MRCET CAMPUS

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

LABORATORY MANUAL & RECORD

Name:.....

Roll No:.....Branch:.....

Year:.....Sem:.....

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(AUTONOMOUS INSTITUTION - UGC, GOVT. OF INDIA)

Affiliated to JNTUH; Approved by AICTE, NBA-Tier 1 & NAAC with A-GRADE | ISO 9001:2015

Maisammaguda, Dhulapally, Komaplly, Secunderabad - 500100, Telangana State, India

Certificate

Certified that this is the Bonafide Record of the Work Done by

Mr./Ms.....Roll.No.....of

B.Tech.....year..... Semester for Academic year.....

in.....Laboratory.

Date: Faculty Incharge HOD

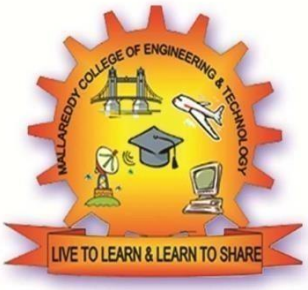
Internal Examiner External Examiner

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

OPERATING SYSTEMS LABMANUAL

B.TECH



(II YEAR – I SEM)
(2023-24)

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

(AIML)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGCACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally(Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Vision

To be a premier centre for academic excellence and research through innovative interdisciplinary collaborations and making significant contributions to the community, organizations, and society as a whole.

Mission

- To impart cutting-edge Artificial Intelligence technology in accordance with industry norms.
- To instill in students a desire to conduct research in order to tackle challenging technical problems for industry.
- To develop effective graduates who are responsible for their professional growth, leadership qualities and are committed to lifelong learning.

Quality Policy

- To provide sophisticated technical infrastructure and to inspire students to reach their full potential.
- To provide students with a solid academic and research environment for a comprehensive learning experience.
- To provide research development, consulting, testing, and customized training to satisfy specific industrial demands, thereby encouraging self-employment and entrepreneurship among students.

Department of Computer Science & Engineering

(Artificial Intelligence & Machine Learning)

Programme Educational Objectives (PEO):

PEO1: To possess knowledge and analytical abilities in areas such as maths, science, and fundamental engineering.

PEO2: To analyse, design, create products, and provide solutions to problems in Computer Science and Engineering.

PEO3: To leverage the professional expertise to enter the workforce, seek higher education, and conduct research on AI-based problem resolution.

PEO4: To be solution providers and business owners in the field of computer science and engineering with an emphasis on artificial intelligence and machine learning.

Programme Specific Outcomes (PSO):

After successful completion of the program a student is expected to have specific abilities to:

PSO1: To understand and examine the fundamental issues with AI and ML applications.

PSO2: To apply machine learning, deep learning, and artificial intelligence approaches to address issues in social computing, healthcare, vision, language processing, speech recognition, and other domains.

PSO3: Use cutting-edge AI and ML tools and technology to further your study and research.

PROGRAMOUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), thosewho come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab withthe synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm,Procedure, Program, Expected Output, etc.,) filled in for the lab session.

b. Laboratory Record updated up to the last session experiments and other utensils (if any)needed in the lab.

c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer systemallotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observationnote book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which shouldbe utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours willbe treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

IIYear B.Tech CSE AIML- I Sem **L T/P/D C**

- -/3/- 1.5

(R22A0587) OPERATING SYSTEMS LAB

OBJECTIVES:

1. To provide an understanding of the design aspects of operating system concepts through simulation
2. Introduce basic Linux commands, system call interface for process management, inter-process communication and I/O in Unix.
3. Student will learn various process and CPU scheduling Algorithms through simulation programs
4. Student will have exposure to System calls and simulate them.
5. Student will learn deadlocks and process management & Inter Process communication and simulate

WEEK 1:

Practice File handling utilities, Process utilities, Disk utilities, Networking commands, Filters, Text processing utilities and Backup utilities.

WEEK 2:

Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or directory and reports accordingly. Whenever the argument is a file it reports no of lines present in it.

WEEK 3:

Simulate the following CPU scheduling algorithms. a)FCFS b) SJF c) Round Robin d) Priority.

WEEK 4:

Simulate Bankers Algorithm for Dead Lock Avoidance; Simulate Bankers Algorithm for Dead Lock Prevention.

WEEK 5:

a) Write a C program to simulate the concept of Dining-philosophers problem.
b) Write a C program to simulate producer-consumer problem using Semaphores

WEEK 6:

a) Write a C program to implement kill(), raise() and sleep()functions.
b) Write a C program to implement alarm(), pause() and abort()functions
c) Write a program that illustrate communication between two process using unnamed pipes

WEEK 7:

a) Write a program that illustrates communication between two process using named pipes or FIFO.
b) Write a C program that receives a message from message queue and display them.

WEEK 8:

Write a C program that illustrates two processes communicating using Shared memory.

WEEK 9:

Simulate all page replacement algorithms a) FIFO b) LRU c) OPTIMAL

WEEK 10:

Write a C program that takes one or more file/directory names as command line input and reports following information A) File Type B) Number Of Links C) Time of last Access D) Read, write and execute permissions

WEEK 11

a) Implement in c language the following UNIX commands using system calls i) cat ii) ls iii) Scanning Directories (Ex: opendir(),readdir(),etc.)
b) Write a C program to create child process and allow parent process to display “parent” and the child to display “child” on the screen

WEEK 12:

Write a C program to simulate disk scheduling algorithms. a) FCFS b) SCAN c) C-SCAN

REFERENCE BOOKS:

1. Operating Systems – Internals and Design Principles, William Stallings, Fifth Edition–2005, Pearson Education/PHI
2. Operating System - A Design Approach-Crowley, TMH.
3. Modern Operating Systems, Andrew S Tanenbaum, 2nd edition, Pearson/PHI
4. UNIX Programming Environment, Kernighan and Pike, PHI/Pearson Education
5. UNIX Internals: The New Frontiers, U. Vahalia, Pearson Education

COURSE OUTCOMES:

1. To provide an understanding of the design aspects of operating system concepts through simulation
2. Introduce basic Linux commands, system call interface for process management, inter process communication and I/O in Unix.
3. Student will learn various process and CPU scheduling Algorithms through simulation programs
4. Student will have exposure to System calls and simulate them.
5. Student will learn deadlocks and process management & Inter Process communication and simulate them

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(UGC-Autonomous Institution , Govt. of India)
(Permanently Affiliated to JNTUH, Approved by AICTE-Accredited by NBA & NAAC- A-Grade; ISO 9001:2008 Certified) Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
Operating Systems Lab Manual (R22A0587))
TABLE OF CONTENTS

WEEK 1

AIM : Practice File handling utilities, Process utilities, Disk utilities, Networking commands, Filters, Text processing utilities and Backup utilities.

FILE HANDLING UTILITIES

Cat Command: cat linux command concatenates files and print it on the standard output.

To Create a new file:

cat > file1.txt

This command creates a new file file1.txt. After typing into the file press control+d(^d) simultaneously to end the file.

To Append data into the file: To append data into the same file use append operator >> to write into thefile, else the file will be overwritten (i.e., all of its contents will be erased).

cat >> file1.txt

To display a file: This command displays the data in the file.cat file1.txt

Toconcatenate several files and display:

cat file1.txt file2.txt

The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you canprint those files in a new file or display the file using less command.

cat file1.txt file2.txt | less

Toconcatenate several files and to transfer the output to anotherfile.

cat file1.txt file2.txt > file3.txt

In the above example the output is redirected to new file file3.txt.

rm COMMAND:

rm linux command is used to remove/delete the file from the directory.

