JEPPIAAR INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

Self-Belief | Self Discipline | Self Respect

Kunnam, Sunguvarchatram, Sriperumbudur-631604

CS3461 – OPERATING SYSTEMS LABORATORY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NAME :

REG NO. :

YEAR :

SEMESTER :

BRANCH :

JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self Belief || Self Discipline || Self Respect"

Sunguvarchatram, Sriperumbudur, Chennai - 631604

BONAFIDE CERTIFICATE

This is a certified Bonafide Record Work of Mr./Ms._____

Register No._____submitted for the Anna University Practical Examination held on_____in **CS3461-OPERATING SYSTEMS** Laboratory during the year **2024-2025.**

Signature of the Lab In-charge Head of the Department

Internal Examiner **External Examiner**

Date:_____

INSTITUTE VISION

Jeppiaar Institute of Technology aspires to provide technical education in futuristic technologies with theperspective of innovative, industrial and social application for the betterment of humanity.

INSTITUTE MISSION

IM1: To produce competent and disciplined high-quality professionals with the practical skills necessaryto excel as innovative professionals and entrepreneurs for the benefit of the society.

IM2: To improve the quality of education through excellence in teaching and learning, research, leadership and by promoting the principles of scientific analysis, and creative thinking.

IM3: To provide excellent infrastructure, serene and stimulating environment that is most conducive tolearning.

IM4: To strive for productive partnership between the Industry and the Institute for research anddevelopment in the emerging fields and creating opportunities for employability.

IM5: To serve the global community by instilling ethics, values and life skills among the studentsneeded to enrich their lives.

DEPARTMENT VISION

To impart futuristic technological education, innovation and collaborative research in the field of Computer Science Engineering and develop Quality Professional for the improvement of society and industry.

DEPARTMENT MISSION

DM1: Devise students as professionally competent and disciplined engineers for the benefit of the country's development.

DM2: Produce excellent to adopt latest technologies, industry-institute interaction and encouraging research activities.

DM3: Provide multidisciplinary technical skills to pursue search activities, higher studies, entrepreneurship and perpetual learning.

DM4: Enrich students with professional integrity and ethical standards to handle social challenges successfully in their life.

PROGRAM EDUCATIONAL OBJECTIVES(PEO'S)

Graduates can

PEO1 To support students with substantial knowledge for developing and resolving mathematical, scientific and engineering problems.

PEO2 To provide students with adequate training and opportunities to work as a collaborator with informative and administrative qualities.

PEO3 To motivate students for extensive learning to prepare them for graduate studies, R&D and competitive exams.

PEO4 To cater students with industrial exposure in an endeavour to succeed in the emerging cutting-edge technologies.

PEO5 To shape students with principled values and to follow the code of ethics in social and professional life.

PROGRAM SPECIFIC OUTCOMES(PSO'S)

The Students will be able to

PSO1 Analyse, design, and implement quality software by applying fundamental and programming concepts of Computer Science and Engineering.

PSO2 Design and develop solutions for scientific, business and real time applications through analytical, logical and problems solving skills.

PSO3 Provide efficient solutions for industrial and society needs with acquired knowledge through emerging technical skills.

PROGRAM OUTCOMES

Engineering Graduates will be able to:

1. **Engineering knowledge:** (K3) Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** (K4) 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:** (K4) Design solutions forcomplex engineering problems and design system components or processes that meet the specified needswith appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: (K5) Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of theinformation to provide valid conclusions.
5. **Modern tool usage:** (K3, K5, K6) Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: (A3) 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:** (A2) 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:** (A3) Apply ethical principles and commit to professional ethics and responsibilities and normsof the engineering practice.
9. **Individual and team work:** (A3) Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** (A3) 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:** (A3) Demonstrate knowledge and understanding of the engineeringand management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** (A2) 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.

ANNA UNIVERSITY SYLLABUS

CS3461-OPERATING SYSTEMS LABORATORY L T P C

0 0 3 1.5

1. Installation of Operating system : Windows/ Linux
2. Illustrate UNIX commands and Shell Programming
3. Process Management using System Calls : Fork, Exec, Getpid, Exit, Wait, Close
4. Write C programs to implement the various CPU Scheduling Algorithms
5. Illustrate the inter process communication strategy
6. Implement mutual exclusion by Semaphores
7. Write a C program to avoid Deadlock using Banker's Algorithm
8. Write a C program to Implement Deadlock Detection Algorithm
9. Write C program to implement Threading
10. Implement the paging Technique using C program
11. Write C programs to implement the following Memory Allocation Methods

    a. First Fit b. Worst Fit c. Best Fit

12. Write C programs to implement the various Page Replacement Algorithms
13. Write C programs to Implement the various File Organization Techniques
14. Implement the following File Allocation Strategies using C programs

    a. Sequential b. Indexed c. Linked

15. Write C programs for the implementation of various disk scheduling algorithms
16. Install any guest operating system like Linux using VMware.

INDEX

Ex.No:1 **Installation of Windows Operating System**
Date :

Aim: To study the installation of Windows operating system.

Procedure:

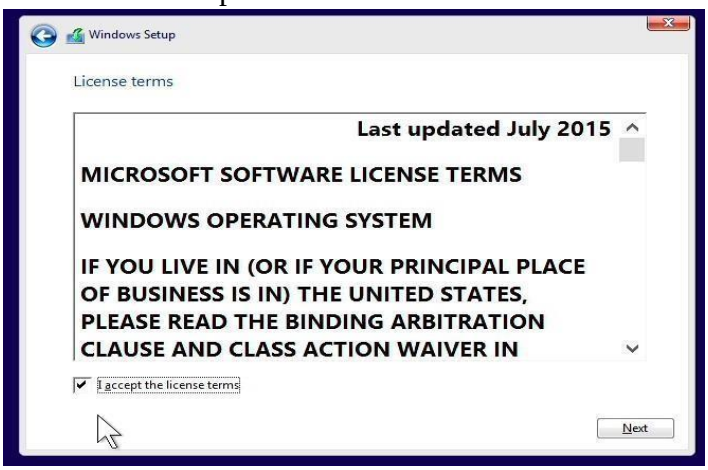
Follow the steps below to proceed with the Windows 10 installation.

1. Select the following Language, Time, and Keyboard Layout and select "Next".

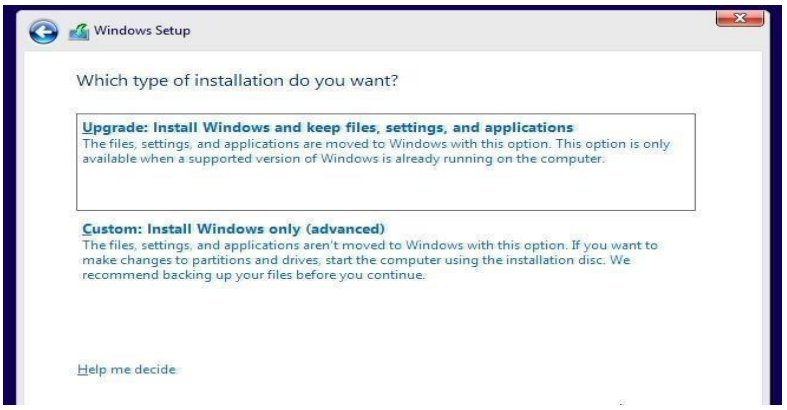
2. Select "Install now" option.

3. Enter the product key to activate Windows and press "Next", or press "Skip" to enter a valid product key later.

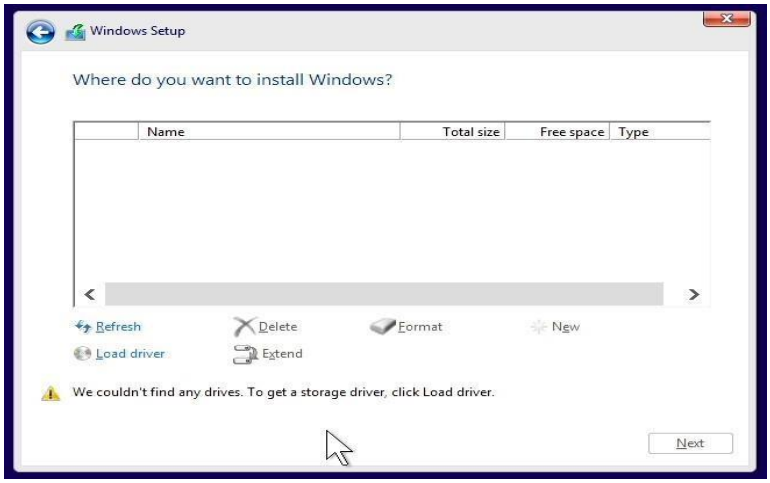
4. Check the box next to "I accept the license terms" and select "Next".



5. Choose the type of installation to perform.
   - To upgrade to Windows 10 from an earlier version of Windows, select the "Upgrade" option below.
   - To perform a clean OS installation, select "Custom: Install Windows only (advanced)" option below. For instructional purposes, this option was selected below.



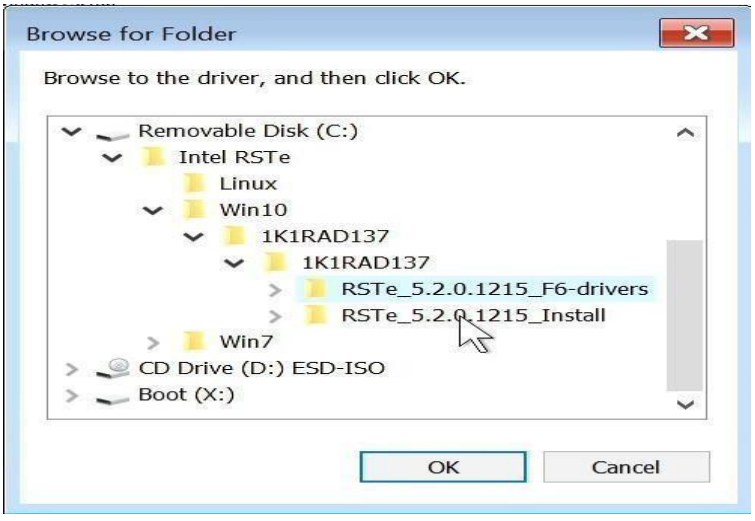
6. Select the "Load driver" option to load the appropriate driver for the storage device.



7. Make sure to load the appropriate storage device driver onto a CD, DVD, or USB flash drive before completing the next step.

   - For storages devices attached to the Intel storage controller, load the Intel Rapid Storage Technology enterprise (RSTe) driver.

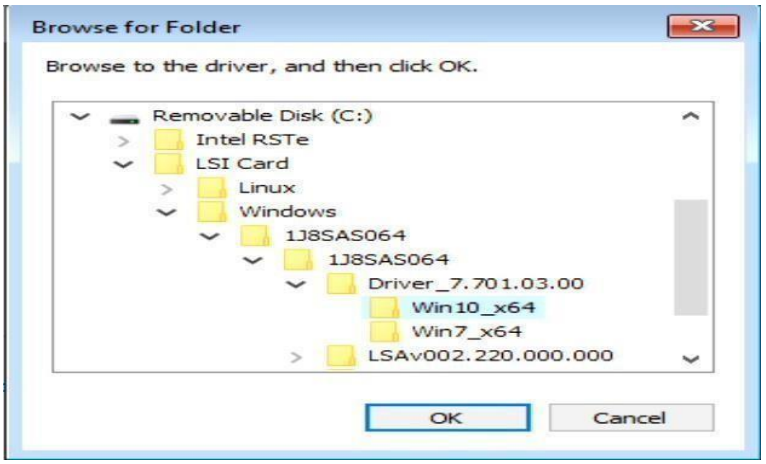
     - Select “Browse”, and browse to the CD, DVD, or USB flash drive to where the storage device driver is located and select “OK”.



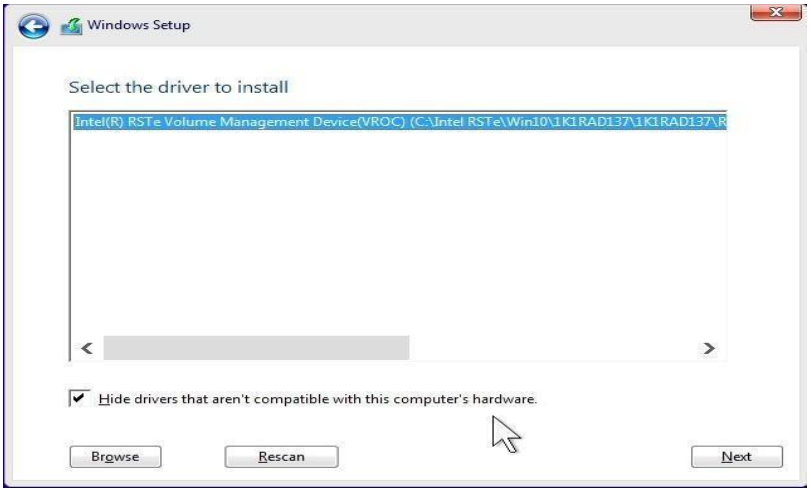
-

- For storages devices attached to the Broadcom controller, load the Broadcom storage driver.

