

**EXP NO:** 1a**) INSTALLATION AND CONFIGURATION OF LINUX**

**DATE:22/1/25**

**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

8. Choose puppy-linux.iso then kernel version

9. Select CPU and RAM limits

10.Create default disk image to 8 GB

11.Click finish for creating the new VM with PuppyLinux

**EXP NO:** 1b**) BASIC LINUX COMMANDS**

**DATE:22/1/25**

**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

**2. 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

**4. 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

**5. 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

**6. The ‘who am i’ command**

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

SYNTAX: $ who am i

**7. The ‘id’ command**

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

SYNTAX: $ id

**8. The ‘tty’ command**

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

SYNTAX: $ tty

**9. The ‘clear’ command**

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

SYNTAX: $ clear

**10. The ‘man’ command**

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

SYNTAX: $ man [command]

**11. 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

**12. 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

**2. The ‘mkdir’ command:**

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

SYNTAX: $ mkdir dirname

EXAMPLE: $ mkdir receee

**3. 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

**4. The ‘cd’ command:**

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

SYNTAX: $ cd dirname

EXAMPLE: $ cd receee

**5. 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

**2. The ‘Display contents of a file’ command:**

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

SYNTAX: $ cat filename

**3. 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

**4. 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.

**5. 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

**6. The ‘file’ command:**

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

SYNTAX: $ file filename

EXAMPLE: $ file receee

**7. 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

**8. 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

**9. 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

**10. 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

**11. 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.

**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.

**13. 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.

**14. 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

**2. 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

**3. 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.

**2. 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.

**3. The more filter:**

The pg command shows the file page by page.

SYNTAX: $ ls –l | more

**4. 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

**5. 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

**6. 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

**7. 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

**2. 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

**4. 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

**5. 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

**6. 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

**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

**EXP NO:** 2a**) WRITE A SHELLSCRIPT TO DISPLAY BASIC CALCULATOR**

**DATE:31/1/25**

**PROGRAM:**

#!/bin/bash

echo "-------------------------"

echo "   Basic Calculator"

echo "-------------------------"

echo "Enter first number:"

read a

echo "Enter second number:"

read b

echo "Select operation:"

echo "1. Addition (+)"

echo "2. Subtraction (-)"

echo "3. Multiplication (\*)"

echo "4. Division (/)"

read choice

case $choice in

  1)

    result=$((a + b))

    echo "Result: $a + $b = $result"

    ;;

  2)

    result=$((a - b))

    echo "Result: $a - $b = $result"

    ;;

  3)

    result=$((a \* b))

    echo "Result: $a \* $b = $result"

    ;;

  4)

    if [ $b -ne 0 ]; then

      result=$(echo "scale=2; $a / $b" | bc)

      echo "Result: $a / $b = $result"

    else

      echo "Error: Division by zero not allowed."

    fi

    ;;

  \*)

    echo "Invalid choice."

    ;;

esac

**OUTPUT:**

-------------------------

Basic Calculator

-------------------------

Enter first number:

5

Enter second number:

4

Select operation:

1. Addition (+)

2. Subtraction (-)

3. Multiplication (\*)

4. Division (/)

1

Result: 5 + 4 = 9

**EXP NO:** 2b**) WRITE A SHELLSCRIPT TO CHECK LEAP YEAR OR NOT**

**DATE:31/1/25**

**PROGRAM:**

#!/bin/bash

# Prompt the user to enter a year

echo "Enter a year:"

read year

# Check if the year is divisible by 4

if (( year % 4 == 0 )); then

    # If it is divisible by 100, it should also be divisible by 400 to be a leap year

    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 is not a leap year"

fi

**OUTPUT:**

Enter a year:

2004

2004 is a leap year

**EXP NO:** 3a**) WRITE A SHELLSCRIPT TO REVERSE OF DIGIT**

**DATE:5/2/25**

**PROGRAM:**

#!/bin/bash

# Prompt the user to enter a number

echo "Enter a number:"

read num

# Initialize variables

rev=0

# Loop to reverse the number

while [ $num -gt 0 ]

do

    # Get the last digit of the number

    digit=$((num % 10))

    # Add the digit to the reversed number

    rev=$((rev \* 10 + digit))

    # Remove the last digit from the number

    num=$((num / 10))

done

# Display the reversed number

echo "Reversed number: $rev"

**OUTPUT:**

Enter a number:

