RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS23431 – OPERATING SYSTEMS

(REGULATION 2023)

RAJALAKSHMI ENGINEERING COLLEGE
Thandalam, Chennai-602015

Name: JANANI V R . . . . . .

Register No: 2116231801065. . . . . .

Year / Branch / Section: . . .II/ B.Tech.AI&DS/ "FA ". . . . . .

Semester: . . . . . IV. . . . . .

Academic Year: . . . . . 2024-2025. . . . . .

INDEX

8	08/03/25	Producer Consumer using Semaphores		
9	27/03/25	Bankers Deadlock Avoidance algorithms		
10 a	29/03/25	Best Fit		
10 b	3/04/25	First Fit		
11a	5/04/25	FIFO		
11b	7/04/25	LRU		
11c	7/04/25	Optimal		
12	7/04/25	File Organization Technique- single and Two level directory		

EXP NO: 1a)	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

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

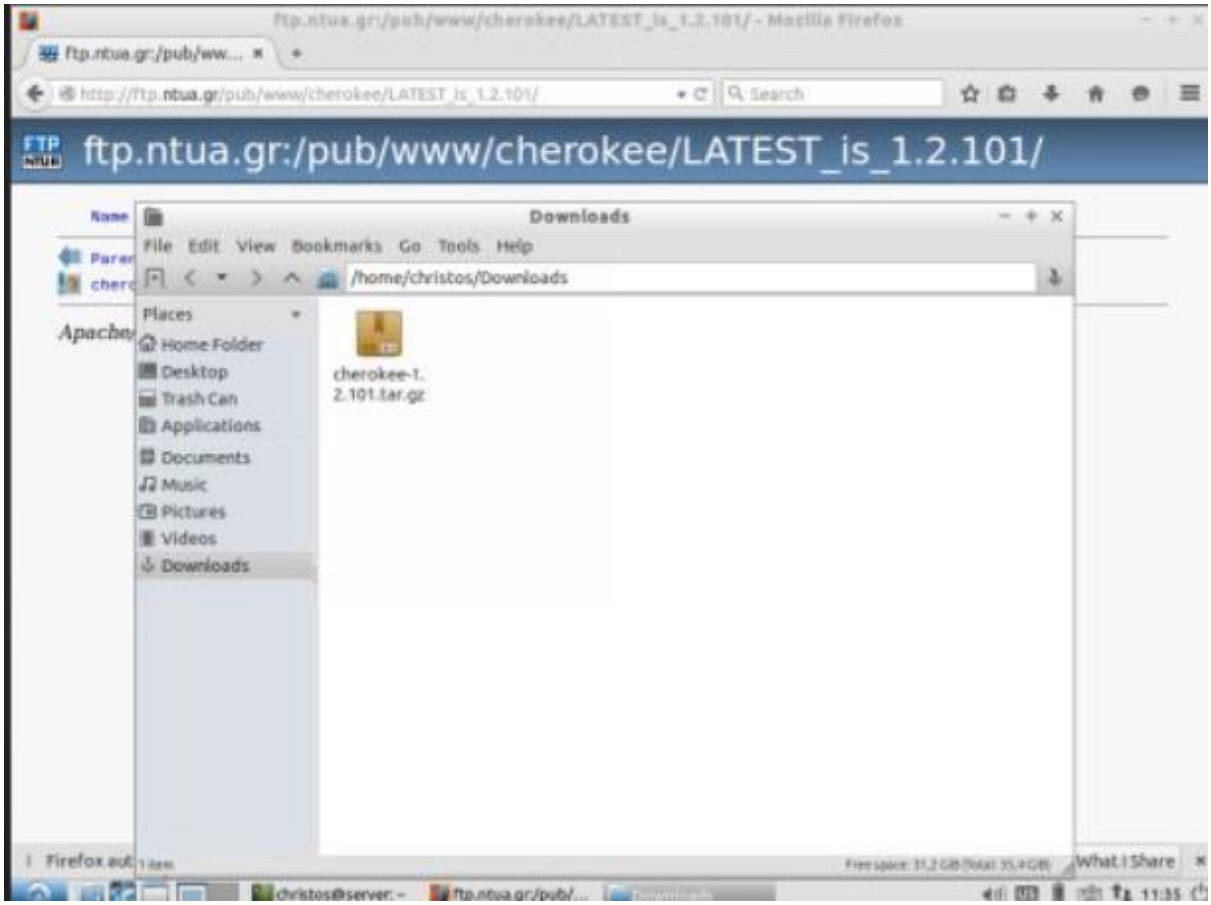
8. Choose puppy-linux.iso then kernel version