To Remove / Delete a file: Here rm command will remove/delete the file file1.txt.rm file1.txt

To delete a directory tree:

rm -ir tmp

This rm command recursively removes the contents of all subdirectories of the tmp directory,prompting you regarding the removal of each file, and then removes the tmpdirectory itself.

To remove more files at once: rm command removes file1.txt and file2.txt files at the same time.rm file1.txt file2.txt

cd COMMAND: cd command is used to change the directory.

cd linux-command

This command will take you to the sub-directory(linux-command) from its parent directory.

Ex:

cd ..
This will change to the parent-directory from the current working directory/sub-directory.

cd ~
This command will move to the user's home directory which is "/home/username".

cp COMMAND:
cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

Copy two files:
cp file1.txt file2.txt
The above cp command copies the content of file1.txt to file2.txt

ls COMMAND:
ls command lists the files and directories under current working directory. Display root directory contents:

ls /
lists the contents of root directory.

Display hidden files and directories:
ls -a
lists all entries including hidden files and directories.

Display inode information:
ls –i

ln COMMAND:
ln command is used to create link to a file (or) directory. It helps to provide soft link for desired files.

Inode will be different for source and destination.
ln -s file1.txt file2.txt

Creates a symbolic link to 'file1.txt' with the name of 'file2.txt'. Here inode for 'file1.txt' and 'file2.txt'will be different.

mkdir command: Use this command to create one or more new directories.

Include one or more instances of the "**<DIRECTORY**" variable (separating each with a whitespace), and set each to the complete path to the new directory to be created.

mkdir OPTION <DIRECTORY>

rmdir command:

mv command:

diff command:

comm command:

wc command**:**

PROCESS UTILITIES:

ps Command:
ps command is used to report the process status. ps is the short name for Process Status.

1. **ps:** List the current running processes.

Output:
PID TTY TIME CMD
2540 pts/1 00:00:00 bash

2. **ps –f :** Displays full information about currently running processes.

Output:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
nirmala	2540	2536	0	15:31	pts/1	00:00:00	bash

3. **kill COMMAND:** kill command is used to kill the background process.

Step by Step process:

- Open a process music player or any file.xmms

press ctrl+z to stop the process.

- To know group id or job id of the background task.jobs -l

It will list the background jobs with its job id as,

- xmms 3956
- kmail 3467

To kill a job or process.

- **kill 3956**

kill command kills or terminates the background process xmms.

Disk utilities:
du (abbreviated from disk usage) is a standard Unix program used to estimate file spaceusage—space used under a particular directory or files on a file system.

$du kt.txt pt.txt /* the first column displayed the file's disk usage */
8 kt.txt
4 pt.txt

Using -h option: As mentioned above, -h option is used to produce the output in humanreadable format.

$du -h kt.txt pt.txt
8.0K kt.txt4.0K pt.txt
/*now the output is in human readable format i.e in Kilobytes */

Using -a option
$du -a kartik
8 kartik/kt.txt
4 kartik/pt.txt
4 kartik/pranjal.
png
4 kartik/thakral.png
4 kartik/thakral
24 kartik

/*so with -a option used all the files (under directory kartik) disk usage info is displayed alongwith the thakral sub-directory */

df command : Report file system disk space usage

$df kt.txt

Filesystem 1K-blocks Used Available Use% Mounted on
/dev/the2 1957124 1512 1955612 1% /snap/core
/* the df only showed the disk usage details of the file system that contains file kt.txt */

//using df without any filename //
$df
/* in this case df displayed the disk usage details of all mounted file systems */

Using -h : This is used to make df command display the output in human-readable format.

//using -h with df//
$df -h kt.txt
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/the2 1.9G 1.5M 1.9G 1% /snap/core
/*this output is easily understandable by the user and all cause of -h option */

NETWORKING COMMANDS

ping
The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not.
Syntax: $ping hostname or ip-address

The above command starts printing a response after every second. To come out of thecommand, you can terminate it by pressing CNTRL + C keys.

$ping google.com
PING google.com (74.125.67.100) 56(84) bytes of data.
64 bytes from 74.125.67.100: icmp_seq=1 ttl=54 time=39.4 ms

ftp: ftp stands for File Transfer Protocol. This utility helps you upload and download your filefrom one computer to another computer.
Syntax $ftp hostname or ip-address

$ftp amrood.com
Connected to amrood.com.
220 amrood.com FTP server (Ver 4.9 Thu Sep 2 20:35:07 CDT 2009)Name (amrood.com:amrood): amrood
331 Password required for amrood.Password:
230 User amrood logged in.ftp> dir
200 PORT command successful.
....
ftp> quit
221 Goodbye.

telnet:
Telnet is a utility that allows a computer user at one site to make a connection, login and then conduct work on a computer at another site. Once you login using Telnet, you can perform all the activities on your remotely connected machine.

C:>telnet amrood.comTrying...
Connected to amrood.com.Escape character is '^]'. login: amrood
amrood's Password:
**********WELCOME TO AMROOD.COM *
****
$ logoutLINUX PROGRAMMING LAB021-2022

Connection closed.C:>

Finger:
The finger command displays information about users on a given host. The host can be either local or remote.

Check all the logged-in users on the local machine –

$ finger

Login	Name	Tty	Idle	Login Time	Office
amrood		pts/0	Jun 25	08:03	(62.61.164.115)

Check all the logged-in users on the remote machine –

$ finger @avtar.com

Login Name Tty Idle Login Time Office amrood pts/0 Jun 25 08:03 (62.61.164.115)

Get the information about a specific user available on the remote machine –

$ finger amrood@avtar.com

Ifconfig: Ifconfig is used to configure the network interfaces.

FILTERS

more COMMAND:

more command is used to display text in the terminal screen. It allows only backwardmovement.

1. ***more -c index.txt***
   Clears the screen before printing the file .
2. ***more -3 index.txt***
   Prints first three lines of the given file. Press Enter to display the file line by line.

head COMMAND:

head command is used to display the first ten lines of a file, and also specifies how many linesto display.

1. **head index.php**
   This command prints the first 10 lines of 'index.php'.
2. **head -5 index.php**
   The head command displays the first 5 lines of 'index.php'.
3. **head -c 5 index.php**
   The above command displays the first 5 characters of 'index.php'.

tail COMMAND:

tail command is used to display the last or bottom part of the file. By default it displays last10 lines of a file.

1. **tail index.php**
   It displays the last 10 lines of 'index.php'.
2. **tail -2 index.php**
   It displays the last 2 lines of 'index.php'.

3. **tail -n 5 index.php**

It displays the last 5 lines of 'index.php'.

4. **tail -c 5 index.php**

It displays the last 5 characters of 'index.php'.

cut COMMAND:

cut command is used to cut out selected fields of each line of a file. The cut command uses delimiters to determine where to split fields.

cut -c1-3 text.txt

Output:

Thi

Cut the first three letters from the above line.

paste COMMAND:

paste command is used to paste the content from one file to another file. It is also used to set column format for each line.

paste test.txt>test1.txt

Paste the content from 'test.txt' file to 'test1.txt' file.

sort COMMAND:

sort command is used to sort the lines in a text file.

1. ***sort test.txt***

Sorts the 'test.txt'file and prints result in the screen.

2. ***sort -r test.txt***

Sorts the 'test.txt' file in reverse order and prints result in the screen.

uniq

Report or filter out repeated lines in a file.

uniq myfile1.txt > myfile2.txt - Removes duplicate lines in the first file1.txt and outputs the results to the second file.

