

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

LAB MANUAL

**CS23431 – OPERATING SYSTEMS**

**(REGULATION 2023)**

**RAJALAKSHMI ENGINEERING COLLEGE**

**Thandalam, Chennai-602015**

Name: K.ASWIN

Register No.: 231501027

Year / Branch / Section: II/AIML/FA Semester: IV

Academic Year: 2024-2025

# INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EXP.NO** | **Date** | **Title** | **Page No** | **Signature** |
| 1a | 29/03/2025 | Installation and Configuration of Linux | 4 |  |
| 1b | 21/02/2025 | Basic Linux Commands | 8 |  |
| 2 | 21/02/2025 | Study of Unix editors : sed,vi,emacs | 20 |  |
| 3 a) | 21/02/2025 | Shell script   1. Arithmetic Operation -using expr command 2. Check leap year using if-else | 24 |  |
| 3 b) | 21/02/2025 | 1. Reverse the number using while loop 2. Fibonacci series using for loop | 26 |  |
| 4 | 05/03/2025 | Text processing using Awk script   1. Employee average pay 2. Results of an examination | 28 |  |
| 5 | 05/03/2025 | System calls –fork(), exec(), getpid(),opendir(), readdir() | 32 |  |
| 6a | 05/03/2025 | FCFS | 35 |  |
| 6b | 05/03/2025 | SJF | 38 |  |
| 6c | 05/03/2025 | Priority | 41 |  |
| 6d | 05/03/2025 | Round Robin | 44 |  |
| 7. | 21/03/2025 | Inter-process Communication using Shared Memory | 48 |  |
| 8 | 26/03/2025 | Producer Consumer using Semaphores | 53 |  |
| 9 | 28/03/2025 | Bankers Deadlock Avoidance algorithms | 57 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 a | 02/04/2025 | Best Fit | 59 |  |
| 10 b | 04/04/2025 | First Fit | 62 |  |
| 11a | 04/04/2025 | FIFO | 65 |  |
| 11b | 09/04/2025 | LRU | 69 |  |
| 11c | 11/04/2025 | Optimal | 73 |  |
| 12 | 11/04/2025 | File Organization Technique- single and Two level directory | 76 |  |

**Ex No: 1a)**

**Date:**

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

**Output:**

**Result :**

Thus, Virtual Machine was installed and configured in a Linux Operating System successfully.

**Ex No: 1b)**

**Date:**

# BASIC LINUX COMMANDS

* 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”

1. 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

# 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

# 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 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

1

Execute permission

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

# 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.

# 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.

**S**ynopsis- 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

^C

--- 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

1. **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

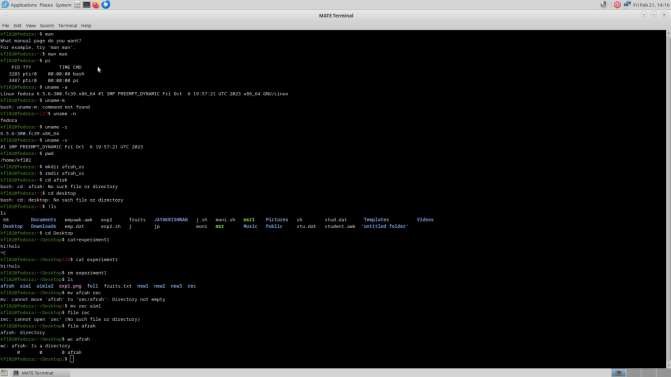
[root@localhost ~]# traceroute [www.rajalakshmi.org](http://www.rajalakshmi.org/)

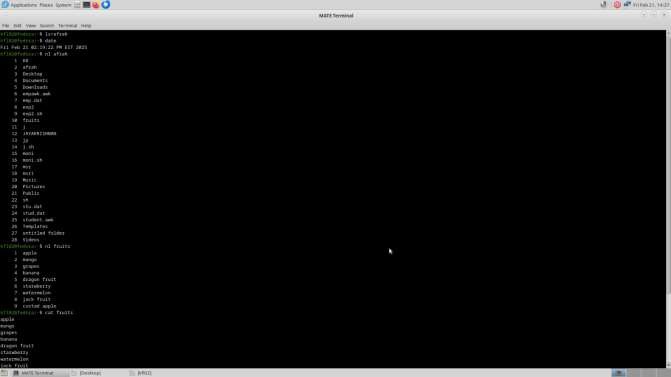
traceroute to [www.rajalakshmi.org](http://www.rajalakshmi.org/) (220.227.30.51), 30 hops max, 60 byte

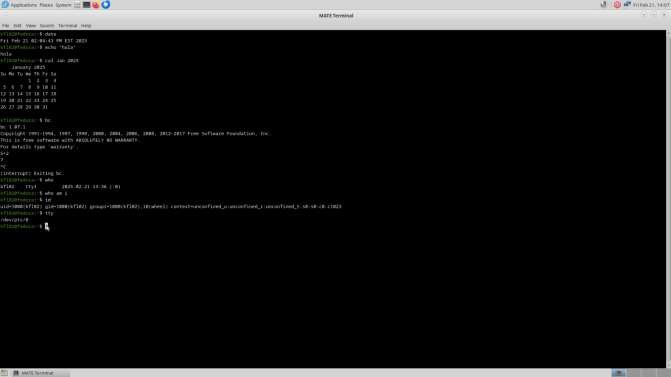
packets 1 gateway (172.16.4.1) 0.299 ms 0.297 ms 0.327 ms

2 220.225.219.38 (220.225.219.38) 6.185 ms 6.203 ms 6.189 ms

**Sample Input and Output**

****

****



**Result:**

Thus, basic linux commands were implemented successfully

**Ex. no: 2a) Date:**

**Shell Script**

**Aim:**

To write a Shellscript to to display basic calculator.

**Program:**

echo "Enter two numbers" read num1

read num2

# Perform arithmetic operations sum=$((num1 + num2)) diff=$((num1 - num2)) mul=$((num1 \* num2))

# Handle division and modulo to prevent division by zero if [ $num2 -eq 0 ]; then

div="undefined (division by zero)" mod="undefined (modulo by zero)" else

div=$((num1 / num2)) mod=$((num1 % num2)) fi

# Output the results echo "add $sum" echo "sub $diff" echo "mul $mul" echo "div $div" echo "mod $mod"

**Sample Input and Output**

**Run the program using the below command**

[REC@local host~]$ sh arith.sh

Enter two no 5

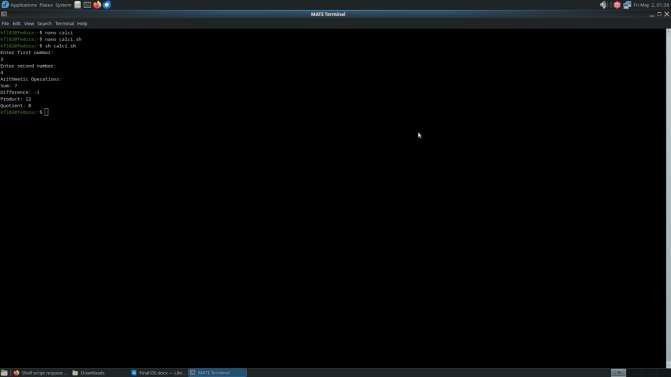
10

add 15

sub -5

mul 50

div 0 mod 5c"



**Result:**

Thus, a Shell script was written to display basic calculator successfully.