1234

Reversed number: 4321

**EXP NO:** 3b**) WRITE A SHELLSCRIPT TO FIBONACCI SERIES**

**DATE:5/2/25**

**PROGRAM:**

#!/bin/bash

# Prompt the user to enter a number

echo "Enter a number:"

read num

# Initialize the first two numbers in the Fibonacci series

a=0

b=1

# Print the Fibonacci series

echo "Fibonacci series up to $num:"

echo $a

echo $b

# Loop to generate Fibonacci numbers

for (( i=2; i<=num; i++ ))

do

    # Calculate the next Fibonacci number

    fib=$((a + b))

    # Print the Fibonacci number

    echo $fib

    # Update a and b for the next iteration

    a=$b

    b=$fib

done

**OUTPUT:**

Enter a number:

5

Fibonacci series up to 5:

0

1

1

2

3

5

**EXP NO:** 4a**) EMPLOYEE AVERAGE PAY**

**DATE:12/2/25**

**PROGRAM:**

#!/bin/bash

# Initialize variables

total\_pay=0

employee\_count=0

# Read the number of employees

echo "Enter the number of employees:"

read num\_employees

# Loop to get employee details

for ((i=1; i<=num\_employees; i++))

do

    echo "Enter name of employee $i:"

    read name

    echo "Enter salary of $name:"

    read salary

    # Add the salary to the total pay

    total\_pay=$((total\_pay + salary))

    employee\_count=$((employee\_count + 1))

done

# Calculate average pay

if [ $employee\_count -gt 0 ]; then

    average\_pay=$(echo "scale=2; $total\_pay / $employee\_count" | bc)

else

    average\_pay=0

fi

# Display results

echo "No of employees are = $employee\_count"

echo "Total pay = $total\_pay"

echo "Average pay = $average\_pay"

**OUTPUT:**

Enter the number of employees:

2

Enter name of employee 1:

faisal

Enter salary of faisal:

10000000

Enter name of employee 2:

mohan

Enter salary of mohan:

20000000

No of employees are = 2

Total pay = 30000000

Average pay =1,50,00,000

**EXP NO:** 4b**) RESULT OF EXAMINATION**

**DATE:12/2/25**

**PROGRAM:**

#!/bin/bash

# Read the file line by line

while read -r line

do

    # Split the line into an array

    arr=($line)

    # Get the student's name

    name=${arr[0]}

    # Get the marks of the student

    marks=${arr[@]:1}

    # Initialize a variable to track fail status

    fail=false

    # Check if any of the marks is less than 45

    for mark in ${marks[@]}

    do

        if [ $mark -lt 45 ]; then

            fail=true

            break

        fi

    done

    # Print the pass/fail status based on the condition

    if [ "$fail" = true ]; then

        echo "$name FAIL"

    else

        echo "$name PASS"

    fi

done < students.dat

**OUTPUT:**

BEN FAIL

: integer expression expected0

TOM PASS

: integer expression expected0

RAM PASS

: integer expression expected7

JIM PASS

**EXP NO:** 5 **SYSTEM CALLS PROGRAMMING**

**DATE:12/2/25**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main() {

    // Declare an integer variable pid to store the process ID

    pid\_t pid;

    // Create a new process using fork()

    pid = fork();

    // Print the statement executed twice (both parent and child process)

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

    // Check if fork() failed (pid == -1)

    if (pid == -1) {

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

        exit(0);  // Exit if the process creation fails

    }

    // If the process is the child (pid == 0)

    if (pid == 0) {

        // Print the process ID of the child and the parent process ID of the child

        printf("Child Process ID: %d\n", getpid());

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

        // Optionally, we could use execlp() to execute a new program, e.g., /bin/ls

        // execlp("/bin/ls", "ls", (char \*) NULL);  // Uncomment if you want to run 'ls'

    }

    // If the process is the parent (pid > 0)

    else {

        // Print the process ID of the parent and the parent's parent process ID

        printf("Parent Process ID: %d\n", getpid());

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

    }

    // Final print statement executed by both parent and child

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

    return 0;

}

**OUTPUT:**

THIS LINE EXECUTED TWICE

Parent Process ID: 2306

Parent's Parent Process ID: 2299

IT CAN BE EXECUTED TWICE

THIS LINE EXECUTED TWICE

Child Process ID: 2307

Parent Process ID of Child: 2306

IT CAN BE EXECUTED TWICE

**EXP NO:** 6a) **FIRST COME FIRST SERVE**

**DATE:16/2/25**

**PROGRAM:**

#include <stdio.h>

int main() {

    int n;

    int burstTime[10], waitingTime[10], turnAroundTime[10];

    int totalWaitingTime = 0, totalTurnAroundTime = 0;