TEXT PROCESSING UTILITIES

echo: display a line of text or echo command prints the given input string to standard output.eg.
echo I love India
echo $HOME

wc: print the number of newlines, words, and bytes in fileseg. wc file1.txt

nl: which lets you number lines in files.

eg. $ nl file11 hi

join- Join command is used for merging the lines of different sorted files based on the presence of common field into a single line. The second line will be appended at the end ofthe first line and cursor is placed at the end of line after joining.

Grep (Global Regular Expression Searching for a pattern), fgrep and egrep
$ grep -sales director**||** emp1 emp2

$fgrep _good bad great' userfile
$egrep _good | bad | great' userfile
cat, head, tail, sort, uniq, cut, paste and etc.

BACKUP UTILITIES
Linux backup and restore can be done using backup commands tar, cpio, dump and restore.

Backup Restore using tar command

tar: tape archive is used for single or multiple files backup and restore on/from a tape or file.
$tar cvf /dev/rmt/0 *

Options: c -> create ; v -> Verbose ; f->file or archive device ; * -> all files and directories .

$tar cvf /home/backup *
Create a tar called backup in home directory, from all file and directories s in the currentdirectory.

Viewing a tar backup on a tape or file
$tar tvf /dev/rmt/0 ## view files backed up on a tape device.
$tar tvf /home/backup ## view files backed up inside the backup
Note: t option is used to see the table of content in a tar file.

Extracting tar backup from the tape
$tar xvf /home/backup ## extract / restore files in to current directory.
Note : x option is used to extract the files from tar file. Restoration will go to present directoryor original backup path depending on relative or absolute path names used for backup.

Backup restore using cpio command

Using cpio command to backup all the files in current directory to tape.
find . -depth -print | cpio -ovcB > /dev/rmt/0

cpio expects a list of files and find command provides the list, cpio has to put these file onsome destination and a > sign redirect these files to tape. This can be a file as well .

Viewing cpio files on a tape

cpio -ivtB < /dev/rmt/0
Options i -> input ; v->verbose; t-table of content; B-> set I/O block size to 5120 bytes

Restoring a cpio backup
cpio -ivcB < /dev/rmt/0
Options i -> input ; v->verbose; t-table of content; B-> set I/O block size to 5120 bytes

WEEK 2

AIM : Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or directory and reports accordingly. Whenever the argument is a file it reports no of lines present in it.

ALGORITHM:

step 1: if arguments are less than 1 print Enter at least one input file name and goto step 9
Step 2: selects list a file from list of arguments provided in command line
Step 3: check for whether it is directory if yes print is directory and goto step 9
step 4: check for whether it is a regular file if yes goto step 5 else goto step 8
Step 5: print given name is regular file
step 6: print No of lines in file
step 7: goto step
step 8: print not a file or adirectory
step 9: stop

Script name: 2a.sh

```
for x in $*
do
if [ -f $x ]
then
echo " $x is  a file "
echo " no of lines in the file are "
wc -l $x
elif [ -d $x ]
then
echo " $x is  a directory "
else
echo " enter valid filename or directory name "
fi
done
```

OUTPUT

```
guest-glcbIs@ubuntu:~$sh 2a.sh dir1 d1
dir1 is a directory
d1 is a file
no of lines in the file are 2
```

WEEK 3

AIM : To write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the following.

a) FCFS b) SJF c) Round Robin d) Priority

DESCRIPTION

Assume all the processes arrive at the same time.

FCFSCPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

SJF CPU SCHEDULING ALGORITHM

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

ROUND ROBIN CPU SCHEDULING ALGORITHM

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

PRIORITY CPU SCHEDULING ALGORITHM

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

PROGRAM

a) FCFS CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<conio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
```

```
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

INPUT

```
Enter the number of processes --        3
Enter Burst Time for Process  0 --     24
Enter Burst Time for Process  1 --      3
Enter Burst Time for Process  2 --      3
```

OUTPUT

PROCESS	BURST TIME	WAITING TIME	RNAROUND TIME
P0	24	0	24
P1	3	24	27
P2	3	27	30

Average Waiting Time-- 17.000000
Average Turnaround Time -- 27.000000

b) SJF CPU SCHEDULING ALGORITHM

```
  #include<stdio.h>
int main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
float wtavg, tatavg;
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
for(i=0;i<n;i++) for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
```

```
bt[k]=temp;

temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

SAMPLE INPUT

```
Enter the number of processes --        4
Enter Burst Time for Process  0 --      6
Enter Burst Time for Process  1 --      8
Enter Burst Time for Process  2 --      7
Enter Burst Time for Process  3 --      3
```

SAMPLE OUTPUT

PROCESS	BURST TIME	WAITING TIME	URNAROUND TIME
P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000

C)ROUND ROBIN CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
main()
{
int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;float awt=0,att=0,temp=0;
printf("Enter the no of processes -- ");
scanf("%d",&n);

for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for process %d -- ", i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnter the size of time slice -- ");scanf("%d",&t);
max=bu[0]; for(i=1;i<n;i++)
if(max<bu[i])
                max=bu[i];for(j=0;j<(max/t)+1;j++)
for(i=0;i<n;i++)
if(bu[i]!=0)
```

```
if(bu[i]<=t)

{
tat[i]=temp+bu[i];
temp=temp+bu[i];
bu[i]=0;
}
else
{
bu[i]=bu[i]-t; temp=temp+t;

}

for(i=0;i<n;i++)
{wa[i]=tat[i]-ct[i];att+=tat[i];
awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n); printf("\nThe Average Waiting time is
",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
}
```

SAMPLE INPUT

```
Enter the no of processes – 3
Enter Burst Time for process 1 –        24
Enter Burst Time for process 2 --       3
Enter Burst Time for process 3 --       3

Enter the size of time slice – 3
```

PROCESS	BURST TIME	AITING TIME	TURNAROUND TIME
1	24	6	30
2	3	4	7
3	3	7	10

SAMPLE OUTPUT

```
The Average Turnaround time is – 15.666667
The Average Waiting time is --          5.666667
```

d) PRIORITY CPU SCHEDULING ALGORITHM

```
#include<stdio.h>main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;float wtavg, tatavg;
printf("Enter the number of processes --- ");scanf("%d",&n);

for(i=0;i<n;i++)
{
p[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ",i);scanf("%d %d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(pri[i] > pri[k])
{
temp=p[i];p[i]=p[k]; p[k]=temp;

temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;

temp=pri[i]; pri[i]=pri[k];pri[k]=temp;


}
wtavg = wt[0] = 0;
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = tat[i-1] + bt[i];

wtavg = wtavg + wt[i]; tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
for(i=0;i<n;i++)
printf("\n%d \t\t %d \t\t  %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n); printf("\nAverage Turnaround Time is ---
%f",tatavg/n);
}
```

INPUT

```
Enter the number of processes --5Enter the Burst Time & Priority of
Process 0 ---  10
                                Enter the Burst Time & Priority of Process 1 ---  1
                                Enter the Burst Time & Priority of Process 2 ---  2
                                Enter the Burst Time & Priority of Process 3 ---  1
                                Enter the Burst Time & Priority of Process 4 ---  5
```

EXPECTED OUTPUT

PROCESS

	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
1	1	1	0	1
4	2	5	1	6
0	3	10	6	16
2	4	2	16	18
3	5	1	18	19

Average Waiting Time is --- 8.200000 Average Turnaround Time is --- 12.000000

WEEK 4

AIM: To Simulate Bankers Algorithm for Dead Lock Avoidance; Simulate Bankers Algorithm for Dead Lock Prevention.