**Ex. no: 2b) Date:**

**Shell Script**

**Aim:**

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

**Program:**

read -p "Enter a year: " year if ((year % 4 == 0)); then

if ((year % 100 == 0)); then if ((year % 400 == 0)); then echo "$year is a leap year." else

echo "$year is not a leap year." fi

else

echo "$year is a leap year." fi

else

echo "$year

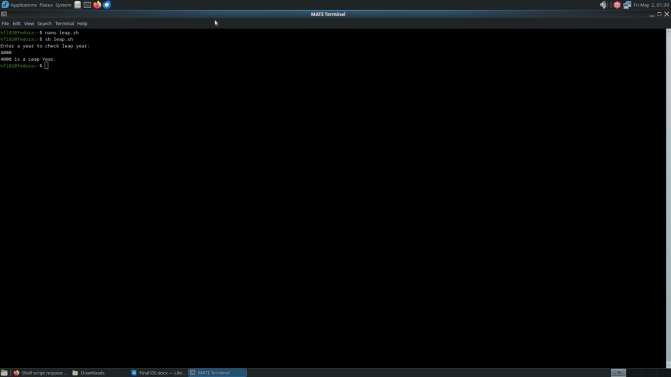
**Sample Input and Output**

**Run the program using the below command**

[REC @ local host~]$ sh leap.sh enter number

12

leap year



**Result:**

Thus, a Shell script was written to test given year is leap or not using conditional statement successfully

**Ex. No.: 3a)**

**Date: Aim:**

**Shell Script – Reverse of Digit**

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

**Program:**

read -p "Enter a number: " num reverse=0

while [ $num -gt 0 ]; do reverse=$((reverse \* 10 + num % 10)) num=$((num / 10))

done

echo "Reversed”

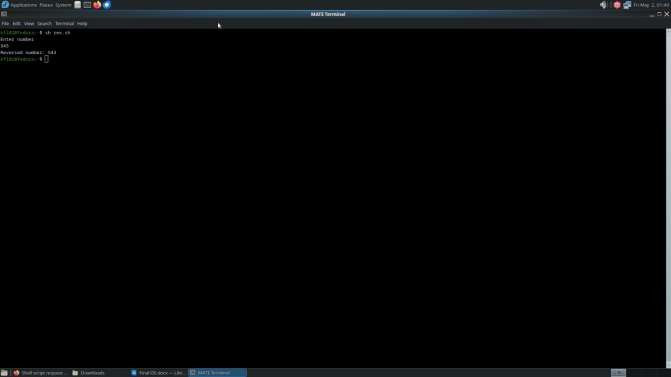
**Sample Input and Output**

**Run the program using the below command**

[REC@local host~]$sh indhu.sh

enter number 123

321



**Result:**

Thus, a Shell script was written to reverse a given digit using looping statement successfully.

**Ex. No.: 3b)**

**Date: Aim:**

**Shell Script – Fibbonacci Series**

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

**Program:**

read -p "Enter the number of terms: " n a=0

b=1

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

echo -n "$a " fn=$((a + b)) a=$b

b=$fn done

**Sample Input and Output**

**Run the program using the below command**

[REC@local host~]$sh indhu.sh

enter number 21

fibonacci series 0

1

1

2

3

5

8

13

21

34

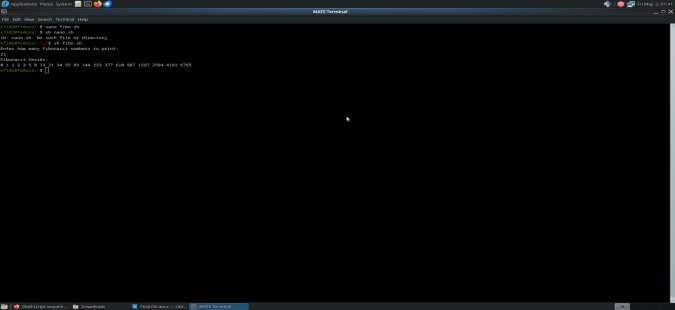
55

89

144

233

377



**Result:**

Thus, a Shell script was written to generate a Fibonacci series using for loop successfully.

**Ex. No.: 4a)**

**Date:**

**Aim:**

# EMPLOYEE AVERAGE PAY

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
   1. If Salary is greater than 6000 and number of days worked is more than 4, then print name and salary earned
   2. Compute total pay of employee
4. Print the total number of employees satisfying the criteria and their average pay.

**Program Code:**

BEGIN {

total\_pay = 0

count = 0

print "EMPLOYEES DETAILS"

}

{

if ($2 > 6000 && $3 > 4) { employee\_pay = $2 \* $3

printf "%s %d\n", $1, employee\_pay total\_pay += employee\_pay count++

}

}

END {

if (count > 0) {

printf "no of employees are= %d\n", count printf "total pay= %d\n", total\_pay

printf "average pay= %.1f\n", total\_pay/count

} else {

print "No employees meet the criteria"

}

}

**Sample Input:**

//emp.dat – Col1 is name, Col2 is Salary Per Day and Col3 is //no. of days worked JOE 8000 5

RAM 6000 5

TIM 5000 6

BEN 7000 7

AMY 6500 6

**Output:**

**Run the program using the below commands** [student@localhost ~]$ vi emp.dat [student@localhost ~]$ vi emp.awk [student@localhost ~]$ gawk -f emp.awk emp.dat.

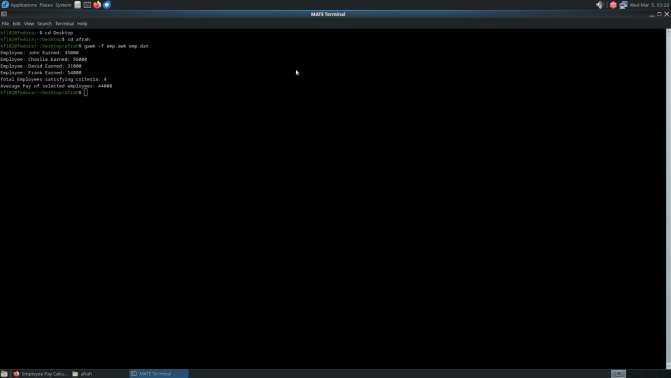
EMPLOYEES DETAILS JOE 40000

BEN 49000

AMY 39000

no of employees are= 3 total pay= 128000

average pay= 42666.7 [student@localhost ~]$



**Result:**

Thus, found the average pay of all employees whose salary is more than 6000 and no. of days worked is more than 4 successfully.