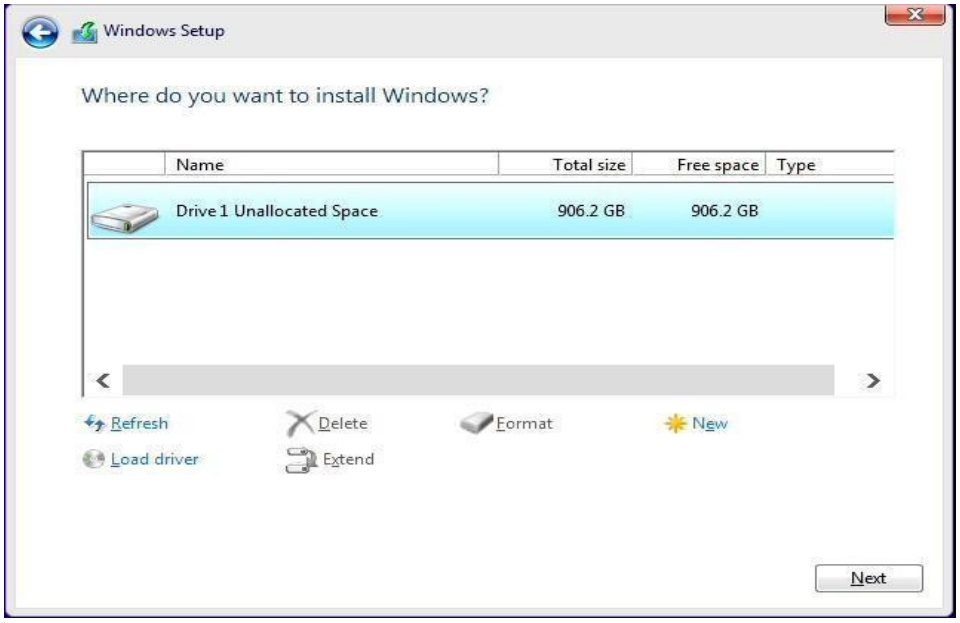
- Select "Browse", browse to the CD, DVD, or USB flash drive to where the storage device driver is located and select "OK".



8. Select the driver to install.

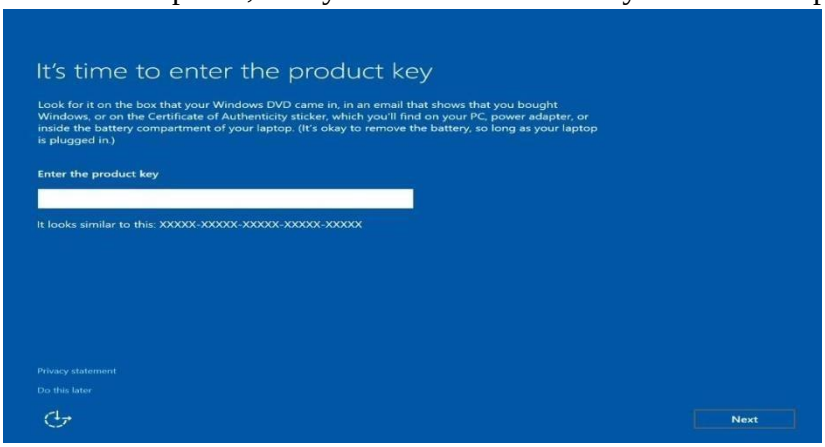


9. Select the drive to install Windows and select "Next" at the bottom.



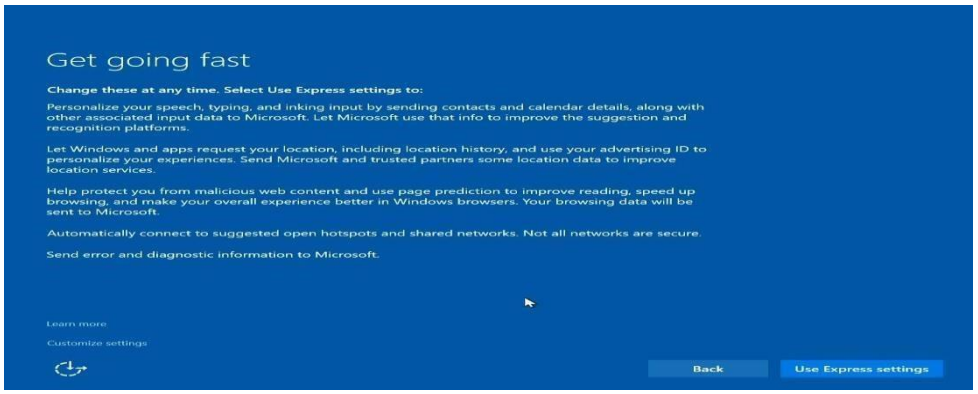
10. Installing Windows screen.

11. After the installation completes, the system will automatically reboot. If no product key

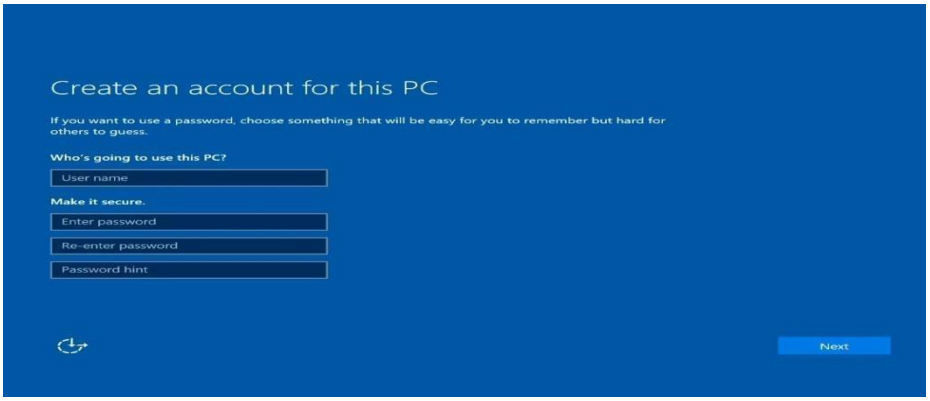


was entered above, then it'll prompt to enter a product key again. Either enter the product key here and select "Next" or select "Do this later" at the bottom left.

12. Select "Use Express Settings" at the bottom right to use the default settings or select "Customize settings" at the bottom left. For instructional purposes, "Use Express settings" was selected.



13. Create an account by typing a "User name". Optionally, type a password to make it secure.



14. Windows 10 desktop screen.

15. At this point, download and install the appropriate device drivers, i.e. Intel Chipset, Intel AMT, Ethernet, Graphics, etc.
16. Restart the system after installation.

Result:

Thus the Windows operating system installation has been studied successfully.

Ex.No:2a **Basic Commands in UNIX**
Date :

Aim:

To study and execute UNIX commands.

UNIX Commands

1. **Cat Command**

It is used to create a file, display the contents of a file and concatenating the files.

$cat > filename /for create
$cat filename /for display
$cat file1 file2 >file3 /for concatenate

2. **Date Command**

It is used to display the current date, time, month and year.

$date +%d /display date
$date +%m /display month
$date +%h /display month in words
$date +%y /display year
$date +%R /display the time with hour and mins
$date +%T /display time with hour,mins and sec

1. **Calendar Command**

It is used to display the calendar of the given month and year**.**

$cal year /display the year calendar
$cal month year /display the calendar of given month

2. **Who / who am i Command**

It is used to display the data about all the users, who are currently logged into the system.

$who /display all users
$who am I /display about user

3. **Man command**

It is used to display the description and usage of particular command.

$man command name /display the description of the cmd

4. **Head Command**

It is used to display the text from top of the file content to mentioned line.

$head – (option) filename / display text from top

5. Tail command

It is used to display the text from bottom of the file content to mentioned line.

$tail – (option) filename / display text from bottom

6. Wc Command

It is used to count the number of lines, words and characters in the given file.

$wc filename /display no of lines, words and characters

7. Copy Command

It is used to copy the content from one file to another file.

$cp source destination /copy content from source to destination.

8. Move Command

It is used to rename the file.

$mv filename1 filename2 /for renaming

9. Compare Command

It is used to compare two sorted files line by line.

$cmp file1 file2 /to compare file1& 2

10. Echo Command

It is used to display whatever **message** we want to display on the screen.

$echo message /display the given msg

11. Read Command

It is used to get the user input from keyboard.

$read variable name /get input from keyboard

12. Write Command

It is used to send message to any logged in users. It's a two way communications.

$write username /send msg

13. Link Command

It is used to link the content from one file to another file. It's same as copy command

$ln source destination

Or

$link file1 file2 /Link the two files

14. Directory Commands

It is used to making, changing and removing directories.

$mkdir dirname /to create directory

$cd dirname /to change working directory
$rmdir dirname /to remove the directory
$cd .. /to close the working directory

15. List Command

It is used to display list of files in the current working directory.

$ls –(option)

Options entries format. order order
a- List all directory
l- List files in long
r- List files in reverse
t- List files in recently used
s- List no of blocks(memory) used by the file

16. Remove Command

It is used to remove files from a directory.

$rm filename
Or
$rm –(option) filename /remove the file

Options

i- Ask user whether he wants to delete the file or not
r- Delete entries / entire content of the file recursively
f- Forcing to delete

17. Pwd Command

It is used to display current working directory.

$pwd /display current directory

18. Print Command

It is used to print the content of file.

$lp filename /print the file

19. Sort Command

It is used to sort the content in the file.

$sort filename /sort the content

20. Tty Command

It is used to know the terminal name that we are using.

$tty /display the terminal name

21. Bc Command

It is used as an online calculator.

$bc /open an online calculator

22. Message Command

It is used to avoid message from other users.

$mesg /to avoid the msg

23. Mail Command

It is used as a simple email utility available on UNIX system

$mail username /sends mail

24. Wall Command

It is used to send message to all users, those who are currently logged in.

$wall message /send msg to all users

25. News Command

It is used to permit users to read messages published by the system administrator.

$news /allow to read admin msg.

26. Grep Command (Global Regular Expression and Print)

It is used to search and print specified patterns from a file.

$grep text filename /search and print given text from file

27. Cut Command

It is used to select specified field from a line of text.

$cut –c(option) filename /cut a text

28. Paste Command

It is used to paste back the cut characters.

$paste filename /paste back the text

29. Common Command

It is used to compare two sorted files and compares each line of the first file with its corresponding line in the second file.

$comm file1 file2 /to compare the files

30. Difference Command

It is used to display file differences.

$diff file1 file2 /find the difference from the two identical files

31. Finger Command

It is used to gather and display the information about users, which includes login name, realname, home directory etc...

$finger username /display user details.

32. Password Command

It is used to change the password.

$passwd /to change password

33. Nl Command

It is used to add line number to file content.

$nl filename /add no to the file content

34. Which Command

It is used to report the path to the command or the shell alias in use

$which /to display the path

37. Clear Command

It is used to clear the screen.

$tput clear /to clear the screen

38. Reply Command

It is used to send reply to the specified user.

$reply username /to send reply

39. More Command

It is used to scroll your screen when your file content is too large.

$more filename /to scroll the screen

40. Compress Command

It is used to compress the file and save it as file.z.

$compress filename /to compress

Output:

FILE COMMANDS

1. Cat Command

```
[examuser1@linux ~]$ cat > file1 hi
Welcome
This is my first unix file
Thank You

[examuser1@linux ~]$ cat >file2
This is my second File
Thank You

[examuser1@linux ~]$ cat file1
hi
Welcome
This is my first unix file
Thank You
```

```
[examuser1@linux ~]$ cat file2
This is my second File
Thank You

[examuser1@linux ~]$ cat file1 file2 > file3

[examuser1@linux ~]$ cat file3
hi
Welcome
This is my first unix file
Thank You
This is my second File
Thank You
```

2. Copy Command

```
[examuser1@linux ~]$ cp file1 file4

[examuser1@linux ~]$ cat file4
hi
Welcome
This is my first unix file
Thank You
```

3. Move Command

```
[examuser1@linux ~]$ mv file4 file5

[examuser1@linux ~]$ cat file5
hi
Welcome
This is my first unix file
Thank You
```

4. Remove Command

```
[examuser1@linux ~]$ rm file5
[examuser1@linux ~]$ cat file5
cat: file5: No such file or directory
```

5. WC Command

```
[examuser1@linux ~]$ wc file3
     6      17      81 file3
```

WORKING WITH DIRECTORIES

6. Creating A Directory

[examuser1@linux ~]$ mkdir unix

7. Changing the working Directory

[examuser1@linux ~]$ cd unix

8. Current working Directory

[exam1@redhat unix]$ pwd
/home/exam1/unix

[exam1@redhat unix]$ cd ..