9. Select CPU and RAM limits

10.Create default disk image to 8 GB

11.Click finish for creating the new VM with PuppyLinux

Output:



Result :

Thus the installation and configuration of Linux was successful.

EXP NO: 1b)	BASIC LINUX COMMANDS

1.1 GENERAL PURPOSE COMMANDS

1. The 'date' command:

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

SYNTAX: $ date

The date command can also be used with following format.

Format Purpose Example

+ %m To display only month $ date + %m

+ %h To display month name $ date + %h

+ %d To display day of month $ date + %d

+ %y To display last two digits of the year $ date + %y

+ %H To display Hours $ date + %H

+ %M To display Minutes $ date + %M

+ %S To display Seconds $ date + %S

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

Aim:

To write a Shellscript to to display basic calculator.

PROGRAM:

```
echo "   Basic Calculator"
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 [ $b -ne 0 ]; then
      result=$((a / b) )
      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
```

Result:

Thus the program was executed Successfully.

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

Aim:

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

PROGRAM:

```
echo "Enter a year:"
read 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 is not a leap year"
fi
```

OUTPUT:

Enter a year: 2004

2004 is a leap year

Result:

Thus the program was executed Successfully

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

Aim:

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

PROGRAM:

```
echo "Enter a number:"
read num
rev=0
while [ $num -gt 0 ];
do
   digit=$((num % 10))
   rev=$((rev * 10 + digit))
   num=$((num / 10))
done
echo "Reversed number: $rev"
```

OUTPUT:

```
Enter a number:
1234
Reversed number: 4321
```

Result:

Thus the program was executed Successfully.

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

Aim:

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

PROGRAM:

```
echo "Enter a number:"
read num
a=0
echo "Fibonacci series up to $num:"
echo $a
echo $b
for (( i=2; i<=num; i++ ))
do
   fib=$((a + b))

   echo $fib
   a=$b
   b=$fib
done
```

OUTPUT:

Enter a number:

5

Fibonacci series up to 5:

0

1

1

2

3

5

Result:

Thus the program was executed Successfully

EXP NO: 4a)	EMPLOYEE AVERAGE PAY

Aim:

To find out the average pay of all employees whose salary is more than 6000 and no. of days worked is more than 4.

Algorithm:

1. Create a flat file emp.dat for employees with their name, salary per day and number of days worked and save it.

2. Create an awk script emp.awk

3. For each employee record do

a. If Salary is greater than 6000 and number of days worked is more than 4, then print name and salary earned

b. Compute total pay of employee

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

PROGRAM:

```
total_pay=0

employee_count=0

echo "Enter the number of employees:"

for ((i=1; i<=num_employees; i++))

do

   echo "Enter name of employee $i:"

   read name


   echo "Enter salary of $name:"

   read salary

   total_pay=$((total_pay + salary))

   employee_count=$((employee_count + 1))

done

if [ $employee_count -gt 0 ]; then
```

```
    average_pay=$(echo "scale=2; $total_pay / $employee_count" | bc)
else
    average_pay=0
fi
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
```

Result:

Thus the program was executed Successfully

EXP NO: 4b)	RESULT OF EXAMINATION

Aim:

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

Algorithm:

1. Read the data from file
2. Get a data from each column
3. Compare the all subject marks column

a. If marks less than 45 then print Fail

b. else print Pass

PROGRAM:

```
while read -r line
do
   arr=($line)
   name=${arr[0]}
   marks=${arr[@]:1}
   fail=false
   for mark in ${marks[@]}
   do
      if [ $mark -lt 45 ]; then
         fail=true
         break
      fi
   done
   if [ "$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

Result:

Thus the program was executed Successfully

EXP NO: 5	SYSTEM CALLS PROGRAMMING

Aim:

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

Algorithm:

1. Start

o Include the required header files (stdio.h and stdlib.h).

2. Variable Declaration

o Declare an integer variable pid to hold the process ID.

3. Create a Process

o 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

o Print the statement:

scss

Copy code

THIS LINE EXECUTED TWICE

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

5. Check for Process Creation Failure

o If pid == -1:

▪ Print:

Copy code

CHILD PROCESS NOT CREATED

▪ Exit the program using exit(0).

6. Child Process Execution

o If pid == 0 (child process):

▪ Print:

▪ Process ID of the child process using getpid().

▪ Parent process ID of the child process using getppid().

7. Parent Process Execution

o If pid > 0 (parent process):

▪ Print:

▪ Process ID of the parent process using getpid().

▪ Parent's parent process ID using getppid().

8. Final Print Statement

o Print the statement:

objectivec

Copy code

IT CAN BE EXECUTED TWICE

(This line is executed by both parent and child processes). 9. End

PROGRAM:

Sender

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<string.h>
int main(){
    key_t key=1234;
    int shmid;
    char *shared_memory;

    shmid=shmget(key,1024,IPC_CREAT|0666);
    if(shmid==-1){
        perror("shmid failed");
        exit(1);
    }
    shared_memory=(char*)shmat(shmid,NULL,0);
```

```
    if(shared_memory==(char*)-1){
        perror("shmat failed");
        exit(1);
    }
    strcpy(shared_memory,"Hello Sender!");
    printf("Sender:Message is written to sender");
    sleep(5);
    shmdt(shared_memory);
}
```

RECEIVER

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<string.h>
int main(){
    int shmid;
    key_t key=1234;
    char *shared_memory;

    shmid=shmget(key,1024,IPC_CREAT|0666);
    if(shmid==-1){
        perror("shmid failed");
        exit(1);
    }

    shared_memory=(char*)shmat(shmid,NULL,0);
    if(shared_memory==(char*)-1){
        perror("shmat failed");
        exit(1);
```

```
    }
    printf("Message is received");
    shmdt(shared_memory);
    sleep(5);
    shmctl(shmid,IPC_RMID,NULL);

}
```

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
```

Result:

Thus the program was executed Successfully

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

Aim:

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

Algorithm:

1. Get the number of processes from the user.

2. Read the process name and burst time.

3. Calculate the total process time.

4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process. 6. Display the total waiting time, average waiting time, turnaround time

PROGRAM:

```
#include<stdio.h>
int main(){
    int n;
    printf("Enter the number of processes:");
    scanf("%d",&n);

    int burst_time[n],waiting_time[n],turnaround_time[n];
    int total_waiting_time=0,total_turnaround_time=0;
    char process_name[n][10];
    float avg_waiting_time,avg_turnaround_time;

    for(int i=0;i<n;i++){
        printf("Enter the name of the process %d:",i+1);
        scanf("%s",process_name[i]);
        printf("Enter the burst time for process %s: ",process_name[i]);
```

```
        scanf("%d",&burst_time[i]);
    }
    turnaround_time[0]=burst_time[0];
    for(int i=1;i<n;i++){
        turnaround_time[i]=turnaround_time[i-1]+burst_time[i];
    }
    for(int i=0;i<n;i++){
        waiting_time[i]=turnaround_time[i]-burst_time[i];
    }
    for(int i=0;i<n;i++){
        total_turnaround_time+=turnaround_time[i];
        total_waiting_time+=waiting_time[i];
    }
    avg_turnaround_time=(float)total_turnaround_time/n;
    avg_waiting_time=(float)total_waiting_time/n;
    printf("Processe\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(int i=0;i<n;i++){

printf("%s\t\t%d\t\t%d\t\t%d\n",process_name[i],burst_time[i],waiting_time[i],turnaround_ti
me[i]);
    }
    printf("Average Turnaround Time:%.2f\n",avg_turnaround_time);
    printf("Average Waiting Time:%.2f",avg_waiting_time);
}
```

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

Result:

Thus the program was executed Successfully.