**Ex. No.: 4b)**

**Date:**

**Aim:**

# RESULTS OF EXAMINATION

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 Code:**

**//marks.awk**

BEGIN {

# Print header

printf "%-10s %-6s %-6s %-6s %-6s %-6s %-6s %-10s\n",

"NAME", "SUB-1", "SUB-2", "SUB-3", "SUB-4", "SUB-5", "SUB-6", "STATUS"

print " "

}

{

status = "PASS" # Default status

# Check each subject mark (columns 2-7) for (i = 2; i <= 7; i++) {

if ($i < 45) { status = "FAIL"

break # No need to check further if one subject fails

}

}

# Print student record with status

printf "%-10s %-6d %-6d %-6d %-6d %-6d %-6d %-10s\n",

$1, $2, $3, $4, $5, $6, $7, status

}

**Input:**

**//marks.dat**

**//**Col1- name, Col 2 to Col7 – marks in various subjects BEN 40 55 66 77 55 77

TOM 60 67 84 92 90 60

RAM 90 95 84 87 56 70

JIM 60 70 65 78 90 87

**Output:**

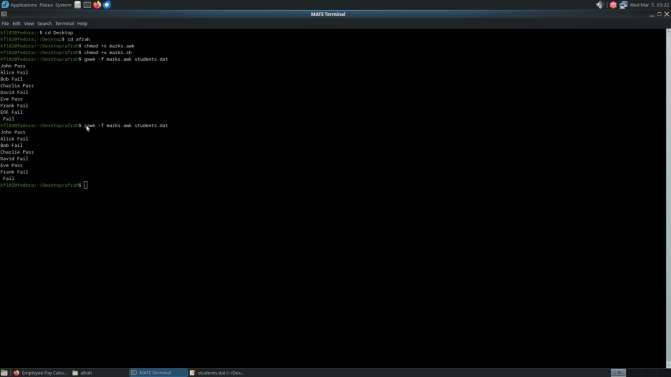
**Run the program using the below command**

[root@localhost student]# gawk -f marks.awk marks.dat

NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS

BEN 40 55 66 77 55 77 FAIL TOM 60 67 84 92 90 60 PASS RAM 90 95 84

87 56 70 PASS JIM 60 70 65 78 90 87 PASS



**Result:**

Thus, printed the pass/fail status of a student in a class successfully.

**Ex. No.: 5**

**Date:**

**System Calls Programming**

**Aim:**

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

**Algorithm:**

1. **Start**
   * Include the required header files (stdio.h and stdlib.h).
2. **Variable Declaration**
   * Declare an integer variable pid to hold the process ID.
3. **Create a Process**
   * Call the fork() function to create a new process. Store the return value in the pid variable:
     + If fork() returns:
       - -1: Forking failed (child process not created).
       - 0: Process is the child process.
       - Positive integer: Process is the parent process.
4. **Print Statement Executed Twice**
   * Print the statement:

scss

Copy code

THIS LINE EXECUTED TWICE

(This line is executed by both parent and child processes after fork()).

1. **Check for Process Creation Failure**
   * If pid == -1:
     + Print:

Copy code

CHILD PROCESS NOT CREATED

* + - Exit the program using exit(0).

1. **Child Process Execution**
   * If pid == 0 (child process):
     + Print:
       - Process ID of the child process using getpid().
       - Parent process ID of the child process using getppid().
2. **Parent Process Execution**
   * If pid > 0 (parent process):
     + Print:
       - Process ID of the parent process using getpid().
       - Parent's parent process ID using getppid().
3. **Final Print Statement**
   * Print the statement:

objectivec

Copy code

IT CAN BE EXECUTED TWICE

(This line is executed by both parent and child processes).

1. **End Program:**

#include <stdio.h> #include <stdlib.h> #include <unistd.h>

int main() { int pid;

pid = fork();

if (pid == -1) {

printf("CHILD PROCESS NOT CREATED\n");

exit(0);

}

printf("THIS LINE EXECUTED TWICE\n");

if (pid == 0) {

printf("Child Process:\n");

printf("Process ID of the child: %d\n", getpid()); printf("Parent Process ID of the child: %d\n", getppid());

}

else if (pid > 0) { printf("Parent Process:\n");

printf("Process ID of the parent: %d\n", getpid()); printf("Parent's Parent Process ID: %d\n", getppid());

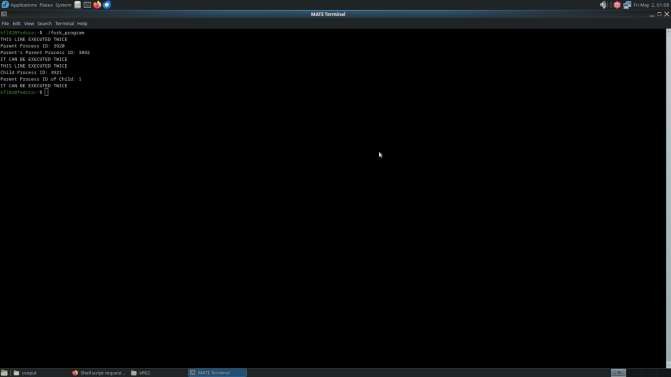
}

printf("IT CAN BE EXECUTED TWICE\n");

return 0;

}

**Output:**

****

**Result:**

Thus, experimented system calls using fork(), execlp() and pid() functions successfully.

**Ex. No.: 6a) Date:**

# 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 Code:**

echo "Enter the number of process:"; read n for ((i=0; i<n; i++)); do

echo "Enter burst time for process $i:"; read bt[i]

done

wt[0]=0; tat[0]=${bt[0]}

for ((i=1; i<n; i++)); do wt[i]=$((wt[i-1] + bt[i-1]))

tat[i]=$((wt[i] + bt[i])) done

echo -e "Process\tBurst Time\tWaiting Time\tTurn Around Time" for ((i=0; i<n; i++)); do

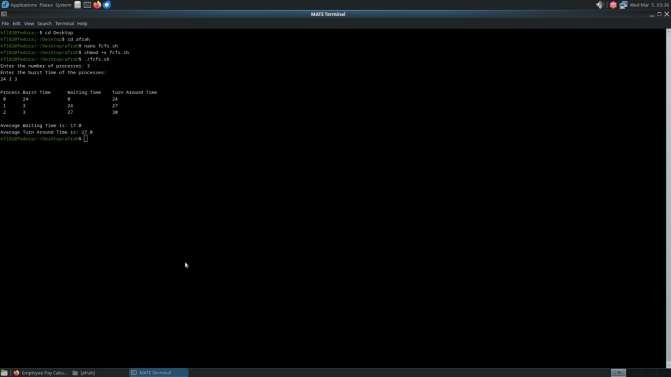
echo -e "$i\t${bt[i]}\t\t${wt[i]}\t\t${tat[i]}" twt=$((twt + wt[i]))

ttat=$((ttat + tat[i])) done

echo "Average waiting time is: $(echo "scale=1; $twt / $n" | bc)"

echo "Average Turn around Time is: $(echo "scale=1; $ttat / $n" | bc)"

**Sample Output:**

****

**Result:**

Thus, implemented First-come First- serve (FCFS) scheduling technique successfully.