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

    scanf("%d", &n);

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

    for (int i = 0; i < n; i++) {

        scanf("%d", &burstTime[i]);

    }

    waitingTime[0] = 0;

    for (int i = 1; i < n; i++) {

        waitingTime[i] = burstTime[i - 1] + waitingTime[i - 1];

    }

    for (int i = 0; i < n; i++) {

        turnAroundTime[i] = burstTime[i] + waitingTime[i];

    }

    printf("\nProcess Burst Time Waiting Time Turn Around Time\n");

    for (int i = 0; i < n; i++) {

        printf("%d\t\t%d\t\t%d\t\t%d\n", i, burstTime[i], waitingTime[i], turnAroundTime[i]);

    }

    for (int i = 0; i < n; i++) {

        totalWaitingTime += waitingTime[i];

        totalTurnAroundTime += turnAroundTime[i];

    }

    float averageWaitingTime = (float)totalWaitingTime / n;

    float averageTurnAroundTime = (float)totalTurnAroundTime / n;

    printf("\nAverage waiting time is: %.2f\n", averageWaitingTime);

    printf("Average Turnaround Time is: %.2f\n", averageTurnAroundTime);

    return 0;

}

**OUTPUT:**

Enter the number of processes: 2

Enter the burst time of the processes:

5

8

Process Burst Time Waiting Time Turn Around Time

0 5 0 5

1 8 5 13

Average waiting time is: 2.50

Average Turnaround Time is: 9.00

**EXP NO:**6b)**SHORTEST JOB FIRST**

**DATE:16/2/25**

**PROGRAM:**

#include <stdio.h>

struct Process {

    int processID;

    int burstTime;

    int waitingTime;

    int turnAroundTime;

};

void sortProcesses(struct Process processes[], int n) {

    struct Process temp;

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (processes[i].burstTime > processes[j].burstTime) {

                temp = processes[i];

                processes[i] = processes[j];

                processes[j] = temp;

            }

        }

    }

}

int main() {

    int n;

    float totalWaitingTime = 0, totalTurnAroundTime = 0;

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

    scanf("%d", &n);

    struct Process processes[n];

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

    for (int i = 0; i < n; i++) {

        processes[i].processID = i + 1;  // Assigning process ID

        scanf("%d", &processes[i].burstTime);

        processes[i].waitingTime = 0;

        processes[i].turnAroundTime = 0;

    }

    sortProcesses(processes, n);

    processes[0].waitingTime = 0; // The waiting time of the first process is always 0

    for (int i = 1; i < n; i++) {

        processes[i].waitingTime = processes[i - 1].burstTime + processes[i - 1].waitingTime;

    }

    for (int i = 0; i < n; i++) {

        processes[i].turnAroundTime = processes[i].burstTime + processes[i].waitingTime;

    }

    printf("\nProcess Burst Time Waiting Time Turn Around Time\n");

    for (int i = 0; i < n; i++) {

        printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i].processID, processes[i].burstTime,

               processes[i].waitingTime, processes[i].turnAroundTime);

        totalWaitingTime += processes[i].waitingTime;

        totalTurnAroundTime += processes[i].turnAroundTime;

    }

    printf("\nAverage waiting time is: %.2f\n", totalWaitingTime / n);

    printf("Average Turn Around Time is: %.2f\n", totalTurnAroundTime / n);

    return 0;

}

**OUTPUT:**

Enter the number of processes: 2

Enter the burst time of the processes:

4

6

Process Burst Time Waiting Time Turn Around Time

1 4 0 4

2 6 4 10

Average waiting time is: 2.00

Average Turn Around Time is: 7.00

**EXP NO:**6c)**PRIORITY SCHEDULING**

**DATE:16/2/25**

**PROGRAM:**

#include <stdio.h>

struct Process {

    intprocessID;

    intburstTime;

    int priority;

    intwaitingTime;

    intturnAroundTime;

};

voidsortProcesses(struct Process processes[], int n) {

    struct Process temp;

    for (inti = 0; i< n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (processes[i].priority > processes[j].priority) {

                temp = processes[i];

                processes[i] = processes[j];

                processes[j] = temp;

