NPR COLLEGE OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)

Approved by AICTE,New Delhi & Affiliated to Anna University,Chennai| Accredited by NAAC with 'A' GRADE
Recognized by UGC under 2 (f) | ISO 9001:2015 Certified | Web: www.nprcolleges.org | E-Mail: nprcetprincipal@nprcolleges.org
NPR Nagar, Natham - 624 401, Dindigul Dist, Tamil Nadu. Ph: 04544 - 246500, 501, 502.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3461 - OPERATING SYSTEM LABORATORY

Semester IV

Student Lab Manual

Regulation 2021

NPR COLLEGE OF ENGINEERING & TECHNOLOGY, NATHAM

VISION

- To develop students with intellectual curiosity and technical expertise to meet the global needs.

MISSION

- To achieve academic excellence by offering quality technical education using best teaching techniques.
- To improve Industry – Institute interactions and expose industrial atmosphere.
- To develop interpersonal skills along with value based education in a dynamic learning environment.
- To explore solutions for real time problems in the society.

NPR COLLEGE OF ENGINEERING & TECHNOLOGY, NATHAM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

- To produce globally competent technical professionals for digitized society.

MISSION

- To establish conducive academic environment by imparting quality education and value added training.
- To encourage students to develop innovative projects to optimally resolve the challenging social problems.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates of Computer Science and Engineering Program will be able to:

- Develop into the most knowledgeable professional to pursue higher education and Research or have a successful carrier in industries.
- Successfully carry forward domain knowledge in computing and allied areas to solve complex and real world engineering problems.
- Meet the technological revolution they are continuously upgraded with the technical knowledge.

Serve the humanity with social responsibility combined with ethics

CS3461 OPERATING SYSTEM LABORATORY

L T P C
0 0 3 1.5

OBJECTIVES:

- To install windows operating systems.
- To understand the basics of Unix command and shell programming.
- To implement various CPU scheduling algorithms.
- To implement Deadlock Avoidance and Deadlock Detection Algorithms .
- To implement Page Replacement Algorithms
- To implement various memory allocation methods.
- To be familiar with File Organization and File Allocation Strategies.

LIST OF EXPERIMENTS:

1. Installation of windows operating system
2. Illustrate UNIX commands and Shell Programming
3. Process Management using System Calls : Fork, Exit, Getpid, 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 Semaphore
7. Write C programs 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.

TOTAL: 60 PERIODS

OUTCOMES:

On completion of this course, the students will be able to:

CO1 : Define and implement UNIX Commands.

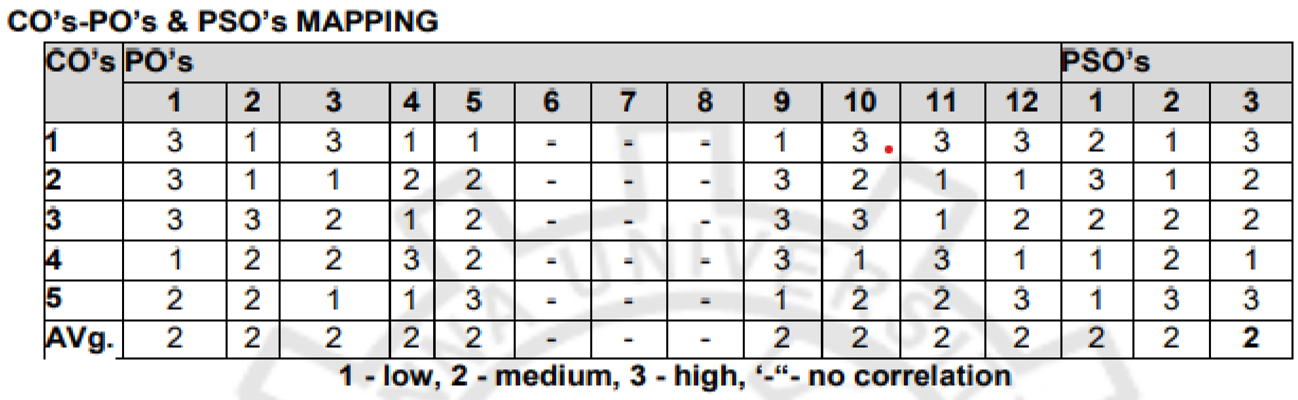
CO2 : Compare the performance of various CPU Scheduling Algorithms.