**Ex. No.: 6b) Date:**

# SHORTEST JOB FIRST

**Aim:**

To implement the Shortest Job First (SJF) scheduling technique.

**Algorithm:**

1. Declare the structure and its elements.
2. Get 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 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 Code:**

echo "Enter the number of processes:" read n

declare -a bt declare -a wt declare -a tat

echo "Enter the burst time of the processes:" for ((i = 0; i < n; i++)); do

read bt[$i] done

sorted\_bt=($(printf "%s\n" "${bt[@]}" | sort -n)) wt[0]=0

tat[0]=${sorted\_bt[0]}

total\_wt=0 total\_tat=${sorted\_bt[0]}

for ((i = 1; i < n; i++)); do

wt[$i]=$((wt[$i - 1] + sorted\_bt[$i - 1]))

tat[$i]=$((wt[$i] + sorted\_bt[$i])) total\_wt=$((total\_wt + wt[$i])) total\_tat=$((total\_tat + tat[$i]))

done

avg\_wt=$(echo "scale=2; $total\_wt / $n" | bc) avg\_tat=$(echo "scale=2; $total\_tat / $n" | bc)

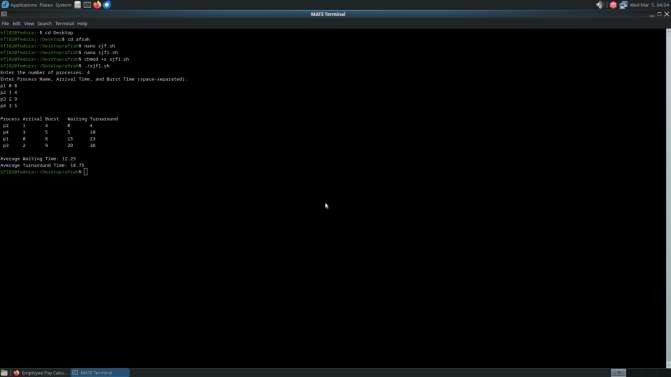
echo "Process Burst Time Waiting Time Turn Around Time" for ((i = 0; i < n; i++)); do

echo " $((i + 1)) ${sorted\_bt[$i]} ${wt[$i]} ${tat[$i]}" done

echo "Average waiting time is: $avg\_wt"

echo "Average Turn Around Time is: $avg\_tat"

**Sample Output:**

****

**Result:**

Thus, implemented the Shortest Job First (SJF) scheduling technique successfully.

**Ex. No.: 6c) Date:**

# PRIORITY SCHEDULING

**Aim:**

To implement priority scheduling technique

**Algorithm:**

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority 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 Code:**

# Read the number of processes

echo -n "Enter number of processes: " read n

# Declare arrays

declare -a burst\_time priority waiting\_time turnaround\_time process

# Read burst time and priority for each process for ((i=0; i<n; i++)); do

process[i]=$((i+1)) # Process ID echo -e "P[$((i+1))]\nBurst Time: " read burst\_time[i]

echo -n "Priority: " read priority[i]

done

# Sort processes by priority (lower number = higher priority) for ((i=0; i<n-1; i++)); do

for ((j=i+1; j<n; j++)); do

if (( priority[i] > priority[j] )); then # Swap priority temp=${priority[i]} priority[i]=${priority[j]} priority[j]=$temp

# Swap burst time temp=${burst\_time[i]} burst\_time[i]=${burst\_time[j]} burst\_time[j]=$temp

fi done

done

# Swap process ID temp=${process[i]} process[i]=${process[j]} process[j]=$temp

# Calculate waiting time waiting\_time[0]=0

for ((i=1; i<n; i++)); do

waiting\_time[i]=$((waiting\_time[i-1] + burst\_time[i-1])) done

# Calculate turnaround time for ((i=0; i<n; i++)); do

turnaround\_time[i]=$((waiting\_time[i] + burst\_time[i])) done

# Display results

echo -e "\nProcess\tBurst Time\tWaiting Time\tTurnaround Time" total\_wt=0

total\_tat=0

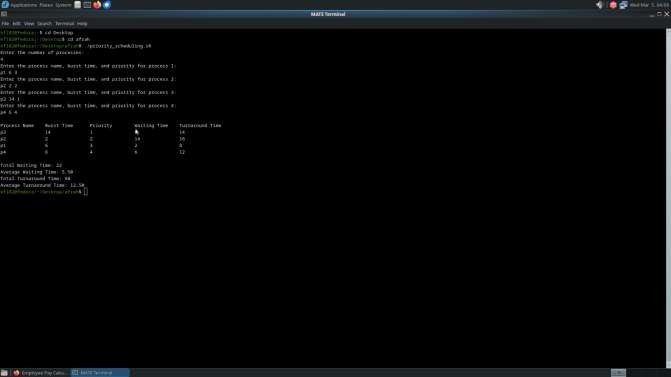
for ((i=0; i<n; i++)); do

echo -e "P[${process[i]}]\t${burst\_time[i]}\t\t${waiting\_time[i]}\t\t${turnaround\_time[i]}" total\_wt=$((total\_wt + waiting\_time[i]))

total\_tat=$((total\_tat + turnaround\_time[i])) done

# Calculate and print average times avg\_wt=$(echo "scale=2; $total\_wt / $n" | bc) avg\_tat=$(echo "scale=2; $total\_tat / $n" | bc) echo -e "\nAverage Waiting Time: $avg\_wt" echo -e "Average Turnaround Time: $avg\_tat"

**Sample Output:**

****

**Result:**

Thus, implemented priority scheduling technique successfully.

**Ex. No.: 6d) Date:**

# ROUND ROBIN SCHEDULING

**Aim:**

To implement the Round Robin (RR) scheduling technique

**Algorithm:**

1. Declare the structure and its elements.
2. Get 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 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
6. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
   1. If rem\_bt[i] > quantum
      1. t = t + quantum
      2. bt\_rem[i] -= quantum;
   2. 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
7. Calculate the waiting time and turnaround time for each process.
8. Calculate the average waiting time and average turnaround time.
9. Display the results.