DESCRIPTION

In a multiprogramming environment, several processes may compete for a finite number of resources. A processrequests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

DEADLOCK AVOIDANCE : PROGRAM

```
#include<stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("********** Banker's Algo ************\n");
input();
show();
cal();
getch();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
```

```
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}

}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}
}
void show()
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
for(j=0;j<r;j++)
{
printf("%d ",alloc[i][j]);
}
printf("\t");
for(j=0;j<r;j++)
{
printf("%d ",max[i][j]);
}
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
printf("%d ",avail[j]);
}
}
}
void cal()
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
```

```
int safe[100];
int i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)

{
need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
int c=0;
for(j=0;j<r;j++)
{
if((finish[i]==0)&&(need[i][j]<=avail[j]))
{
c++;
if(c==r)
{
for(k=0;k<r;k++)
{
avail[k]+=alloc[i][j];
finish[i]=1;
flag=1;
}
printf("P%d->",i);
if(finish[i]==1)
{
i=n;
}
}
}
}
}
}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
```

```
c1++;
}
else
{
printf("P%d->",i);
}

}
if(c1==n)
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
```

OUTPUT:

```
Enter the no of processes 5
Enter the no of resources instances 3
Enter the max matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter available resources 3 2 2
P1->p3->p4->p2->p0->
The system is in safe state.
```

WEEK 5

AIM : a)To Write a C program to simulate the concept of Dining-philosophers problem.

DESCRIPTION

The dining-philosophers problem is considered a classic synchronization problem because it is an example of a large class of concurrency-control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner. Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, eachbelonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors).

A philosopher may pick up only one chopstick at a time. Obviously, she cam1ot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again. The dining-philosophers problem may lead to a deadlock situation and hence some rules have to be framed to avoid the occurrence of deadlock.

PROGRAM

```
#include<stdio.h>
  #include<stdlib.h>
  int tph, philname[20], status[20], howhung, hu[20], cho;
  main()
  {
  int i;
  printf("\n\nDINING PHILOSOPHER PROBLEM");
  printf("\nEnter the total no. of philosophers: ");
  scanf("%d",&tph);
  for(i=0;i<tph;i++)
  {
  philname[i]=(i+1);
  status[i]=1;
  }
  Printf("How many are hungry : ");
  scanf("%d", &howhung);
  if(howhung==tph)
  {
  printf("\n All are hungry..\nDead lock stage will occur");
  printf("\nExiting\n");
  }
  else
  {
  for(i=0;i<howhung;i++)
  {
  printf("Enterphilosopher%dposition:",(i+1));
  scanf("%d",&hu[i]);
  status[hu[i]]=2;
```

```
}
do
{
printf("1.One can eat at a time\t2.Two can eat at a time\t3.Exit\nEnter your choice:");
scanf("%d", &cho);
switch(cho)
{
case 1:one();
break;
case
2:two();
break;
case 3: exit(0);
default: printf("\nInvalid option..");
}
}while(1);
}
}
one()
{
int pos=0, x, i;
printf("\nAllow one philosopher to eat at any time\n");
for(i=0;i<howhung; i++, pos++)
{
printf("\nP %d is granted to eat", philname[hu[pos]]);
for(x=pos;x<howhung;x++)
printf("\nP %d is waiting", philname[hu[x]]);
}
}
two()
{
int i, j, s=0, t, r, x;
printf("\n Allow two philosophers to eat at same time\n");
for(i=0;i<howhung;i++)
{
for(j=i+1;j<howhung;j++)
{
if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)
{
printf("\n\ncombination %d \n", (s+1));
t=hu[i];
r=hu[j];
s++;
printf("\nP %d and P %d are granted to eat", philname[hu[i]], philname[hu[j]]);
```

```
for(x=0;x<howhung;x++)
{
if((hu[x]!=t)&&(hu[x]!=r))
printf("\nP %d is waiting", philname[hu[x]]);
}
}
}
}
}
```

INPUT

DINING PHILOSOPHER PROBLEM
Enter the total no. of philosophers: 5How many are hungry : 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

EXPECTED OUTPUT

1. One can eat at a time2.Two can eat at a time3.ExitEnter your choice: 1

Allow one philosopher to eat at any timeP 3 is granted to eat
P 3 is waitingP 5 is waitingP 0 is waiting
P 5 is granted to eatP 5 is waiting
P 0 is waiting
P 0 is granted to eatP 0 is waiting
1.One can eat at a time 2.Two can eat at a time 3.ExitEnter your choice: 2

Allow two philosophers to eat at same timecombination 1
P 3 and P 5 are granted to eatP 0 is waiting

combination 2
P 3 and P 0 are granted to eatP 5 is waiting

combination 3
P 5 and P 0 are granted to eatP 3 is waiting

1.One can eat at a time 2.Two can eat at a time 3.ExitEnter your choice: 3

AIM : b) To Write a C program to simulate producer-consumer problem using semaphores.

DESCRIPTION

Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

PROGRAM

```
#include<stdio.h>
int main()
{
int buffer[10], bufsize, in, out, produce, consume,choice=0;
in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n1.Produce\t 2.Consume\t 3.Exit");
printf("\nEnter your choice:");
scanf("%d",&choice);
switch(choice)
{
case 1: if((in+1)%bufsize==out)
printf("\nBuffer is Full");
else
{
printf("\nEnter the value:");
scanf("%d", &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
}
break;
case 2:
if(in == out) printf("\nBuffer is Empty"); else
{
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
}break;
}
}
}
```

SAMPLE OUTPUT

1. Produce 2. Consume 3. ExitEnter your choice: 2
Buffer is Empty
1. Produce 2. Consume 3. ExitEnter your choice: 1
Enter the value: 100
1. Produce 2. Consume 3. ExitEnter your choice: 2
The consumed value is 100
1. Produce 2. Consume 3. ExitEnter your choice: 3

SAMPLE OUTPUT

2. Produce 2. Consume 3. ExitEnter your choice: 2
Buffer is Empty
1. Produce 2. Consume 3. ExitEnter your choice: 1
Enter the value: 100
1. Produce 2. Consume 3. ExitEnter your choice: 2
The consumed value is 100
1. Produce 2. Consume 3. ExitEnter your choice: 3

WEEK 6

AIM: To Implement kill(), raise() and sleep() functions using a C program.

kill( ) and sleep( ):

Program file name:

kill.c

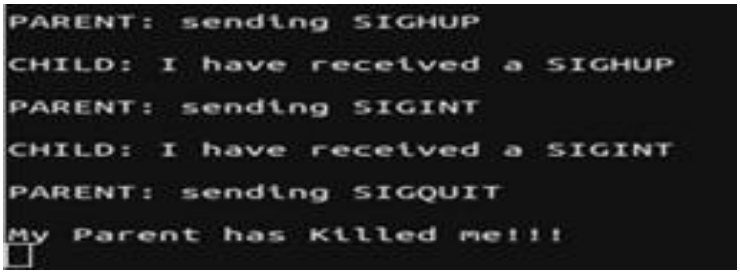
```
#include <signal.h>
  #include <stdio.h>
  #include <stdlib.h>
  #include <sys/types.h>
  #include <unistd.h>
  // function declarationvoid
  void sighup();
  void sigint();
  void sigquit();
  // driver codevoid main()
  int main(void)
  {
  int pid;
  /* get child process */
  if ((pid = fork()) < 0) {
  perror("fork");
  exit(1);
  }
  if (pid == 0)
  { /* child */ signal(SIGHUP, sighup);
  signal(SIGINT, sigint);
  signal(SIGQUIT, sigquit);
  for (;;)
  ; /* loop for ever */
  }
  else /* parent */
  { /* pid hold id of child */
  printf("\nPARENT: sending SIGHUP\n\n");
  kill(pid, SIGHUP);
  sleep(3); /* pause for 3 secs */
  printf("\nPARENT: sending SIGINT\n\n");
  kill(pid, SIGINT);
  sleep(3); /* pause for 3 secs */
  printf("\nPARENT: sending SIGQUIT\n\n");
  kill(pid, SIGQUIT);
  sleep(3);
  }
  }
  // sighup() function definition
```

```
void sighup()
{

signal(SIGHUP, sighup); /* reset signal */
printf("CHILD: I have received a SIGHUP\n");
}
// sigint() function definitionvoid
void sigint()
{
signal(SIGINT, sigint); /* reset signal */
printf("CHILD: I have received a SIGINT\n");
}
// sigquit() function definition
void sigquit()
{
printf("My DADDY has Killed me!!!\n");
exit(0);
}
```

