DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS

LAB MANUAL

(R22A0587)

B. TECH CSE

(II YEAR – II SEM)

R22 REGULATION
(2024-25)

Name : _____

Roll no:_____

Section:_____

Year :_____

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
Recognized under 2(f) and 12 (B) of UGC ACT 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 AND ENGINEERING

Vision

To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students in to competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

- To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address thechallenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

- To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

- To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

- To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science andEngineering, the graduates will have the following Program Specific Outcomes:

1.Fundamentals and critical knowledge of the Computer System:- Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2.The comprehensive and Applicative knowledge of Software Development: Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3.Applications of Computing Domain & Research: Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (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

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who 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 labwith the 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 system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note 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 should be 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 workinghours will be 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**

INDEX

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

INDEX

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 endthe 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

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 canprint those files ina new file or display the file using less command.

```
cat file1.txt file2.txt | less
```

To concatenate 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 directorycontents:

```
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 bedifferent.

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 seteach 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.

ps: List the current running processes.
Output:
PID TTY TIME CMD
2540 pts/1 00:00:00 bash

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

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 -lIt 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 usedunder 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.

4 kartik/thakral.png
4 kartik/thakral
24 kartik.png

/*so with -a option used all the files (under directory kartik) disk usage info is displayed alongwiththe 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 filefromone 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> dir200 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 theactivities on your remotely connected machine.

C:>telnet amrood.comTrying...
Connected to amrood.com.Escape character is '^]'. login: amroodamrood'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 eitherlocal 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

lines to 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 last10lines 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 usesdelimiters 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 setcolumn 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 theresults 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 directoryororiginal 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

onsomedestination 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 a directory
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

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.

FCFS CPU 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 thesame 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 twojobs 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;
   clrscr();
   printf("\nEnter the number of processes -- ");
   scanf("%d", &n);
   for(i=0;i<n;i++)
   {
        printf("\nEnter Burst Time for Process %d -- ", 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);
        getch();
   }
}
```

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

a)SJF CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<conio.h>
main()
{
     int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
     float wtavg, tatavg;
     clrscr();
     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;
               }
          np=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);
          getch();
}
```

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
```

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;

clrscr();

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 --   %f ",awt/n);

printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\t
TURNAROUND 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]);

getch();
}
```

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

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

OUTPUT

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;

   clrscr();
   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);getch();
}
```

INPUT

Enter the number of processes –5
Enter 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

OUTPUT

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 process requests 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>
#include<conio.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:

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, each belonging 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

```
int tph, philname[20], status[20], howhung, hu[20], cho;

main()
{
     int i;

     clrscr();

     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)
     {
```

```
}
else
{
     printf("\nAll are hungry..\nDead lock stage will occur");
     printf("\nExiting..");

     for(i=0;i<howhung;i++)
     {
          printf("Enter philosopher %d position: ",(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++)
     {
     }
}


two()
{
     int i, j, s=0, t, r, x;

     printf("\nP %d is granted to eat", philname[hu[pos]]);
     for(x=pos;x<howhung;x++)
          printf("\nP %d is waiting", philname[hu[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: 5
How many are hungry : 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5

OUTPUT

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>

void 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 \t3. 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;
}     }     }
```

OUTPUT

WEEK 6

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

Algorithm:

Create two processes, one is fifo server_two way and another one is fifo client_two way.

Algorithm for fifo server_two way :

1. step 1:Start
2. step 2: Creates a named pipe (using library function mkfifo())with name —fifo_two way‖ in /tmp directory, if not created.
3. step 3: Opens the named pipe for read and write purposes.
4. step 4: Here, created FIFO with permissions of read and write for Owner. Read for Group and no permissions for Others.
5. step 5: Waits infinitely for a message from the client.
6. 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.
7. step 7:stop.

Algorithm for client :

1. Step 1: start
2. Step 2: Opens the named pipe for read and write purposes.
3. Step 3: Accepts string from the user.
4. 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.
5. 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.
6. Step 6: Repeats infinitely until the user enters the string —end‖.
7. 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[last]
```

```
        str[last] = temp;
        first++;
        last--;
}

return;
}
```

OUTPUT:

```
/* 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:

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 =0 Retrieve the next message on the queue, regardless of its mtype. PositiveGet the next message with an mtype equal to the specifiedmsgtyp.

Negative Retrieve the first message on the queue whose mtype fieldis less than or equal 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 = 1234 Step 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);
}
```

Program:

```
//IPC_msgq_rcv.c

#include <sys/types.h> #include <sys/ipc.h>
#include <sys/msg.h> #include <stdio.h> #include <stdlib.h>
#define MAXSIZE128

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);
```

OUTPUT:

WEEK 7

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

PROGRAM:

```
#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("SharedMemoryID=%u",shmid);
if (fork() == 0)
{/* Child */
/* Attach to shared memory and print the pointer */
shared = shmat(shmid, (void *)0, 0);
printf("Child pointer %ls\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 %ls\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:

WEEK 8

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> #include<conio.h>main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;clrscr(); 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(k==f)
{


}

if(m[k]==rs[i])
break;
```

```
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

LRU PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h> #include<conio.h> #define high 37 void main()
{
int fframe[10],used[10],index;
int count,n1,k,nf,np=0,page[high],tmp; int flag=0,pf=0;
clrscr();
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); getch();
}
```

OUTPUT:

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 main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2,
flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

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

   printf("Enter page reference string: ");

   for(i = 0; i < no_of_pages; ++i){
      scanf("%d", &pages[i]);
   }

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

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

      for(j = 0; j < no_of_frames; ++j){
         if(frames[j] == pages[i]){
               flag1 = flag2 = 1;
               break;
           }
      }

      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
               faults++;
               frames[j] = pages[i];
               flag2 = 1;
               break;
            }
         }
      }
```

```
    if(flag2 == 0){
            flag3 =0;

       for(j = 0; j < no_of_frames; ++j){
            temp[j] = -1;

            for(k = i + 1; k < no_of_pages; ++k){
                    if(frames[j] == pages[k]){
                            temp[j] = k;
                            break;
                    }
            }
       }

       for(j = 0; j < no_of_frames; ++j){
            if(temp[j] == -1){
                    pos = j;
                    flag3 = 1;
                    break;
            }
       }

       if(flag3 ==0){
            max = temp[0];
            pos = 0;

            for(j = 1; j < no_of_frames; ++j){
                    if(temp[j] > max){
                            max = temp[j];
                            pos = j;
                    }
            }
       }

                    frames[pos] = pages[i];
                    faults++;
    }

    printf("\n");

    for(j = 0; j < no_of_frames; ++j){
       printf("%d\t", frames[j]);
    }
}

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

```
    return 0;
}
```

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

WEEK 9

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 file
Step6: print number of links
Step 7:print last time access
Step 8: Print Read,write and execute permissions
Step 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))
{

}
else
{

}
 printf("is a Directory file\n");
printf("is Regular file\n");
printf("******File Properties********\n"); printf("Inode Number:%d\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:%d\n",a.st_size); printf("Blocks Allocated:%d\n",a.st_blocks);
printf("Last modification time %s\n",ctime(&a.st_atime));
}
}
```

OUTPUT:

WEEK 10

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 disk scheduling. 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 disk arm 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 main()
{
int t[20], n, I, j, tohm[20], tot=0;float avhm; clrscr();
printf(“enter the no.of tracks”); scanf(“%d”,&n);
printf(“enter the tracks to be traversed”); for(i=2;i<n+2;i++)
scanf(“%d”,&t*i+); for(i=1;i<n+1;i++)
{
tohm[i]=t[i+1]-t[i]; if(tohm[i]<0) tohm[i]=tohm[i]*(-1);
}
for(i=1;i<n+1;i++)
tot+=tohm[i]; avhm=(float)tot/n;
printf(“Tracks traversed\tDifference between tracks\n”); for(i=1;i<n+1;i++)
printf(“%d\t\t\t%d\n”,t*i+,tohm*i+); printf("\nAverage header
movements:%f",avhm); getch();
}
```

INPUT

```
Enter no.of tracks:9
Enter track position:55  58  60  70  18  90  150  160  184
```

OUTPUT

SCAN DISK SCHEDULING ALGORITHM

```
#include<conio.h> #include<stdio.h> int main()
{
int i,j,sum=0,n; int d[20];
int disk; //loc of head int temp,max;
int dloc; //loc of disk in array clrscr();
printf("enter number of location\t"); scanf("%d",&n);
printf("enter position of head\t"); scanf("%d",&disk);
printf("enter elements of disk queue\n");
for(i=0;i<n;i++)
{
scanf("%d",&d[i]);
}
d[n]=disk; n=n+1;
for(i=0;i<n;i++) // sorting disk locations
{
for(j=i;j<n;j++)
{
if(d[i]>d[j])
{
temp=d[i]; d[i]=d[j]; d[j]=temp;
}
}
}
max=d[n];
for(i=0;i<n;i++) // to find loc of disc in array
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
printf("%d -->",d[i]);
}
printf("0 -->"); for(i=dloc+1;i<n;i++)
{
printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum); getch();
return 0;
}
```

OUTPUT:

C-SCAN DISK SCHEDULING ALGORITHM

```
#include<stdio.h> int main()
{
int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],
temp1=0,temp2=0;
float avg;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the initial head position\n");
scanf("%d",&head);
printf("Enter the size of queue request\n");
scanf("%d",&n);
printf("Enter the queue of disk positions to be read\n");
for(i=1;i<=n;i++)
{
scanf("%d",&temp);
if(temp>=head)
{

}
else
{

}
}
queue1[temp1]=temp;
temp1++;
queue2[temp2]=temp;
temp2++;
for(i=0;i<temp1-1;i++)
{
for(j=i+1;j<temp1;j++)
{
if(queue1[i]>queue1[j])
{
temp=queue1[i]; queue1[i]=queue1[j]; queue1[j]=temp;
}
```

```
}
}
for(i=0;i<temp2-1;i++)
{
for(j=i+1;j<temp2;j++)
{
if(queue2[i]>queue2[j])
{
temp=queue2[i]; queue2[i]=queue2[j]; queue2[j]=temp;
}
}
}
for(i=1,j=0;j<temp1;i++,j++) queue[i]=queue1[j]; queue[i]=max; queue[i+1]=0;
for(i=temp1+3,j=0;j<temp2;i++,j++) queue[i]=queue2[j]; queue[0]=head;
for(j=0;j<=n+1;j++)
{
diff=abs(queue[j+1]-queue[j]); seek+=diff;
printf("Disk head moves from %d to %d with
seek          %d\n",queue[j],queue[j+1],diff);
}
printf("Total seek time is %d\n",seek); avg=seek/(float)n;
printf("Average seek time is %f\n",avg); return 0;
}
```

OUTPUT