CO3 : Compare and contrast various Memory Allocation Methods.

CO4 : Define File Organization and File Allocation Strategies.

CO5 : Implement various Disk Scheduling Algorithms.

OPERATING SYSTEM LABORATORY



CO's-PO's & PSO's MAPPING

CO's	PO's												PSO's		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
1	3	1	3	1	1	-	-	-	1	3	3	3	2	1	3
2	3	1	1	2	2	-	-	-	3	2	1	1	3	1	2
3	3	3	2	1	2	-	-	-	3	3	1	2	2	2	2
4	1	2	2	3	2	-	-	-	3	1	3	1	1	2	1
5	2	2	1	1	3	-	-	-	1	2	2	3	1	3	3
AVg.	2	2	2	2	2	-	-	-	2	2	2	2	2	2	2

1 - low, 2 - medium, 3 - high, '-"- no correlation

List of Experiments with COs, POs and PSOs

Exp. No.	Name of the Experiment	COs	POs	PSOs
1	Installation Of Windows Operating System	CO1	1,2,5,10	1
2	Illustrate UNIX Commands And Shell Programming	CO1	1,2,5,10	1
3	Process Management Using System Calls: Fork, Exit, Getpid,Wait,Close	CO1	1,2,5,10	1
4	Write C Programs To Implement The Various CPU Scheduling Algorithms	CO2	1,2,4,5	1
5	Illustrate The Inter Process Communication Strategy	CO2	1,2,4,5	1
6	Implement Mutual Exclusion By Semaphore	CO2	1,2,4,5	1
7	Write C Programs To Avoid Deadlock Using Banker's Algorithm	CO3	1,2,3,4	1,2
8	Write A C Program To Implement Deadlock Detection Algorithm	CO3	1,2,3,4	1,2
9	Write C Program To Implement Threading	CO4	1,2,3,4	1
10	Implement The Paging Technique Using C Program	CO4	1,2,3,4	1
11	Write C Programs To Implement The Following Memory Allocation Method a. First Fit B. Worst Fit C. Best Fit	CO4	1,2,3,4	1
12	Write C Programs To Implement The Various Page Replacement Algorithms	CO4	1,2,3,4	1
13	Write C Programs To Implement The Various File Organization Techniques	CO4	1,2,3,4	1
14	Implement The Following File Allocation Strategies Using C Programs A. Sequential B. Indexed C. Linked	CO4	1,2,3,4	1

15	Write C Programs For The Implementation Of Various Disk Scheduling Algorithms	CO5	1,2,3,4	1,2
16	Install Any Guest Operating System Like Linux Using VM ware	CO5	1,2,3,4	1,2

Program Outcomes

1. Engineering Knowledge
2. Problem Analysis
3. Design/Development of Solutions
4. Conduct Investigations of Complex Problems
5. Modern Tool Usage
6. The Engineer and Society
7. Environment and Sustainability
8. Ethics
9. Individual and Team Work
10. Communication
11. Project Management and Finance
12. Life-long Learning

Program Specific Outcomes

At the end of the program students will be able to

- Deal with real time problems by understanding the evolutionary changes in computing, applying standard practices and strategies in software project development using open-ended programmingenvironments.
- Employ modern computer languages, environments and platforms in creating innovative career pathsby inculcating moral values and ethics.
- Achieve additional expertise through add-on and certificate programs.