OP: $ ./a.out

EXPECTED OUTPUT:



raise( ):

Program file name: raise.c

```
#include  <stdio.h>
 #include <stdlib.h>
 #include <signal.h>

void signal_handler(int signal)
{
/* Display a message indicating we have received a signal */
if (signal == SIGUSR1) printf("Received a SIGUSR1 signal\n");

/* Exit the application */exit(0);
}

int main(int argc, const char * argv[])
{
/* Display a message indicating we are registering the signal handler */printf("Registering the signal handler\n");
```

```
/* Register the signal handler */ signal(SIGUSR1, signal_handler);
/* Display a message indicating we are raising a signal */printf("Raising a SIGUSR1 signal\n");
/* Raise the SIGUSR1 signal */raise(SIGUSR1);
/* Display a message indicating we are leaving main */printf("Finished main\n");


return 0;
}
```

SAMPLE OUTPUT:

```
Registering the signal handler
Raising a SIGUSR1
ignal Received a SIGUSR1 signal
```

AIM: Implement alarm(), pause() and abort() functions using a C program.

Program file name: alarmpause.c

```
#define  _POSIX_SOURCE
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <time.h>

void catcher(int signum)
{
puts("inside catcher.  ");
 }

void timestamp()
{
time_t t;
time(&t);
 printf("the time is %s", ctime(&t));
 }

 main()
 {
 struct sigaction sigact;

 sigemptyset(&sigact.sa_mask);
 sigact.sa_flags = 0;
 sigact.sa_handler = catcher;
 sigaction(SIGALRM, &sigact, NULL);


 alarm(10);
 printf("before pause... ");
 timestamp();
 pause();
 printf("after pause... ");
 timestamp();
 }
```

SAMPLE OUTPUT:

```
before pause... the time is Fri Jun 16 09:42:29 2001inside catcher...

after pause... the time is Fri Jun 16 09:42:39 2001
```

abort():

/* abort.c -- terminates execution abnormally */

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

int main(void)
{
abort();
printf("\"abort() called prior to printf()\"\n");

return 0;
}
```

AIM:- To write a C program that illustrate communication between two process usingunnamed pipes

Program file name: unnamed_pipe.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<fcntl.h>
void server(int,int);
void   client(int,int);
int main()
{
int p1[2],p2[2],pid;
pipe(p1);
pipe(p2);
pid=fork();
if(pid==0)
{
close(p1[1]);
close(p2[0]);
server(p1[0],p2[1]);
return 0;
}
close(p1[0]);
close(p2[1]);
client(p1[1],p2[0]);
wait();
return 0;
}
```

```
void client(int wfd,int rfd)
{
int i,j,n;
char fname[2000];
char buff[2000];
printf("ENTER THE FILE NAME :");
scanf("%s",fname);
printf("CLIENT SENDING THEREQUEST..... PLEASE WAIT\n");
sleep(10);
write(wfd,fname,2000);
n=read(rfd,buff,2000);
buff[n]='\0';
printf("THE RESULTS OF CLIENTS ARE..... \n");
write(1,buff,n);
}

void server(int rfd,int wfd)
{
int i,j,n;
char fname[2000];char buff[2000];
n=read(rfd,fname,2000);fname[n]='\0';
int fd=open(fname,O_RDONLY);sleep(10);
if(fd<0)
write(wfd,"can't open",9);
else
n=read(fd,buff,2000);
write(wfd,buff,n);
}
```

EXPECTED OUTPUT

Enter File name:file.txt
CLIENT SENDING THEREQUEST PLEASE WAIT

WEEK 7

AIM :a) To Write a program that illustrates communication between two process using named pipes or FIFO.

Algorithm:

Create two processes, one is fifoserver_twoway and another one is fifoclient_twoway.

Algorithm for fifoserver_twoway :

step 1:Start

step 2: Creates a named pipe (using library function mkfifo())with name -fifo_twoway**||** in /tmp directory, if not created.

step 3: Opens the named pipe for read and write purposes.

step 4: Here, created FIFO with permissions of read and write for Owner. Read for Group and nopermissions for Others.

step 5: Waits infinitely for a message from the client.

step 6: If the message received from the client is not -end||, prints the message and reverses the string. The reversed string is sent back to the client. If the message is –end**||**, closes the fifo and ends the process.

step 7:stop.

Algorithm for client :

Step 1: start

Step 2: Opens the named pipe for read and write purposes.Step 3: Accepts string from the user.

Step 4: Checks, if the user enters -end|| or other than -end**||**. Either way, it sends a message to the server. However, if the string is —end||, this closesthe FIFO and also ends the process.

Step 5: If the message is sent as not -end||, it waits for the message (reversed string) from the client and prints the reversed string.

Step 6: Repeats infinitely until the user enters the string -end||. Step 7: stop

Programs:

```
/* Filename: fifoserver_twoway.c */
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define FIFO_FILE "/tmp/fifo_twoway"
void reverse_string(char *);
int main()
{
int fd;

char readbuf[80];

char end[10];

int to_end;
int read_bytes;
```

```
 /* Create the FIFO if it does not exist */mkfifo(FIFO_FILE, S_IFIFO|0640);
 strcpy(end, "end");
 fd = open(FIFO_FILE, O_RDWR);
 while(1)
 {
 read_bytes = read(fd, readbuf, sizeof(readbuf));
 readbuf[read_bytes] = '\0';
printf("FIFOSERVER: Received string: \"%s\" and length is %d\n", readbuf,(int)strlen(readbuf));
 to_end = strcmp(readbuf, end);

if (to_end == 0)
{
close(fd);
break;
 }
 reverse_string(readbuf);
printf("FIFOSERVER: Sending Reversed String: \"%s\" and length is %d\n", readbuf, (int)strlen(readbuf));
 write(fd, readbuf, strlen(readbuf));
 /*
 sleep - This is to make sure other process reads this, otherwise thisprocess would retrieve the message
 */ sleep(2);
 }
 return 0;
 }


 void reverse_string(char *str)
 {
  int last, limit, first;
 char temp;
 last = strlen(str) - 1;limit = last/2;
 first = 0;

 while (first < last)
{
temp = str[first];
str[first] =
str[laststr[last] =
temp; first++;
 last--;
 }
 return;
 }
```