9. The Path

[examuser1@linux ~]$ echo $HOME
/home/exam1

10. Moving files within directories

[examuser1@linux ~]$ mv file4 unix
mv: cannot stat `file4': No such file or directory

[examuser1@linux ~]$ cat > file6
hello unix
world

[examuser1@linux ~]$ mv file6 unix

[examuser1@linux ~]$ cd unix

[exam1@redhat unix]$ cat file6
hello unix
world

11. Removing Directory

```
[examuser1@linux ~]$ cd unix [exam1@redhat
unix]$ rm file6 [exam1@redhat unix]$ cd ..
[examuser1@linux ~]$ rmdir unix
```

CALENDAR AND DATE COMMANDS

12. Calendar command - Year

```
[examuser1@linux ~]$ cal 2010
```

2010

January

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

February

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27

March

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

April

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

May

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

June

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

July

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

August

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

September

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

October

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

November

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

December

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	31

13. Calendar command – Month of the Year

```
[examuser1@linux ~]$ cal 5 2012

     May 2012
Su  Mo  Tu  We  Th  Fr  Sa
          1     2     3    4     5
 6    7    8     9     10 11  12
13  14   15   16   17  18  19
20   21 22    23   24 25  26
27   28  29   30  31
```

14. Date Commands

```
[examuser1@linux ~]$ date
Tue Jan  1 00:26:15 IST 2002
[examuser1@linux ~]$ date +%d
01
[examuser1@linux ~]$ date +%m
01
[examuser1@linux ~]$ date +%h
Jan
[examuser1@linux ~]$ date +%y
02
[examuser1@linux ~]$ date +%R
00:26
[examuser1@linux ~]$ date +%T
00:27:03
```

PIPES

15. Pipes in who command

```
[examuser1@linux ~]$ who | wc -l
     1
```

16. Longway pipeline

```
[examuser1@linux ~]$ ls | sort | wc -l
     5
```

17. Capturing output while using pipes : tee

```
[examuser1@linux ~]$ cat file3|wc|tee file4
     6      17      81
```

```
[examuser1@linux ~]$  cat file4
    6     17     81
```

OTHER BASIC COMMANDS

18. Who

```
[examuser1@linux ~]$  who
exam1    pts/1        Jan  1 00:02 (10.0.5.18)
```

19. who am i Command

```
[examuser1@linux ~]$  who am I
exam1    pts/1        Jan  1 00:02 (10.0.5.18)
```

20. Man command

```
[examuser1@linux ~]$  man cat
man cat
CAT(1)                          FSF                          CAT(1)
NAME
     cat - concatenate files and print on the standard output

SYNOPSIS
     cat [OPTION] [FILE]...

DESCRIPTION
     Concatenate FILE(s), or standard input, to standard output.

     -A, --show-all
          equivalent to -vET

     -b, --number-nonblank
number nonblank output lines

     -e     equivalent to -vE

     -E, --show-ends
          display $ at end of each line

     -n, --number
          number all output lines
```

21. List Command

```
[examuser1@linux ~]$ ls -t
postfix.l file7 file4 file3 file2 file1 new
```

22. Print Command

```
[examuser1@linux ~]$ lp file3 lp:
error - scheduler not responding!
```

23. Sort Command

```
[examuser1@linux ~]$ sort file3
hi
Thank You
Thank You
This is my first unix file
This is my second File
Welcome
```

24. Password Command

```
[examuser1@linux ~]$ passwd
Changing password for user exam1.
Changing password for exam1
(current) UNIX password:
You must wait longer to change your password
passwd: Authentication token manipulation error
```

25. Which Command

```
[examuser1@linux ~]$ which
Usage: /usr/bin/which [options] [--] programname [...]
Options: --version, -[vV] Print version and exit successfully.
    --help,         Print this help and exit successfully.
    --skip-dot      Skip directories in PATH that start with a dot.
    --skip-tilde    Skip directories in PATH that start with a tilde.
    --show-dot      Don't expand a dot to current directory in output.
    --show-tilde    Output a tilde for HOME directory for non-root.
    --tty-only      Stop processing options on the right if not on tty.
--all, -a       Print all matches in PATH, not just the first       --read-alias,
-i Read list of aliases from stdin.
    --skip-alias    Ignore option --read-alias; don't read stdin.
    --read-functions Read shell functions from stdin.
    --skip-functions Ignore option --read-functions; don't read stdin.
```

Result:

Thus the basic UNIX commands have been studied and executed.

Ex: No: 2b **Shell Programming**

Date :

i. Biggest of Three Numbers

Aim:

To write a shell program for finding the biggest among 3 numbers

Algorithm:

1. Get the 3 values
2. Check if 'a' is greater than 'b'
3. If step 2 is true, go to step 4. else, go to step 7
4. Check if 'a' is bigger than 'c'
5. If step 4 is true, print "a is big". else print "c is big"
6. Go to step 9
7. Check if 'b' is greater than 'c'
8. If step 7 is true, print "b is big". Else, print "c is big"
9. End of program

Program:

```
echo "enter a" read a
echo "enter b" read b
echo "enter c" read c if [
$a -gt $b -a $a -gt $c ]
then echo "a is big" elif [
$b -gt $c ] then echo "b is
big"
else echo "c is big"
fi
```

Output:

```
[examuser1@linux ~]$ sh biggest.sh
enter a
5 enter
b 4
enter c
2 a is
big
Result:
```

Result:

Thus the shell program for finding the biggest among 3 numbers, has been written and executed successfully.

ii. Checking Odd or Even Aim:

To write a shell program to find the given number is odd or even **Algorithm:**

1. Get a number from the user, say num.
2. check if (num % 2) = = 0
   2.1 Display the given number is even otherwise
   2.2 Display the given number is odd
3. Stop the program

Program:

```
echo "enter any number" read
num
if [ `expr $num % 2` -eq 0 ] then
echo number is even else
echo number is odd
fi
```

Output:

```
[examuser1@linux ~]$ sh oddeven.sh enter
any number
5
number is odd
[examuser1@linux ~]$ sh oddeven.sh enter
any number
4
number is even
```

Result:

Thus the shell program for finding whether the given number is odd or even has been written and executed successfully.

iii. Finding Factorial Aim:

To write a shell program to find factorial of given number.

Algorithm:

1. Read a number say n.
2. Initialize i=1,f=1
3. Repeat until I is leass than n.
   3.1.f=f*i
3. 2.i=i+1
4. Display factorial (f) of n
5. Stop

Program:

```
echo "enter number"
read n
i=1 f=1
while [ $i -le $n ]
do f=`expr $f \*
$i` i=`expr $i +
1` done
echo "Factorial is.. $f"
```

Output:

```
[examuser1@linux ~]$ sh fact.sh enter
number
5
Factorial is.. 120
```

Result:

Thus the shell program to find factorial of given number has been written and executed successfully.

iv. Arithmetic Operations Aim:

To write a shell program for implementing arithmetic operations.

Algorithm:

1. Display menu to user.
   Say, 1.Add 2. Sub 3. Mul 4. Div 5.exit
2. Prompt user to enter 2 values (say a, b) and enter choice of operation.
3. Using switch case statement, use choice value for processing output accordingly.
   3.1. If choice=1, return sum (a+b)
   3.2. If choice=2, return difference (a-b)
   3.3. If choice=3, return product (a*b)
   3.4. If choice=4, return quotient (a/b)

4Display result.

.

Program:

```
echo "Enter two numbers"
read a b
echo "1.Add 2.Sub 3.Mul 4.Div 5.Exit"
read op case $op in 1)c=`expr $a +
$b`;;
2)c=`expr $a - $b`;;
3)c=`expr $a \* $b`;;
4)c=`expr $a / $b`;;
5)exit
esac echo
$c
```

Output:

```
[examuser1@linux ~]$ sh case.sh
Enter two numbers
5 4
1.Add 2.Sub 3.Mul 4.Div 5.Exit
1
9
[examuser1@linux ~]$ sh case.sh
Enter two numbers
5 4
1.Add 2.Sub 3.Mul 4.Div 5.Exit
2
1
[examuser1@linux ~]$ sh case.sh
Enter two numbers
5 4
```

```
1.Add 2.Sub 3.Mul 4.Div 5.Exit
3
20
[examuser1@linux ~]$ sh case.sh
Enter two numbers
8 4
1.Add 2.Sub 3.Mul 4.Div 5.Exit
4
2
```

Result:

Thus the shell program for implementing arithmetic operations has been written and executed successfully.

EX. NO. 3a Implementation of Process Management using fork and getpid system calls
Date :

Aim:

To create a new child process using fork system call and implement getpid system call.

Algorithm

1. Declare a variable *x* to be shared by both child and parent.
2. Create a child process using fork system call.
3. If return value is -1 then
    a. Print "Process creation unsuccessfull"
    b. Terminate using exit system call.
4. If return value is 0 then
    a. Print "Child process"
    b. Print process id of the child using getpid system call
    c. Print value of *x*
    d. Print process id of the parent using getppid system call
5. Otherwise
    a. Print "Parent process"
    b. Print process id of the parent using getpid system call
    c. Print value of *x*
    d. Print process id of the shell using getppid system call.
6. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
 main()
{
 pid_t pid; int x=5; pid=fork();
x++;
if(pid<0)
 {
printf("process creation error");
exit(-1);
 }
else if(pid==0)
{
printf("child process:");
printf("\n process id is %d",getpid());
printf("\n value of x is %d",x);
printf("\n process id of parent is%d\n",getppid());
} else { printf("\n .
```

Output:

```
[examuser1@linux ~]$ cc fork.c
[examuser1@linux ~]$ ./a.out
child process:

 process id is 7353
value of x is 6
 process id of parent is7352

 parent process:

 process id is 7352
value of x is 6
 process id of shell is 7122
```

Result

Thus a child process is created with copy of its parent's address space.

EX. NO. 3b Implementation of Process Management using wait system call
Date :

Aim:

To block a parent process until child completes using wait system call.

Algorithm:

1. Create a child process using fork system call.
2. If return value is -1 then
   a. Print "Process creation unsuccessfull" 3.
   Terminate using exit system call.
4. If return value is > 0 then
   a. Suspend parent process until child completes using wait system call
   b. Print "Parent starts"
   c. Print even numbers from 0–10
   d. Print "Parent ends"
5. If return value is 0 then
   a. Print "Child starts"
   b. Print odd numbers from 0–10
   c. Print "Child ends" 6. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h
>
#include<sys/wait.h>
main() { int i,status;
pid_t pid; pid=fork();
if(pid<0) {
printf("\n process creation failure\n");
exit(-1); }
else if(pid>0)
{
wait(NULL);
printf("\n parent starts \n even nos:");
for(i=2;i<=10;i+=2)
printf("%3d",i);
printf("\n parent
ends\n");
}
else if(pid==0)
```

```
{
printf("child starts \n odd nos:");
for(i=1;i<10;i+=2)
printf("%3d",i); printf("\n child
ends\n");
}
}
```

Output:

```
[examuser1@linux ~]$ cc wait.c
[examuser1@linux ~]$ ./a.out

 child starts  odd nos:
1  3  5  7  9  child ends

 parent starts  even nos:
2  4  6  8 10  parent ends
```

Result

Thus using wait system call zombie child processes were avoided.

EX. NO. 3c Implementation of Process Management using exec system call
Date :

Aim:

To load an executable program in a child processes exec system call.

Algorithm:

1. If no. of command line arguments □ 3 then stop.
2. Create a child process using fork system call.
3. If return value is -1 then
    a. Print "Process creation unsuccessfull"
    b. Terminate using exit system call.
4. If return value is > 0 then
    a. Suspend parent process until child completes using wait system call
    b. Print "Child Terminated".
    c. Terminate the parent process.
5. If return value is 0 then
    a. Print "Child starts"
    b. Load the program in the given path into child process using exec system call.
    c. If return value of exec is negative then print the exception and stop.
    d. Terminate the child process.
6. Stop

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
main(int argc,char *argv[])
{ pid_t
pid; int i;
if(argc!=3)
{ printf("\n insufficient arguments load
program"); printf("\n usage:./a.out <path>
<cmd> \n"); exit(-1); }
switch(pid=fork())
{ case -1:
printf("fork
failed"); exit(-1);
case 0:
printf("child process\n"); i=execl(argv[1],argv[2],0);
if(i<0)
{
printf("%s program not loaded using exec system call \n",argv[2]); exit(-1);
```

```
}
default:

wait(NULL);
printf("child terminated \n");
exit(0);
}
}
```

Output:

```
[examuser1@linux ~]$ cc exec.c
[examuser1@linux ~]$ ./a.out /usr/bin/who who
child process
root     :0           Jan 11 22:51 exam1     pts/1
Jan 11 22:58 (10.0.5.25) exam43  pts/4       Jan
11 23:16 (10.0.5.134) exam40   pts/5        Jan
11 23:16 (10.0.5.136) exam31   pts/6        Jan
11 23:18 (10.0.5.138) exam42   pts/7        Jan
11 23:20 (10.0.5.135) exam39   pts/8        Jan
11 23:21 (10.0.5.137) exam34   pts/3        Jan
12 00:02 (10.0.5.146) exam35   pts/9        Jan
12 00:20 (10.0.5.148) exam33   pts/10       Jan
12 00:26 (10.0.5.151) exam38   pts/11       Jan
12 00:32 (10.0.5.156) exam28   pts/12       Jan
12 00:42 (10.0.5.157) exam24   pts/2        Jan
12 00:42 (10.0.5.158) exam36   pts/0        Jan
12 00:45 (10.0.5.149) exam32   pts/13       Jan
12 00:49 (10.0.5.152) child terminated
```

Result

Thus the child process loads a binary executable file into its address space.

EX. NO. 3d. Implementation of Process Management using system calls: open, read, write, close, exit

Date :

Aim:

To implement UNIX I/O system calls open ,read , write,close and exit.

Algorithm:

1. Declare a character buffer *buf* to store 100 bytes.
2. Get the new filename as command line argument.
3. Create a file with the given name using open system call with O_CREAT and O_TRUNC options.
4. Check the file descriptor.
    a) If file creation is unsuccessful, then stop.
5. Get input from the console until user types Ctrl+D
    a) Read 100 bytes (max.) from console and store onto buf using read system call
    b) Write length of *buf* onto file using write system call.
6. Close the file using close system call.

7. Stop

Program:

```
#include <stdlib.h> #include
<string.h> #include <fcntl.h>
main(int argc, char *argv[])
{ int fd, n, len; char
buf[100]; if (argc !=
2)
{
printf("Usage: ./a.out <filename>\n"); exit(- 1);
}
fd = open(argv[1], O_WRONLY|O_CREAT|O_TRUNC, 0644); if(fd <
0) { printf("File creation problem\n");
exit(-1); }
printf("\n The execution of creat & write system calls:\n"); printf("Press Ctrl+D at end
in a new line:\n");

while((n = read(0, buf, sizeof(buf))) > 0)
{
len = strlen(buf); write(fd, buf,
len);
}
close(fd);

        printf("\nFile has been created and contents are written in the file\n");

        fd = open(argv[1], O_RDONLY); if(fd
        == -1)
        {
        printf("%s file does not exist\n", argv[1]); exit(-
        1); } printf("\nExecution of read system call:\n");
        printf("\nOpening  the file %s to
        read\n",argv[1]); printf("Contents of the file %s
        is : \n", argv[1]); while(read(fd, buf, sizeof(buf))
        > 0) printf("%s", buf); close(fd);
        }
```

Output:

```
[examuser1@linux ~]$ cc orw.c
[examuser1@linux ~]$ ./a.out os.txt

 The execution of creat & write system calls:
Press Ctrl+D at end in a new line:

System calls provide interface between a process and the operating system.

File has been created and contents are written in the file
```

Execution of read system call:

Opening the file os.txt to read Contents
of the file os.txt is :

System calls provide interface between a process and the operating system.

Result:

Thus the program for implementing system calls open,read, write,close and exit have been executed successfully.

Ex.No.: 4 Implementation of various CPU Scheduling Algorithms

Ex.No.4a FIRST COME FIRST SERVE SCHEDULING
Date :

Aim:

To write a program in C to implement the FCFS Gantt Chart

Algorithm:

1. Start the program.
2. Get the number of process to be executed
3. Get the process name and its burst time.
4. Calculate the waiting time and turn around time for each process
5. Draw the Gantt chart using the graphics mode.
6. Stop the program

Program:

```
#include<stdio.h>
struct process
{
int btime,wtime,ttime;
}
p[50];
main()
{
   int n,i,j,h,c;
float tot_turn=0.0,tot_wait=0.0,avg_turn=0.0,avg_wait=0.0;
printf("\n\n\t\t\t\tFIRST COME FIRST SERVE
SCHEDULING\n\n");
printf("\t\t\t\t**********************************\n");
printf("Enter the number of process=");
scanf("%d",&n); printf("\n");
for(i=1;i<=n;i++)
{
printf("Enter the burst time %d:",i);
scanf("%d",&p[i].btime);
}
i=1; p[i].wtime=0;
p[i].ttime=p[i].btime;
tot_wait=p[i].wtime;
tot_turn=p[i].ttime;
for(i=2;i<=n;i++)
{

p[i].wtime=p[i-1].wtime+p[i-1].btime;
```

```
p[i].ttime=p[i].wtime+p[i].btime;
tot_wait=tot_wait+p[i].wtime;

tot_turn=tot_turn+p[i].ttime;
}
avg_wait=tot_wait/n;
avg_turn=tot_turn/n;
printf("\nProcess No \tBurst Time\tWaiting Time\tTurn Around
Time");
for(i=1;i<=n;i++)
{
printf("\n%d \t\t\t%d\t\t\t%d
\t\t\t\t%d",i,p[i].btime,p[i].wtime,p[i].ttime);
}
printf("\n\nAverage Waiting Time=%f",avg_wait);
printf("\nAverage Turn Around Time=%f",avg_turn);
printf("\n");
printf("\n\t\t\t\tGANTT CHART");
printf("\n\t\t\t\t**********\n\n");
for(i=1;i<=n;i++)
{
   printf("%d",p[i].wtime);
   for(j=1;j<=p[i].btime;j++)
   printf("_"); }
   for(i=1;i<=n;i++)
{
c=p[i].wtime+p[i].btime;
}
printf("%d",c);
printf("\n\n");
return 0;
}
```

Output:

```
                FIRST COME FIRST SERVE SCHEDULING

                *********************************
Enter the number of process=3

Enter the burst time 1:8
Enter the burst time 2:6
Enter the burst time 3:2

Process No  Burst Time  Waiting Time    Turn Around Time
1           8           0               8
2           6           8               14
3           2           14              16

Average Waiting Time=7.333333
Average Turn Around Time=12.666667

                GANTT CHART
                ***********

0________8______14__16
```

Result:

Thus the program to implement FCFS scheduling algorithm has been written and executed successfully.

Ex.No.4b Shortest Job First Scheduling
Date :

Aim:

To write a program in C to implement the SJF scheduling algorithm.

Algorithm:

1. Start the process.
2. Declare the array size.
3. Get the number of elements to be inserted.
4. Select the process which has shortest burst time will execute first.
5. If two processes have same burst length then FCFS scheduling algorithm used.
6. Make the average waiting length of next process.
7. Start with the first process from its selection as above and let the other process in queue.
8. Calculate the total number of burst time
9. Display the values.
10. Terminate the process.

Program:

```
#include<stdio.h>
main()
{
int i,j,n,t,d,h,tot=0,tt=0,p[20],c[20],a[20];
printf("\n\t\t\t\tSHORTEST JOB FIRST SCHEDULING\n");
printf("\t\t\t\t*****************************\n\n");
printf("Enter the number of process:");
scanf("%d",&n);
printf("\nEnter the %d process\n",n);
for(i=0;i<n;i++)
scanf("%d",&p[i]);
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
if(p[i]>p[j])
{
   t=p[i];
p[i]=p[j];

p[j]=t;
   }
printf("\nSorted Process\n");
for(i=0;i<n;i++)
printf("%d\n",p[i]); c[0]=0; for(i=0;i<n-1;i++) c[i+1]=c[i]+p[i];
for(i=0;i<n;i++) a[i]=c[i]+p[i];
printf("\nP.No \tProcess \tWaiting Time \tTurn Around Time");
for(i=0;i<n;i++)
{
printf("\n%d\t%d\t\t%d\t\t%d",i+1,p[i],c[i],a[i]);
tot=tot+c[i]; tt=tt+a[i]; }
```

```
printf("\n\nAverage Waiting Time %f",((float)tot/n));
printf("\nAverage Turn Around Time %f",((float)tt/n));
printf("\n");
printf("\n\n\t\t\t\t\tGANTT CHART");
printf("\n\n\t\t\t\t\t**********");
printf("\n\n\t\t\t\t");
for(i=0;i<n;i++)
{
printf("%d",c[i]);
for(j=1;j<p[i];j++)
printf("_"); }
for(i=1;i<n;i++)
{
   d=c[i]+p[i];
}
printf("%d",d);
printf("\n\n"); return 0;
}
```

Output:

```
                SHORTEST JOB FIRST SCHEDULING
                *****************************

Enter the number of process:3

Enter the 3 process
9
7
3

Sorted Process
3
7
9

P.No    Process     Waiting Time    Turn Around Time
1       3           0               3
2       7           3               10
3       9           10              19

Average Waiting Time 4.333333
Average Turn Around Time 10.666667


                    GANTT CHART

                    ***********

               0__3_____10_______19
```

Result:

Thus the program to implement shortest job first scheduling algorithm has been written and executed successfully

Ex.No.4c **Priority Scheduling**

Date :

Aim:

To write a program in C to implement the priority scheduling algorithm.

Algorithm:

1. Start the program.
2. Initialize the variables in structure.
3. Get the number of process, priority and burst time from the user.
4. Start the process execution according to the priority.
5. The total execution time is calculated by adding the burst time.
6. Calculate the average waiting time and turnaround time using total execution and waiting time
7. Terminate the program.

Program:

```
#include<stdio.h>
main()
{ int n,b[10],w[10],i,j,h,t,tt,d;
int stime[10],a[10],p[10];
float avg=0;
printf("\n\t\t\t\tPRIORITY SCHEDULING ALGORITHM");
printf("\n\t\t\t\t*****************************\n");
printf("Enter how many jobs:");
scanf("%d",&n);
printf("\nEnter burst time & priority for corresponding job\n\n");
for(i=1;i<=n;i++)
{
   printf("Process %d:",i);
scanf("%d %d",&b[i],&p[i]);
a[i]=i; }
for(i=1;i<=n;i++)
for(j=i;j<=n;j++)

if(p[i]>p[j])
{
   t=b[i];
tt=a[i];
b[i]=b[j];
a[i]=a[j];
b[j]=t;
a[j]=tt;
   }
w[1]=0;
printf("\nProcess %d Waiting Time:0",a[1]);
for(i=2;i<=n;i++)
```

```
    {
    w[i]=b[i-1]+w[i-1];
    printf("\nProcess %d waiting time:%d",a[i],w[i]); avg+=w[i]; }
    printf("\nTotal Waiting Time:%f",avg);
    printf("\nAverage Waiting Time=%f\n",avg/n);
    printf("\nGANTT CHART"); printf("\n**********\n\n");
    for(i=1;i<=n;i++)
    {
       printf("%d ",b[i]); }
    printf("\n\n");
    for(i=1;i<=n;i++)
    {
    printf("%d",w[i]);
    for(j=1;j<=b[i];j++)
    printf("_"); }
    for(i=1;i<=n;i++)
    { d=w[i]+b[i];
    }

    printf("%d",d);
    return 0;
    }
```

Output:

```
                PRIORITY SCHEDULING ALGORITHM
                ****************************
Enter how many jobs:3

Enter burst time & priority for corresponding job

Process 1:4 2
Process 2:6 1
Process 3:10 3

Process 2 Waiting Time:0
Process 1 waiting time:6
Process 3 waiting time:10
Total Waiting Time:16.000000
Average Waiting Time=5.333333

GANTT CHART
***********

6 4 10

0______6____10__________20
```

Result:

Thus the program to implement priority scheduling algorithm has been written and executed successfully.

Ex.4D **Round Robin Scheduling**
Date:

Aim:

To write a program to implement the Round Robin CPU scheduling Algorithm.

Algorithm:

1. Start the program
2. Get the number of processors
3. Get the Burst time(BT) of each processors
4. Get the Quantum time(QT) or time slice.
5. Execute each processor until reach the QT or BT
6. Time of reaching processor's BT is it's Turn Around Time(TAT)
7. Time waits to start the execution, is the waiting time(WT) of each processor
8. Calculation of Turn Around Time and Waiting Time
   - 8.1.tot_TAT = tot_TAT + cur_TAT
   - 8.2.avg_TAT = tot_TAT/num_of_proc
   - 8.3.tot_WT = tot_WT + cur_WT
8. 4.avg_WT = tot_WT/num_of_proc
9. Display the result
10. Stop the program

Program:

```
#include<stdio.h>
int n,b[10],z[10],q,i,j,r,m[50],e=0,avg=0;
float f;
int rr();
int main()
{
    printf("\n\n\t\t\t\tROUND ROBIN\n\n");
printf("\t\t\t\t**********\n");
printf("Enter how many jobs:");
scanf("%d",&n);

printf("\nEnter burst time for corresponding job..\n");
printf("\n");
for(i=1;i<=n;i++)
{
printf("Process %d:",i);
scanf("%d",&b[i]); z[i]=b[i];
}
printf("\nEnter the time slice value:");
scanf("%d",&q);
rr();//no return type with no argument function average();
return 0; }
int rr()
{
   int max=0; max=b[1]; for(j=1;j<=n;j++) if(max<=b[j]) max=b[j];
```

```
if((max%q)==0)
r=(max/q);
else
r=(max/q)+1;
for(i=1;i<=r;i++)
{
printf("\n\nRound %d",i);
for(j=1;j<=n;j++)
{
   if(b[j]>0) {
      b[j]=b[j]-q;
   }
if(b[j]<=0)
{
     b[j]=0;
printf("\nProcess %d is completed",j);
}
else
{
printf("\nProcess %d remaining time is %d",j,b[j]);
}
}
}
return 0;
}
int average()
{
   for(i=1;i<=n;i++)
{ e=0;
for(j=1;j<=r;j++)
{
   if(z[i]!=0) {
if(z[i]>=q)
{
m[i+e]=q; z[i]-=q; }
else
{ m[i+e]=z[i];
z[i]=0; } }
else
m[i+e]=0; e=e+n;
} }
for(i=2;i<=n;i++)
for(j=1;j<=i-1;j++) avg=avg+m[j];
for(i=n+1;i<=r*n;i++)
{
   if(m[i]!=0) {
      for(j=i-(n-1);j<=i-1;j++)
avg=m[j]+avg;
} }
f=avg/n;
printf("\n\nTotal Waiting:%d",avg);
```

```
printf("\n\nAverage Waiting Time:%f\n",f);
printf("\n\t\t\t\tGANTT CHART");
printf("\n\t\t\t\t**********\n\n");
for(i=1;i<=r*n;i++) {
   if(m[i]!=0) {
      if(i%n==0) {
         printf("P%d",(i%n)+(n));
}
else
{
   printf("P%d",(i%n)); for(j=1;j<=m[i];j++)
   printf("_",n);
}
} }
printf("\n\n\n");
return 0; }
```

Output:

```
                    ROUND ROBIN

                    *****************
Enter how many jobs:3

Enter burst time for corresponding job..