Ex.No:01 **Date:**	**INSTALLATION OF WINDOWS OPERATING SYSTEM**

Aim:

To INSTALLATION OF WINDOWS OPERATING SYSTEM

Procedure to Install :

How to Install Windows 10

1. Creating an Installation Disc or Drive

1 **Connect a blank USB flash drive or insert a blank writable DVD.** You can install Windows 10 by creating a bootable USB flash drive or DVD that contains the Windows 10 installation files. You'll need a USB flash drive that's at least 8GB, or any blank DVD to get started.[1]

- If you already have Windows 10 installed on the PC and just want to reinstall it, it'll be easiest to reinstall it from within Windows 10 instead of creating installation media.
- If you want to upgrade from Windows 7 or Windows 8.1, you won't need to create an installation disc or drive. However, you *will* need to follow most of this method to start the upgrade.

2 **Make sure you have a product key.** If you bought Windows 10 through Microsoft using your Microsoft account, your product key is already linked to your account. If you bought Windows 10 from another retailer, you'll have a 25-character product key that you'll need to have handy to activate Windows.[2]

- If you don't have a product key or you're installing Windows 10 on a new hard drive, make sure you've linked your Windows 10 digital license to your Microsoft account before you start the installation.[3] Head to **Settings** > **Update & Security** > **Activation** from the current installation—if the activation status says Windows is activated with a digital license, click **Add an account** and follow the on-screen instructions to link your Microsoft account.
- If you're upgrading from an earlier version and your PC qualifies for a free upgrade, you won't need a product key.

3 **Go to https://www.microsoft.com/en-us/software-download/windows10%20.** This is the official download site for Windows 10.

4 **Click Download tool now.** This is a blue button in the middle of the page. This downloads the Media Creation Tool, which you'll use to create your installation media (or start your upgrade).

5 **Double-click the downloaded file.** Its name begins with "MediaCreationTool" and ends with ".exe." You'll find it in your default download folder, which is usually called Downloads.

6 **Click Accept to accept the license.** It's in the bottom-right corner of the window.

n

7 **Select "Create installation media" and click OK.** This option lets you create a Windows installation disc or drive that will work on any compatible PC, not just the one you're using now.

8 **Select your preferences and click Next.** If you're installing Windows on the current PC, you can keep the default options. If you need to install on a different PC, make sure you choose the language and edition for which you have a license, and select the architecture (64-bit or 32-bit) that matches the PC you're going to install on.

9 **Choose an installation type and click Next.** An ISO file is a type of file that can be burned to a DVD, so choose that option if you plan to create a DVD. Otherwise, choose the USB flash drive option.

10 **Create your installation media.** The steps are a little different depending on what you're doing:

- **Flash drive:** Select your flash drive from the list, click **Next**, and wait for the installation files to install. When the process is complete, click **Finish**.
- **DVD/ISO:** Click **Save** to save the ISO file to your computer—it may take a while because the file is large and has to be downloaded. Once downloaded, you'll see a progress screen that monitors the download. When the download is complete, click **Open DVD burner** on the "Burn the ISO file to a DVD" screen, select your DVD burner, and then click **Burn** to create your DVD.

PART 2

Booting from Windows 10 Installation Media

1 **Connect your Windows 10 installation media.** If you created a flash drive, connect it to the PC on which you want to install Windows 10. If you made a DVD, insert it into the drive now.

2. **Boot the PC into the BIOS.** If your PC is not already set up to boot from your flash or optical drive, rebooting from your installation media won't work. You'll need to make a quick change in your BIOS to change the boot order. There are a few ways to get in:

- **Windows 8.1 or 10:** From Windows, open **Settings**, select **Update & Recovery** or **Update & Security**, and go to **Recovery** > **Restart now** > **Troubleshoot** > **Advanced Options** > **UEFI Firmware Settings** > **Restart**.
- **Any PC:** Reboot the PC and immediately start pressing (over and over again) the keyboard key required by your PC to enter "Setup," or the BIOS. The key varies by computer, but here are some of the most common keys:
  - Acer and Asus: F2 or Del
  - Dell: F2 or F12
  - HP: ESC or F10
  - Lenovo: F1, F2, or Fn + F2
  - Lenovo ThinkPads: Enter + F1.
  - MSI: DEL

3 **Go to the Boot tab.** You'll use the arrow keys to select it.