OUTPUT:

```
FIFOSERVER: Received string: "LINUX IPCs" and length is 10
FIFOSERVER: Sending Reversed String: "sCPI XUNIL" and length is 10
FIFOSERVER: Received string: "Inter Process Communication" and length is 27
FIFOSERVER: Sending Reversed String: "noitacinummoC ssecorP retnI" and length is 27
FIFOSERVER: Received string: "end" and length is 3
```

```
/* Filename: fifoclient_twoway.c */
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#define FIFO_FILE "/tmp/fifo_twoway"
int main()
{
int fd;
int end_process;int stringlen;
int read_bytes; char readbuf[80];char end_str[5];
printf("FIFO_CLIENT: Send messages, infinitely, to end enter \"end\"\n");
fd = open(FIFO_FILE, O_CREAT|O_RDWR);
strcpy(end_str, "end");

while (1)
{
printf("Enter string:  ");
fgets(readbuf, sizeof(readbuf), stdin);
stringlen = strlen(readbuf);
readbuf[stringlen - 1] = '\0';
end_process = strcmp(readbuf, end_str);

//printf("end_process is %d\n", end_process);
if (end_process != 0)
{
write(fd, readbuf, strlen(readbuf));
printf("FIFOCLIENT: Sent string: \"%s\" and string length is %d\n", readbuf,(int)strlen(readbuf));
read_bytes = read(fd, readbuf, sizeof(readbuf));
readbuf[read_bytes] = '\0';
printf("FIFOCLIENT: Received string: \"%s\" and length is %d\n", readbuf,(int)strlen(readbuf));
}
else
{
write(fd, readbuf, strlen(readbuf));
printf("FIFOCLIENT: Sent string: \"%s\" and string length is %d\n", readbuf,(int)strlen(readbuf));
close(fd);
break;
}
}
return 0;
}
```

OUTPUT:

FIFO_CLIENT: Send messages, infinitely, to end enter "end"
Enter string: LINUX IPCs
FIFOCLIENT: Sent string: "LINUX IPCs" and string length is 10
FIFOCLIENT: Received string: "sCPI XUNIL" and length is 10
Enter string: Inter Process Communication
FIFOCLIENT: Sent string: "Inter Process Communication" and string length is 27
FIFOCLIENT: Received string: "noitacinummoC ssecorP retnI" and length is 27
Enter string: end
FIFOCLIENT: Sent string: "end" and string length is 3

AIM : b) Write a C program that receives a message from message queue and display them.

ALGORITHM:

Step 1:Start
Step 2:Declare a message queue
typedef struct msgbuf
{
long mtype;
char mtext[MSGSZ];
}
message_buf;
Mtype =0Retrieve the next message on the queue, regardless of its mtype.
PositiveGet the next message with an mtype equal to the specifiedmsgtyp.
NegativeRetrieve the first message on the queue whose mtype fieldis less than orequal to the absolute value of the msgtyp argument.Usually mtype is set to1
mtext is the data this will be added to the queue.
Step 3:Get the message queue id for the "name" 1234, which was created by the serverkey = 1234Step 4 : if ((msqid = msgget(key, 0666< 0) Then print error The msgget() function shall return the message queue identifier associated with the argument key.
Step 5: Receive message from message queue by using msgrcv function
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
#include < sys/msg.h>
(msgrcv(msqid, &rbuf, MSGSZ, 1, 0)msqid: message queue id &sbuf: pointer to user defined structure MSGSZ: message sizeMessage type: 1
Message flag:The msgflg argument is a bit mask constructed by ORing together zero or more of the following flags: IPC_NOWAIT or MSG_EXCEPT or MSG_NOERROR
Step 6:if msgrcv <0 return error
Step 7:otherwise print message sent is sbuf.mextStep 8:stop

Program:

```
//IPC_msgq_send.c

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXSIZE                 128
void die(char *s)
{
perror(s);
exit(1);
}
typedef struct msgbuf
{
long                    mtype;
char                    mtext[MAXSIZE];
};
```

```
main()
{
int msqid;
int msgflg = IPC_CREAT | 0666;
key_t key;
struct msgbuf sbuf;
size_t buflen;

key = 1234;

   if ((msqid = msgget(key, msgflg )) < 0) //Getthe message queue ID for the given key
die("msgget");

//Message Typesbuf.mtype = 1;

printf("Enter a message to add to messagequeue : ");
scanf("%[^\n]",sbuf.mtext);
getchar();

buflen = strlen(sbuf.mtext) + 1 ;

 if (msgsnd(msqid, &sbuf, buflen,
IPC_NOWAIT) < 0)
{
       printf ("%d, %d, %s, %d\n", msqid,sbuf.mtype, sbuf.mtext, buflen);
die("msgsnd");
}

else
printf("Message Sent\n");

exit(0);
}
```

EXPECTED OUTPUT:

```
//IPC_msgq_rcv.c

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 128

void die(char *s)
{
perror(s);exit(1);
}

typedef struct msgbuf
{
long                    mtype;
char                    mtext[MAXSIZE];
} ;
main()
{
int msqid;key_t key;
struct msgbuf rcvbuffer;key = 1234;

if ((msqid = msgget(key, 0666)) < 0)die("msgget()");

//Receive an answer of message type 1.
 if (msgrcv(msqid, &rcvbuffer, MAXSIZE, 1, 0) < 0)die("msgrcv");

printf("%s\n", rcvbuffer.mtext);exit(0);
}
```

EXPECTED OUTPUT:

WEEK 8

AIM: To write a C program that illustrates two processes communicating using Shared memory.

Algorithm:-

the shared memory identifier associated with key Theargument key is equal to IPC_PRIVATE. so that
step1.Start
step 2.Include header files required for the program are#include <sys/types.h>
#include <sys/ipc.h>#include <sys/shm.h>#include <unistd.h> #include <string.h> #include <errno.h>
step 3.Declare the variable which are required aspid_t pid
int *shared /* pointer to the shm */int shmid
step 4.Use shmget function to create shared memory#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg)
The shmget() function shall return the operating system selects the next availablekey for a newly created shared block of memory.Size represents size of shared memory block Shmflg shared memory permissions which are represented by octalinteger shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0666);

print the shared memory idstep 5.if fork()==0 Then
begin
shared = shmat(shmid, (void *) 0, 0)
print the shared variable(shared) *shared=2print *shared sleep(2)
print *shared

end

step 6.else
begin
shared = shmat(shmid, (void *) 0, 0) print the shared variable(shared) print *shared sleep(1) *shared=30
printf("Parent value=%d\n", *shared);sleep(5)
shmctl(shmid, IPC_RMID, 0)

end

step 7.stop.

Sha.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <errno.h>
```

```
int main(void)
{
pid_t pid;
int *shared; /* pointer to the shm */ int shmid;
shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0666);
printf("Shared MemoryID=%u",shmid);
if (fork() == 0) { /* Child */
/* Attach to shared memory and print the pointer */
shared = shmat(shmid, (void *) 0,0);
printf("Child pointer %u\n", shared);
*shared=1;
printf("Child value=%d\n", *shared);
sleep(2); printf("Child value=%d\n", *shared);
}
else { /* Parent */
/* Attach to shared memory and print the pointer */
shared = shmat(shmid, (void *) 0,0);
printf("Parent pointer %u\n", shared); printf("Parent value=%d\n", *shared);
sleep(1);
*shared=42;
printf("Parent value=%d\n", *shared);
sleep(5);shmctl(shmid, IPC_RMID, 0);
}
}
```

OUTPUT:

```
$cc shared_mem.c
$ ./a.out
Shared Memory ID=65537
Child pointer 3086680064
Child value=1
Shared Memory ID=65537
Parent pointer 3086680064
Parent value=1
Parent value=42
Child value=42
```

WEEK 9

AIM: To Simulate all page replacement algorithms a) FIFO b) LRU c) OPTIMAL

DESCRIPTION

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.

PROGRAM

FIFO PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h>
int main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
        scanf("%d",&rs[i]);
        printf("\n Enter no. of frames -- ");
        scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;

printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
                                if(m[k]==rs[i])
                                break;
}
if(k==f)
{
                                m[count++]=rs[i];
                                pf++;
}
```

```
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f)
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
}
```