EXP NO: 6b)	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:

```
#include<stdio.h>
int main(){
    int n;
    printf("Enter the number of processes:");
    scanf("%d",&n);
    int arrival_time[n],burst_time[n],turnaround_time[n],waiting_time[n];
    char process_name[n][10];
    int total_waiting_time=0,total_turnaround_time=0;
    float avg_waiting_time,avg_turnaround_time;

    for(int i=0;i<n;i++){
        printf("Enter the Process name %d:",i+1);
        scanf("%s",process_name[i]);
        printf("Enter the Arrival Time for process %s:",process_name[i]);
        scanf("%d",&arrival_time[i]);
        printf("Enter the Burst Time for process %s:",process_name[i]);
        scanf("%d",&burst_time[i]);
```

```
    }

    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            if(arrival_time[i]>arrival_time[j] || (arrival_time[i]==arrival_time[j] &&
burst_time[i]>burst_time[j])){
                int temp=burst_time[i];
                burst_time[i]=burst_time[j];
                burst_time[j]=temp;

                int temp_arrival=arrival_time[i];
                arrival_time[i]=arrival_time[j];
                arrival_time[j]=temp_arrival;

                char temp_p;
                for(int k=0;k<n;k++){
                    temp_p=process_name[i][k];
                    process_name[i][k]=process_name[j][k];
                    process_name[j][k]=temp_p;
                }
            }
        }
    }
    turnaround_time[0]=burst_time[0];
    for(int i=1;i<n;i++){
        turnaround_time[i]=turnaround_time[i-1]+burst_time[i];
    }
    for(int i=0;i<n;i++){
        waiting_time[i]=turnaround_time[i]-burst_time[i];
    }
    for(int i=0;i<n;i++){
```

```
        total_turnaround_time+=turnaround_time[i];
        total_waiting_time+=waiting_time[i];
    }
    avg_turnaround_time=(float)total_turnaround_time/n;
    avg_waiting_time=(float)total_waiting_time/n;
    printf("\nProcess\tArrival Time\tBurst Time\tTurnaround Time\nWaiting Time\n");
    for(int i=0;i<n;i++){

printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\t\t\n",process_name[i],arrival_time[i],burst_time[i],turnar
ound_time[i],waiting_time[i]);
    }
    printf("Average turnaround time:%.2f",avg_turnaround_time);
    printf("Average Waiting time:%.2f",avg_waiting_time);
}
```

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

Result:

Thus the program was executed Successfully

EXP NO: 6c)	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:

```
#include<stdio.h>
int main(){
   int n;
   printf("Enter the number of processes:");
   scanf("%d",&n);
   int arrival_time[n],burst_time[n],turnaround_time[n],waiting_time[n],priority[n];
   char process_name[n][10];
   int total_waiting_time=0,total_turnaround_time=0;
   float avg_waiting_time,avg_turnaround_time;

   for(int i=0;i<n;i++){
      printf("Enter the Process name %d:",i+1);
      scanf("%s",process_name[i]);
      printf("Enter the Arrival Time for process %s:",process_name[i]);
      scanf("%d",&arrival_time[i]);
      printf("Enter the Burst Time for process %s:",process_name[i]);
      scanf("%d",&burst_time[i]);
      printf("Enter the priority of process %s:",process_name[i]);
```

```
    scanf("%d",&priority[i]);
  }
  for(int i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
      if(priority[i]>priority[j]){
        int temp=priority[i];
        priority[i]=priority[j];
        priority[j]=temp;

        int temp_arrival=arrival_time[i];
        arrival_time[i]=arrival_time[j];
        arrival_time[j]=temp_arrival;

        int temp_burst=burst_time[i];
        burst_time[i]=burst_time[j];
        burst_time[j]=temp_burst;
      }
    }
  }
  turnaround_time[0]=burst_time[0];
  for(int i=1;i<n;i++){
    turnaround_time[i]=turnaround_time[i-1]+burst_time[i];
  }
  for(int i=0;i<n;i++){
    waiting_time[i]=turnaround_time[i]-burst_time[i];
  }
  for(int i=0;i<n;i++){
    total_turnaround_time+=turnaround_time[i];
    total_waiting_time+=waiting_time[i];
  }
```

```
    avg_turnaround_time=total_turnaround_time/n;
    avg_waiting_time=total_waiting_time/n;


    printf("\nProcess\tArrival Time\tBurst Time\tPriority\tTurnaround Time\nWaiting
Time\n");
    for(int i=0;i<n;i++){

printf("%s\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\n",process_name[i],arrival_time[i],burst_time[i],
priority[i],turnaround_time[i],waiting_time[i]);
    }
    printf("Average turnaround time:%.2f",avg_turnaround_time);
    printf("Average Waiting time:%.2f",avg_waiting_time);
}
```

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

Result:

Thus the program was executed Successfully

EXP NO: 6d)	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

7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.

a- If rem_bt[i] > quantum

(i) t = t + quantum

(ii) bt_rem[i] -= quantum;

b- Else // Last cycle for this process

(i) t = t + bt_rem[i];

(ii) wt[i] = t - bt[i]

(iii) bt_rem[i] = 0; // This process is over

8. Calculate the waiting time and turnaround time for each process.

9. Calculate the average waiting time and average turnaround time.

10. Display the results.

PROGRAM:

```
#include <stdio.h>

int main() {
   int n;

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

```
int processes[n];  // Process IDs
int burst_time[n]; // Burst times
int arrival_time[n]; // Arrival times
int remaining_burst_time[n];
int waiting_time[n];
int turn_around_time[n];
int completion_time[n];

// Input burst and arrival times
for (int i = 0; i < n; i++) {
    processes[i] = i + 1;
    printf("Enter burst time for process %d: ", processes[i]);
    scanf("%d", &burst_time[i]);
    printf("Enter arrival time for process %d: ", processes[i]);
    scanf("%d", &arrival_time[i]);
}

int quantum;
printf("Enter the time quantum: ");
scanf("%d", &quantum);

// Initialize arrays
for (int i = 0; i < n; i++) {
    remaining_burst_time[i] = burst_time[i];
    waiting_time[i] = 0;
    turn_around_time[i] = 0;
    completion_time[i] = 0;
}

int time = 0;

// Round Robin scheduling loop
while (1) {
    int done = 1;
    for (int i = 0; i < n; i++) {
        if (remaining_burst_time[i] > 0 && arrival_time[i] <= time) {
            done = 0;

            if (remaining_burst_time[i] > quantum) {
                time += quantum;
                remaining_burst_time[i] -= quantum;
            } else {
                time += remaining_burst_time[i];
                waiting_time[i] = time - burst_time[i] - arrival_time[i];
                remaining_burst_time[i] = 0;
```

```
                completion_time[i] = time;
            }
        }
    }
    if (done == 1)
        break;
}

// Combined loop: calculate turnaround time, total waiting and total turnaround
float total_waiting_time = 0;
float total_turnaround_time = 0;

for (int i = 0; i < n; i++) {
    turn_around_time[i] = completion_time[i] - arrival_time[i];
    total_waiting_time += waiting_time[i];
    total_turnaround_time += turn_around_time[i];
}

float avg_waiting_time = total_waiting_time / n;
float avg_turnaround_time = total_turnaround_time / n;

// Output the results
printf("\nProcess | Arrival Time | Burst Time | Waiting Time | Turnaround Time |
Completion Time\n");
for (int i = 0; i < n; i++) {
    printf("  %d   |     %d     |    %d    |     %d     |      %d       |       %d\n",
          processes[i], arrival_time[i], burst_time[i], waiting_time[i], turn_around_time[i],
completion_time[i]);
}

printf("\nTotal Waiting Time: %.2f\n", total_waiting_time);
printf("Total Turnaround Time: %.2f\n", total_turnaround_time);
printf("Average Waiting Time: %.2f\n", avg_waiting_time);
printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);

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

Result:

Thus the program was executed Successfully

EXP NO: 7	IPC-USING SHARED MEMORY

Aim:

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:

Sender.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<string.h>
int main(){
   key_t key=1234;
```

```
    int shmid;
    char *shared_memory;

    shmid=shmget(key,1024,IPC_CREAT|0666);
    if(shmid==-1){
        perror("shmid failed");
        exit(1);
    }
    shared_memory=(char*)shmat(shmid,NULL,0);
    if(shared_memory==(char*)-1){
        perror("shmat failed");
        exit(1);
    }
    strcpy(shared_memory,"Hello Sender!");
    printf("Sender:Message is written to sender");
    sleep(5);
    shmdt(shared_memory);
}
```

receiver.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
#include<string.h>
int main(){
    int shmid;
    key_t key=1234;
    char *shared_memory;
```

```
    shmid=shmget(key,1024,IPC_CREAT|0666);
    if(shmid==-1){
        perror("shmid failed");
        exit(1);
    }

    shared_memory=(char*)shmat(shmid,NULL,0);
    if(shared_memory==(char*)-1){
        perror("shmat failed");
        exit(1);
    }
    printf("Message is received");
    shmdt(shared_memory);
    sleep(5);
    shmctl(shmid,IPC_RMID,NULL);

}
```

OUTPUT:

Message Sent: Welcome to Shared Memory.

Message Received: Welcome to Shared Memory.

Result:

Thus the program was executed Successfully

EXP NO: 8	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:

```
#include<stdio.h>
#include<pthread.h>
#include<unistd.h>
#include<semaphore.h>
#define BUFFER_SIZE 5
sem_t empty,full,mutex;
int buffer[BUFFER_SIZE];
int in=0,out=0,count=0;
void producer(){
   if(count==BUFFER_SIZE){
      printf("The buffer is full!\n");
      return;
   }
   static int item=1;
```

```
    sem_wait(&empty);
    sem_wait(&mutex);
    buffer[in]=item;
    printf("producer produces an item %d\n",item);
    in=(in+1)%BUFFER_SIZE;
    count++;
    sem_post(&mutex);
    sem_post(&full);
}
void consumer(){
    if(count==0){
        printf("Buffer is empty!\n");
        return;
    }
    int item;
    sem_wait(&full);
    sem_wait(&mutex);
    item=buffer[out];
    printf("consumer consumes an item %d\n",item);
    out=(out+1)%BUFFER_SIZE;
    count--;
    sem_post(&mutex);
    sem_post(&empty);
}
int main(){
    sem_init(&empty,0,BUFFER_SIZE);
    sem_init(&full,0,0);
    sem_init(&mutex,0,1);
    int choice;
    printf("1.Producer\n2.Consumer\n3.exit\n");
```

```
    while(1){
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1:
                producer();
                break;
            case 2:
                consumer();
                break;
            case 3:
                sem_destroy(&empty);
                sem_destroy(&full);
                sem_destroy(&mutex);
                return 0;
                break;
            default:
                printf("Invalid choice!\n");
        }
    }
}
```

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

Result:

Thus the program was executed Successfully

EXP NO: 9	DEADLOCK AVOIDANCE

Aim:

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

Algorithm:

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

finish[i]=false and Needi<= work

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

PROGRAM:

```
#include<stdio.h>
#include<stdbool.h>
int main(){
    int p,r;
    printf("Enter the number of processes:");
    scanf("%d",&p);
    printf("Enter the number of resources:");
    scanf("%d",&r);
    int alloc[p][r],max[p][r],avail[r],need[p][r],work[r];
    int safeseq[p];
    int count=0,finish[p];
    printf("Enter Allocation Matrix:\n");
    for(int i = 0; i < p; i++){
        for(int j = 0; j < r; j++){
            scanf("%d", &alloc[i][j]);
```

```
}}
    // Input max matrix
    printf("Enter Max Matrix:\n");
    for(int i = 0; i < p; i++){
        for(int j = 0; j < r; j++){
            scanf("%d", &max[i][j]);
}}
    // Input available resources
    printf("Enter Available Resources:\n");
    for(int i = 0; i < r; i++){
        scanf("%d", &avail[i]);
    }
    for(int i=0;i<p;i++){
        for(int j=0;j<r;j++){
        need[i][j]=max[i][j]-alloc[i][j];
    }}
    for(int i = 0; i < r; i++){
            work[i]=avail[i];
}
   for(int i = 0; i < p; i++){
        finish[i]=0;
}
    while(count<p){
        bool found=true;
        for(int i=0;i<p;i++){
            if(finish[i]==0){
              bool canAlloc=true;

            for(int j=0;j<r;j++){
               if(need[i][j]>work[j]){
```

```
                canAlloc=false;
                break;
              }
            }
            if(canAlloc){
               for(int k=0;k<r;k++){
                  work[k]+=alloc[i][k];
               }
            safeseq[count++]=i;
            found=true;
            finish[i]=1;

            }
}
            }
        if(!found){
            printf("There is no safesequence");
        }
    }
    printf("The safeseq is:");
    for(int i=0;i<p;i++){
        printf("P%d%s",safeseq[i],(i<p-1)?"->":"\n");
    }
    return 0;
}
```

OUTPUT:

The SAFE Sequence is:

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

Result:

Thus the program was executed Successfully

EXP NO: 10a)	**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:

```
#include <stdio.h>

#define MAX_BLOCKS 10
#define MAX_PROCESSES 10

int main() {
    int blocks[MAX_BLOCKS], processes[MAX_PROCESSES];
    int blockCount, processCount;

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

    for (int i = 0; i < blockCount; i++) {
        printf("Enter size of Block %d: ", i + 1);
        scanf("%d", &blocks[i]);
    }
```

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

    for (int i = 0; i < processCount; i++) {
        printf("Enter size of Process %d: ", i + 1);
        scanf("%d", &processes[i]);
    }

    for (int i = 0; i < processCount; i++) {
        int bestFitIdx = -1;
        int minWaste = 9999;

        for (int j = 0; j < blockCount; j++) {
            if (blocks[j] >= processes[i] && blocks[j] - processes[i] < minWaste) {
                bestFitIdx = j;
                minWaste = blocks[j] - processes[i];
            }
        }

        if (bestFitIdx != -1) {
            printf("Process %d allocated to Block %d\n", i + 1, bestFitIdx + 1);
            blocks[bestFitIdx] -= processes[i]; // Allocate block
        } else {
            printf("Process %d could not be allocated\n", i + 1);
        }
    }

    return 0;
}
```

OUTPUT:

Process No.	Process Size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

Result: Thus the program was executed Successfully

EXP NO: 10b)	**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:

```
#include <stdio.h>


#define MAX_BLOCKS 10

#define MAX_PROCESSES 10


int main() {

   int blocks[MAX_BLOCKS], processes[MAX_PROCESSES];

   int blockCount, processCount;


   printf("Enter number of memory blocks: ");

   scanf("%d", &blockCount);


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

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

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

   }
```

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

    for (int i = 0; i < processCount; i++) {
        printf("Enter size of Process %d: ", i + 1);
        scanf("%d", &processes[i]);
    }

    for (int i = 0; i < processCount; i++) {
        int allocated = 0;
        for (int j = 0; j < blockCount; j++) {
            if (blocks[j] >= processes[i]) {
                printf("Process %d allocated to Block %d\n", i + 1, j + 1);
                blocks[j] -= processes[i]; // Allocate block
                allocated = 1;
                break;
            }
        }
        if (!allocated)
            printf("Process %d could not be allocated\n", i + 1);
    }

    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

Result: Thus the program was executed Successfully.

EXP NO: 11a)	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
5. Insert the page require memory into the queue
6. Check for bad replacement and page fault
7. Get the number of processes to be inserted
8. Display the values

PROGRAM:

```
#include<stdio.h>
int main(){
   int pages[100],frames[100];
   int p,f,page_faults=0,found,index=0;
   printf("Enter the no of paages:");
   scanf("%d",&p);
   printf("Enter the reference string:");
   for(int i=0;i<p;i++){
      scanf("%d",&pages[i]);
   }
   printf("Enter the number of frames:");
   scanf("%d",&f);
   for(int i=0;i<f;i++){
      frames[i]=-1;
```

```
}
for(int i=0;i<p;i++){
    found=0;
    for(int j=0;j<f;j++){
        if(frames[j]==pages[i]){
            found=1;
            break;
        }

        if(found==0){
            frames[index]=pages[i];
            index=(index+1)%f;
            page_faults++;
        }
        printf("%d\n",pages[i]);
        if(frames[j]!=-1){
            printf("%d\n",frames[j]);
        }
        else{
            printf("-");
        }}
        if(found==0){
            printf("\tyes\n");
        }
        else{
            printf("\tNo\n");
        }