Process 1:7
Process 2:4
Process 3:3

Enter the time slice value:3


Round 1
Process 1 remaining time is 4
Process 2 remaining time is 1
Process 3 is completed

Round 2
Process 1 remaining time is 1
Process 2 is completed
Process 3 is completed

Round 3
Process 1 is completed
Process 2 is completed
Process 3 is completed
```

Result:

Thus the program to implement Round Robin scheduling algorithm has been written and executed successfully.

Ex: No: 5 Illustration of Interprocess Communication using Shared Memory
Date :

Aim:

To write a C program to implement inter process communication using shared memory .

Algorithm:

1. Start the Program
2. Obtain the required process id 3. Increment the *ptr=*ptr+1;
4. Print the process identifier.
5. Check the values of sem_num, sem_op, sem_flg.
6. Stop the execution.

Program:

```
#include<stdio.h>
#include<sys/shm.h>
#include<sys/ipc.h>
int main() {
int child,shmid,i;
char *shmptr;
//child=fork();
if(!child)
{
shmid=shmget(2041,32,0666|IPC_CREAT);
shmptr=shmat(shmid,0,0);
printf("\nParent writing\n");
for(i=0;i<10;i++)
{
   shmptr[i]='a'+i;
   putchar(shmptr[i]);
}
   }
shmid=shmget(2041,32,0666);
shmptr=shmat(shmid,0,0);
printf("\nChild is reading\n");
for(i=0;i<10;i++)
putchar(shmptr[i]);
shmdt(NULL);
shmctl(shmid,IPC_RMID,NULL);
return 0;
}
```

Output:

```
[examuser1@linux ~]$cc memory.c
[examuser1@linux ~]$./a.out

 Parent writing
abcdefghij

 child is reading
abcdefghij
```

Result:

Thus the program to implement interprocess communication using shared memory has been written and executed successfully.

Ex.No:6 Implementation of Mutex for Producer Consumer Problem by Semaphores

Date :

Aim:

To write a C-program to implement mutex for the producer – consumer problem by semaphores.

Algorithm:

1. Start the program.
2. Declare the required variables.
3. Initialize the buffer size and get maximum item you want to produce.
4. Get the option, which you want to do either producer, consumer or exit from the operation.
5. If you select the producer, check the buffer size if it is full the producer should not produce the item or otherwise produce the item and increase the value buffer size.
6. If you select the consumer, check the buffer size if it is empty the consumer should not consume the item or otherwise consume the item and decrease the value of buffer size.
7. If you select exit come out of the program.
8. Stop the program.

Program:

```
#include <stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main() {

  int n;

void producer();

void consumer();

int wait(int);

  int signal(int);
```

```
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
  while(1)
{
printf("\nENTER YOUR CHOICE\n");
  scanf("%d",&n);
  switch(n)
{
case 1:
  if((mutex==1)&&(empty!=0)) producer();
else
printf("BUFFER IS FULL");
break;
  case 2:
if((mutex==1)&&(full!=0)) consumer();
else
printf("BUFFER IS EMPTY");
break;
  case 3:
  exit(0);
   break;
}
}
    return 0;
}
 int wait(int s) { return(--s);
} int signal(int s) { return(++s); }
void producer()
{
```

```
mutex=wait(mutex); full=signal(full); empty=wait(empty); x++;

printf("\nproducer produces the item%d",x);

 mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex); full=wait(full); empty=signal(empty);

 printf("\n consumer consumes item%d",x); x--;

mutex=signal(mutex);

}
```

Output:

```
[examuser1@linux ~]$vi semaphore.c

[examuser1@linux ~]$cc semaphore.c

[examuser1@linux ~]$ ./a.out
1.PRODUCER

2.CONSUMER

3.EXIT


ENTER YOUR CHOICE  1

producer produces the item1

ENTER YOUR CHOICE 1
```

producer produces the item2

ENTER YOUR CHOICE 2

consumer consumes item2

ENTER YOUR CHOICE 2

consumer consumes item1

ENTER YOUR CHOICE 2

BUFFER IS EMPTY

ENTER YOUR CHOICE 1

producer produces the item1

ENTER YOUR CHOICE 1

producer produces the item2

ENTER YOUR CHOICE 3

Result:

Thus the C program to implement mutex for the producer – consumer problem by semaphores has been written and executed successfully.

Ex.No:7 Implementation of Bankers algorithm for Deadlock Avoidance

Date :

Aim:

To write a C program to implement bankers algorithm for dead lock avoidance **Algorithm:**

1. Start the Program
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate. If possible it is safe state
6. If the new request comes then check that the system is in safety or not if we allow the request.
7. Stop the execution

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;
input();
show();
cal();
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 the available Resources
3 3 2
Process  Allocation      Max     Available
P1        0 1 0       7 5 3   3 3 2
P2        2 0 0       3 2 2
P3        3 0 2       9 0 2
P4        2 1 1       2 2 2
P5        0 0 2       4 3 3
P1->P3->P4->P2->P0->
 The system is in safe state
```

Result:

Thus the program to implement Bankers algorithm for deadlock avoidance has been written and executed successfully.

Ex.No:8 Implementation of Deadlock Detection Algorithm

Date :

Aim:

To write a C program to implement Deadlock Detection algorithm

Algorithm:

1. Start the Program
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Stop the execution

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("********** Deadlock Detection Algorithm**********\n");
input();  show(); cal();       return 0; } void input() {
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resource 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 dead[100];  int safe[100];  int i,j; for(i=0;i<n;i++)
{
finish[i]=0;
}
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
need[i][j]=max[i][j]-alloc[i][j];
   } } 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;
}
    if(finish[i]==1)
                {
i=n;
}
}
}
}
}          }    j=0;
flag=0;
for(i=0;i<n;i++)
{
if(finish[i]==0)
{
dead[j]=i;
j++;
   flag=1;
      }   }
if(flag==1)
{
printf("\n\nSystem is in Deadlock and the Deadlock processes are\n");
for(i=0;i<n;i++)
{
printf("P%d\t",dead[i]);
}
}
else
{
printf("\nNo Deadlock Occur");
}
}
```

Output:

```
[examuser1@linux ~]$ vi ddet.c
[examuser1@linux ~]$ cc ddet.c
[examuser1@linux ~]$ ./a.out

********** Deadlock Detection Algo ************
Enter the no of Processes       3
Enter the no of resource instances      3
Enter the Max Matrix
3 6 8
4 3 3
3 4 4
Enter the Allocation Matrix
3 3 3
2 0 3
1 2 4
Enter the available Resources
1 2 0
Process  Allocation      Max     Available
P1       3 3 3       3 6 8          1 2 0
P2       2 0 3       4 3 3
P3       1 2 4       3 4 4



System is in Deadlock and the Deadlock processes are
P0                              P1              P2
```

Result:

Thus the program to implement deadlock detection algorithm has been written and executed successfully.

Ex.No:9 **Implementation of Threading**
Date :

Aim:

To write a C program to implement Threading & Synchronization

Algorithm:

1. Start the Program
2. Initialize the process thread array.
3. Print the job started status.
4. Print the job finished status.
5. Start the main function
6. Check for the process creation if not print error message.
7. Stop the execution

Program:

```
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
pthread_t tid[2];
int counter;
void* doSomeThing(void *arg)
{
   unsigned long i = 0;
   counter += 1;
   printf("\n Job %d started\n", counter);
   for(i=0; i<(0xFFFFFFFF);i++);
printf("\n Job %d finished\n", counter);
return NULL;
}
int main(void)
{
   int i = 0;
int err;
while(i < 2)
{
err = pthread_create(&(tid[i]), NULL, &doSomeThing, NULL);
if (err != 0)
printf("\ncan't create thread :[%s]", strerror(err));
i++;
```

```
}
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
return 0;
}
```

Output:

```
$ ./tgsthreads
Job 1 started
Job 2 started
Job 2 finished
Job 2 finished
```

Result:

Thus the program to implement threading and synchronization application has been written and executed successfully.

Ex: No: 10 Implementation of Paging Technique

Date :

Aim:

To write a C program to implement paging technique.

Algorithm:

1. Start the program.
2. Get the number of pages in the process.
3. Get the size of the pages.
4. Get the page table values in frame numbers.
5. Insert the pages into the memory using the formula Z=l[i/m]*m+(i/m)
6. Display the memory allocation
7. Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main(void)
{ int i,m,n,k,z,l[30];
char data[25][10],mem[50][10];
clrscr();
for(i=0;i<50;i++) strcpy(mem[i]," ");
printf("Enter the number of pages:");
scanf("%d",&n); printf("\nEnter the page size:");
scanf("%d",&m);
k=m*n;
printf("\nEnter the %d number of data:\n",k);
for(i=0;i<k;i++)
scanf("%s",data[i]);
printf("Enter the %d page table values:\n",n);
for(i=0;i<n;i++)
scanf("%d",&l[i]);
for(i=0;i<k;i++)
{ z=l[i/m]*m+(i%m);
strcpy(mem[z],data[i]);
}
printf("\t Memory allocation\n");
for(i=0;i<30;i++)
printf("\t%d\t\t%s\n",i,mem[i]);
getch();
}
```

Output:

```
Enter the number of pages:3

Enter the page size:3

Enter the 9 number of data:
a
b
c
d
e
f
g
h
i
Enter the 3 page table values:
1
2
3
     Memory allocation
    0
    1
    2
    3       a
    4       b
    5       c
    6       d
    7       e
    8       f
    9       g
    10      h
    11      i
    12
    13
    14
    15
    16
```

```
Enter the number of pages:3

Enter the page size:3

Enter the 9 number of data:
a
b
c
d
e
f
g
h
i
Enter the 3 page table values:
0
2
4
     Memory allocation
    0       a
    1       b
    2       c
    3
    4
    5
    6       d
    7       e
    8       f
    9
    10
    11
    12      g
    13      h
    14      i
    15
    16
```

Result:

Thus the c program to implement the paging technique has been written and executed successfully.

Ex.No:11 **Implementation of the following Memory Allocation Methods for fixed partition**

Date :

Aim:

To write a program to implement memory allocation method for fixed partition using first fit,worst fit,best fit algorithms.

Algorithm:

1. Start the process.
2. Declare the size.
3. Get the number of processes to be inserted.
4. For first fit
   a. Allocate the first hole that is big enough for searching.
   b. Start from the beginning set of holes.
   c. If not start at the hole, which is sharing the previous first fit search end.
   d. Compare the hole.
   e. If large enough, then stop searching in the procedure.
5. For Worst Fit
   a. Allocate the largest free hole available in the memory that is sufficient enough to hold the process within the system.
   b. Search the complete memory for available free partitions
   c. Allocate the process to the memory partition which is the largest out of all.
6. For best fit
   a. Allocate the best hole that is small enough for searching.
   b. Start at the best of the set of holes.
   c. If not start at the hole, which is sharing the previous best fit search end.
   d. Compare the hole.
   e. If small enough, then stop searching in the procedure.
   f. Display the values.
7. Terminate the process.

Program:

```
#include<stdio.h>
int main()
{
int n,p,i,j,tmp,t;
int size[10],first[10],best[10],worst[10];
printf(" Memory Allocation Strategy \n\n Enter the number of holes in the Main Memory: ");
scanf("%d",&n);
printf(" Mention their sizes.\n");
for (i=0;i<n;i++)
{
printf("Hole %d : ",i+1);
scanf("%d",&size[i]);
}
printf(" Holes and their sizes \n\n");
for (i=0;i<n;i++)
{
printf(" Hole %d : %d\n",i+1,size[i]);
first[i]=size[i];
best[i]=size[i];
worst[i]=size[i];
}
printf("Enter the size of new process : ");
scanf("%d",&p);
printf("\n FIRST - FIT \n ********** \n");
for (i=0;i<n;i++)
{
if (size[i]>=p)
{
first[i]=size[i]-p;
break;
}
}
if
(n==i+1)
{
printf("New process of size %d cannot be stored in any holes",p);
goto l;
}
for (i=0;i<n;i++)
{
printf("\tHole %d : %d\n",i+1,first[i]);
}
l:printf("\n BEST - FIT \n ********** \n"); t=0;
for (i=0;i<n;i++)
best[i]=size[i]-p; tmp=best[0];
for (i=1;i<n;i++)
{
if (best[i]>0)
{
if (best[i]<tmp)
{
tmp=best[i]; t=i;     }
}
}
for (i=0;i<n;i++) best[i]=size[i];
```

```
if (best[t]>=p) best[t]=best[t]-p;
else
{
printf("New process of size %d cannot be stored in any holes.",p);
goto l1;
}
for (i=0;i<n;i++)
printf("\tHole %d : %d\n",i+1,best[i]);
l1: printf("\n WORST - FIT \n ********** \n"); t=0;
for (i=0;i<n;i++)    best[i]=size[i]-p; tmp=best[0];
for (i=1;i<n;i++)
{
if (best[i]>0)
{
if (best[i]>tmp)
{
tmp=best[i]; t=i;    }
}
}
for (i=0;i<n;i++) worst[i]=size[i];
if (worst[t]>=p) worst[t]=worst[t]-p;
else
{
printf(".    New process of size %d cannot be stored in any holes.",p);
goto l2;
}
for (i=0;i<n;i++)
printf("\tHole %d :%d\n",i+1,worst[i]);
l2: printf("\nProgram Ended");
}
```

Output:

```
Memory Allocation Strategy

Enter the number of holes in the Main Memory: 3
Mention their sizes.
Hole 1 : 50
Hole 2 : 100
Hole 3 : 150
Holes and their sizes

Hole 1 : 50
Hole 2 : 100
Hole 3 : 150
Enter the size of new process : 50

FIRST - FIT
***********
   Hole 1 : 0
   Hole 2 : 100
   Hole 3 : 150

BEST - FIT
***********
   Hole 1 : 0
   Hole 2 : 100
   Hole 3 : 150

WORST - FIT
***********
   Hole 1 :50
   Hole 2 :100
   Hole 3 :100
```

Result:

Thus the c program to implement memory allocation methods for fixed partitions has been written and executed successfully.

Ex: No: 12a **Implementation Of FIFO Page Replacement Algorithm**

Date :

Aim:

To write a program to implement FIFO page replacement algorithm.

Algorithm:

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

Program:

```
#include<stdio.h>
void main() {
int n,ref[50],f,frame[10],i,fault=0,k=0,j;
printf("\n Enter the number of reference string:");
scanf("%d",&n); printf("\n Enter the reference string values:");
for(i=0;i<n;i++)
scanf("%d",&ref[i]);
printf("\n Enter the frame size:");
scanf("%d",&f); printf("\n FIFO page replacement \n");
for(i=0;i<f;i++)
{
   frame[i]=ref[i];
printf("%d\t",frame[i]);
}
fault=f; while(i<n)
{
for(j=0;j<f;j++)
{
if(ref[i]==frame[j])
{
break;
} }
if(f==j)
{
```

```
fault++;
frame[ k]=ref[ i];
 k++;
if(k== 3) k=0;
printf("\n");
for(j=0;j<f;j++)
printf("%d\t",frame[j]);
} i++;
}
printf("\n Number of page fault is %d",fault);
}
```

Output:

```
 Enter the number of reference string:6

 Enter the reference string values:2
3
4
2
5
6

 Enter the frame size:3

 FIFO page replacement
2   3   4
5   3   4
5   6   4
 Number of page fault is 5
```

```
 Enter the number of reference string:6

 Enter the reference string values:2
3
4
5
6
7

 Enter the frame size:3

 FIFO page replacement
2   3   4
5   3   4
5   6   4
5   6   7
 Number of page fault is 6
```

Result:

Thus the c program to implement FIFO page replacement has been written and executed successfully.

Ex: No: 12b Implementation Of LRU Page Replacement Algorithm
Date :

Aim:

To write a program to implement LRU page replacement

Algorithm:

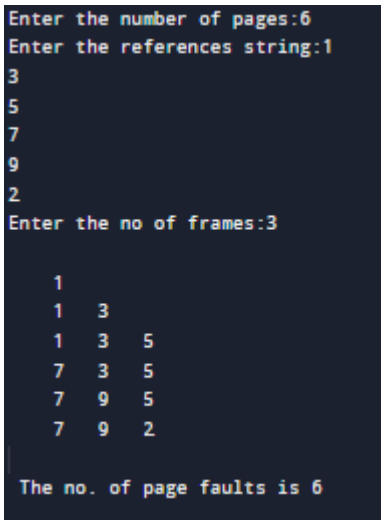
1. Start the process.
2. Declare the size.
3. Get the number of pages to be inserted.
4. Get the value.
5. Declare the counter and stack value.
6. Select the least recently used by counter value.
7. Stack them according to the selection
8. Display the values.
9. Stop the process.

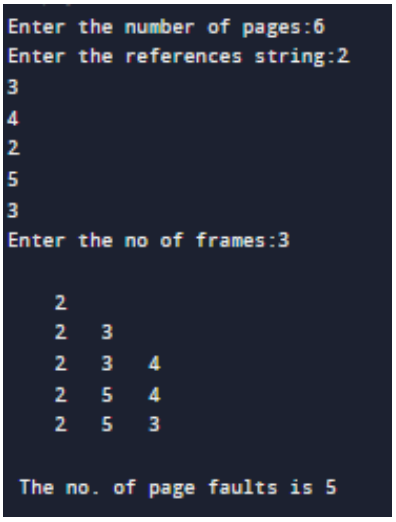
Program:

```
#include<stdio.h>
int main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter the number of pages:");
scanf("%d",&n);
printf("Enter the references string:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter the no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++; k++;
for(i=1;i<n;i++)
{
   c1=0;
for(j=0;j<f;j++)
{
if(p[i]!=q[j]) c1++; }
if(c1==f)
{ c++; if(k<f)
{
q[k]=p[i];
k++;
for(j=0;j<k;j++)
```

```
printf("\t%d",q[ j]);
printf("\n");
}
else {
   for(r=0;r<f;r++)
{ c2[r]=0;
for(j=i-1;j<n;j--)
{
   if(q[r]!=p[j]) c2[r]++;
else break; } }
for(r=0;r<f;r++) b[r]=c2[r];
for(r=0;r<f;r++)
{
for(j=r;j<f;j++)
{
   if(b[r]<b[j])
   { t=b[r]; b[r]=b[j];
b[j]=t; }
} }
for(r=0;r<f;r++)
{ if(c2[r]==b[0])
q[r]=p[i];
printf("\t%d",q[r]);
}
printf("\n");
}
} }
printf("\n The no. of page faults is %d",c); return 0; }
```

Output:





Result:

Thus the c program to implement LRU page replacement has been written and executed successfully.

Ex. No: 12c Optimal (LFU) Page Replacement Algorithm
Date :

Aim:

To write a program to implement LFU page replacement.

Algorithm:

1. Start the process.
2. Declare the size.
3. Get the number of pages to be inserted.
4. Get the value.
5. Declare the counter and stack value.
6. Select the least frequently used by counter value. 7. Stack them according to the selection 8. Display the values.
8. Stop the process.

Program:

```
#include<stdio.h> #include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0; int optvictim();
void main() {    clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
printf("\n. ..... ");
  printf("\nEnter the no.of frames");
scanf("%d",&nof);
  printf("Enter the no.of reference string");
scanf("%d",&nor);    printf("Enter the
reference string");   for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\n.....");     printf("\nThe given string");
printf("\n.....\n");     for(i=0;i<nor;i++)
printf("%4d",ref[i]);
  for(i=0;i<nof;i++)
  {       frm[i]=-
1;
optcal[i]=0;
  }
  for(i=0;i<10;i++)
     recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
  {
flag=0;
     printf("\n\tref no %d ->\t",ref[i]);
```

```
        for(j=0;j<nof;j++)
        {
      if(frm[j]==ref[i])
      {
       flag=1;break;
      }
        }
    if(flag==0)
        {
                    count++;
                    if(count<=nof)
                    victim++; else
                    victim=optvictim(i); pf++;
                frm[victim]=ref[i];
                  for(j=0;j<nof;j++)
                            printf("%4d",frm[j]);
        }
      }
      printf("\n Number of page faults: %d",pf);
      getch(); }
    int optvictim(int index)
    {    int
    i,j,temp,notfound;
      for(i=0;i<nof;i++)
      {
    notfound=1;
        for(j=index;j<nor;j++)
                        if(frm[i]==ref[j])
                        {
                           notfound=0;
          optcal[i]=j;              break;
        }       if(notfound==1)
                          return i;
      }
      temp=optcal[0];
    for(i=1;i<nof;i++)
        if(temp<optcal[i])
          temp=optcal[i];     for(i=0;i<nof;i++)
        if(frm[temp]==frm[i])
                          return i;
     return 0;
    }
```

Output:

```
                    OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames. ..3
 Enter no.of reference string. 6

 Enter reference string..
6 5 4 2 3 1


          OPTIMAL PAGE REPLACEMENT ALGORITHM
     The given reference string:
      ……. 6  5  4  2  3  1

     Reference NO 6->      6  -1  -1
     Reference NO 5->      6   5  -1
     Reference NO 4->      6   5   4
     Reference NO 2->      2   5   4
     Reference NO 3->      2   3   4
     Reference NO 1->      2   3   1

     No.of page faults...6
```

Result:

Thus the c program to implement Optimal (LFU) page replacement has been written and executed successfully.

Ex.No: 13 a **Single Level Directory**
Date :

Aim:

To write a C program to implement File Organization concept using the technique Single level directory.

Algorithm:

1. Start the Program
2. Initialize values gd=DETECT,gm,count,i,j,mid,cir_x.
3. Initialize graph function
4. Set back ground color with setbkcolor();
5. Read number of files in variable count.
6. Check i<count; mid=640/count;
7. Stop the execution

Program:

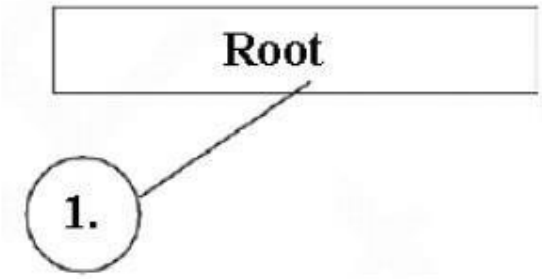
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h> void
main()
{
   int gd=DETECT, gm, count, i, j, mid, cir_x;
char fname[10][20]; clrscr();
initgraph(&gd, &gm, "c:\c\bgi");
cleardevice(); setbkcolor(GREEN);
puts("Enter no. of Files fo you have?");
scanf("%d", &count);

for(i=00;i<count;i++)
{
cleardevice();
setbkcolor(GREEN); printf("Enter
File %d Name", i+1); scanf("%s",
fname[i]);
setfillstyle(1,MAGENTA); mid =
640 / count;

cir_x=mid/3;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4); settextjustify(1,1);
outtextxy(320,125,"Root Directory"); setcolor(BLUE);
for(j=0;j<=i;j++,cir_x+=mid)
```

```
{
   line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[j]);
} getch();
}
}
```

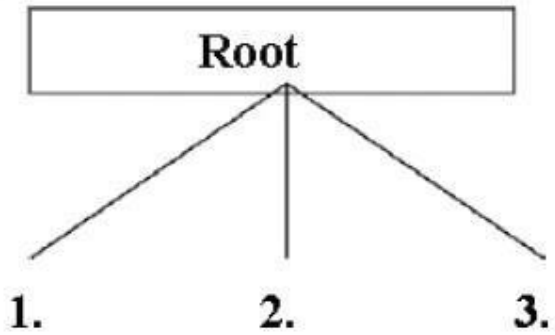
Output:

Enter no. of Files do you have? 3

Enter file1 name : 1.c



Enter file3 name : 3.c



Result:

Thus the C program to implement File Organization concept using the technique Single level directory has been written and executed successfully.

Ex.No: 13 b **Two Level Directoryies**
Date :

Aim:

To write a C program to implement File Organization concept using the technique two level directories.

Algorithm:

1. Start the Program
2. Initialize structure elements
3. Start main function
4. Set variables gd =DETECT, gm;
5. Create structure using create (&root,0,"null",0,639,320);
6. initgraph(&gd,&gm,"c:\tc\bgi");
7. Stop the execution

Program:

```
#include<stdio.h>
#include<graphics.h> struct
tree_element
{
   char name[20];     int
x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element node; void
main()
{
   int gd=DETECT,gm;
node *root;     root=NULL;
clrscr();
create(&root,0,"null",0,639,320); clrscr();
initgraph(&gd,&gm,"c:\\tc\\bgi");
display(root);
getch(); closegraph();
}
create(node **root, int lev, char *dname,
                                int lx,int rx,int x)
{     int
i,gap;
   if(*root==NULL)
{
 (*root)=(node*)malloc(sizeof(node));
  printf("Enter Name of Dir/File under %s):",dname);
fflush(stdin);  gets((*root)->name);
if(lev==0||lev==1)  (*root)->ftype=1;  else
```

```
 (*root)->ftype=2;
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx; (*root)->rx=rx;
for(i=0;i<5;i++)  (*root)-
>link[i]=NULL;
  if((*root)->ftype==1)
    {
if(lev==0||lev==1)
{   if((*root)-
>level==0)
printf("How many Users :");
   else
printf("How many Files :");
printf("(for%s):", (*root)->name);
scanf("%d",& (*root)->nc);
}else(*root)->nc=0; if((*root)-
>nc==0) gap=rx-lx; else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,
(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
   } }
display(node *root)
{
   int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
   if(root!=NULL)
   {
for(i=0;i<root->nc;i++)
{
  line(root->x,root->y,root->link[i]->x,root->link[i]->y);
} if(root->ftype==1) bar3d(root->x-20, root->y-10,root-
>x+20,root->y+10,0,0); else  fillellipse(root->x,root-
>y,20,20); outtextxy(root->x,root->y,root->name);
for(i=0;i<root->nc;i++)
 {
display(root->link[i]);
  }  }
  }
```

Output:

```
Enter Name of Dir/File (under null) : sld
How many Users (for sld) : 2
Enter Name of Dir/File (under sld) : tld
How many Files (for tld) : 2
Enter Name of Dir/File (under tld) : hir
Enter Name of Dir/File (under tld) : dag
Enter Name of Dir/File (under sld) : bin
How many Files (for bin) : 2
Enter Name of Dir/File (under bin) : exe
Enter Name of Dir/File (under bin) : obj
```

Result:

Thus the C program to implement File Organization concept using the technique two level directories has been written and executed successfully.

Ex.No: 13 c **Hierarchical Directories**
Date :

Aim:

To write a C program to implement File Organization concept using the technique hierarchical level directories.

Algorithm:

1. Start the Program
2. Define structure and declare structure variables
3. In main declare variables
4. Check a directory tree structure
5. Display the directory tree in graphical mode.
6. Stop the execution

Program:

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
 {
   char name[20];    int
 x,y,ftype,lx,rx,nc,level;
 struct tree_element *link[5];
 };
 typedef struct tree_element node; void
 main()
 {
   int gd=DETECT,gm;
 node *root;    root=NULL;
   clrscr();
   create(&root,0,"root",0,639,320);
 clrscr();
   initgraph(&gd,&gm,"c:\\tc\\BGI");
   display(root);
 getch();    closegraph();
 }
 create(node **root,int lev, char *dname,
 int lx,int rx,int x)
 {     int
 i,gap;
    if(*root==NULL)
 {
 (*root)=(node *)malloc(sizeof(node));
 printf("Enter Name of Dir/File (under %s) : ",dname);
 fflush(stdin); gets((*root)->name);
 printf("Enter 1 for Dir/2 for File :"); scanf("%d",&(*root)->ftype);
 (*root)->level=lev;
 (*root)->y=50+lev*50;
 (*root)->x=x;
```

```
(*root)->lx=lx;
(*root)->rx=rx; for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype== 1)
{
   printf("No. of Sub Directories / Files (for %s) :",
                          (*root)->name);
   scanf("%d",&(*root)->nc);     if((*root)-
>nc==0)
  gap=rx-lx;
   else  gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++) create(&((*root)-
>link[i]),lev+1,(*root)->name,
  lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
   }
   else(*root)->nc=0;
   }
   }
   display(node *root)
   {
int i; settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
   }
 if(root->ftype==1)
bar3d(root->x-20, root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x, root->y,20,20); outtextxy(root->x,
root->y, root->name);    for(i=0;i<root->nc;i++)
   {
display(root->link[i]);
   }
                        }
   }
```