INPUT

Enter the length of reference string – 20
Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter no. of frames -- 3

OUTPUT

The Page Replacement Process is –

-1	-1	PF No. 1
0	-1	PF No. 2
0	1	PF No. 3
0	1	PF No. 4
0	1	
3	1	PF No. 5
3	0	PF No. 6
3	0	PF No. 7
2	0	PF No. 8
2	3	PF No. 9
2	3	PF No. 10
2	3	
2	3	
1	3	PF No. 11
1	2	PF No. 12
1	2	
1	2	
1	2	PF No. 13
0	2	PF No. 14
0	1	PF No. 15

The number of Page Faults using FIFO are 15

LRU PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h>
#define high 37
void main()
{
int fframe[10],used[10],index,i;
int count,n1,k,nf,np=0,page[high],tmp;
int flag=0,pf=0;
printf("Enter no. of frames:");
scanf("%d",&nf);
for(i=0;count<nf;count++)
```

```
fframe[count]=-1;
printf(" lru page replacement algorithm in c ");
printf("Enter pages (press -999 to exit):\n");
for(count=0;count<high;count++)
{
scanf("%d",&tmp);
if(tmp==-999) break;
page[count]=tmp;
np++;
}
for(count=0;count<np;count++)
{
flag=0;
for(n1=0;n1<nf;n1++)
{
if(fframe[n1]==page[count])
{
printf("\n\t");
flag=1;break;
}
}
  /* program for lru page replacement algorithm in c */
  if(flag==0)
  {
  for(n1=0;n1<nf;n1++) used[n1]=0;
  for(n1=0,tmp=count-1;n1<nf-1;n1++,tmp--)
  {
  for(k=0;k<nf;k++)
  {
  if(fframe[k]==page[tmp])
  used[k]=1;
  }
  }
  for(n1=0;n1<nf;n1++)
  if(used[n1]==0)
  index=n1;
  fframe[index]=page[count];
  printf("\nFault: ");
  pf++;
  }
  for(k=0;k<nf;k++)
  printf("%d\t",fframe[k]);
  } // lru algorithm in c
  printf("\nnumber of total page faults is: %d ",pf);
  }
```

EXPECTED OUTPUT

```
lru page replacement algorithm in c
Enter no. of frames Enter pages (press -999 to exit):
2 3 2 1
5 2 4 5
```

```
-1 -1 2 Fault:
-1 -1 2 Fault:
-1 3 2 Fault:
1 3 2 Fault:
1 3 2
1 5 2 Fault:
4 5 2 Fault:
4 5 2
4 5 3 Fault:
2 5 3 Fault:
2 5 3
2 5 3
```

OPTIMAL PAGE REPLACEMENT ALGORITHM

DESCRIPTION

Optimal page replacement algorithm has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly. The basic idea is to replace the page that will not be used for the longest period of time. Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames. Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

PROGRAM

```
#include<stdio.h>
int fr[3], n, m;
void display();
void main()
{
int i,j,page[20],fs[10];
 int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
float pr;
printf("Enter length of the reference string:");
scanf("%d",&n);
printf("Enter the reference string: ");
for(i=0;i<n;i++)
scanf("%d",&page[i]);
printf("Enter no of frames:");
scanf("%d",&m);
for(i=0;i<m;i++)
fr[i]=-1;
pf=m;
for(j=0;j<n;j++)
{
flag1=0; flag2=0;
for(i=0;i<m;i++)
{
if(fr[i]==page[j])
{
flag1=1; flag2=1; break;
```

```
}
}
if(flag1==0)
{
for(i=0;i<m;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j]; flag2=1;break;
}
}
}
if(flag2==0)
{
for(i=0;i<m;i++) lg[i]=0; for(i=0;i<m;i++)
{
for(k=j+1;k<=n;k++)
{
if(fr[i]==page[k])
{
lg[i]=k-j; break;
}
}
}
found=0;
for(i=0;i<m;i++)
{
if(lg[i]==0)
{
index
=i;
found =1;
break;
}
}
if(found==0)
{
max=lg[0]; index=0; for(i=0;i<m;i++)
{
if(max<lg[i])
{
max=lg[i]; index=i;
}
}
}
fr[index]=page [j];
pf++;
}
display();
}
printf("Number of page faults :%d\n",pf);
pr=(float)pf/n*100;
```

```
printf("Page fault rate = %f \n", pr);
//getch();
}
void display()
{
int i;
for(i=0;i<m;i++)
printf("%d\t",fr[i]);
printf("\n");
}
```

INPUT

Enter number of page references -- 10
Enter the reference string -- 1 2 3 4 5 2 5 2 5 1 4 3
Enter the available no. of frames --- 3

OUTPUT

The Page Replacement Process is –

1	-1	-1	PF No. 1
1	2	-1	PF No. 2
1	2	3	PF No. 3
4	2	3	PF No. 4
5	2	3	PF No. 5
5	2	3	
5	2	3	
5	2	1	PF No. 6
5	2	4	PF No. 7
5	2	3	PF No. 8

Total number of page faults -- 8

WEEK 10

AIM: To write a C program that takes one or more file/directory names as command line input and reports following information A) File Type B) Number Of Links C) Time of last Access D) Read, write and execute permissions

Algorithm:

Step 1:start
Step 2:Declare struct stat a
Step 3:read arguments at command line
Step 4: set the status of the argument using stat(argv[i],&a);
Step 5:Check whether the given file is Directory file by using S_ISDIR(a.st_mode)if it is a directory file
print Directory file
Else
print is Regular fileStep6: print number of links
Step 7:print last time access
Step 8:Print Read,write and execute permissionsStep 9:stop

Program File name: 6.c

```
#include<stdio.h>
#include<sys/stat.h>
#include<time.h>
int main(int argc,char *argv[])
{
int i,j; struct stat a;
for (i=1;i<argc;i++)
{
printf("%s : ",argv[i]);
stat(argv[i],&a);
if(S_ISDIR(a.st_mode))
{
                                printf("is a Directory file\n");
}
else
{
                                printf("is Regular file\n");
}
printf("******File Properties********\n");
printf("Inode Number:%ld\n",a.st_ino);
printf("UID:%o\n",a.st_uid);
printf("GID:%o\n",a.st_gid);
printf("No of Links:%d\n",a.st_nlink);
printf("Last Access time:%s",asctime(localtime(&a.st_atime)));
```