}
printf("Total number of page faults%d\n:",page_faults);
```

}

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

Result:

Thus the program was executed Successfully

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

Aim:

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

Algorithm:

1: Start the process

2: Declare the size

3: Get the number of pages to be inserted

4: Get the value

5: Declare counter and stack

6: Select the least recently used page by counter value

7: Stack them according the selection.

8: Display the values

9: Stop the process

PROGRAM:

```
#include <stdio.h>


int main() {
    int pages[100], frames[100];
    int p, f, page_faults = 0, found, index = 0;

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

    printf("Enter the reference string:\n");
    for (int i = 0; i < p; i++) {
        scanf("%d", &pages[i]);
    }
```

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

for (int i = 0; i < f; i++) {
    frames[i] = -1; // initialize all frames as empty
}

printf("\nPage\tFrames\t\tPage Fault\n");

for (int i = 0; i < p; i++) {
    found = 0;

    // Check if the page is already in any frame
    for (int j = 0; j < f; j++) {
        if (frames[j] == pages[i]) {
            found = 1;
            break;
        }
    }

    // If not found, it's a page fault and needs replacement
    if (found == 0) {
        frames[index] = pages[i];
        index = (index + 1) % f;
        page_faults++;
    }

    // Print the page and frame status
    printf("%d\t", pages[i]);
    for (int j = 0; j < f; j++) {
```

```
            if (frames[j] != -1)
                printf("%d ", frames[j]);
            else
                printf("- ");
        }

        if (found == 0)
            printf("\tYes\n");
        else
            printf("\tNo\n");
    }

    // Final output
    printf("\nTotal number of page faults: %d\n", page_faults);

    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

Result: Thus the program was executed Successfully.

EXP NO: 11c)	OPTIMAL PAGE REPLACEMENT

Aim:

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 main() {
    int pages[100], frames[10];
    int n, f, i, j, k;
    int page_faults = 0;

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

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

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

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

// Initialize all frames as empty (-1)
for (i = 0; i < f; i++) {
    frames[i] = -1;
}

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

    // Check if the page is already in the frame (Page Hit)
    for (j = 0; j < f; j++) {
        if (frames[j] == pages[i]) {
            found = 1;
            break;
        }
    }

    // If not found, it's a Page Fault
    if (!found) {
        int replace = -1, farthest = i + 1;

        // Find an empty frame first
        for (j = 0; j < f; j++) {
            if (frames[j] == -1) {
                replace = j;
                break;
            }
        }
```

```
    // If no empty frame, find the page that won't be used for the longest time
    if (replace == -1) {
        int latest = -1;
        for (j = 0; j < f; j++) {
            int found_future = 0;
            for (k = i + 1; k < n; k++) {
                if (frames[j] == pages[k]) {
                    if (k > latest) {
                        latest = k;
                        replace = j;
                    }
                    found_future = 1;
                    break;
                }
            }
            if (!found_future) {
                replace = j;
                break;
            }
        }
    }

    frames[replace] = pages[i]; // Replace the page
    page_faults++;
}

// Print current frame status
for (j = 0; j < f; j++) {
    if (frames[j] != -1)
```

```
            printf("%d ", frames[j]);
        else
            printf("- ");
    }
    printf("\n");
  }

  printf("Total Page Faults = %d\n", page_faults);
  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
```

Result:

Thus the program was executed Successfully

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

AIM:

To implement File Organization Structures in C are

a. Single Level Directory

b. Two-Level Directory

c. Hierarchical Directory Structure

d. Directed Acyclic Graph Structure

a. Single Level

Directory

ALGORITHM

1. Start

2. Declare the number, names and size of the directories and file names. 3. Get the values for the declared variables.

4. Display the files that are available in the directories.

5. Stop.

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

b. Two-level directory Structure

ALGORITHM:

1. Start

2. Declare the number, names and size of the directories and subdirectories and file names.

3. Get the values for the declared variables.

4. Display the files that are available in the directories and subdirectories. 5. Stop.

PROGRAM:

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

Result:

Thus the program was executed Successfully