Output:

```
Enter Name of Dir/File (under root) : ROOT
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for ROOT) :2
Enter Name of Dir/File (under ROOT) : USER 1
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for USER 1) :1
Enter Name of Dir/File (under USER 1) : SUBDIR
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for SUBDIR) :2
Enter Name of Dir/File (under SUBDIR) : JAVA
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for JAVA) :0
Enter Name of Dir/File (under SUBDIR) : VB
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for VB) :0
Enter Name of Dir/File (under ROOT) : USER 2
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for USER 2) :2
Enter Name of Dir/File (under USER 2) : SUBDIR 2
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for SUBDIR 2) :2
Enter Name of Dir/File (under SUBDIR 2) : PPL
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for PPL) :2
Enter Name of Dir/File (under PPL) : B
Enter 1 for Dir/2 for File :2
Enter Name of Dir/File (under PPL) : C
Enter 1 for Dir/2 for File :2
Enter Name of Dir/File (under SUBDIR 2) : AI
Enter 1 for Dir/2 for File :1
No. of Sub Directories / Files (for AI) :2
Enter Name of Dir/File (under AI) : D
Enter 1 for Dir/2 for File :2
Enter Name of Dir/File (under AI) : E
Enter 1 for Dir/2 for File :2
```

Result:

Thus the C program to implement File Organization concept using the technique hierarchical level directory has been written and executed successfully.

Ex.No: 13 d **Directed Acyclic Graph Directory**
Date :

Aim:

To write a C program to implement File Organization concept using the technique directed acyclic graph directory.

Algorithm:

1. Start the Program
2. Define structure and declare structure variables
3. In main declare variables
4. Check a directory tree structure
5. Display the directory tree in graphical mode 6. Stop.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h> struct
tree_element
{
   char name[20];     int
x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element node; typedef
struct
{     char from[20];
char to[20]; }link;
link L[10]; int nofl;
node *root; void
main()
{
   int gd=DETECT, gm;
   root=NULL; clrscr();
create(&root,0,"root",0,639,320); read_links();
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
draw_link_lines(); display(root);
getch(); closegraph();
}
read_links()
{
   int i;
   printf("How many Links :");
scanf("%d",&nofl);
```

```
    for(i=0;i<nofl;i++)
    {
     printf("File / Dir :");  fflush(stdin);
     gets(L[i].from);          printf("Username :");
     fflush(stdin);   gets(L[i].to);
    } }
draw_link_lines()
{    int
i,x1,y1,x2,y2;
    for(i=0;i<nofl;i++)
    {
                          search(root,L[i].from,&x1,&y1);
     search(root,L[i].to,&x1,&y1);
     setcolor(LIGHTGREEN);      setlinestyle(3,0,1);
     line(x1,y1,x2,y2);       setcolor(YELLOW);
     setlinestyle(0,0,1);
    } }
search(node *root,char *s,int *x,int *y)
{
    int i;
    if(root!=NULL)
   {
                          if(strcmpi(root->name,s)==0)
{   *x=root->x;
*y=root->y;
return; } else {
    for(i=0;i<root->nc;i++)
                          search(root->link[i],s,x,y);
}
    } }
create(node **root,int lev,char *dname,int lx,
     int rx,int x)
{
    int i,gap;
    if(*root==NULL)
    {
(*root)=(node *)malloc(sizeof(node));
printf("Enter Name of Dir / File (under %s):",dname);
fflush(stdin); gets((*root)->name);
printf("Enter 1 for Dir / 2 for File :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;
(*root)->lx=lx; (*root)->rx=rx;
for(i=0;i<5;i++) (*root)-
```

```
>link[i]=NULL; if((*root)-
>ftype==1)
{
printf("No. of Sub-Directories / Files (for %s) :",
(*root)->name);
scanf("%d",&(*root)->nc);
if((*root)->nc==0) gap=rx-lx;
else gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]), lev+1,
 (*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
     }          else
                              *root)->nc=0;
   }
}
/* Displays the Constructed Tree in Grpahics mode  */ display(node
*root)
{
   int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
   if(root!=NULL)
   {
for(i=0;i<root->nc;i++)
{
  line(root->x,root->y,root->link[i]->x,  root->link[i]->y);
} if(root->ftype==1)
   bar3d(root->x-20, root->y-10, root->x+20,
root->y+10,0,0); else
   fillellipse(root->x, root->y, 20, 20);     outtextxy(root-
>x, root->y, root->name);     for(i=0;i<root->nc;i++)
   {
                              display(root->link[i]);
   }
}
}
```

Output :

```
Enter Name of Dir/File (under root) : ROOT
Enter 1 for Dir / 2 for File : 1
No. of Sub-Directories / Files (for ROOT) : 2
Enter Name of Dir/File (under ROOT) : USER 1
Enter 1 for Dir / 2 for File : 1
No. of Sub-Directories / Files (for USER 1) : 2
Enter Name of Dir/File (under USER 1) : VB
Enter 1 for Dir / 2 for File : 1
No. of Sub-Directories / Files (for VB) : 2
Enter Name of Dir/File (under VB) : A
Enter 1 for Dir / 2 for File : 2
Enter Name of Dir/File (under VB) : B
Enter 1 for Dir / 2 for File : 2
Enter Name of Dir/File (under USER 1) : C
Enter 1 for Dir / 2 for File : 2
Enter Name of Dir/File (under ROOT) : USER 2
Enter 1 for Dir / 2 for File : 1
No. of Sub-Directories / Files (for USER2) : 1
Enter Name of Dir/File (under USER 2) : JAVA
Enter 1 for Dir / 2 for File : 1
No. of Sub-Directories / Files (for JAVA) : 2

Enter Name of Dir/File (under JAVA) : D
Enter 1 for Dir / 2 for File : 2
Enter Name of Dir/File (under JAVA) : HTML
Enter 1 for Dir / 2 for File : 1
No. of Sub-Directories / Files (for JAVA) : 0
How many Links : 2
File/Dir : B
User Name : USER 2
File/Dir : HTML
User Name : USER 1
```

Result:

Thus the C program to implement File Organization concept using directed acyclic graph directory has been written and executed successfully.

Ex.No.14a **Sequential File Allocation**
Date :

Aim:

To implement sequential file allocation technique.

Algorithm:

1. Start the program.
2. Get the number of files.
3. Get the memory requirement of each file.
4. Allocate the required locations to each in sequential order.
   4.1. Randomly select a location from available location s1= random(100);
   4.2. Check whether the required locations are free from the selected location.
   4.3. Allocate and set flag=1 to the allocated locations.
5. Print the results file number, length , Blocks allocated.
6. Stop the program.

Program:

```
#include<stdio.h>
int main() {
int f[50],i,st,j,len,c,k,count=0;
for(i=0;i<50;i++) f[i]=0; X: printf("\n enter
starting block & length of files");
scanf("%d%d",&st,&len);
printf("\n file not allocated(yes-1/no-0)");
for(k=st;k<(st+len);k++)
if(f[k]==0) count++;
if(len==count)
{
for(j=st;j<(st+len);j++)
if(f[i]==0) { f[j]=1;
printf("\n%d\t%d",j,f[j]);
if(j==(st+len-1))
printf("\n the file is allocated to disk");
} }
else
printf("file is not allocated");
count=0; printf("\n if u want to enter more
files(y-1/n-0)");
scanf("%d",&c);
if(c==1) goto X;
else exit(0);
return 0;
}
```

Output:

```
[examuser1@linux ~]$ cc sequential.c

[examuser1@linux ~]$ ./a.out

enter starting block & length of files3
5

file not allocated(yes-1/no-0)
3       1
4       1
5       1
6       1
7       1

the file is allocated to disk if u want
to enter more files(y-1/n-0)1

enter starting block & length of files0
3

file not allocated(yes-1/no-0)
0       1
1       1
2       1

the file is allocated to disk if u want
to enter more files(y-1/n-0)1

enter starting block & length of files5
3

file not allocated(yes-1/no-0)file is not allocated if
u want to enter more files(y-1/n-0)1

enter starting block & length of files8
3

file not allocated(yes-1/no-0)
8       1
9       1
10      1

the file is allocated to disk
if u want to enter more files(y-1/n-0)0
```

Result:

Thus the C program to implement sequential file allocation has been written and executed successfully.

Ex.No: 14 b **Indexed File Allocation Strategy**
Date :

Aim:

To write a C program to implement File Allocation concept using indexed allocation Technique.

Algorithm:

1. Start the Program
2. Get the number of files.
3. Get the memory requirement of each file.
4. Allocate the required locations by selecting a location randomly.
5. Print the results file no,length, blocks allocated.
6. Stop the execution.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct file { char
n[20]; int ind;
}s[20];
int no,i= -1,a,b,f,j= -1,fe,t;
char tem[20];
void create();
void display();
void del(); void
main()
{
clrscr();
while(1) {
printf("\n \n Menu" );
printf("\n 1.Create \n 2.Display \n 3.Delete \n 4.Exit "); printf("Enter
Your Choice : ");
scanf("%d",&no);
switch(no) { case
1: create(); break;
case 2 : display();
break; case 3:
del(); break; case
4: exit(0); default
: printf("Wrong
Choice");
}
```

```
} } void
create() {
i++;
printf("\n Enter the name of the record : ");
scanf("%s",&s[i].n); printf("\n Enter the
Index no. :"); scanf("%d",&s[i].ind); j++;
} void display() { for(a=0;a<i;a++) {
for(b=0;b<i;b++) {
if(s[b].ind > s[b+1].ind)
{ t = s[b].ind; s[b].ind
= s[b+1].ind;
s[b+1].ind = t;
strcpy(tem,s[b].n);
strcpy(s[b].n,s[b+1].n);
strcpy(s[b+1].n,tem);
                              }
else
continue;
} }
printf("\n \t  Index      Recordname");
for(i=0;i<=j;i++)  {  printf("\n  \t  %d
\t",s[i].ind); printf("\t %s",s[i].n);
 } i--; }  void del() {
int de,index= -1, k=0,l;
if(i!= -1) {
printf("Enter Index no. to be Deleted : ");
scanf("%d", &de); index = de;
while(s[k].ind!= de)
{
k++;
printf("\n \t \t \t %d",k);
} for(l=k;l<=j;l++)
s[l] = s[l+1];
i--; j--;
printf("\n Index no. %d File is deleted",index);
 }
}
```

Output :

```
Menu
1.Create
2.Display
3.Delete 4.Exit

Enter Your Choice : 1
Enter the name of the record : a.java
Enter the index no : 0
Enter the Field no : 1

Menu
1.Create
2.Display
3.Delete
4.Exit
Enter your Choice : 1
Enter the name of the record : b.c
Enter the index no : 1
Enter the Field no : 2

Menu
1.Create
2.Display
3.Delete
4.Exit
Enter your choice : 2
- - - - - - - - - - - - - - - - - - - - - - - -
Index                Recordname          FieldNo
0    a.java          1 1      b.c        2
- - - - - - - - - - - - - - - - - - - - - - - -
Menu
1.Create
2.Display
3.Delete
4.Exit
Enter your Choice : 4
```

Result:

Thus the C program to implement indexed file allocation has been written and executed successfully.

Ex.No.: 14c **Linked File Allocation Strategy**
Date :

Aim:

To write a C program to implement File Allocation concept using Linked List Technique.

Algorithm:

1. Start the Program 2.
Get the number of files.
3. Allocate the required locations by selecting a location randomly
4. Check whether the selected location is free.
5. If the location is free allocate and set flag =1 to the allocated locations.
6. Print the results file no, length, blocks allocated.
7. Stop the execution

Program:

```
#include<stdio.h>
#include<conio.h> void
main()
{
int f[50],p,i,j,k,a,st,len,n;
char c; for(i=0;i<50;i++)
f[i]=0;
printf("enter how many blocks already allocated");
scanf("%d",&p);
printf("\nenter the blocks nos"); for(i=0;i<p;i++)
 {
scanf("%d",&a);
f[a]=1;  }
printf("enter index starting block & length");
scanf("%d%d",&st,&len);
k=len; if(f[st]==0)
 {
for(j=st;j<(k+st);j++)
 { if(f[j]==0)
 { f[j]=1;
printf("\n%d->%d",j,f[j]);
 }
else
{
printf("\n %d->file is already allocated",j); k++;
 }
 }
 } else
printf("\nif u enter one more (yes-1/no-0)");
```

```
scanf("%d",&c);
if(c==1) goto X;
else exit();
getch();
}
```

Output:

```
enter how many blocks already allocated4

enter the blocks nos4
8
2 7
enter index sarting block & length1
10

1->1
 2->file is already allocated
3->1
 4->file is already allocated
5->1
6->1
 7->file is already allocated
 8->file is already allocated
9->1
10->1
11->1
12->1
13->1
14->1
```

Result:

Thus the C program to implement linked file allocation has been written and executed successfully.

Ex.No.: 15 Implementation of various Disk Scheduling Algorithms

Ex.No.: 15a FCFS Disk Scheduling

Aim:

To write a C program to implement FCFS disk scheduling.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seeks count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

Program:

```
#include<stdio.h>
#include<stdlib.h
> intmain() {
intRQ[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);

// logic for FCFS disk scheduling

for(i=0;i<n;i++)
{
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
}

printf("Total head moment is %d",TotalHeadMoment); return0;
```

Output:

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Total head movement is 644
```

Result:

Thus the C program to implement FCFS disk scheduling has been written and executed successfully.

Ex.No.: 15b SSTF Disk Scheduling
Date :

Aim:

To write a C program to implement SSTF disk scheduling.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested,'head' is the position of disk head.
2. Find the positive distance of all tracks in the request array from head.
3. Find a track from requested array which has not been accessed/serviced yet and has minimum distance from head.
4. Increment the total seek count with this distance.
5. Currently serviced track position now becomes the new head position.
6. Go to step 2 until all tracks in request array have not been serviced.

Program:

```
#include<stdio.h>
#include<stdlib.h
> int main() {
   int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;
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);

   // logic for sstf disk scheduling

      /* loop will execute until all process is completed*/
while(count!=n)
   {
      int min=1000,d,index;
      for(i=0;i<n;i++)
      {
        d=abs(RQ[i]-initial);
if(min>d)
        {
min=d;
index=i;
        }
```

```
        }
TotalHeadMoment=TotalHeadMoment+min;
        initial=RQ[index];
        // 1000 is for max
        // you can use any number
RQ[index]=1000;
        count++;
    }

printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

Output:

```
Enter the number of Request
8
Enter Request Sequence
95 180 34 119 11 123 62 64
Enter initial head Position
50
Total head movement is 236
```

Result:

Thus the C program to implement SSTF disk scheduling has been written and executed successfully

Ex.No.: 15c SCAN Disk Scheduling
Date :

Aim:

To write a C program to implement SCAN disk scheduling.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let direction represents whether the head is moving towards left or right.
3. In the direction in which head is moving service all tracks one by one.
4. Calculate the absolute distance of the track from the head.
5. Increment the total seek count with this distance.
6. Currently serviced track position now becomes the new head position.
7. Go to step 3 until we reach at one of the ends of the disk.
8. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

Program:

```
#include<stdio.h>
#include<stdlib.h
> int main() {
   int   RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
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);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

   // logic for Scan disk scheduling

      /*logic for sort the request array */
for(i=0;i<n;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
if(RQ[j]>RQ[j+1])
         {              int
temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
```

```
            }

        }
    }

    int index;
for(i=0;i<n;i++)
    {
        if(initial<RQ[i])
        {
index=i;
break;
        }
    }

    // if movement is towards high value
if(move==1)
    {
        for(i=index;i<n;i++)
        {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
        // last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
        initial = size-1;
        for(i=index-1;i>=0;i--)
        {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
             initial=RQ[i];

        }
    }
    // if movement is towards low value
else
    {
        for(i=index-1;i>=0;i--)
        {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
        }
        // last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
initial =0;         for(i=index;i<n;i++)
        {
```

```
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];

      }
   }

printf("Total head movement is %d",TotalHeadMoment);     return 0;
}
```

Output:

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0 1
Total head movement is 337
```

Result:

Thus the C program to implement SCAN disk scheduling has been written and executed successfully

Ex No: 15 d **CSCAN Disk Scheduling**

Date:

Aim:

To write a C program to implement CSCAN disk scheduling.

Algorithm:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. The head services only in the right direction from 0 to the size of the disk.
3. While moving in the left direction does not service any of the tracks.
4. When we reach the beginning (left end) reverse the direction.
5. While moving in the right direction it services all tracks one by one.
6. While moving in the right direction calculates the absolute distance of the track from the head.
7. Increment the total seeks count with this distance.
8. Currently serviced track position now becomes the new head position.
9. Go to step 6 until we reach the right end of the disk.
10. If we reach the right end of the disk reverse the direction and go to step 3 until all tracks in the request array have not been serviced.

Program:

```
#include<stdio.h>
#include<stdlib.h
> int main() {
   int  RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
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);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

   // logic for C-Scan disk scheduling

      /*logic for sort the request array */
for(i=0;i<n;i++)
   {
for( j=0;j<n-i-1;j++)
      {
if(RQ[j]>RQ[j+1])
         {
int temp;
```

```
temp=RQ[j];
RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
          }

       }
   }

   int index;
for(i=0;i<n;i++)
   {
      if(initial<RQ[i])
      {
index=i;
break;
      }
   }

   // if movement is towards high value
if(move==1)
   {
      for(i=index;i<n;i++)
      {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
      }
      //  last movement for max size
TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
      /*movement max to min disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
initial=0;
for( i=0;i<index;i++)
      {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
   }
   // if movement is towards low value
else
   {       for(i=index-
1;i>=0;i--)
      {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
      }
```

```
        // last movement for min size
TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        for(i=n-1;i>=index;i--)
        {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }

printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

Output:

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter total disk size
200
Enter the head movement direction for high 1 and for low 0 1
Total head movement is 382
```

Result:

Thus the C program to implement CSCAN disk scheduling has been written and executed successfully.

Ex No: 15 D **CLOOK Disk Scheduling**

Date:

Aim:

To write a C program to implement CLOOK disk scheduling.

Algorithm:

1. Let Request array represents an array storing indexes of the tracks that have been requested in ascending order of their time of arrival and head is the position of the disk head.
2. The initial direction in which the head is moving is given and it services in the same direction.
3. The head services all the requests one by one in the direction it is moving.
4. The head continues to move in the same direction until all the requests in this direction have been serviced.
5. While moving in this direction, calculate the absolute distance of the tracks from the head.
6. Increment the total seeks count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 5 until we reach the last request in this direction.
9. If we reach the last request in the current direction then reverse the direction and move the head in this direction until we reach the last request that is needed to be serviced in this direction without servicing the intermediate requests.
10. Reverse the direction and go to step 3 until all the requests have not been serviced.

Program:

```
#include<stdio.h>
#include<stdlib.h
> int main() {
   int  RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
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);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

   // logic for C-look disk scheduling
```

```
        /*logic for sort the request array */
for(i=0;i<n;i++)
   {
for( j=0;j<n-i-1;j++)
      {
if(RQ[j]>RQ[j+1])
         {          int
temp;
temp=RQ[j];
RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
         }

      }
   }

   int index;
for(i=0;i<n;i++)
   {
      if(initial<RQ[i])
      {
index=i;
break;
      }
   }

   // if movement is towards high value
if(move==1)
   {
      for(i=index;i<n;i++)
      {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
      }

for( i=0;i<index;i++)
      {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];

      }
   }
   // if movement is towards low value
else
   {
      for(i=index-1;i>=0;i--)
      {
```

```
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];
      }

      for(i=n-1;i>=index;i--)
      {
TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
initial=RQ[i];

      }
   }

printf("Total head movement is %d",TotalHeadMoment);
return 0;
}
```

Output:

```
Enter the number of Request
8
Enter the Requests Sequence
95 180 34 119 11 123 62 64
Enter initial head position
50
Enter the head movement direction for high 1 and for low 0 1
Total head movement is 322
```

Result:

Thus the C program to implement CLOOK disk scheduling has been written and executed successfully.

Ex No: 16 **Install Linux Using VMWare**

Date:

Aim:

To install linux using VMWare.

Procedure:

1. Download the VMWARE which is available in the website. VMware workstation player includes everything that is needed for standard virtual machine task.
   Launch the installer and follow the installation wizard. Select the option to install an Enhanced Keyboard Driver.
2. proceed through the installation wizard and restart windows when prompted.

Create Linux Virtual Machine:

Start by launching VMware Workstation player, When ready to create VM.

1. Click **Create a new virtual machine**.
2. Select the default option, **Installer disc image file** (iso).
3. Click **Browse** to find ISO file.
4. With guest OS selected click **Next**.
5. Select **Linux** as a guest operating system type.
6. Under **version,** scroll through the list and select the OS.
7. Click **Next** to proceed and if necessary, input a **Virtual Machine Name**.
8. Confirm the storage **Location** and change if needed.

After selecting and configuring the operating system, build the virtual machine.

1. Under **Specify Disk Capacity** adjust **Maximum Disk Size** if required (Default is enough).
2. Select **Split virtual disk in to multiple files** as this makes moving the VM to a new PC easy.
3. Click Next then confirm the details on the next screen.
4. If anything seems wrong click **Back**, otherwise click **Finish**.

Now the Linux virtual machine will be added to VMware Workstation Player.

To installing Linux in VMware is follow the steps given below:

1. Download the free VMware workstation player.
2. Install, and restart windows.
3. Create and configure your virtual machine.
4. Install Linux in the Virtual Machine.

Restart the Virtual Machine and use Linux

Result:

Thus the Linux Using VMware installation has been studied successfully.

Ex NO: 17 **Dead Lock Prevention**

Date:

AIM:

To implement deadlock prevention technique

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

DESCRIPTION:

Data structures
n-Number of process,
m-number of resource types.
Available: Available[j]=k, k – instance of resource type Rj is available.
Max: If max[i, j]=k, Pi may request at most k instances resource Rj.
Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj Need:
If Need[I, j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]-Allocation[I, j];

Safety Algorithm

Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
Find an i such that both Finish[i] =False
Need<=Work

If no such I exists go to step 4.
work=work+Allocation, Finish[i] =True;
if Finish[1]=True for all I, then the system is in safe state

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.

5. Check whether its possible to allocate.

6. If it is possible then the system is in safe state.

7. Else system is not in safety state

8. Stop the process.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
void main()
{
 char job[10][10];
int time[10],avail,tem[10],temp[10];
int safe[10]; int
ind=1,i,j,q,n,t;
clrscr();
printf("Enter no of jobs: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter name and time:");
scanf("%s%d",&job[i],&time[i]);
}
printf("Enter the available resources:");
scanf("%d",&avail);
for(i=0;i<n;i++)
{
temp[i]=time[i];
tem[i]=i;
}
for(i=0;i<n;i++)
for(j=i+1;j<n;j++)
{
if(temp[i]>temp[j])
{
t=temp[i];
temp[i]=temp[j];
```

```
temp[j]=t;
t=tem[i];
tem[i]=tem[j];
tem[j]=t;
}
 }
for(i=0;i<n;i++)
 {
q=tem[i];
if(time[q]<=avail)
{
safe[ind]=tem[i];
avail=avail-tem[q];
printf("%s",job[safe[ind]]);
ind++;
 }
else {
printf("No safe sequence\n");
}
}
 printf("Safe sequence is:");
for(i=1;i<ind; i++)
printf("%s %d\n",job[safe[i]],time[safe[i]]);
getch();
}
```

Output:

Enter no of jobs:4
Enter name and time: A 1
Enter name and time: B 4
Enter name and time: C 2
Enter name and time: D 3 Enter the available resources: 20
Safe sequence is: A 1, C 2, D 3, B 4.

Result:

Thus the program to implement deadlock Prevention algorithm has been written and executed successfully.

Ex NO: 18 **Implement Contiguous file allocation technique**

Date:

Aim:

To write a program for implement the contiguous file allocation technique.

Algorithm:

1. Start the program
2. Declare the size
3. Get the number of files to be inserted.
4. Get the capacity of each file.
5. Get the starting address.
6. The file is allocated in memory
7. The file is not allocated if the contiguous memory is not available.
8. Display the result
9. Stop the program.

Program :

```
#include<stdio.h>
main()
{ int nf, fc[20], mb[100], i, j, k, fb[100], fs[20],
        mc=0; clrscr();
        printf("\nEnter the number of files: ");
        scanf("%d",&nf);
        for(i=0;i<nf;i++)
        { printf("\nEnter the capacity of file %d:
                ",i+1); scanf("%d",&fc[i]); printf("\nEnter
                the starting address of file %d:
                ",i+1); scanf("%d",&fs[i]);
        }
        printf("\n---CONTIGUOUS FILE ALLOCATION---\n");
        for(i=0;i<100;i++) fb[i]=1; for(i=0;i<nf;i++)
        { j=fs[i];

                { if(fb[j]==1)
                        { for(k=j;k<(j+fc[i]);k++)
                                { if(fb[k]==1)
                                        mc++;
                                }
                                if(mc==fc[i]){
                                for(k=fs[i];k<(f
                                s[i]+fc[i]);k++)
```

```
                                { fb[k]=0;
                                }
                                printf("\nFile %d allocated in memory %d
                            to %d...",i+1,fs[i],fs[i]+fc[i]-1); }

                        }
                        else
                        printf("\nFile %d not allocated since %d contiguous memory not
                    available from %d...",i+1,fc[i],fs[i]); }


                    mc=0;
                }
        }
```

Output :

```
Enter the number of files: 3
Enter the capacity of file 1: 21
Enter the starting address of file 1: 21
Enter the capacity of file 2: 24
Enter the starting address of file 2: 36
Enter the capacity of file 3: 2
Enter the starting address of file 3: 54
---CONTIGUOUS FILE ALLOCATION---
File 1 allocated in memory 21 to 41...
File 2 not allocated since 24 contiguous memory not available from 36...
File 3 allocated in memory 54 to 55...
```

Result:

Thus the C program to implement sequential file allocation has been written and executed successfully.