```
printf("Permission flag:%o\n",a.st_mode%512);
printf("size in bytes:%ld\n",a.st_size);
printf("Blocks Allocated:%d\n",a.st_blocks);
printf("Last modification time %s\n",ctime(&a.st_atime));
}
}
```

EXPECTED OUTPUT

WEEK 11

AIM: a)To Implement in c language the following UNIX commands using system calls i) cat ii) ls iii) Scanning Directories (Ex: opendir(),readdir(),etc.)

DESCRIPTION:

(i) cat COMMAND:cat linux command concatenates files and print it on the standardoutput. SYNTAX:
cat [OPTIONS] [FILE]...

OPTIONS:

-A	Show all.
-b	Omits line numbers for blank space in the output.
-e	A $ character will be printed at the end of each line prior to a new line.
-E	Displays a $ (dollar sign) at the end of each line.
-n	Line numbers for all the output lines.
-s	If the output has multiple empty lines it replaces it with one empty line.
-T	Displays the tab characters in the output.
-v	Non-printing characters (with the exception of tabs, new-lines & form-feeds) are printed visibly.

Operations With cat Command:

1. To Create a new file:

$cat > file1.txt

This command creates a new file file1.txt. After typing into the file press control+d (^d) simultaneously toend the file.

2. To Append data into the file:

$cat >> file1.txt

To append data into the same file use append operator >> to write into the file, else the file will be overwritten (i.e., all of its contents will be erased).

3. To display a file:

$cat file1.txt

This command displays the data in the file.

4. To concatenate several files and display:

$cat file1.txt file2.txt

The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you can print those files in a new file or display the file using less command.

cat file1.txt file2.txt | less

5. To concatenate several files and to transfer the output to another file.

$cat file1.txt file2.txt > file3.txt

In the above example the output is redirected to new file file3.txt. The cat command will create new file file3.txt and store the concatenated output into file3.txt.

Algorithm:
Step 1:Start
Step 2:read arguments from keyboard at command line
Step 3:if no of arguments are less than two print ENTER CORRECT ARGUMENTSElse goto step 4
Step4:read the date from specified file and write it to destination fileStep 5 :stop

Program file name: 5a.c

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/stat.h>
int main(int argc,char *argv[])
{
int fd,n;
char buff[512];
if(argc!=2)
printf("ENTER CORRECT ARGUMENTS :");
if((fd=open(argv[1],4))<0)
{
printf("ERROR");
return 0;
}
while(n=read(fd,buff,sizeof(buff))>0)write(1,buff,n);
}
```

(ii) ls
Description:
ls command is used to list the files present in a directory

Algorithm:
Step 1. Start.
Step 2. open directory using opendir( ) system call. Step 3. read the directory using readdir( ) system call.Step 4. print dp.name and dp.inode .
Step 5. repeat above step until end of directory.Step 6: Stop.

Program name: 5b.c

```
#include<stdio.h> #include<dirent.h> void quit(char*,int);
int main(int argc,char **argv )
{
DIR *dirop;
struct dirent *dired;if(argc!=2)
{
printf("Invalid number of arguments\n");
```

```
}
if((dirop=opendir(argv[1]))==NULL) printf("Cannot open directory\n");
while((dired=readdir(dirop))!=NULL)
        printf("%10d %s\n",dired>d_ino,dired>d_name);closedir(dirop);
}
```

iii)Scanning directories (using system calls)

DESCRIPTION:

Scanning directories is used to opendir(), readdir(), rewinddir(), closedir(), etc

Program File name: 5c.c

```
#include <stdio.h>
#include <dirent.h>

int main(void)
{
   struct dirent *de; // Pointer for directory entry

   // opendir() returns a pointer of DIR type.
   DIR *dr = opendir(".");

   if (dr == NULL) // opendir returns NULL if couldn't open directory
   {
      printf("Could not open current directory" );
      return 0;
   }


   while ((de = readdir(dr)) != NULL)
         printf("%s\n", de->d_name);

   closedir(dr);
   return 0;

}
```

OUTPUT

All files and subdirectories of current directory

b) To write a C program to create child process and allow parent process to display “parent” and the child to display “child” on the screen

ALGORITHM:

Step 1: start
Step2: call the fork() function to create a childprocess fork function returns 2 values
step 3: which returns 0 to child process
step 4:which returns process id to the parentprocess step 5:stop

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
int pid,pid1,pid2;
pid=fork();
if(pid==-1)
{
printf("ERROR IN
PROCESS
CREATION \n");
exit(0);
}
if(pid!=0)
{
pid1=getpid();
printf("\n the parent
process ID is %d",
pid1);
}
else
{
pid2=getpid();
printf("\n the child
process ID is %d\n",
pid2);
}
    }
```

Output:

```
[root@dba ~]# cc -
o 8 8a.c[root@dba
~]# ./8 the child
process ID is 4485
the parent process
ID is 4484
```

WEEK 12

AIM: Write a C program to simulate disk scheduling algorithms. a) FCFS b) SCAN c) C-SCAN

DESCRIPTION

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as diskscheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the diskarm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip

PROGRAM

FCFS DISK SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int RQ[100],i,n,TotalHeadMoment=0,initial;
printf("Enter the number of Requests\n");
scanf("%d",&n);
printf("Enter the Requests sequence\n");
  for(i=0;i<n;i++)
  scanf("%d",&RQ[i]);
  printf("Enter initial head position\n");
  scanf("%d",&initial);
  for(i=0;i<n;i++)
  {
  TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
  initial=RQ[i];
  }
  printf("Total head moment is %d",TotalHeadMoment);
  return 0;
        }
```

INPUT

Enter the number of Request 8
Enter the Requests Sequence 95 180 34 119 11 123 62 64
Enter initial head position 50

Expected Output:

Total head moment is 644

OUTPUT:

SCAN DISK SCHEDULING ALGORITHM

```
#include<stdio.h>
int main( )
{
int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
clrscr();
printf("enter the no of tracks to be traversed ");
scanf("%d'",&n);
printf("enter the position of head ");
scanf("%d",&h);
t[0]=0;
t[1]=h;
printf("enter the tracks ");
for(i=2;i<n+2;i++)
scanf("%d",&t[i]);
for(i=0;i<n+2;i++)
{
for(j=0;j<(n+2)-i-1;j++)
{
if(t[j]>t[j+1])
{
temp=t[j];
t[j]=t[j+1];
t[j+1]=temp;
} } }
for(i=0;i<n+2;i++)
if(t[i]==h)
j=i;k=i;
p=0;
```

```
while(t[j]!=0)
{

atr[p]=t[j];
j--;
p++;
}
atr[p]=t[j];
for(p=k+1;p<n+2;p++,k++)
atr[p]=t[k+1];
for(j=0;j<n+1;j++)
{
if(atr[j]>atr[j+1])
d[j]=atr[j]-atr[j+1];
else
d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("\nAverage header movements:%f",(float)sum/n);
}
```

.

INPUT

enter the no of tracks to be traversed 9
enter the position of head 50
enter the tracks
65
78
23
44
65
76
78
89
75

Expected Output:

Average header movements:-116849760.000000

C-SCAN DISK SCHEDULING ALGORITHM

```
#include<stdio.h>
int main()
{
int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
printf("enter the no of tracks to be traversed ");
scanf("%d",&n);
printf("enter the position of head");
scanf("%d",&h);
t[0]=0;
t[1]=h;
printf("enter total tracks ");
scanf("%d",&tot);
t[2]=tot-1;
printf("enter the tracks ");
for(i=3;i<=n+2;i++)
scanf("%d",&t[i]);
for(i=0;i<=n+2;i++)
for(j=0;j<=(n+2)-i-1;j++)
if(t[j]>t[j+1])
{
temp=t[j];
t[j]=t[j+1];
t[j+1]=temp;
}
for(i=0;i<=n+2;i++)
{
if(t[i]==h)
j=i;
break;
}
p=0;
while(t[j]!=tot-1)
{
atr[p]=t[j];
j++;
p++;
}
atr[p]=t[j];
p++;
i=0;
while(p!=(n+3) && t[i]!=t[h])
{
atr[p]=t[i];
i++;
```

```
p++;
}
for(j=0;j<n+2;j++)
{
if(atr[j]>atr[j+1])
d[j]=atr[j]-atr[j+1];
else
d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("total header movements%d\n",sum);
printf("avg is %f",(float)sum/n);
}
```

INPUT:

enter the no of tracks to be traveresed10
enter the position of head100
enter total tracks10
enter the tracks55 58 60 70 18 90 15 01 84 164

Expected Output:

total header movements12590526avg is 1259052.625000

OUTPUT: