX: $ ps

EXAMPLE: $ ps

$ ps –e

$ps -aux

1. The ‘uname’ command

The uname command is used to display relevant details about the operating system on the standard output.

-m -> Displays the machine id (i.e., name of the system hardware) -n -> Displays the name of the network node. (host name) -r -> Displays the release number of the operating system.

-s -> Displays the name of the operating system (i.e.. system name)

-v -> Displays the version of the operating system.

-a -> Displays the details of all the above five options.

SYNTAX: $ uname [option]

EXAMPLE: $ uname -a

1.2 DIRECTORY COMMANDS

1. The ‘pwd’ command:

The pwd (print working directory) command displays the current working directory. SYNTAX: $ pwd

1. The ‘mkdir’ command:

The mkdir is used to create an empty directory in a disk.

SYNTAX: $ mkdir dirname

EXAMPLE: $ mkdir receee

1. The ‘rmdir’ command:

The rmdir is used to remove a directory from the disk. Before removing a directory, the directory must be empty (no files and directories).

SYNTAX: $ rmdir dirname

EXAMPLE: $ rmdir receee

1. The ‘cd’ command:

The cd command is used to move from one directory to another.

SYNTAX: $ cd dirname

EXAMPLE: $ cd receee

1. The ‘ls’ command:

The ls command displays the list of files in the current working directory.

SYNTAX: $ ls

EXAMPLE: $ ls

$ ls –l

$ ls –a

1.3 FILE HANDLING COMMANDS

1. The ‘cat’ command:

The cat command is used to create a file.

SYNTAX: $ cat > filename

EXAMPLE: $ cat > rec

1. The ‘Display contents of a file’ command:

The cat command is also used to view the contents of a specified file.

SYNTAX: $ cat filename

1. The ‘cp’ command:

The cp command is used to copy the contents of one file to another and copies the file from one place to another.

SYNTAX: $ cp oldfile newfile

EXAMPLE: $ cp cse ece

1. The ‘rm’ command:

The rm command is used to remove or erase an existing file

SYNTAX: $ rm filename

EXAMPLE: $ rm rec

$ rm –f rec

Use option –fr to delete recursively the contents of the directory and its subdirectories.

1. The ‘mv’ command:

The mv command is used to move a file from one place to another. It removes a specified file from its original location and places it in specified location.

SYNTAX: $ mv oldfile newfile

EXAMPLE: $ mv cse eee

1. The ‘file’ command:

The file command is used to determine the type of file.

SYNTAX: $ file filename

EXAMPLE: $ file receee

1. The ‘wc’ command:

The wc command is used to count the number of words, lines and characters in a file. SYNTAX: $ wc filename

EXAMPLE: $ wc receee

1. The ‘Directing output to a file’ command:

The ls command lists the files on the terminal (screen). Using the redirection operator ‘>’ we can send the output to file instead of showing it on the screen.

SYNTAX: $ ls > filename

EXAMPLE: $ ls > cseeee

1. The ‘pipes’ command:

The Unix allows us to connect two commands together using these pipes. A pipe ( | ) is an mechanism by which the output of one command can be channeled into the input of another command. SYNTAX: $ command1 | command2 EXAMPLE: $ who | wc -l

1. The ‘tee’ command:

While using pipes, we have not seen any output from a command that gets piped into another command. To save the output, which is produced in the middle of a pipe, the tee command is very useful. SYNTAX: $ command | tee filename

EXAMPLE: $ who | tee sample | wc -l

1. The ‘Metacharacters of unix’ command:

Metacharacters are special characters that are at higher and abstract level compared to most of other characters in Unix. The shell understands and interprets these metacharacters in a special way. \* - Specifies number of characters

?- Specifies a single character

[ ]- used to match a whole set of file names at a command line.

! – Used to Specify Not

EXAMPLE:

$ ls r\*\* - Displays all the files whose name begins with ‘r’

$ ls ?kkk - Displays the files which are having ‘kkk’, from the second characters irrespective of the first character.

$ ls [a-m] – Lists the files whose names begins alphabets from ‘a’ to ‘m’

$ ls [!a-m] – Lists all files other than files whose names begins alphabets from ‘a’ to ‘m’

12. The ‘File permissions’ command:

File permission is the way of controlling the accessibility of file for each of three users namely Users, Groups and Others.

There are three types of file permissions are available, they are r-read w-write x-execute

The permissions for each file can be divided into three parts of three bits each.

|  |  |
| --- | --- |
| First three bits | Owner of the file |
| Next three bits | Group to which the owner of the file belongs |
| Last three bits | Others |

EXAMPLE: $ ls college

-rwxr-xr-- 1 Lak std 1525 jan10 12:10 college

Where,

-rwx The file is readable, writable and executable by the owner of the file.

Lak Specifies Owner of the file.

r-x Indicates the absence of the write permission by the Group owner of the file. Std Is the Group Owner of the file. r-- Indicates read permissions for others.

1. The ‘chmod’ command:

The chmod command is used to set the read, write and execute permissions for all categories of users for file.

SYNTAX: $ chmod category operation permission file

|  |  |  |
| --- | --- | --- |
| Category | Operation | permission |
| u-users | + assign | r-read |
| g-group | -Remove | w-write |
| o-others | = assign absolutely | x-execute |
| a-all |  |  |

EXAMPLE:

$ chmod u –wx college

Removes write & execute permission for users for ‘college’ file. $ chmod u +rw, g+rw college

Assigns read & write permission for users and groups for ‘college’ file. $ chmod g=wx college

Assigns absolute permission for groups of all read, write and execute permissions for ‘college’ file.

1. The ‘Octal Notations’ command:

The file permissions can be changed using octal notations also. The octal notations for file permission are

|  |  |
| --- | --- |
| Read permission | 4 |
| Write permission | 2 |

EXAMPLE:

$ chmod 761 college

|  |  |
| --- | --- |
| Execute permission | 1 |

Assigns all permission to the owner, read and write permissions to the group and only executable permission to the others for ‘college’ file.

1.4 GROUPING COMMANDS

1. The ‘semicolon’ command:

The semicolon(;) command is used to separate multiple commands at the command line. SYNTAX: $ command1;command2;command3…………….;commandn EXAMPLE: $ who;date

1. The ‘&&’ operator:

The ‘&&’ operator signifies the logical AND operation in between two or more valid Unix commands.It means that only if the first command is successfully executed, then the next command will executed.

SYNTAX: $ command1 && command && command3…………….&&commandn EXAMPLE: $ who && date.

1. The ‘||’ operator:

The ‘||’ operator signifies the logical OR operation in between two or more valid Unix commands.It means, that only if the first command will happen to be un successfully,it will continue to execute next commands.

SYNTAX: $ command1 || command || command3…………….||commandn EXAMPLE: $ who || date

1.5 FILTERS

1. The head filter

It displays the first ten lines of a file.

SYNTAX: $ head filename

EXAMPLE: $ head college Display the top ten lines. $ head -5 college Display the top five lines.

1. The tail filter

It displays ten lines of a file from the end of the file.

SYNTAX: $ tail filename

EXAMPLE: $ tail college Display the last ten lines. $tail -5 college Display the last five lines.

1. The more filter:

The pg command shows the file page by page.

SYNTAX: $ ls –l | more

1. The ‘grep’ command:

This command is used to search for a particular pattern from a file or from the standard input and display those lines on the standard output. “Grep” stands for “global search for regular expression.”

SYNTAX: $ grep [pattern] [file\_name]

EXAMPLE: $ cat> student

Arun cse

Ram ece

Kani cse

$ grep “cse” student

Arun cse

Kani cse

1. The ‘sort’ command:

The sort command is used to sort the contents of a file. The sort command reports only to the screen, the actual file remains unchanged.

SYNTAX: $ sort filename EXAMPLE: $ sort college OPTIONS:

|  |  |
| --- | --- |
| Command | Purpose |
| Sort –r college | Sorts and displays the file contents in reverse order |
| Sort –c college | Check if the file is sorted |
| Sort –n college | Sorts numerically |
| Sort –m college | Sorts numerically in reverse order |

|  |  |
| --- | --- |
| Sort –u college | Remove duplicate records |
| Sort –l college | Skip the column with +1 (one) option.Sorts according to second column |

1. The ‘nl’ command:

The nl filter adds lines numbers to a file and it displays the file and not provides access to edit but simply displays the contents on the screen.

SYNTAX: $ nl filename

EXAMPLE: $ nl college

1. The ‘cut’ command:

We can select specified fields from a line of text using cut command.

SYNTAX: $ cut -c filename EXAMPLE: $ cut -c college OPTION:

-c – Option cut on the specified character position from each line.

1.5 OTHER ESSENTIAL COMMANDS

1. free

Display amount of free and used physical and swapped memory system. synopsis- free [options] example

[root@localhost ~]# free -t total used free shared buff/cache available Mem: 4044380 605464 2045080 148820 1393836 3226708 Swap: 2621436 0 2621436

Total: 6665816 605464 4666516

1. top

It provides a dynamic real-time view of processes in the system.

synopsis- top [options] example

[root@localhost ~]# top top - 08:07:28 up 24 min, 2 users, load average: 0.01, 0.06, 0.23 Tasks: 211 total, 1 running, 210 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.8 us, 0.3 sy, 0.0 ni, 98.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 4044380 total, 2052960 free, 600452 used, 1390968 buff/cache KiB Swap: 2621436 total,

2621436 free, 0 used. 3234820 avail Mem PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1105 root 20 0 175008 75700 51264 S 1.7 1.9 0:20.46 Xorg 2529 root 20 0 80444 32640 24796 S 1.0 0.8 0:02.47 gnome-term

3. ps

It reports the snapshot of current processes synopsis- ps [options] example

[root@localhost ~]# ps -e

PID TTY TIME CMD

1. ? 00:00:03 systemd
2. ? 00:00:00 kthreadd
3. ? 00:00:00 ksoftirqd/0

1. vmstat

It reports virtual memory statistics synopsis- vmstat [options] example

[root@localhost ~]# 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 1879368 1604 1487116 0 0 64 7 72 140 1 0 97 1 0

1. df

It displays the amount of disk space available in file-system.

Synopsis- df [options] example

[root@localhost ~]# df

Filesystem 1K-blocks Used Available Use% Mounted on

devtmpfs 2010800 0 2010800 0% /dev tmpfs 2022188 148 2022040 1% /dev/shm tmpfs 2022188 1404 2020784 1% /run /dev/sda6 487652 168276 289680 37% /boot

1. ping

It is used verify that a device can communicate with another on network. PING stands for Packet Internet Groper. synopsis- ping [options]

[root@localhost ~]# ping 172.16.4.1

PING 172.16.4.1 (172.16.4.1) 56(84) bytes of data.

64 bytes from 172.16.4.1: icmp\_seq=1 ttl=64 time=0.328 ms

64 bytes from 172.16.4.1: icmp\_seq=2 ttl=64 time=0.228 ms

64 bytes from 172.16.4.1: icmp\_seq=3 ttl=64 time=0.264 ms 64 bytes from 172.16.4.1: icmp\_seq=4 ttl=64 time=0.312 ms

--- 172.16.4.1 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3000ms rtt min/avg/max/mdev =

0.228/0.283/0.328/0.039 ms

7. ifconfig

It is used configure network interface. synopsis- ifconfig [options] example

root@localhost ~]# ifconfig

enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500 inet 172.16.6.102 netmask

255.255.252.0 broadcast 172.16.7.255 inet6 fe80::4a0f:cfff:fe6d:6057 prefixlen 64 scopeid 0x20<link> ether 48:0f:cf:6d:60:57 txqueuelen 1000 (Ethernet)

RX packets 23216 bytes 2483338 (2.3 MiB)

RX errors 0 dropped 5 overruns 0 frame 0

TX packets 1077 bytes 107740 (105.2 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 8.

traceroute

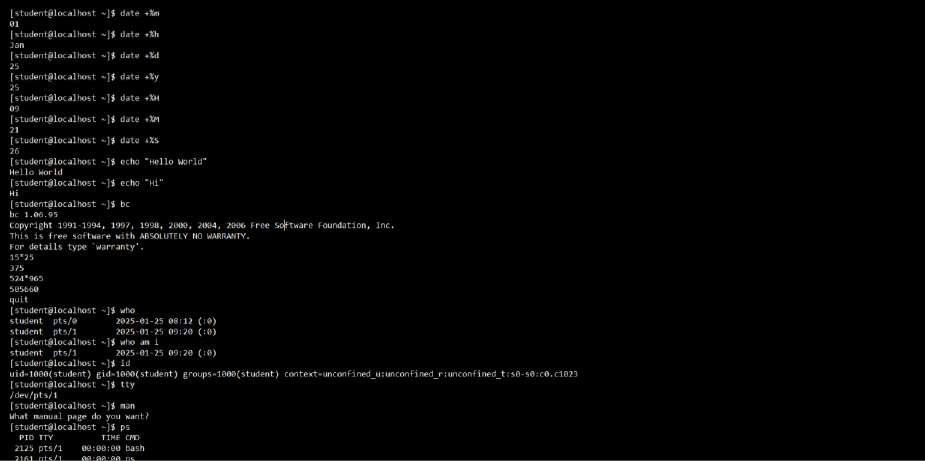
It tracks the route the packet takes to reach the destination. synopsis- traceroute [options] example

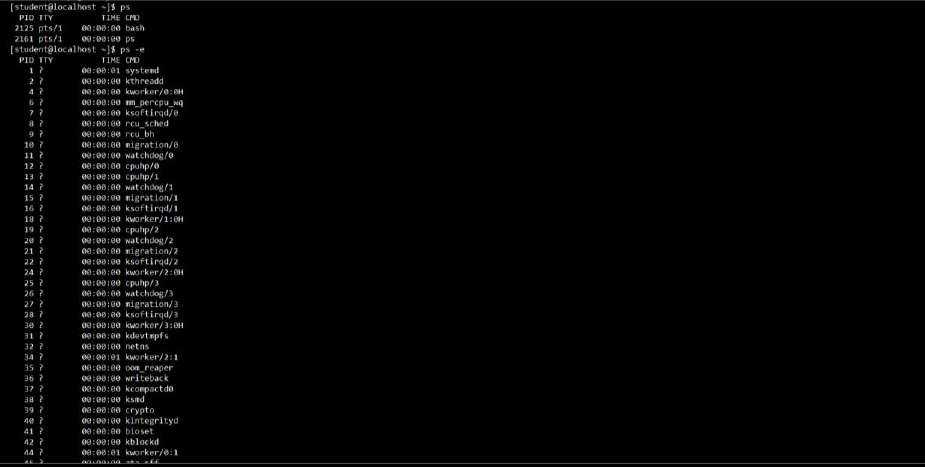
[root@localhost ~]# traceroute www.rajalakshmi.org

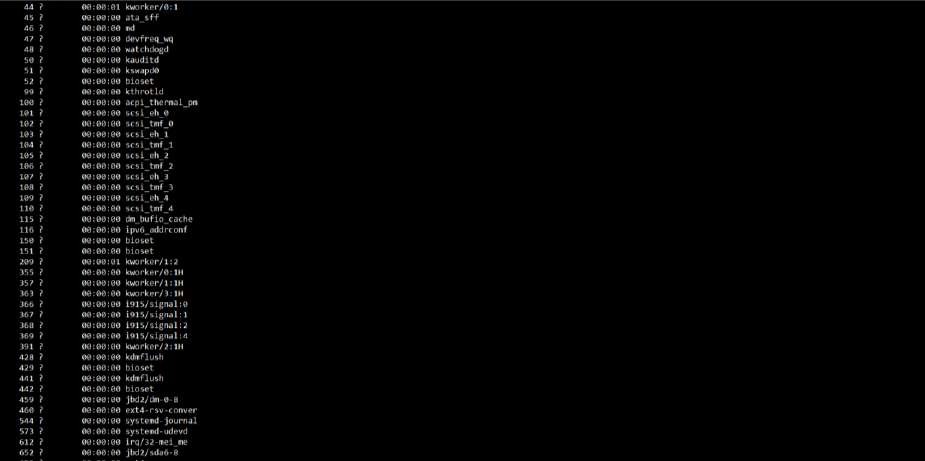
traceroute to www.rajalakshmi.org (220.227.30.51), 30 hops max, 60 byte packets 1 gateway (172.16.4.1) 0.299 ms 0.297 ms 0.327 ms 2

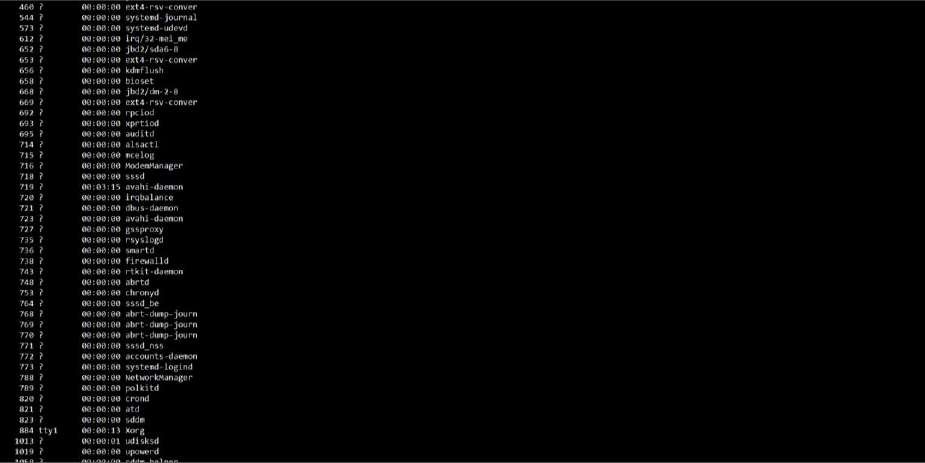
220.225.219.38 (220.225.219.38) 6.185 ms 6.203 ms 6.189 ms

**OUTPUT:**

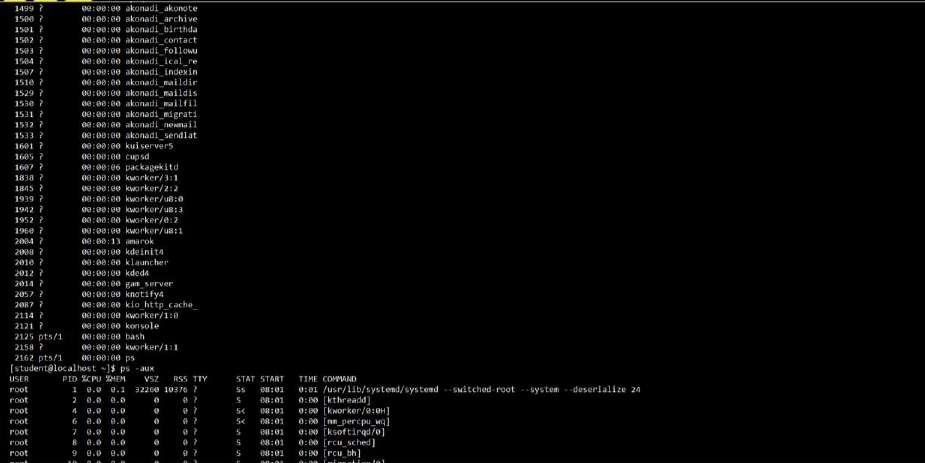


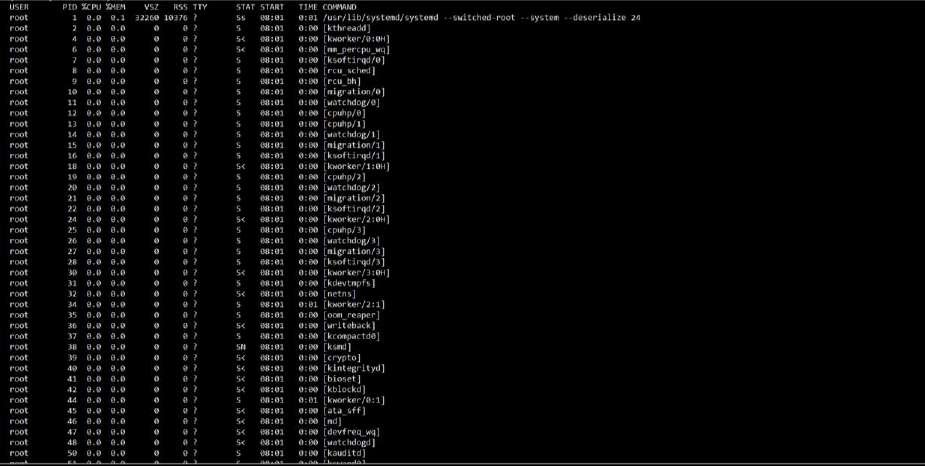


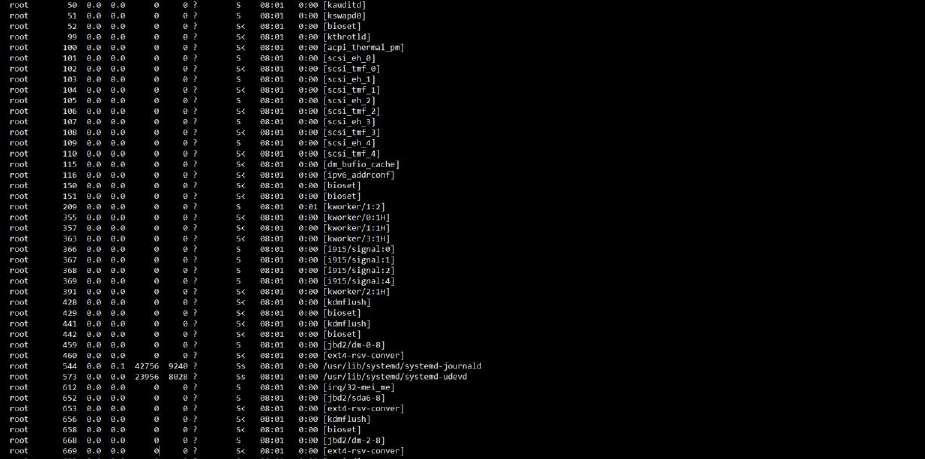


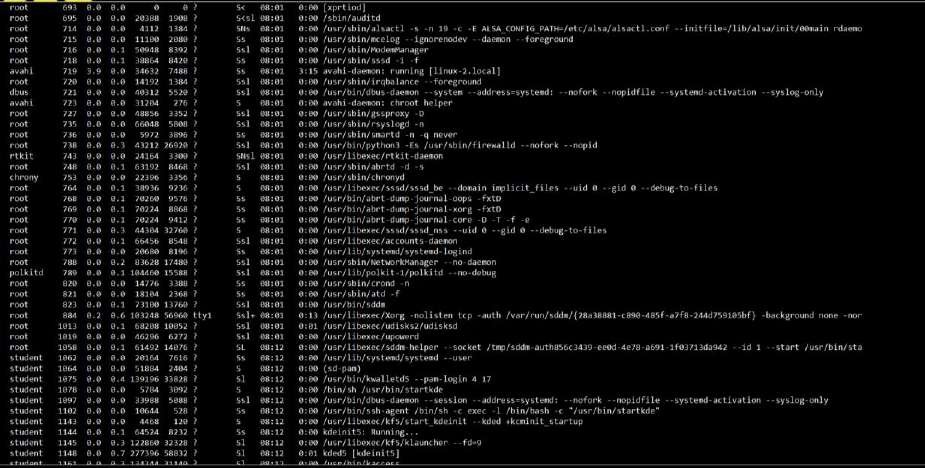


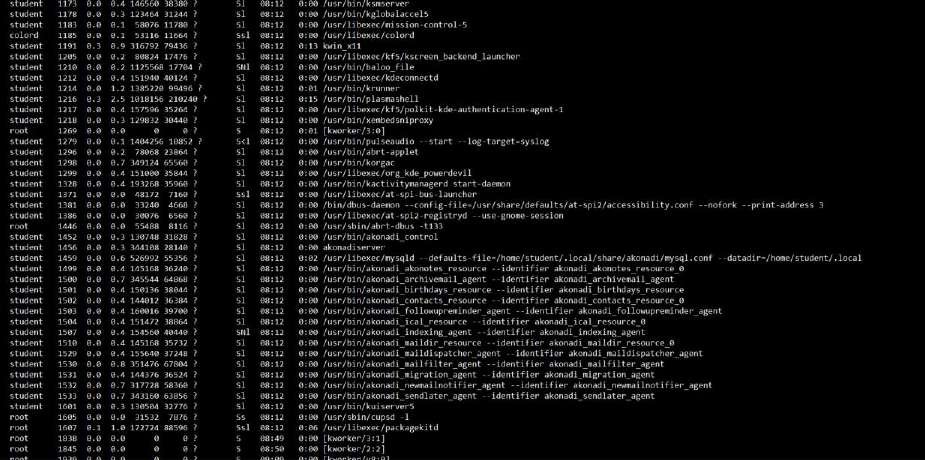


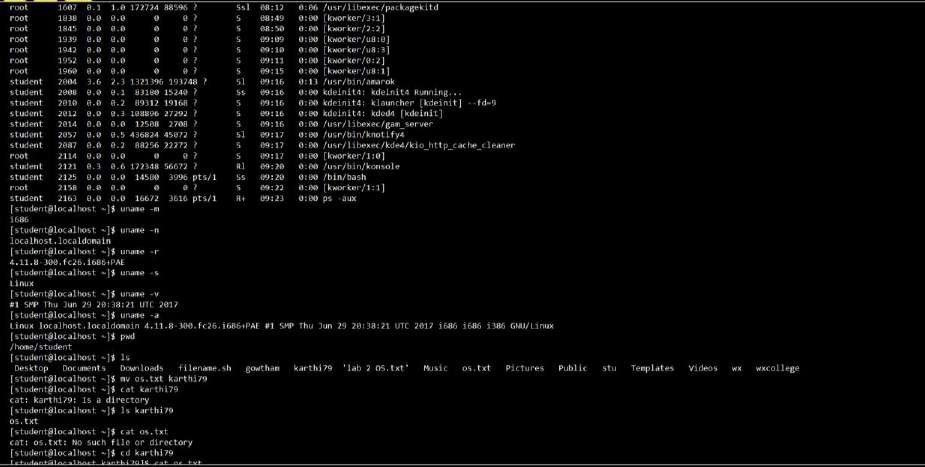


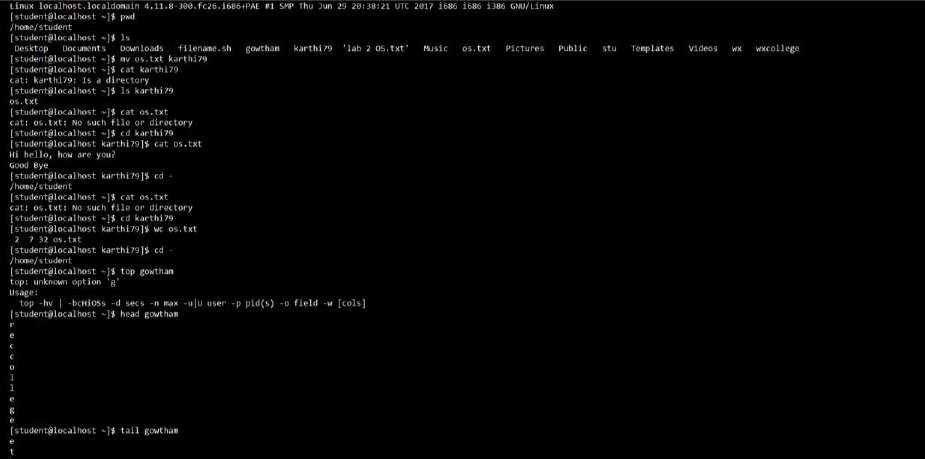


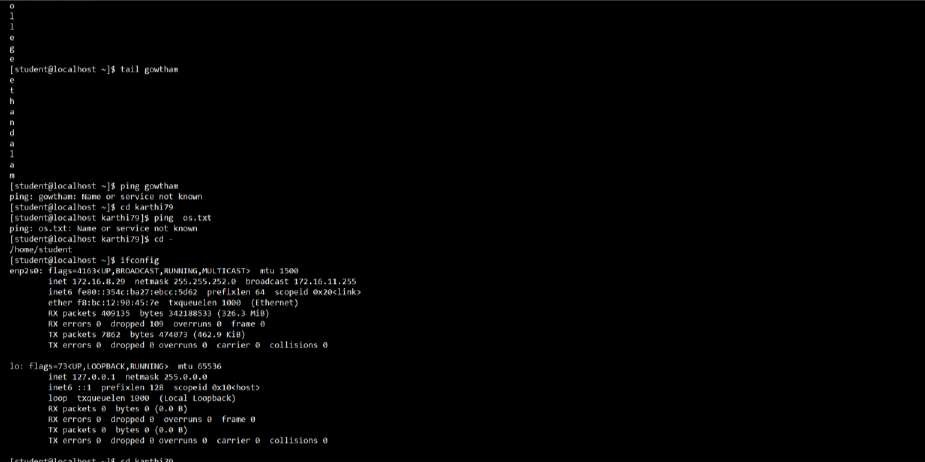


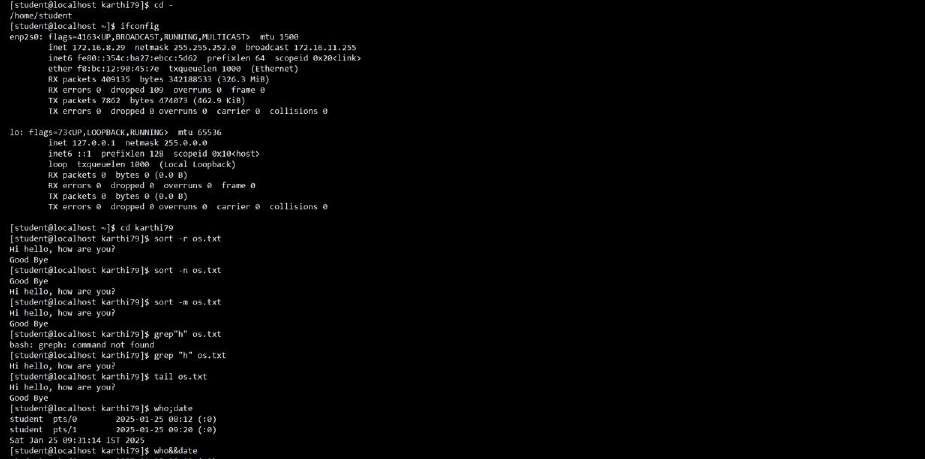


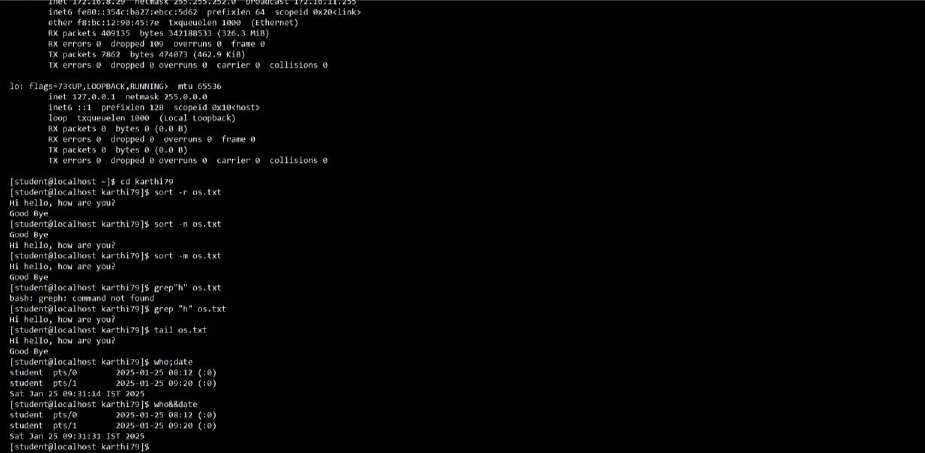












**RESULT:**

Thus, the program of basic Linux commands has been executed and the output has been verified.

**Ex. No: 2a**

**Date: 24/1/25**

# Shell Script

**AIM:**

To write a Shell script to display a basic calculator.

**PROGRAM:**

#!/bin/bash

while true; do

echo "========================"

echo " Basic Calculator"

echo "========================"

echo "1. Addition" echo "2. Subtraction" echo "3. Multiplication" echo "4. Division" echo "5. Exit" echo -n "Choose an option (1-5): " read choice

if [[ $choice -eq 5 ]]; then echo "Exiting Calculator. Goodbye!"

exit 0 fi

echo -n "Enter first number: " read num1 echo -n "Enter second number: " read num2

case $choice in 1) result=$((num1 + num2)) echo "Result: $num1 + $num2 = $result"

;;

1. result=$((num1 - num2)) echo "Result: $num1 - $num2 = $result"

;;

1. result=$((num1 \* num2)) echo "Result: $num1 \* $num2 = $result"

;;

1. if [[ $num2 -eq 0 ]]; then

echo "Error: Division by zero is not allowed!" else

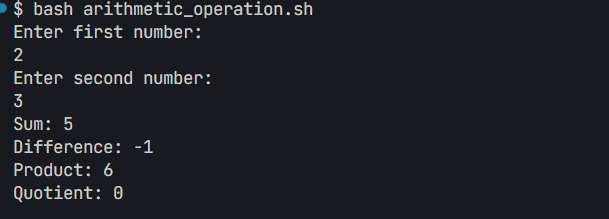
result=$(awk "BEGIN {print $num1 / $num2}") echo "Result: $num1 / $num2 = $result" fi ;;

\*) echo "Invalid option! Please choose between 1-5."

;; esac

echo "------------------------" echo "" done

**OUTPUT:**



**RESULT:**

Thus, the basic calculator program was successfully implemented using shell scripting.

**Ex. No: 2b**

**Date: 24/1/25**

# Shell Script

**AIM:**

To write a Shellscript to test given year is leap or not using conditional statement

**PROGRAM:**

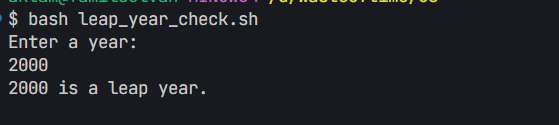
#!/bin/bash

read -p "Enter year: " year

if (( year % 4 == 0 && year % 100 != 0 )) || (( year % 400 == 0 )); then echo "$year is a Leap Year" else

echo "$year is not a Leap Year" fi

**OUTPUT:**



**RESULT:**

Thus, the leap year program was successfully implemented using shell scripting.

**Ex. No: 3a Date: 28/1/25**

# Shell Script – Reverse of Digit

**AIM:**

To write a Shell script to reverse a given digit using looping statement.

**PROGRAM:**

#!/bin/bash

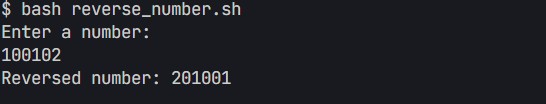
read -p "Enter a number: " num

reverse=0 while [ $num -gt 0 ]; do digit=$(( num % 10 ))

reverse=$(( reverse \* 10 + digit )) num=$(( num / 10 )) done

echo "Reversed number: $reverse"

**OUTPUT:**



**RESULT:**

The shell script to reverse a given digit is successfully implemented.

**Ex. No: 3b Date: 28/1/25**

# Shell Script – Fibonacci Series

**AIM:**

To write a Shell script to generate a Fibonacci series using a for loop.

**PROGRAM:**

#!/bin/bash

read -p "Enter the number of terms: " n

a=0 b=1

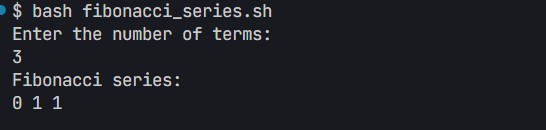
echo "Fibonacci Series:" for (( i=0; i<n; i++ )); do echo -n "$a " temp=$((a + b)) a=$b

b=$temp

done

echo

**OUTPUT**



**RESULT:**

The Shell Script to generate the Fibonacci series is successfully implemented.

**Ex. No: 4a**

**Date: 3/2/25**

# EMPLOYEE AVERAGE PAY

**AIM:**

To find out the average pay of all employees whose salary is more than 6000 and no. of days worked is

more than 4.

**ALGORITHM:**

1. Create a flat file emp.dat for employees with their name, salary per day and number of days worked and save it.
2. Create an awk script emp.awk
3. For each employee record do
4. If the Salary is greater than 6000 and number of days worked is more than 4, then print the name and salary earned
5. Compute total pay of employee

4. Print the total number of employees satisfying the criteria and their average pay.

**PROGRAM:**

#!/usr/bin/awk -f

BEGIN {

count = 0; total\_pay = 0;

}

{ salary = $2; days = $3;

if (salary > 6000 && days > 4) { pay = salary \* days;

print "Employee:", $1, "Total Pay:", pay; total\_pay += pay; count++;

}

}

END {

if (count > 0) {

avg\_pay = total\_pay / count; print "\nTotal Employees:", count; print "Total Pay:", total\_pay; print "Average Pay:", avg\_pay;

} else {

print "No employees satisfy the criteria.";

}

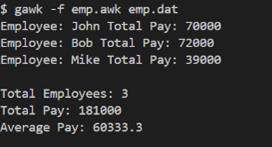
}

**INPUT: John 7000 10**

**Alice 5000 12**

**Bob 8000 9**

**Mike 6500 6 OUTPUT:**



**RESULT:**

To find the average salary whose salary is above 6000 is successfully implemented.

**Ex. No: 4b**

**Date: 3/2/25**

# RESULTS OF EXAMINATION

**AIM:**

To print the pass/fail status of a student in a class.

**ALGORITHM:**

1. Read the data from file
2. Get a data from each column
3. Compare the all subject marks column
   1. If marks less than 45 then print Fail
   2. else print Pass

**PROGRAM:**

//marks.awk

#!/usr/bin/gawk -f

{ name = $1; pass = 1; for (i = 2; i <= NF; i++) { if ($i < 45) { pass = 0; break;} } if(pass) { print name, "Pass"; } else { print name, "Fail";}

}

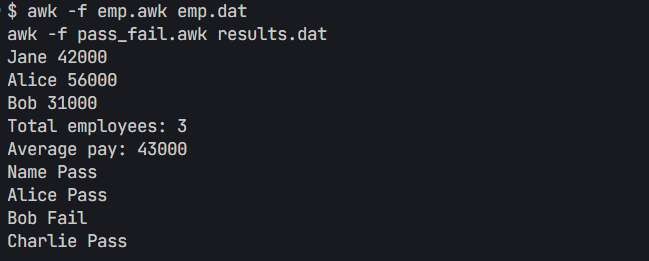
**INPUT:**

**//marks.dat**

John 50 60 45 70 80 Alice 40 55 30 65 75 Bob 80 85 90 78 88

Mike 35 40 50 60 45

**OUTPUT:**



**RESULT:**

To print the Pass/Fail Status of a student in a class is successfully implemented.

**Ex. No: 5**

**Date: 8/2/25**

# System Calls Programming

**AIM:**

To experiment system calls using fork(), execlp() and pid() functions.

**ALGORITHM:**

1. **Start**
2. **Include Header Files** o Include stdio.h for input/output functions o Include stdlib.h for general utility functions
3. **Variable Declaration** o Declare an integer variable pid to store the process ID returned by fork()
4. **Create a New Process** o Call the fork() function and assign its return value to pid  If fork() returns:
   * + - -1: Process creation failed
       - 0: This is the **child** process
       - A positive integer: This is the **parent** process
5. **Print Statement Executed by Both Processes** o Print: "THIS LINE EXECUTED TWICE"
6. **Check for Process Creation Failure** o If pid == -1:
   * Print: "CHILD PROCESS NOT CREATED"
   * Exit the program using exit(0)
7. **Child Process Execution Block** o If pid == 0:
   * + Print:
     + "Process ID of child: " followed by getpid()
     + "Parent Process ID of child: " followed by getppid()
8. **Parent Process Execution Block** o If pid > 0:
   * + Print:
     + "Process ID of parent: " followed by getpid()
     + "Parent’s Parent Process ID: " followed by getppid()
9. **Final Print Statement (Executed by Both Processes)**

o Print: objectives

IT CAN BE EXECUTED TWICE

1. **End**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() { int pid; pid = fork();

printf("This Line Executed Twice\n");

if (pid < 0) {

printf("Child Process Not Created\n"); exit(1);

}

if (pid == 0) { printf("Child Process:\n"); printf("Process ID: %d\n", getpid()); printf("Parent Process ID: %d\n", getppid()); execlp("/bin/ls", "ls", NULL); perror("execlp failed"); exit(1);

} else { // Parent process printf("Parent Process:\n"); printf("Process ID: %d\n", getpid()); printf("Parent's Parent Process ID: %d\n", getppid()); printf("Child Process Completed\n");

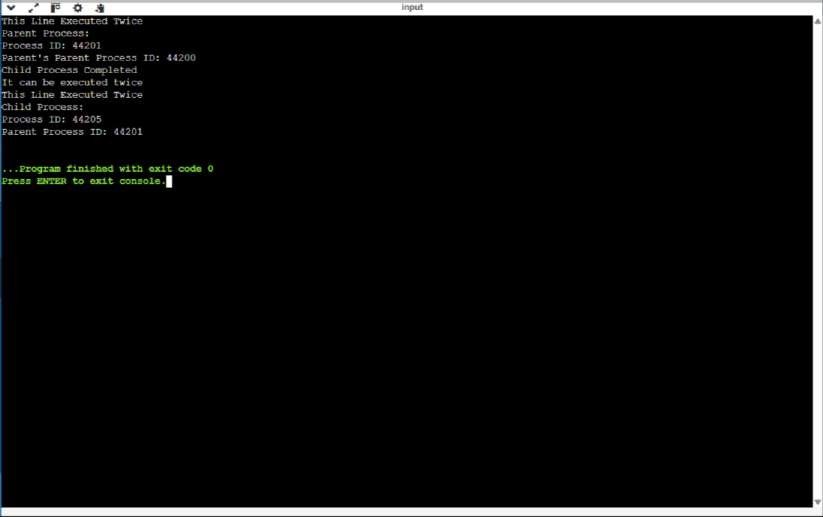
}

printf("It can be executed twice\n");

return 0;

}

**OUTPUT:**



**RESULT:**

To Program is implemented using fork(),execlp() and pid() Functions.

**Ex. No: 6a**

**Date: 15/2/25**

# FIRST COME FIRST SERVE

**AIM:**

To implement First-come First- serve (FCFS) scheduling technique

ALGORITHM**:**

1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process
5. Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time.

**PROGRAM:**

#include <stdio.h>

int main() { int pid[15], bt[15], wt[15], n;

float twt = 0, ttat = 0;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter process ID of all the processes:\n");

for (int i = 0; i < n; i++) { scanf("%d", &pid[i]);

}

printf("Enter burst time of all the processes:\n");

for (int i = 0; i < n; i++) { scanf("%d", &bt[i]);

}

wt[0] = 0;

// Calculate waiting time for all other processes

for (int i = 1; i < n; i++) { wt[i] = wt[i - 1] + bt[i - 1];

}

printf("\nProcess ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

for (int i = 0; i < n; i++) { int tat = bt[i] + wt[i]; twt += wt[i]; ttat += tat;

printf("%d\t\t%d\t\t%d\t\t%d\n", pid[i], bt[i], wt[i], tat);

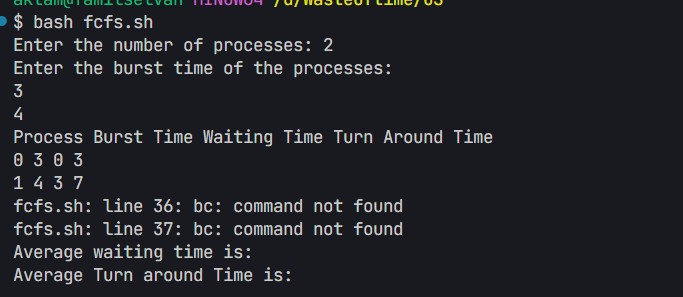
}

printf("\nAverage waiting time = %.2f\n", twt / n); printf("Average turnaround time = %.2f\n", ttat / n);

return 0;

}

**OUTPUT:**



**RESULT:**

The Program of first come first serve is successfully implemented.

**Ex. No: 6b**

**Date: 15/2/25**

# SHORTEST JOB FIRST

**AIM:**

To implement the Shortest Job First (SJF) scheduling technique

**ALGORITHM:**

1. Declare the structure and its elements.
2. Get a number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero.
5. Sort based on the burst time of all processes in ascending order.
6. Calculate the waiting time and turnaround time for each process. 7. Calculate the average waiting time and average turnaround time.

8. Display the results.

**PROGRAM:**

#include <stdio.h>

int main() {

int A[100][4]; // A[i][0]=PID, A[i][1]=BT, A[i][2]=WT, A[i][3]=TAT int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Burst Time:\n"); for (i = 0; i < n; i++) { printf("P%d: ", i + 1); scanf("%d", &A[i][1]); A[i][0] = i + 1; // Assign process ID

}

for (i = 0; i < n; i++) { index = i; for (j = i + 1; j < n; j++) { if (A[j][1] < A[index][1]) index = j;

}

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0]; A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

for (i = 1; i < n; i++) { A[i][2] = 0;

for (j = 0; j < i; j++) {

A[i][2] += A[j][1];

}

total += A[i][2];

}

avg\_wt = (float) total / n;

total = 0;

printf("\nProcess\tBT\tWT\tTAT\n"); for (i = 0; i < n; i++) {

A[i][3] = A[i][1] + A[i][2]; // TAT = BT + WT total += A[i][3];

printf("P%d\t%d\t%d\t%d\n", A[i][0], A[i][1], A[i][2], A[i][3]);

}

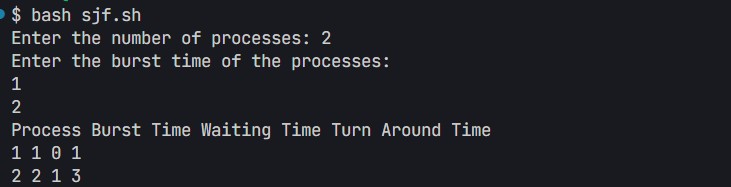
avg\_tat = (float) total / n;

printf("\nAverage Waiting Time = %.2f", avg\_wt); printf("\nAverage Turnaround Time = %.2f\n", avg\_tat);

return 0;

}

**OUTPUT:**



**RESULT:**

The Program Shortest Job First is successfully implemented.

**Ex. No: 6c**

**Date: 16/2/25**

# PRIORITY SCHEDULING

**AIM:**

To implement a priority scheduling technique

**ALGORITHM:**

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of the process.
3. Sort based on burst time of all processes in ascending order based on priority 4. Calculate the total waiting time and total turnaround time for each process
4. Display the process name & burst time for each process.
5. Display the total waiting time, average waiting time, turnaround time.

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

void swap(int \*a, int \*b) { int temp = \*a; \*a = \*b;

\*b = temp;

}

int main() { int n;

printf("Enter number of processes: "); scanf("%d", &n);

int \*burst = (int\*)malloc(n \* sizeof(int)); int \*priority = (int\*)malloc(n \* sizeof(int)); int \*pid = (int\*)malloc(n \* sizeof(int)); int total\_wait = 0, total\_turnaround = 0;

for (int i = 0; i < n; i++) {

printf("Enter Burst Time and Priority for Process %d: ", i + 1); scanf("%d %d", &burst[i], &priority[i]);

pid[i] = i + 1;

}

for (int i = 0; i < n - 1; i++) { for (int j = i + 1; j < n; j++) { if (priority[j] > priority[i]) { swap(&priority[i], &priority[j]); swap(&burst[i], &burst[j]);

swap(&pid[i], &pid[j]);

}

}

}

int wait\_time = 0;

printf("\nProcess Burst Time Wait Time Turnaround Time\n");

for (int i = 0; i < n; i++) {

int turnaround\_time = wait\_time + burst[i]; total\_wait += wait\_time;

total\_turnaround += turnaround\_time;

printf("P%d %d %d %d\n", pid[i], burst[i], wait\_time, turnaround\_time);

wait\_time += burst[i];

}

printf("\nAverage Waiting Time: %.2f\n", (float)total\_wait / n); printf("Average Turnaround Time: %.2f\n", (float)total\_turnaround / n);

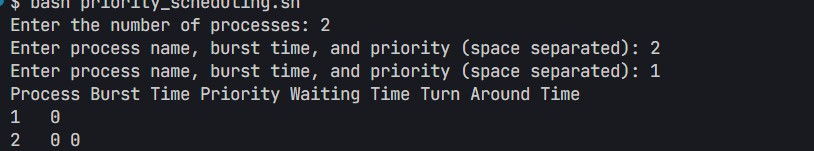
free(burst); free(priority);

free(pid);

return 0;

}

**OUTPUT:**



**RESULT:**

The Program of Priority scheduling is successfully implemented.

**Ex. No: 6d**

**Date: 16/2/25**

# ROUND ROBIN SCHEDULING

**AIM:**

To implement the round-robin (RR) scheduling technique

**ALGORITHM:**

1. Declare the structure and its elements.
2. Get a number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array rem\_bt[] to keep track of the remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array wt[] to store waiting times of processes. Initialize this array as 0.
6. Initialize time : t = 0
7. Keep traversing all processes while all processes are not done. Do the following for i'th process if it is not done yet. a- If rem\_bt[i] > quantum (i) t = t + quantum (ii) bt\_rem[i] -= quantum;

b- Else // Last cycle for this process

1. t = t + bt\_rem[i];
2. wt[i] = t - bt[i]
3. bt\_rem[i] = 0; // This process is over
4. Calculate the waiting time and turnaround time for each process.
5. Calculate the average waiting time and average turnaround time.
6. Display the results.

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

int main() { int n, time\_quantum;

printf("Enter number of processes: ");

scanf("%d", &n);

int \*arrival = (int\*)malloc(n \* sizeof(int)); int \*burst = (int\*)malloc(n \* sizeof(int)); int \*remaining = (int\*)malloc(n \* sizeof(int));

int wait\_time = 0, turnaround\_time = 0, total = 0, x = n;

for (int i = 0; i < n; i++) {

printf("Enter arrival time and burst time for process %d: ", i + 1); scanf("%d %d", &arrival[i], &burst[i]);

remaining[i] = burst[i]; }

printf("Enter time quantum: ");

scanf("%d", &time\_quantum);printf("\nProcess\tBurst\tTurnaround\tWaiting\n");

for (int i = 0; x != 0;) { if (remaining[i] > 0) {

if (remaining[i] <= time\_quantum) {

total += remaining[i]; remaining[i] = 0; x--;

printf("P%d\t%d\t%d\t\t%d\n", i + 1, burst[i], total - arrival[i], total - arrival[i] - burst[i]); wait\_time += total - arrival[i] - burst[i]; turnaround\_time += total - arrival[i];

} else {

remaining[i] -= time\_quantum; total += time\_quantum;

}

}

i = (i + 1) % n;

}

printf("\nAverage Waiting Time: %.2f", (float)wait\_time / n);

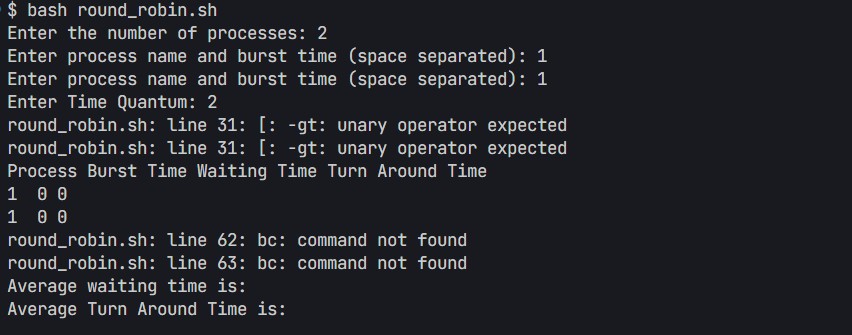
printf("\nAverage Turnaround Time: %.2f\n", (float)turnaround\_time / n);

free(arrival); free(burst); free(remaining);

return 0;

}

**OUTPUT:**



**RESULT:**

The Program of Round Robin Scheduling is successfully implemented.

**Ex. No: 7**

**Date:** 22/2/25

**IPC USING SHARED MEMORY**

**AIM:**

To write a C program to do Inter-Process Communication (IPC) using shared memory between the sender process and the receiver process.

**ALGORITHM:**

**sender**

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat
4. Write a string to the shared memory segment using sprintf
5. Set delay using sleep
6. Detach shared memory segment using shmdt

**receiver**

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat 4. Print the shared memory contents sent by the sender process.

5. Detach shared memory segment using shmdt

**PROGRAM:**

**SENDER**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#define SHMSIZE 1024

typedef struct {

int ready;

char message[SHMSIZE];

} SharedMemory;

int main() {

key\_t key = ftok("sender.c", 65);

int shmid;

SharedMemory \*shm;

shmid = shmget(key, sizeof(SharedMemory), 0666 | IPC\_CREAT); if (shmid == -1) { perror("shmget failed");

exit(1);

}

shm = (SharedMemory \*)shmat(shmid, NULL, 0);

if (shm == (SharedMemory \*)-1) { perror("shmat failed");

exit(1);

}

printf("Sender: Enter a message to send to receiver: "); fgets(shm->message, SHMSIZE, stdin);

shm->message[strcspn(shm->message, "\n")] = '\0';

shm->ready = 1;

sleep(5);

if (shmdt(shm) == -1) { perror("shmdt failed");

exit(1);

}

return 0;

}

**RECEIVER**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#define SHMSIZE 1024

typedef struct { int ready;

char message[SHMSIZE];

} SharedMemory;

int main() {

key\_t key = ftok("sender.c", 65);

int shmid;

SharedMemory \*shm;

shmid = shmget(key, sizeof(SharedMemory), 0666 | IPC\_CREAT); if (shmid == -1) { perror("shmget failed");

exit(1);

}

shm = (SharedMemory \*)shmat(shmid, NULL, 0);

if (shm == (SharedMemory \*)-1) { perror("shmat failed");

exit(1);

}

while (shm->ready == 0) { sleep(1);

}

printf("Receiver: Message received from sender: %s\n", shm->message);

if (shmdt(shm) == -1) { perror("shmdt failed");

exit(1);

}

if (shmctl(shmid, IPC\_RMID, NULL) == -1) { perror("shmctl failed");

exit(1);

}

return 0;

}

**OUTPUT:**





**RESULT:**

The IPC Program with Shared Memory is Successfully Implemented.

**Ex. No: 8**

**Date:** 22/2/25

**PRODUCER CONSUMER USING SEMAPHORES**

**AIM:**

To write a program to implement solutions to producer consumer problem using semaphores.

**ALGORITHM:**

1. Initialize semaphore empty, full and mutex.
2. Create two threads- the producer thread and the consumer thread.
3. Wait for target thread termination.
4. Call sem\_wait on empty semaphore followed by mutex semaphore before entry into critical section.
5. Produce/Consume the item in the critical section.
6. Call sem\_post on mutex semaphore followed by full semaphore 7. before exiting the critical section.
7. Allow the other thread to enter its critical section.
8. Terminate after looping ten times in producer and consumer Threads each.

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

int mutex = 1; int full = 0;

int empty = 10, x = 0;

pthread\_mutex\_t lock;

void \*producer(void \*arg)

{

pthread\_mutex\_lock(&lock);

if (empty != 0) {

--mutex;

++full; --empty; x++;

printf("\nProducer produces item %d\n", x);

++mutex; } else {

printf("Buffer is full!\n");

}

pthread\_mutex\_unlock(&lock); return NULL;

}

void \*consumer(void \*arg)

{

pthread\_mutex\_lock(&lock);

if (full != 0) { --mutex;

--full;

++empty;

printf("\nConsumer consumes item %d\n", x); x--; ++mutex; } else {

printf("Buffer is empty!\n");

}

pthread\_mutex\_unlock(&lock); return NULL;

}

int main() {

int n, i;

pthread\_t prod\_thread, cons\_thread;

pthread\_mutex\_init(&lock, NULL);

printf("\n1. Press 1 for Producer" "\n2. Press 2 for Consumer"

"\n3. Press 3 for Exit\n");

for (i = 1; i > 0; i++) { printf("\nEnter your choice: "); scanf("%d", &n);

switch (n) { case 1:

if (mutex == 1 && empty != 0) {

pthread\_create(&prod\_thread, NULL, producer, NULL);

pthread\_join(prod\_thread, NULL);

} else {

printf("Buffer is full!\n");

} break;

case 2:

if (mutex == 1 && full != 0) {

pthread\_create(&cons\_thread, NULL, consumer, NULL);

pthread\_join(cons\_thread, NULL);

} else {

printf("Buffer is empty!\n");

}

break;

case 3:

pthread\_mutex\_destroy(&lock); exit(0); break; default:

printf("Invalid choice! Please enter a valid option.\n");

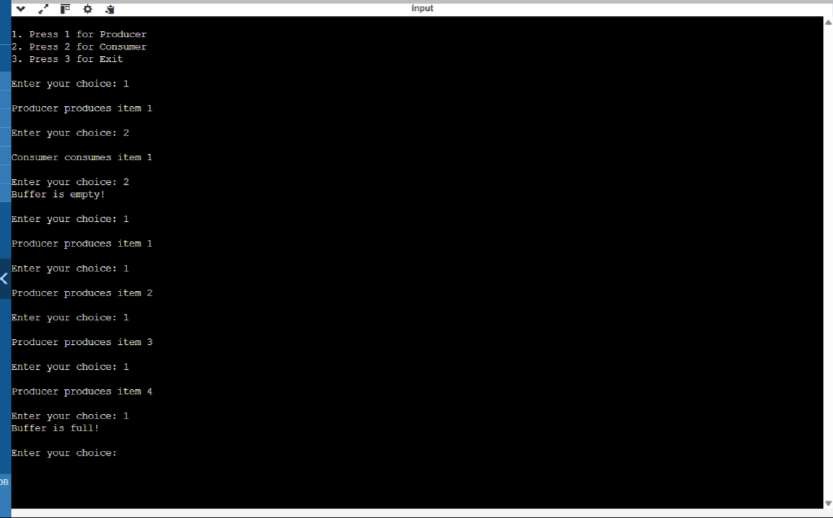
}

}

return 0;

}

**OUTPUT:**



**RESULT:**

The Producer Consumer Program using Semaphore is Successfully Implemented.

**Ex. No.: 9**

**Date:** 22/2/25

## DEADLOCK AVOIDANCE

**AIM:**

To find out a safe sequence using Banker’s algorithm for deadlock avoidance.

**ALGORITHM:**

1. Initialize work=available and finish[i]=false for all values of i 2. Find an i such that both:

finish[i]=false and Needi<= work 3. If no such i exists go to step 6

1. Compute work=work+allocationi
2. Assign finish[i] to true and go to step 2
3. If finish[i]==true for all i, then print safe sequence
4. Else print there is no safe sequence.

**PROGRAM:**

#include <stdio.h>

#include <stdbool.h>

#define MAX 10

void findSafeSequence(int n, int m, int available[], int max[][MAX], int allocation[][MAX]) {

int work[MAX], finish[MAX] = {0}, safeSeq[MAX], need[MAX][MAX];

for (int i = 0; i < m; i++) work[i] = available[i];

for (int i = 0; i < n; i++) for (int j = 0; j < m; j++)

need[i][j] = max[i][j] - allocation[i][j];

int count = 0; while (count < n) { bool found = false; for (int i = 0; i < n; i++) { if (!finish[i]) { bool canAllocate = true; for (int j = 0; j < m; j++)

if (need[i][j] > work[j]) { canAllocate = false; break; } if (canAllocate) {

for (int j = 0; j < m; j++) work[j] += allocation[i][j]; safeSeq[count++] = i;

finish[i] = 1; found = true;

}

}

}

if (!found) { printf("No safe sequence.\n"); return; }

}

printf("Safe sequence: ");

for (int i = 0; i < n; i++) printf("P%d ", safeSeq[i]); printf("\n");

}

int main() {

int n, m, available[MAX], max[MAX][MAX], allocation[MAX][MAX];

printf("Enter processes and resources: "); scanf("%d %d", &n, &m); while (getchar() != '\n');

printf("Enter available resources: "); for (int i = 0; i < m; i++) scanf("%d", &available[i]);

while (getchar() != '\n');

printf("Enter Max matrix: \n"); for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++) scanf("%d", &max[i][j]); while (getchar() != '\n');

printf("Enter Allocation matrix: \n");

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++) scanf("%d", &allocation[i][j]); while (getchar() != '\n');

findSafeSequence(n, m, available, max, allocation); return 0; }

**OUTPUT:**



**RESULT:**

The Safe Sequence is found using Banker’s Algorithm for Deadlock Avoidance.

**Ex. No: 10a**

**Date:** 7/3/25

## BEST FIT

**AIM:**

To implement Best Fit memory allocation technique using Python.

**ALGORITHM:**

1. Input memory blocks and processes with sizes
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process
4. If found then assign it to the current process.
5. If not found then leave that process and keep checking the further processes.

**PROGRAM:** def best\_fit(blocks, processes):

allocation = [-1] \* len(processes)

for i in range(len(processes)): best\_index = -1

for j in range(len(blocks)): if blocks[j] >= processes[i]: if best\_index == -1 or blocks[j] < blocks[best\_index]:

best\_index = j

if best\_index != -1: allocation[i] = best\_index blocks[best\_index] -= processes[i]

print("\nProcess No.\tProcess Size\tBlock No.") for i in range(len(processes)):

print(f"{i + 1}\t\t{processes[i]}\t\t{allocation[i] + 1 if allocation[i] != -1 else 'Not Allocated'}")

if \_\_name\_\_ == "\_\_main\_\_": num\_blocks = int(input("Enter number of memory blocks: "))

blocks = list(map(int, input(f"Enter sizes of {num\_blocks} memory blocks (space-separated): ").split()))

num\_processes = int(input("\nEnter number of processes: "))

processes = list(map(int, input(f"Enter sizes of {num\_processes} processes (space-separated): ").split()))

best\_fit(blocks, processes)

**OUTPUT:**



**RESULT:**

Thus, the Best Fit Memory allocation technique is implemented successfully using Python.

**Ex. No: 10b**

**Date:** 7/3/25

## FIRST FIT

**AIM:**

To write a C program for the implementation of memory allocation methods for a fixed

partition using the first fit.

**ALGORITHM:**

1. Define the max as 25.
2. Declare the variable frag[max],b[max],f[max],i,j, nb,nf, temp, highest=0, bf[max],ff[max].
3. Get the number of blocks, files, size of the blocks using a for loop.
4. In for loop check bf[j]!=1, if so temp=b[j]-f[i]
5. Check the highest.

**PROGRAM:**

#include <stdio.h>

#define MAX 25

int main() {

int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp; static int bf[MAX], ff[MAX];

printf("Enter the number of blocks: "); scanf("%d", &nb);

printf("Enter the number of files: "); scanf("%d", &nf);

printf("Enter the size of the blocks:\n"); for (i = 0; i < nb; i++) { printf("Block %d: ", i + 1);

scanf("%d", &b[i]);

}

printf("Enter the size of the files:\n"); for (i = 0; i < nf; i++) { printf("File %d: ", i + 1);

scanf("%d", &f[i]);

}

for (i = 0; i < nf; i++) { for (j = 0; j < nb; j++) { if (bf[j] != 1) { temp = b[j] - f[i]; if (temp >= 0) { ff[i] = j; bf[j] = 1; frag[i] = temp;

break;

}

}

}

}

printf("\nFile No.\tFile Size\tBlock No.\tBlock Size\tFragment\n"); for (i = 0; i < nf; i++) { if (bf[ff[i]] == 1)

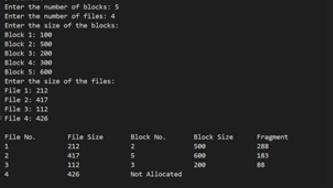
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]); else

printf("%d\t\t%d\t\tNot Allocated\n", i + 1, f[i]); }

return 0;

}

**OUTPUT:**



**RESULT:**

Thus, the First Fit allocation technique is implemented successfully using C.

**Ex. No: 11a**

**Date:** 21/3/25

**FIFO PAGE REPLACEMENT**

**AIM:**

To find out the number of page faults that occur using the First-in First-out (FIFO) page replacement technique.

**ALGORITHM:**

1. Declare the size with respect to page length
2. Check the need for replacement from the page to memory
3. Check the need for replacement from the old page to the new page in memory
4. Form a queue to hold all pages
5. Insert the page required memory into the queue
6. Check for bad replacement and page fault 7. Get the number of processes to be inserted

8. Display the values.

**PROGRAM:**

def fifo\_page\_replacement(pages, frame\_size):

frames = [] page\_faults = 0

front = 0

print("\nPage Replacement Process:")

for page in pages: if page not in frames: if len(frames) < frame\_size: frames.append(page) else:

frames[front] = page front = (front + 1) % frame\_size page\_faults += 1 print(f"Page {page} => {frames} \*Page Fault\*") else:

print(f"Page {page} => {frames}")

print(f"\nTotal Page Faults = {page\_faults}")

if \_\_name\_\_ == "\_\_main\_\_":

n = int(input("Enter the number of pages: ")) pages = []

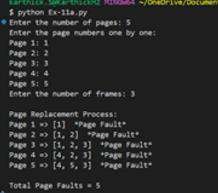
print("Enter the page numbers one by one:") for i in range(n):

page = int(input(f"Page {i+1}: ")) pages.append(page)

frame\_size = int(input("Enter the number of frames: "))

fifo\_page\_replacement(pages, frame\_size)

**OUTPUT:**



**RESULT:**

The Fifo Page Replacement is Successfully Implemented using Python.

**Ex. No: 11b**

**Exp 11 b-LRU**

**Date:** 25/3/25

# LRU

**AIM:**

To write a C program to implement LRU page replacement algorithm.

**ALGORITHM:**

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

**PROGRAM:**

#include <stdio.h>

int main() {

int pages[50], frames[10], counter[10]; int n, frameSize, i, j, k, flag, least, time = 0, faults = 0; printf("Enter the number of frames: ");

scanf("%d", &frameSize);

printf("Enter the number of pages: "); scanf("%d", &n);

printf("Enter the page reference string: ");

for(i = 0; i < n; i++) {

scanf("%d", &pages[i]);

}

for(i = 0; i < frameSize; i++) { frames[i] = -1; counter[i] = 0;

}

for(i = 0; i < n; i++) { flag = 0;

for(j = 0; j < frameSize; j++) { if(frames[j] == pages[i]) { counter[j] = ++time; flag = 1; break;

}

}

if(flag == 0) {

int pos = -1, min = 9999;

for(j = 0; j < frameSize; j++) { if(frames[j] == -1) { pos = j; break;

} else if(counter[j] < min) {

min = counter[j];

pos = j;

}

}

frames[pos] = pages[i]; counter[pos] = ++time;

faults++;

}

printf("Frames after inserting %d: ", pages[i]); for(k = 0; k < frameSize; k++) { if(frames[k] != -1) printf("%d ", frames[k]); else

printf("- ");

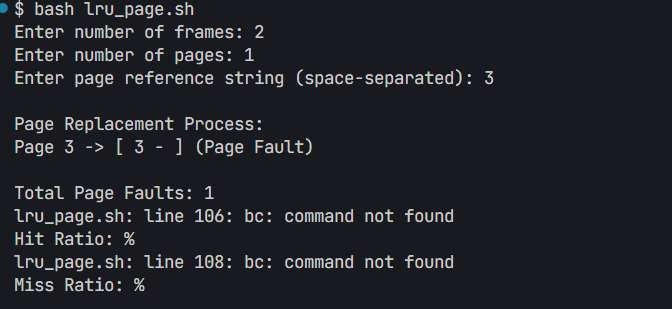
}

printf("\n");

}

printf("\nTotal Page Faults: %d\n", faults); return 0; }

**OUTPUT:**



**RESULT:**

The LRU Program is Successfully Implemented using C.

**Ex. No: 11c Date:** 25/3/25

# Optimal

**AIM:**

To write a c program to implement the Optimal page replacement algorithm

**ALGORITHM:**

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least frequently used page by counter value.

7.Stack them according the selection.

1. Display the values
2. Stop the process

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

int isInFrame(int frame[], int count, int page) {

for (int i = 0; i < count; i++) if (frame[i] == page) return 1;

return 0;

}

int predict(int pages[], int frame[], int n, int index, int count) { int farthest = index, res = -1; for (int i = 0; i < count; i++) {

int j;

for (j = index; j < n; j++) { if (frame[i] == pages[j]) { if (j > farthest) { farthest = j; res = i; } break;

}

}

if (j == n) return i; // If page not found in future

}

return (res == -1) ? 0 : res;

}

int main() {

int n, frameCount, pageFaults = 0, filled = 0;

printf("Enter number of pages: "); scanf("%d", &n);

int\* pages = malloc(n \* sizeof(int));

printf("Enter the page numbers:\n"); for (int i = 0; i < n; i++)

scanf("%d", &pages[i]);

printf("Enter number of frames: "); scanf("%d", &frameCount); int\* frame = malloc(frameCount \* sizeof(int)); for (int i = 0; i < frameCount; i++)

frame[i] = -1;

for (int i = 0; i < n; i++) {

if (!isInFrame(frame, frameCount, pages[i])) {

if (filled < frameCount)

frame[filled++] = pages[i]; else

frame[predict(pages, frame, n, i, frameCount)] = pages[i]; pageFaults++;

}

printf("Frame: "); for (int j = 0; j < frameCount; j++)

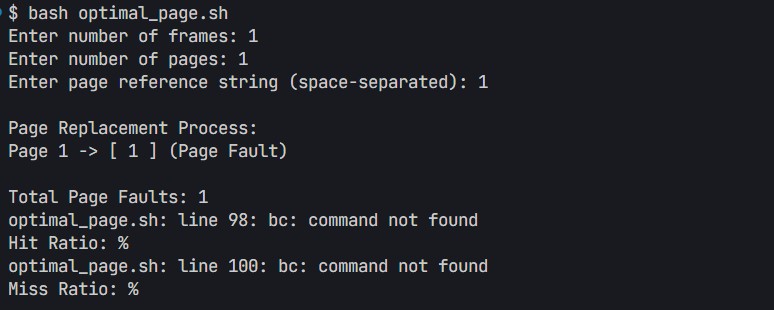
frame[j] == -1 ? printf("- ") : printf("%d ", frame[j]); printf("\n");

}

printf("\nTotal Page Faults = %d\n", pageFaults);

free(pages); free(frame); return 0;

**OUTPUT:**



**RESULT:**

The Optimal page replacement Program is Successfully Implemented using C.

**Ex. No: 12**

**Date:** 1/4/25

## File Organization Technique- Single- and Two-level directory

**AIM:**

To implement File Organization Structures in C are a. Single Level Directory

1. Two-Level Directory
2. Hierarchical Directory Structure
3. Directed Acyclic Graph Structure

**A. SINGLE LEVEL DIRECTORY**

**ALGORITHM:**

1. Start
2. Declare the number, names and size of the directories and file names. 3. Get the values for the declared variables.
3. Display the files that are available in the directories.
4. Stop.

**PROGRAM:**

#include <stdio.h>

#include <string.h>

struct File { char name[20];

};

int main() { int n, i;

struct File files[10];

printf("Enter the number of files: ");

scanf("%d", &n);

if (n <= 0 || n > 10) {

printf("Please enter a valid number of files (1–10).\n"); return 1;

}

for (i = 0; i < n; i++) {

printf("Enter the file %d: ", i + 1);

scanf("%s", files[i].name);

}

printf("\n\nRoot Directory\n");

printf("|\n");

for (i = 0; i < n; i++) {

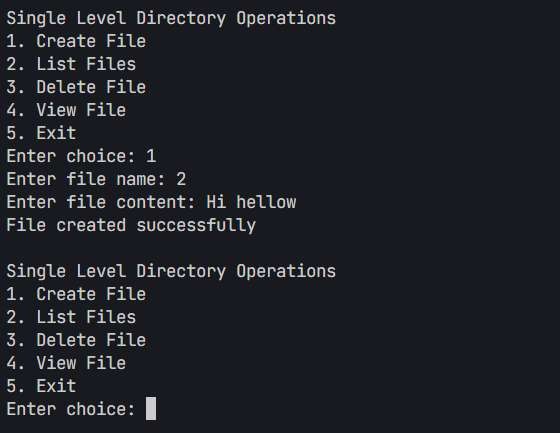
printf("|-- %s\n", files[i].name);

}

return 0;

}

OUTPUT:



**B. TWO-LEVEL DIRECTORY STRUCTURE ALGORITHM:**

1. Start
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories. 5. Stop.

**PROGRAM:**

#include <stdio.h> Implemented using C.

#include <string.h>

struct File { char name[20];

};

struct SubDirectory { char name[20]; struct File files[10]; int fileCount;

};

struct Directory { char name[20]; struct SubDirectory subDirs[10]; int subDirCount;

};

int main() { struct Directory dir;

int i, j;

printf("Enter root directory name: "); scanf("%s", dir.name);

printf("How many subdirectories in '%s'? ", dir.name); scanf("%d", &dir.subDirCount);

for (i = 0; i < dir.subDirCount; i++) {

printf("\nEnter name of subdirectory %d under '%s': ", i + 1, dir.name); scanf("%s", dir.subDirs[i].name);

printf("How many files in '%s'? ", dir.subDirs[i].name); scanf("%d", &dir.subDirs[i].fileCount);

for (j = 0; j < dir.subDirs[i].fileCount; j++) { printf("Enter file %d in '%s': ", j + 1, dir.subDirs[i].name); scanf("%s", dir.subDirs[i].files[j].name);

}

}

printf("\nDirectory Structure:\n"); printf("NULL\n");

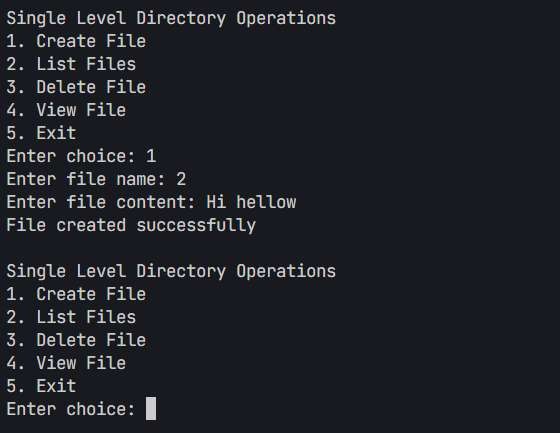
printf("|\_\_ %s\n", dir.name);

for (i = 0; i < dir.subDirCount; i++) { printf(" |\_\_ %s\n", dir.subDirs[i].name); for (j = 0; j < dir.subDirs[i].fileCount; j++) { printf(" |\_\_ %s\n", dir.subDirs[i].files[j].name); }

}

return 0; }

OUTPUT:



**RESULT:**

The File Organization Technique-Single and Two-Level Directory Program is Successfully Implemented using C.