**Program Code:**

echo -n "Enter Total Number of Processes: " read n

declare -a bt at wt tat remaining\_bt for ((i=0; i<n; i++))

do

echo "Enter Details of Process[$((i+1))]" echo -n "Arrival Time: "

read at[i]

echo -n "Burst Time: " read bt[i] remaining\_bt[i]=${bt[i]}

done

echo -n "Enter Time Quantum: " read tq

time=0 done\_processes=0

# Initialize waiting time array for ((i=0; i<n; i++))

do

wt[i]=0 done

while ((done\_processes < n)) do

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

if ((remaining\_bt[i] > 0)) then

if ((remaining\_bt[i] > tq)) then

time=$((time + tq)) remaining\_bt[i]=$((remaining\_bt[i] - tq))

else

time=$((time + remaining\_bt[i])) wt[i]=$((time - bt[i] - at[i])) remaining\_bt[i]=0 ((done\_processes++))

fi

fi done

done

# Calculate turnaround time for ((i=0; i<n; i++))

do

tat[i]=$((bt[i] + wt[i])) done

# Display results

echo -e "\nProcess ID\tBurst Time\tTurnaround Time\tWaiting Time" total\_wt=0

total\_tat=0

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

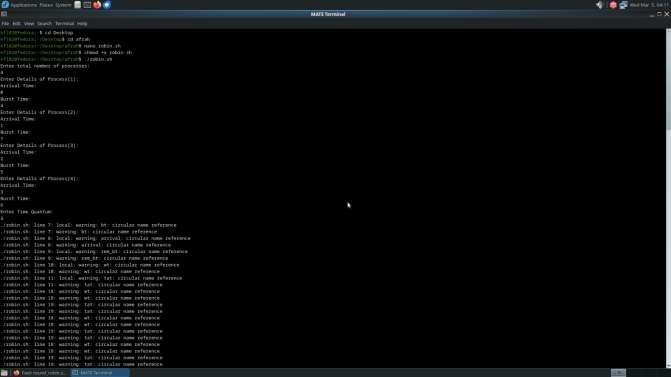
echo -e "Process[$((i+1))]\t${bt[i]}\t\t${tat[i]}\t\t${wt[i]}" total\_wt=$((total\_wt + wt[i]))

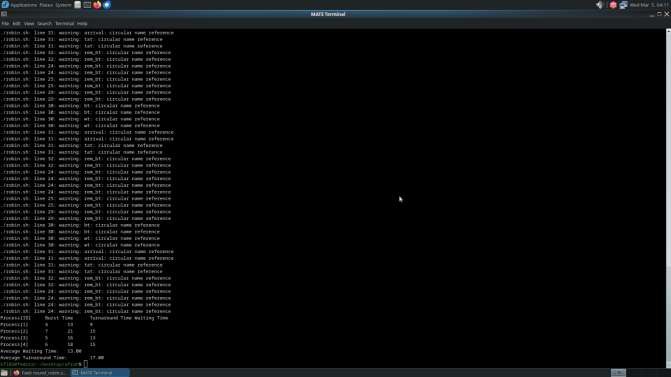
total\_tat=$((total\_tat + tat[i])) done

avg\_wt=$(echo "scale=2; $total\_wt / $n" | bc) avg\_tat=$(echo "scale=2; $total\_tat / $n" | bc)

echo -e "\nAverage Waiting Time: $avg\_wt" echo -e "Average Turnaround Time: $avg\_tat"

**Sample Output:**

****



**Result:**

Thus, implemented the Round Robin (RR) scheduling technique successfully.

**Ex. No.: 7**

**Date:**

**Aim:**

# IPC USING SHARED MEMORY

To write a C program to do Inter Process Communication (IPC) using shared memory

between sender process and 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 Code: sender.c**

#include<stdio.h> #include<stdlib.h> #include<sys/ipc.h> #include<sys/shm.h> #include<unistd.h> #include<string.h>

#define SHM\_SIZE 1024 int main(){

key\_t key = ftok("shmfile",65);

int shmid = shmget(key, SHM\_SIZE, 0666 | IPC\_CREAT); char \*shm = (char \*)shmat(shmid, NULL, 0);

int \*flag = (int \*)shm;

while(1){

printf("Sender: Enter a Message: ");

fgets(shm + sizeof(int), SHM\_SIZE - sizeof(int), stdin); shm[strcspn(shm + sizeof(int), "\n") + sizeof(int)] = 0;

\*flag = 1; while(\*flag==1) sleep(1);

printf("Sender: Received response: %s\n", shm + sizeof(int)); memset(shm + sizeof(int), 0, SHM\_SIZE - sizeof(int));

}

shmdt(shm); return 0

**receiver.c** #include<stdio.h> #include<stdlib.h> #include<sys/ipc.h> #include<sys/shm.h> #include<unistd.h> #include<string.h>

#define SHM\_SIZE 1024 int main(){

key\_t key = ftok("shmfile",65);

int shmid = shmget(key, SHM\_SIZE, 0666 | IPC\_CREAT); char \*shm = (char \*)shmat(shmid, NULL, 0);

int \*flag = (int \*)shm;

while(1){ while(\*flag==0)sleep(1);

printf("Receiver: Received message: %s\n", shm + sizeof(int)); printf("Receiver: Enter a response: ");

fgets(shm + sizeof(int), SHM\_SIZE - sizeof(int), stdin); shm[strcspn(shm + sizeof(int), "\n") + sizeof(int)] = 0;

\*flag = 0;

}

shmdt(shm); return 0;

}

**Sample Output Terminal 1**

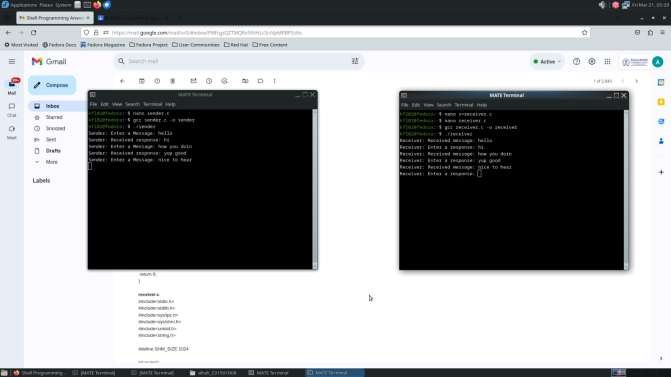
[root@localhost student]# gcc sender.c -o sender [root@localhost student]# ./sender

**Terminal 2**

[root@localhost student]# gcc receiver.c -o receiver [root@localhost student]# ./receiver

Message Received: Welcome to Shared Memory [root@localhost student]#

**Sample Output:**

****

**Result:**

Thus, written a C program to do Inter Process Communication (IPC) using shared memory between sender process and receiver process successfully.

**Ex. No.: 8**

**Date:**

# PRODUCER CONSUMER USING SEMAPHORES

**Aim:**

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

**Algorithm:**

1. Initialize semaphore empty, full and mutex.
2. Create two threads- producer thread and 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 critical section.
6. Call sem\_post on mutex semaphore followed by full semaphore
7. before exiting critical section.
8. Allow the other thread to enter its critical section.
9. Terminate after looping ten times in producer and consumer Threads each.

**Program Code:**

#include <stdio.h> #include <stdlib.h> #include <semaphore.h> #include <pthread.h>

#define BUFFER\_SIZE 3 int buffer[BUFFER\_SIZE]; int in = 0, out = 0;

int next\_item = 1; // Next item number to produce int items\_in\_buffer = 0; // Current items in buffer sem\_t empty;

sem\_t full; sem\_t mutex;

void produce() {

if (sem\_trywait(&empty) == 0) { sem\_wait(&mutex);

buffer[in] = next\_item;

printf("Producer produces the item:%d\n", next\_item); in = (in + 1) % BUFFER\_SIZE;

items\_in\_buffer++;

// Only increment next\_item if buffer wasn't empty before if (items\_in\_buffer >= 1) {

next\_item++;

} else {

// If buffer was empty, keep next\_item as is (it's already 1)

}

sem\_post(&mutex); sem\_post(&full);

} else {

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

}

}

void consume() {

if (sem\_trywait(&full) == 0) { sem\_wait(&mutex);

int item = buffer[out];

printf("Consumer consume product %d\n", item); out = (out + 1) % BUFFER\_SIZE;

items\_in\_buffer--;

// Reset counter when buffer becomes empty if (items\_in\_buffer == 0) {

next\_item = 1;

}

sem\_post(&mutex); sem\_post(&empty);

} else {

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

}

}

int main() { int choice;

// Initialize semaphores sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

sem\_init(&mutex, 0, 1);

// Initialize buffer to 0

for (int i = 0; i < BUFFER\_SIZE; i++) { buffer[i] = 0;

}

do {

printf("1. Produce\n2. Consume\n3. Exit\n"); printf("Enter choice:");

scanf("%d", &choice);

switch(choice) { case 1:

produce(); break;

case 2:

consume(); break;

case 3:

printf("Exiting...\n"); break;

default:

printf("Invalid choice!\n");

}

} while(choice != 3);

// Clean up sem\_destroy(&empty); sem\_destroy(&full); sem\_destroy(&mutex);

return 0;

**}**

**Sample Output:**

1. Producer 2.Consumer 3.Exit

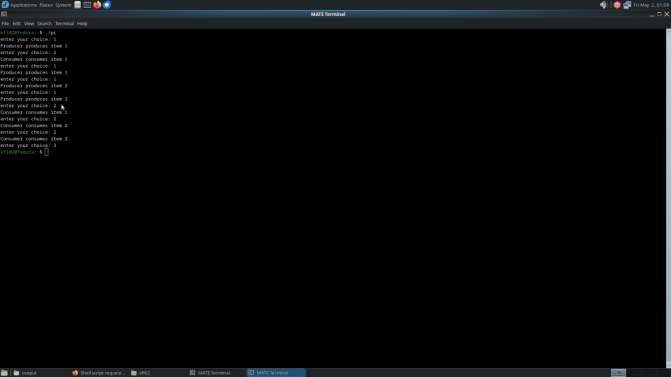
Enter your choice:1 Producer produces the item 1 Enter your choice:2 Consumer consumes item

1 Enter your choice:2 Buffer is empty!!

Enter your choice:1 Producer produces the item 1 Enter your choice:1 Producer produces the item 2 Enter your choice:1 Producer produces the item 3 Enter your choice:1

Buffer is full!! Enter your choice:3

**Sample Output:**

****

**Result:**

Thus, written a program to implement solution to producer consumer problem using semaphores successfully.

**Ex. No.: 9**

**Date:**

# 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

1. If no such i exists go to step 6
2. Compute work=work+allocationi
3. Assign finish[i] to true and go to step 2
4. If finish[i]==true for all i, then print safe sequence
5. Else print there is no safe sequence

**Program Code:**

#include <stdio.h> #include <stdbool.h>

int main() {

int n, m; // n = number of processes, m = number of resources printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter number of resources: "); scanf("%d", &m);

int available[m];

printf("Enter available resources: "); for (int i = 0; i < m; i++) {

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

}

int max[n][m], allocation[n][m], need[n][m]; printf("Enter max resource matrix:\n");

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

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

}

}

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

for (int j = 0; j < m; j++) { scanf("%d", &allocation[i][j]);

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

}

}

bool finish[n];

for (int i = 0; i < n; i++) finish[i] = false;

int safeSequence[n]; 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] > available[j]) { canAllocate = false; break;

}

}

if (canAllocate) {

for (int j = 0; j < m; j++) { available[j] += allocation[i][j];

}

safeSequence[count++] = i; finish[i] = true;

found = true;

}

}

}

if (!found) {

printf("System is not in a safe state.\n"); return 0;

}

}

printf("Safe Sequence: "); for (int i = 0; i < n; i++) {

printf("P%d ", safeSequence[i]);

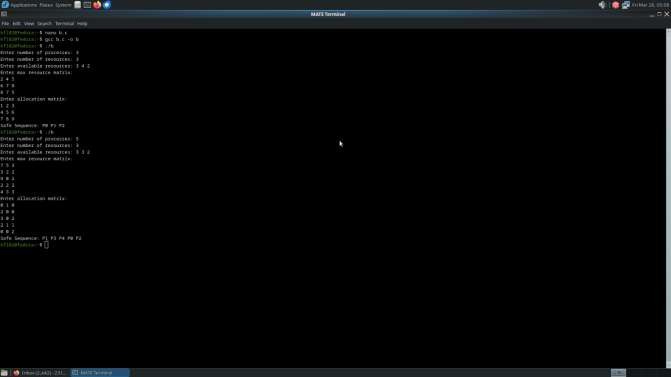
}

printf("\n");

return 0;

}

**Sample Output:**

****

**Result:**

Thus, found a safe sequence using Banker’s algorithm for deadlock avoidance successfully.

**Ex. No.: 10a)**

**Date:**

# 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 Code:**

#include <stdio.h> int main() {

int m, n;

printf("Enter number of memory blocks: "); scanf("%d", &m);

int blockSize[m], blockCopy[m]; printf("Enter block sizes: ");

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

blockCopy[i] = blockSize[i]; // Make a copy for displaying final block usage if needed

}

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

int processSize[n], allocation[n];

printf("Enter %d process sizes: ", n); for (int i = 0; i < n; i++) {

scanf("%d", &processSize[i]); allocation[i] = -1; // Initially no allocation

}

// Best Fit Allocation Logic for (int i = 0; i < n; i++) {

int bestIdx = -1;

for (int j = 0; j < m; j++) {

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) { bestIdx = j;

}

}

}

if (bestIdx != -1) { allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

// Display Output

printf("\nPROCESS NO\tPROCESS SIZE\tBLOCK NO\n"); for (int i = 0; i < n; i++) {

printf("%d\t\t%d\t\t", i + 1, processSize[i]); if (allocation[i] != -1)

printf("%d\n", allocation[i] + 1); else

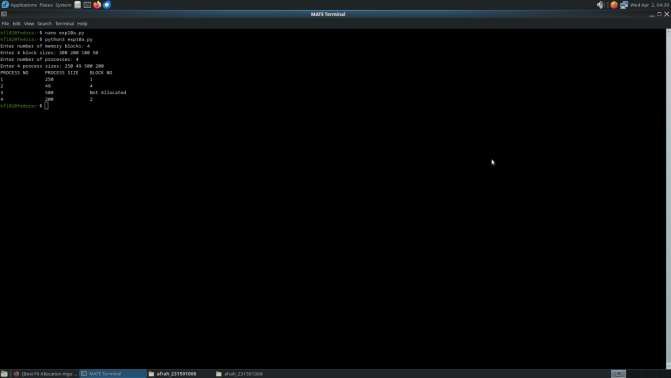
printf("Not Allocated\n");

}

return 0;

}

**Sample Output:**

****

**Result:**

Thus, implemented Best Fit memory allocation technique using Python successfully.

**Ex. No.: 10b)**

**Date:**

# FIRST FIT

**Aim:**

To write a C program for implementation memory allocation methods for fixed partition

using 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 for loop.

4: In for loop check bf[j]!=1, if so temp=b[j]-f[i] 5: Check highest

**Program Code:**

#include <stdio.h> int main() {

int blockCount, fileCount;

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

int blockSize[blockCount], fileSize[fileCount], blockAllocated[blockCount]; int allocation[fileCount], fragment[fileCount], allocatedBlockSize[fileCount];

// Input block sizes

printf("Enter the size of the blocks:\n"); for (int i = 0; i < blockCount; i++) {

printf("Block %d: ", i + 1);

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

blockAllocated[i] = 0; // Initially all blocks are free

}

// Input file sizes

printf("Enter the size of the files:\n"); for (int i = 0; i < fileCount; i++) {

printf("File %d: ", i + 1);

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

allocation[i] = -1; // Initialize as not allocated

}

// First Fit Allocation

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

for (int j = 0; j < blockCount; j++) {

if (!blockAllocated[j] && blockSize[j] >= fileSize[i]) { allocation[i] = j;

fragment[i] = blockSize[j] - fileSize[i]; allocatedBlockSize[i] = blockSize[j]; blockAllocated[j] = 1; // mark block as allocated break;

}

}

}

// Output

printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n"); for (int i = 0; i < fileCount; i++) {

printf("%d\t%d\t\t", i + 1, fileSize[i]); if (allocation[i] != -1) {

printf("%d\t\t%d\t\t%d\n", allocation[i] + 1, allocatedBlockSize[i], fragment[i]);

} else {

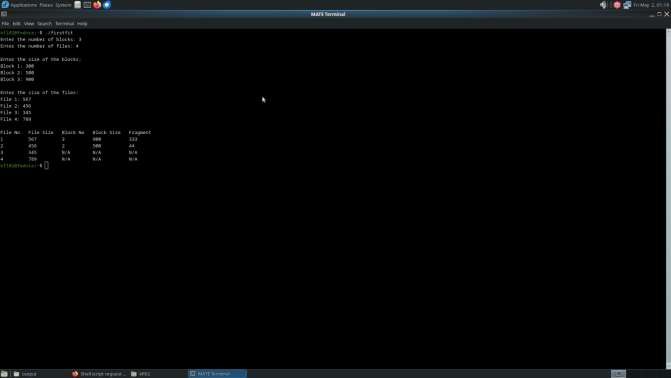
printf("N/A\t\tN/A\t\tN/A\n");

}

}

return 0;

**Sample Output:**

****

**Result:**

Thus, written a C program for implementation memory allocation methods for fixed partition using first fit successfully.

**Ex. No.: 11a)**

**Date:**

# FIFO PAGE REPLACEMENT

**Aim:**

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

replacement technique.

**Algorithm:**

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

**Program Code:**

#include <stdio.h> #include <stdbool.h>

int main() {

int frames[10], refStr[50], n, frameCount, i, j, k, pos = 0, faults = 0; bool isHit;

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

printf("Enter the reference string:\n"); for (i = 0; i < n; i++) {

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

}

printf("Enter number of frames: "); scanf("%d", &frameCount);

// Initialize frames with -1

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

}

printf("\nPage\tFrames\t\tPage Fault\n"); for (i = 0; i < n; i++) {

isHit = false;

// Check if page is already in frame for (j = 0; j < frameCount; j++) {

if (frames[j] == refStr[i]) { isHit = true;

break;

}

}

if (!isHit) {

frames[pos] = refStr[i];

pos = (pos + 1) % frameCount; faults++;

printf("%d\t", refStr[i]);

for (k = 0; k < frameCount; k++) { if (frames[k] != -1)

printf("%d ", frames[k]); else

printf("- ");

}

printf("\tYes\n");

} else {

printf("%d\t", refStr[i]);

for (k = 0; k < frameCount; k++) { if (frames[k] != -1)

printf("%d ", frames[k]); else

printf("- ");

}

printf("\tNo\n");

}

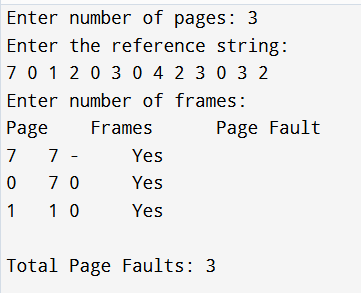
}

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

return 0;

}

**Sample Output:**

****

**Result :**

Thus, found the number of page faults that occur using First-in First-out (FIFO) page replacement technique successfully.

**Ex. No.: 11b)**

**Date:**

**Aim:**

**LRU**

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 Code:**

#include <stdio.h>

int findLRU(int time[], int n) { int i, min = time[0], pos = 0; for (i = 1; i < n; ++i) {

if (time[i] < min) { min = time[i]; pos = i;

}

}

return pos;

}

int main() {

int frames, pages, i, j, k, pos, faultCount = 0; int frame[10], page[30], time[10], counter = 0; int flag1, flag2;

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

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

printf("Enter reference string: "); for (i = 0; i < pages; i++) {

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

}

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

}

for (i = 0; i < pages; i++) { flag1 = flag2 = 0;

for (j = 0; j < frames; j++) { if (frame[j] == page[i]) {

counter++; time[j] = counter; flag1 = flag2 = 1; break;

}

}

if (!flag1) {

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

counter++; faultCount++; frame[j] = page[i]; time[j] = counter; flag2 = 1;

break;

}

}

}

if (!flag2) {

pos = findLRU(time, frames); counter++;

faultCount++; frame[pos] = page[i]; time[pos] = counter;

}

// Print current frame status for (k = 0; k < frames; k++) {

if (frame[k] != -1)

printf("%d ", frame[k]); else

printf("-1 ");

}

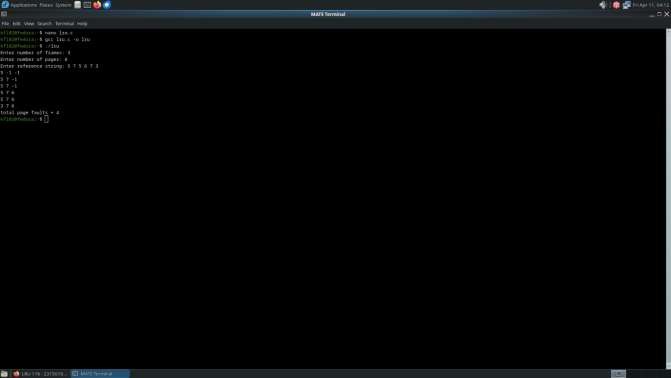
printf("\n");

}

printf("Total page faults = %d\n", faultCount); return 0;

}

**Sample Output:**

****

**Result:**

Thus, written a c program to implement LRU page replacement algorithm successfully.

**Ex. No.: 11c)**

**Date:**

**Aim:**

**Optimal**

To write a c program to implement 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.
8. Display the values
9. Stop the process

# PROGRAM:

#include <stdio.h>

int search(int key, int frame[], int frames) { for (int i = 0; i < frames; i++) {

if (frame[i] == key) return 1;

}

return 0;

}

int predict(int pages[], int frame[], int n, int index, int frames) { int res = -1, farthest = index;

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

for (j = index; j < n; j++) { if (frame[i] == pages[j]) {

if (j > farthest) { farthest = j; res = i;

}

break;

}

}

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

return i;

}

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

}

int main() {

int frames, n, i, pageFaults = 0;

int pages[100], frame[10], index = 0;

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

printf("Enter the page reference string: "); for (i = 0; i < n; i++)

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

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

for (i = 0; i < frames; i++) frame[i] = -1;

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

if (search(pages[i], frame, frames)) {

// Page Hit, do nothing continue;

}

// Page Fault

if (index < frames) { frame[index++] = pages[i];

} else {

int pos = predict(pages, frame, n, i + 1, frames); frame[pos] = pages[i];

}

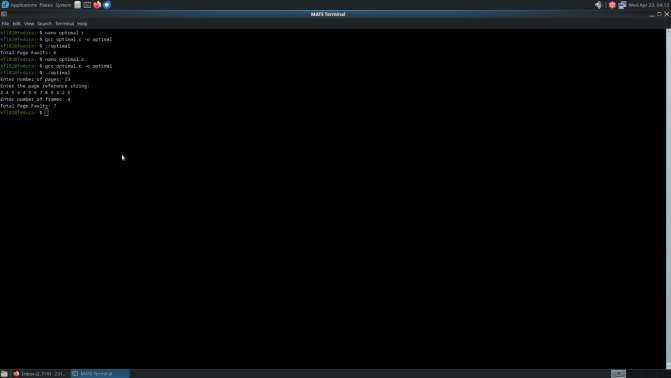
pageFaults++;

}

printf("Total Page Faults = %d\n", pageFaults); return 0;

}

**Sample Output:**

****

**Result:**

Thus, written a c program to implement Optimal page replacement algorithm successfully.

**Ex. No.: 12**

**Date:**

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

# AIM:

To implement File Organization Structures in C are

1. Single Level Directory
2. Two-Level Directory
3. Hierarchical Directory Structure
4. Directed Acyclic Graph Structure
   1. **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.
      4. Display the files that are available in the directories.
      5. Stop.

# PROGRAM:

#include <stdio.h> #include <stdlib.h> #include <string.h>

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

};

int main() { int choice;

printf("Choose Directory Type:\n"); printf("1. Single-Level Directory\n"); printf("2. Two-Level Directory\n"); printf("Enter choice: ");

scanf("%d", &choice);

if (choice == 1) {

// Single-Level Directory struct Directory dir;

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

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

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

}

printf("\nDirectory: %s\n", dir.name); printf("Files:\n");

for (int i = 0; i < dir.fileCount; i++) { printf(" %s\n", dir.files[i]);

}

printf("\nGraphical Representation (Single-Level Directory):\n"); printf("%s\n", dir.name);

for (int i = 0; i < dir.fileCount; i++) { printf(" └── %s\n", dir.files[i]);

}

} else if (choice == 2) {

// Two-Level Directory

struct Directory userDirs[10]; int userCount;

printf("Enter the number of user directories: "); scanf("%d", &userCount);

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

printf("Enter name of user directory %d: ", i + 1); scanf("%s", userDirs[i].name);

printf("Enter number of files in %s: ", userDirs[i].name); scanf("%d", &userDirs[i].fileCount);

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

}

}

printf("\nDirectory Structure:\n"); for (int i = 0; i < userCount; i++) {

printf("User Directory: %s\n", userDirs[i].name); printf("Files:\n");

for (int j = 0; j < userDirs[i].fileCount; j++) { printf(" %s\n", userDirs[i].files[j]);

}

printf("\n");

}

} else {

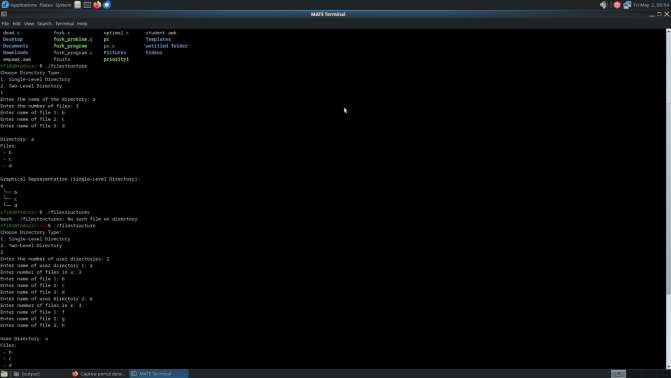
printf("Invalid choice.\n");

}

return 0;

}

**Sample Output:**

****

* 1. **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> #include <stdlib.h> #include <string.h>

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

};

int main() {

struct Directory userDirs[10]; int userCount;

printf("Enter the number of user directories: "); scanf("%d", &userCount);

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

printf("Enter name of user directory %d: ", i + 1); scanf("%s", userDirs[i].name);

printf("Enter number of files in %s: ", userDirs[i].name); scanf("%d", &userDirs[i].fileCount);

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

}

}

printf("\nDirectory Structure:\n"); for (int i = 0; i < userCount; i++) {

printf("User Directory: %s\n", userDirs[i].name); printf("Files:\n");

for (int j = 0; j < userDirs[i].fileCount; j++) {

printf(" %s\n", userDirs[i].files[j]);

}

printf("\n");

}

// Graphical Representation

printf("Graphical Representation (Two-Level Directory):\n"); printf("Root\n");

for (int i = 0; i < userCount; i++) { printf("├── %s\n", userDirs[i].name);

for (int j = 0; j < userDirs[i].fileCount; j++) { printf("│ └── %s\n", userDirs[i].files[j]);

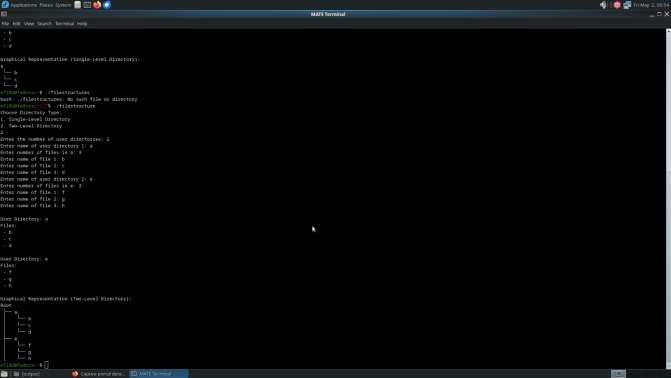
}

}

return 0;

}

**Sample Output:**

****

**Result:**

Thus, implemented Single Level Directory and Two-Level Directory File Organization Structures in C successfully.