            }

        }

    }

}

int main() {

    int n;

    float totalWaitingTime = 0, totalTurnAroundTime = 0;

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

    scanf("%d", &n);

    struct Process processes[n];

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

    for (inti = 0; i< n; i++) {

        processes[i].processID = i + 1;

        printf("Process %d - Burst Time: ", i + 1);

        scanf("%d", &processes[i].burstTime);

        printf("Process %d - Priority: ", i + 1);

        scanf("%d", &processes[i].priority);

        processes[i].waitingTime = 0;

        processes[i].turnAroundTime = 0;

    }

    sortProcesses(processes, n);

    processes[0].waitingTime = 0; // The waiting time of the first process is always 0

    for (inti = 1; i< n; i++) {

        processes[i].waitingTime = processes[i - 1].burstTime + processes[i - 1].waitingTime;

    }

    for (inti = 0; i< n; i++) {

        processes[i].turnAroundTime = processes[i].burstTime + processes[i].waitingTime;

    }

    printf("\nProcess Burst Time Priority Waiting Time Turn Around Time\n");

    for (inti = 0; i< n; i++) {

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].processID, processes[i].burstTime,

               processes[i].priority, processes[i].waitingTime, processes[i].turnAroundTime);

        totalWaitingTime += processes[i].waitingTime;

        totalTurnAroundTime += processes[i].turnAroundTime;

    }

    printf("\nAverage waiting time is: %.2f\n", totalWaitingTime / n);

    printf("Average Turn Around Time is: %.2f\n", totalTurnAroundTime / n);

    return 0;

}

**OUTPUT:**

Enter the number of processes: 2

Enter the burst time and priority of the processes:

Process 1 - Burst Time: 3

Process 1 - Priority: 2

Process 2 - Burst Time: 4

Process 2 - Priority: 1

Process Burst Time Priority Waiting Time Turn Around Time

2 4 1 0 4

1 3 2 4 7

Average waiting time is: 2.00

Average Turn Around Time is: 5.50

**EXP NO:**6d)**ROUND ROBIN SCHEDULING**

**DATE:16/2/25**

**PROGRAM:**

#include <stdio.h>

struct Process {

    intprocessID;

    intburstTime;

    intarrivalTime;

    intremainingTime;

    intwaitingTime;

    intturnAroundTime;

};

int main() {

    int n, quantum;

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

    scanf("%d", &n);

    printf("Enter the time quantum: ");

    scanf("%d", &quantum);

    struct Process processes[n];

    // Reading process details

    for (inti = 0; i< n; i++) {

        processes[i].processID = i + 1;

        printf("Enter the burst time and arrival time of process %d:\n", i + 1);

        scanf("%d %d", &processes[i].burstTime, &processes[i].arrivalTime);

        processes[i].remainingTime = processes[i].burstTime;

        processes[i].waitingTime = 0;

        processes[i].turnAroundTime = 0;

    }

    int t = 0;  // Initialize time

    int completed = 0;

    // Round Robin Scheduling

    while (completed < n) {

        for (inti = 0; i< n; i++) {

            if (processes[i].remainingTime> 0) {

                if (processes[i].remainingTime> quantum) {

                    t += quantum;

                    processes[i].remainingTime -= quantum;

                } else {

                    t += processes[i].remainingTime;

                    processes[i].waitingTime = t - processes[i].burstTime - processes[i].arrivalTime;

                    processes[i].remainingTime = 0;

                    completed++;

                }

            }

        }

    }

// Calculate Turnaround Time

    for (inti = 0; i< n; i++) {

        processes[i].turnAroundTime = processes[i].waitingTime + processes[i].burstTime;

    }

    // Display results

    float totalWaitingTime = 0, totalTurnAroundTime = 0;

    printf("\nProcess Burst Time Arrival Time Waiting Time Turnaround Time\n");

    for (inti = 0; i< n; i++) {

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].processID, processes[i].burstTime,

               processes[i].arrivalTime, processes[i].waitingTime, processes[i].turnAroundTime);

        totalWaitingTime += processes[i].waitingTime;

        totalTurnAroundTime += processes[i].turnAroundTime;

    }

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

    printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);

    return 0;

}

**OUTPUT:**

Enter the number of processes: 2

Enter the time quantum: 2

Enter the burst time and arrival time of process 1:

4

0

Enter the burst time and arrival time of process 2:

6

4

Process Burst Time Arrival Time Waiting Time Turnaround Time

1 4 0 2 6

2 6 4 0 6

Average Waiting Time: 1.00

Average Turnaround Time: 6.00

**EXP NO:**7**IPC-USING SHARED MEMORY**

**DATE:19/2/25**

**PROGRAM:**

**Sender.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

int main() {

    key\_t key = ftok("shmfile", 65); // Create a unique key for shared memory

    intshmid = shmget(key, 1024, 0666 | IPC\_CREAT); // Allocate shared memory segment

    char \*message = (char\*) shmat(shmid, NULL, 0); // Attach shared memory segment

    // Writing a message to shared memory

    sprintf(message, "Welcome to Shared Memory");

    // Set a delay

    sleep(2);

    printf("Message Sent: %s\n", message);

    // Detach the shared memory segment

    shmdt(message);

    return 0;

}

**receiver.c**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

int main() {

    key\_t key = ftok("shmfile", 65); // Create a unique key for shared memory

    intshmid = shmget(key, 1024, 0666); // Get the shared memory segment

    char \*message = (char\*) shmat(shmid, NULL, 0); // Attach shared memory segment

    // Print the message received from the shared memory

    printf("Message Received: %s\n", message);

    // Detach the shared memory segment

    shmdt(message);

    // Remove the shared memory segment

    shmctl(shmid, IPC\_RMID, NULL);

    return 0;

}

**OUTPUT:**

Message Sent: Welcome to Shared Memory.

Message Received: Welcome to Shared Memory.

**EXP NO:**8**PRODUCER CONSUMER USING SEMAPHORES**

**DATE:19/2/25**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 5 // Define buffer size

int buffer[BUFFER\_SIZE]; // Shared buffer

int in = 0, out = 0; // Indices for producer and consumer

sem\_t empty, full, mutex; // Semaphores for synchronization

// Function for Producer

void\* producer(void\* param) {

int item;

for (inti = 0; i< 10; i++) {

item = i + 1; // Item produced

sem\_wait(&empty); // Wait for an empty slot

sem\_wait(&mutex); // Enter critical section

// Produce an item and add it to the buffer

buffer[in] = item;

printf("Producer produces the item %d\n", item);

in = (in + 1) % BUFFER\_SIZE; // Update the index for the next producer

sem\_post(&mutex); // Exit critical section

sem\_post(&full); // Signal that a new item is added to the buffer

sleep(1); // Simulate time taken to produce

}

pthread\_exit(0); // Exit the producer thread

}

// Function for Consumer

void\* consumer(void\* param) {

int item;

for (inti = 0; i< 10; i++) {

sem\_wait(&full); // Wait for an item to consume

sem\_wait(&mutex); // Enter critical section

// Consume an item from the buffer

item = buffer[out];

printf("Consumer consumes item %d\n", item);

out = (out + 1) % BUFFER\_SIZE; // Update the index for the next consumer

sem\_post(&mutex); // Exit critical section

sem\_post(&empty); // Signal that a space is available in the buffer

sleep(1); // Simulate time taken to consume

}

pthread\_exit(0); // Exit the consumer thread

}

int main() {

pthread\_tproducer\_thread, consumer\_thread;

// Initialize semaphores

sem\_init(&empty, 0, BUFFER\_SIZE); // Initially, all buffer slots are empty

sem\_init(&full, 0, 0); // Initially, no items are in the buffer

sem\_init(&mutex, 0, 1); // Mutex to ensure mutual exclusion

// Create producer and consumer threads

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

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

// Wait for threads to finish

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

// Destroy semaphores

sem\_destroy(&empty);

sem\_destroy(&full);

sem\_destroy(&mutex);

printf("Producer and Consumer have finished\n");

return 0;

}

**OUTPUT:**

Producer produces the item 1

Consumer consumes item 1

Producer produces the item 2

Consumer consumes item 2

Producer produces the item 3

Consumer consumes item 3

Producer produces the item 4

Consumer consumes item 4

Producer produces the item 5

Consumer consumes item 5

Producer produces the item 6

Consumer consumes item 6

Producer produces the item 7

Consumer consumes item 7

Producer produces the item 8

Consumer consumes item 8

Producer produces the item 9

Consumer consumes item 9

Producer produces the item 10

Consumer consumes item 10

Producer and Consumer have finished

**EXP NO:**9 **DEADLOCK AVOIDANCE**

**DATE:19/2/25**

**PROGRAM:**

#include <stdio.h>

#include <stdbool.h>

#define P 5  // Number of processes

#define R 3  // Number of resources

// Function to find the safe sequence using Banker's Algorithm

boolisSafe(int processes[], int avail[], int max[][R], int allot[][R], intsafeSeq[]) {

    int need[P][R];

    bool finish[P] = {false};

    int work[R];

    int count = 0;

    // Calculate the 'need' matrix

    for (inti = 0; i< P; i++) {

        for (int j = 0; j < R; j++) {

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

        }

    }

    // Initialize the work vector with available resources

    for (inti = 0; i< R; i++) {

        work[i] = avail[i];

    }

    // Find the safe sequence

    while (count < P) {

        bool found = false;

        for (int p = 0; p < P; p++) {

            if (!finish[p]) {

                inti;

                // Check if all resources needed by process p are available

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

                    if (need[p][i] > work[i]) {

                        break;

                    }

                }

                // If all needs are met, allocate resources and mark process as finished

                if (i == R) {

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

                        work[j] += allot[p][j];

                    }

                    safeSeq[count++] = processes[p];

                    finish[p] = true;

                    found = true;

                }

            }

        }

        // If no process is found that can proceed, return false

        if (!found) {

            return false;

        }

    }

    return true;

}

int main() {

    int processes[] = {0, 1, 2, 3, 4}; // Process IDs

    // Available instances of resources

    int avail[] = {3, 3, 2}; // Example: 3 instances of resource 1, 3 of resource 2, 2 of resource 3

    // Maximum demand of each process for each resource

    int max[][R] = {

        {7, 5, 3},

        {3, 2, 2},

        {9, 0, 2},

        {2, 2, 2},

        {4, 3, 3}

    };

    // Allocation matrix (resources allocated to each process)

    int allot[][R] = {

        {0, 1, 0},

        {2, 0, 0},

        {3, 0, 2},

        {2, 1, 1},

        {0, 0, 2}

    };

    intsafeSeq[P];

    // Check for the safe sequence

    if (isSafe(processes, avail, max, allot, safeSeq)) {

        printf("The SAFE Sequence is:\n");

        for (inti = 0; i< P; i++) {

            printf("P%d -> ", safeSeq[i]);

        }

        printf("\n");

    } else {

        printf("There is no safe sequence\n");

    }

    return 0;

}

**OUTPUT:**

The SAFE Sequence is:

P1 -> P3 -> P4 -> P0 -> P2

**EXP NO:**10a)**BEST FIT**

**DATE:25/2/25**

**PROGRAM:**

#include <stdio.h>

voidbestFit(intblockSize[], int m, intprocessSize[], int n) {

    int allocation[n];

    for (inti = 0; i< n; i++) {

        allocation[i] = -1;

    }

    for (inti = 0; i< n; i++) {

        intbestIdx = -1;

        for (int j = 0; j < m; j++) {

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

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

                    bestIdx = j;

                }

            }

        }

        if (bestIdx != -1) {

            allocation[i] = bestIdx;

            blockSize[bestIdx] -= processSize[i];

        }

    }

    printf("Process No.\tProcess Size\tBlock No.\n");

    for (inti = 0; i< n; i++) {

        if (allocation[i] != -1) {

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

        } else {

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

        }

    }

}

int main() {

    intblockSize[] = { 100, 500, 200, 300, 600 };

    intprocessSize[] = { 212, 417, 112, 426 };

    int m = sizeof(blockSize) / sizeof(blockSize[0]);

    int n = sizeof(processSize) / sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);

    return 0;

}

**OUTPUT:**

Process No. Process Size Block No.

1 212 4

2 417 2

3 112 3

4 426 5

**EXP NO:**10b)**FIRST FIT**

**DATE:25/2/25**

**PROGRAM:**

#include <stdio.h>

#define MAX 25

int main() {

    int frag[MAX], b[MAX], f[MAX], i, j, nb, nf, temp, highest = 0, 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 + 1;

                    bf[j] = 1;

                    frag[i] = temp;

                    break;

                }

            }

        }

    }

    printf("\nFile No.\tFile Size\tBlock No.\tBlock Size\tFragmentation\n");

    for(i = 0; i<nf; i++) {

        if(ff[i] != 0) {

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

        } else {

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

        }

    }

    return 0;

}

**OUTPUT:**

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:

Block 1: 3

Block 2: 4

Block 3: 2

Enter the size of the files:

File 1: 2

File 2: 4

File No. File Size Block No. Block Size Fragmentation

1 2 1 3 1

2 4 2 4 0

**EXP NO:**11a) **FIFO PAGE REPLACEMENT**

**DATE:28/2/25**

**PROGRAM:**

#include <stdio.h>

#define MAX 10

voidfifoPageReplacement(int pages[], int n, int capacity) {

int frame[capacity];

intpageFaults = 0, front = 0, rear = 0, flag;

for (inti = 0; i< capacity; i++) {

frame[i] = -1;

}

printf("Page Reference String: ");

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

printf("%d ", pages[i]);

}

printf("\n");

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

flag = 0;

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

if (frame[j] == pages[i]) {

flag = 1;

break;

}

}

if (flag == 0) {

frame[rear] = pages[i];

rear = (rear + 1) % capacity;

pageFaults++;

printf("Page Fault: ");

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

if (frame[j] != -1) {

printf("%d ", frame[j]);

}

}

printf("\n");

}

}

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

}

int main() {

int pages[MAX], n, capacity;

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

scanf("%d", &n);

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

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

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

}

printf("Enter the number of frames (capacity of memory): ");

scanf("%d", &capacity);

fifoPageReplacement(pages, n, capacity);

return 0;

}

**OUTPUT:**

Enter the number of pages: 2

Enter the page reference string:

3

2

Enter the number of frames (capacity of memory): 4

Page Reference String: 3 2

Page Fault: 3

Page Fault: 3 2

Total Page Faults: 2

**EXP NO:**11b) **LRU PAGE REPLACEMENT**

**DATE:28/2/25**

**PROGRAM:**

#include <stdio.h>

#define MAX 10

voidlruPageReplacement(int pages[], int n, int capacity) {

int frames[capacity], counter[MAX] = {0}, pageFaults = 0;

// Initialize frames

for (inti = 0; i< capacity; i++) {

frames[i] = -1;

}

printf("Page Reference String: ");

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

printf("%d ", pages[i]);

}

printf("\n");

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

int found = 0;

// Check if the page is already in memory

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

if (frames[j] == pages[i]) {

found = 1;

counter[j] = i; // Update the counter with the current index (recent use)

break;

}

}

// If page is not found, replace it using LRU

if (!found) {

intlru = 0;

for (int j = 1; j < capacity; j++) {

if (counter[j] < counter[lru]) {

lru = j;

}

}

frames[lru] = pages[i];

counter[lru] = i; // Update the counter for the replaced page

pageFaults++;

}

// Display the frames after each page reference

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

printf("%d ", frames[j]);

}

printf("\n");

}

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

}

int main() {

int pages[MAX], n, capacity;

printf("Enter number of frames: ");

scanf("%d", &capacity);

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter reference string: ");

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

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

}

lruPageReplacement(pages, n, capacity);

return 0;

}

**OUTPUT:**

Enter number of frames: 3

Enter number of pages: 5

Enter reference string: 2

3

4

3

7

Page Reference String: 2 3 4 3 7

2 -1 -1

3 -1 -1

3 4 -1

3 4 -1

3 4 7

Total Page Faults = 4

**EXP NO:** 11c) **OPTIMAL PAGE REPLACEMENT**

**DATE:28/2/25**

**PROGRAM:**

#include <stdio.h>

#define MAX 10

int findOptimalPage(int frames[], int n, int pages[], int currentIndex, int capacity) {

int farthest = currentIndex;

int optimalPage = -1;

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

int j;

for (j = currentIndex + 1; j < n; j++) {

if (frames[i] == pages[j]) {

if (j > farthest) {

farthest = j;

optimalPage = i;

}

break;

}

}

if (j == n) {

return i; // If the page is not found, return this frame

}

}

return optimalPage;

}

void optimalPageReplacement(int pages[], int n, int capacity) {

int frames[capacity], pageFaults = 0;

// Initialize frames to -1 (empty)

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

frames[i] = -1;

}

printf("Page Reference String: ");

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

printf("%d ", pages[i]);

}

printf("\n");

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

int found = 0;

// Check if the page is already in one of the frames

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

if (frames[j] == pages[i]) {

found = 1;

break;

}

}

// If page is not found, replace the optimal page

if (!found) {

int optimalPage = findOptimalPage(frames, n, pages, i, capacity);

frames[optimalPage] = pages[i];

pageFaults++;

// Display the frames after each page reference

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

printf("%d ", frames[j]);

}

printf("\n");

}

}

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

}

int main() {

int pages[MAX], n, capacity;

printf("Enter number of frames: ");

scanf("%d", &capacity);

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter reference string: ");

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

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

}

optimalPageReplacement(pages, n, capacity);

return 0;

}

**OUTPUT:**

Enter number of frames: 3

Enter number of pages: 6

Enter reference string: 2

3

4

3

4

8

Page Reference String: 2 3 4 3 4 8

2 -1 -1

3 -1 -1

3 4 -1

8 4 -1

Total Page Faults = 4

**EXP NO:** 12 **FILE ORGANIZATION TECHNIQUE-SINGLE AND TWO LEVEL**

**DATE:28/2/25**

**PROGRAM:**

**singleLevel.c**

#include <stdio.h>

#define MAX\_FILES 10

void singleLevelDirectory() {

    int n;

    char fileNames[MAX\_FILES][50];

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

    scanf("%d", &n);

    printf("Enter the names of the files:\n");

    for (int i = 0; i < n; i++) {

        printf("File %d: ", i + 1);

        scanf("%s", fileNames[i]);

    }

    printf("\nFiles in the directory:\n");

    for (int i = 0; i < n; i++) {

        printf("%s\n", fileNames[i]);

    }

}

int main() {

    singleLevelDirectory();

    return 0;

}

**OUTPUT:**

Enter the number of files in the directory: 2

Enter the names of the files:

File 1: base

File 2: bin

Files in the directory:

base

bin

**PROGRAM:**

**twoLevel.c**

#include <stdio.h>

#define MAX\_DIRECTORIES 10

#define MAX\_SUBDIRECTORIES 10

#define MAX\_FILES 10

void twoLevelDirectory() {

    int n, m, k;

    char dirNames[MAX\_DIRECTORIES][50];

    char subDirNames[MAX\_DIRECTORIES][MAX\_SUBDIRECTORIES][50];

    char fileNames[MAX\_DIRECTORIES][MAX\_SUBDIRECTORIES][MAX\_FILES][50];

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

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        printf("Enter the name of directory %d: ", i + 1);

        scanf("%s", dirNames[i]);

        printf("How many subdirectories for %s: ", dirNames[i]);

        scanf("%d", &m);

        for (int j = 0; j < m; j++) {

            printf("Enter the name of subdirectory %d in %s: ", j + 1, dirNames[i]);

            scanf("%s", subDirNames[i][j]);

            printf("How many files in %s: ", subDirNames[i][j]);

            scanf("%d", &k);

            for (int l = 0; l < k; l++) {

                printf("Enter the name of file %d in %s/%s: ", l + 1, dirNames[i], subDirNames[i][j]);

                scanf("%s", fileNames[i][j][l]);

            }

        }

    }

    printf("\nFiles in the directories and subdirectories:\n");

    for (int i = 0; i < n; i++) {

        printf("\nDirectory: %s\n", dirNames[i]);

        for (int j = 0; j < MAX\_SUBDIRECTORIES && subDirNames[i][j][0] != '\0'; j++) {

            printf("\tSubdirectory: %s\n", subDirNames[i][j]);

            for (int l = 0; l < MAX\_FILES && fileNames[i][j][l][0] != '\0'; l++) {

                printf("\t\tFile: %s\n", fileNames[i][j][l]);

            }

        }

    }

}

int main() {

    twoLevelDirectory();

    return 0;

}

**OUTPUT:**

Enter the number of directories: 2

Enter the name of directory 1: code

How many subdirectories for code: 2

Enter the name of subdirectory 1 in code: python

How many files in python: 2

Enter the name of file 1 in code/python: main.py

Enter the name of file 2 in code/python: fib.py

Enter the name of subdirectory 2 in code: java

How many files in java: 2

Enter the name of file 1 in code/java: main.java

Enter the name of file 2 in code/java: stairs.java

Enter the name of directory 2: game

How many subdirectories for game: 1

Enter the name of subdirectory 1 in game: coc

How many files in coc: 2

Enter the name of file 1 in game/coc: clashOfClans

Enter the name of file 2 in game/coc: clashRoyals

Files in the directories and subdirectories:

Directory: code

Subdirectory: python

File: main.py

File: fib.py

Subdirectory: java

File: main.java

File: stairs.java

Directory: game

Subdirectory: coc

File: clashOfClans

File: clashRoyals