- The **Boot** tab may instead say **Boot Options** or **Boot Order**, depending on your computer's manufacturer.

4. **Select a device from which to boot.** You have a couple of options here:For a **USB flash drive**, select the **Removable Devices** option.For a **disc installation**, select the **CD-ROM Drive** or **Optical Drive** option.

5. **Press the + key until your boot option is first.** Once either **Removable Devices** or **CD-ROM Drive** is at the top of the list, your computer will select your choice as its default boot option.

- On some computers, you'll instead press one of the function keys (e.g., **F5** or the arrow keys to navigate an option up to the top of the menu. The key will be listed on the right side of the screen.

6 **Save your settings.** You should see a key prompt (e.g., **F10** at the bottom of the screen that correlates to "Save and Exit". Pressing it will save your settings and restart your computer.

7 **Wait for your computer to restart.** Once your computer finishes restarting, you'll see a window here with your geographical data. You're now ready to begin setting up your Windows 10 installation

.Part 3 Installing Windows 10

1. **Click Next when prompted.** You can also 7change the options on this page (e.g., the setup language) before continuing if need be.

2. **Click Install Now.** It's in the middle of the window.

3. **Enter your Windows 10 key, then click Next.** If you don't have a Windows 10 key, instead click **Skip** in the bottom-right corner of the screen.If you've changed hardware in the PC, such as replacing the motherboard, you can activate Windows after installing by going to **Settings** > **Update & Security** > **Activation** > **Troubleshoot** > **I changed hardware on this device**

RESULT:

Thus installation of Operating system of Windows Os was studied successfully

Ex.No:2.A	BASICS OF UNIX COMMANDS

AIM:

To study of Basic UNIX Commands and various UNIX editors such as vi, ed, ex and EMACS.

CONTENT:

Note: Syn->Syntax

a) date

–used to check the date and time

Syn:$date

Format	Purpose	Example	**RESULT**
+%m	To display only month	$date+%m	06
+%h	To display month name	$date+%h	June
+%d	To display day of month	$date+%d	O1
+%y	To display last two digits of years	$date+%y	09
+%H	To display hours	$date+%H	10
+%M	To display minutes	$date+%M	45
+%S	To display seconds	$date+%S	55

b) cal

–used to display the calendar

Syn:$cal 2 2009

c)echo

–used to print the message on the screen.

Syn:$echo “text”

d)ls

–used to list the files. Your files are kept in a directory.

Syn:$lsls–s

All files (include files with prefix)

ls–l Lodetai (provide file statistics)

ls–t Order by creation time

ls– u Sort by access time (or show when last accessed together with –l)

ls–s Order by size

ls–r Reverse order

ls–f Mark directories with /,executable with* , symbolic links with @, local sockets with =, named pipes(FIFOs)with

ls–s Show file size

ls– h“ Human Readable”, show file size in Kilo Bytes & Mega Bytes (h can be used together with –l or)

ls[a-m]*List all the files whose name begin with alphabets From „a‟ to „m‟
ls[a]*List all the files whose name begins with „a‟ or „A‟
Eg:$ls>my list Output of „ls‟ command is stored to disk file named „my list‟

e)lp
–used to take printouts
Syn:$lp filename
f)man
–used to provide manual help on every UNIX commands.
Syn:$man unix command
$man cat
g)who & whoami
–it displays data about all users who have logged into the system currently. The next command displays about current user only.
Syn:$who$whoami
h)uptime
–tells you how long the computer has been running since its last reboot or power-off.
Syn:$uptime
i)uname
–it displays the system information such as hardware platform, system name and processor, OS type.
Syn:$uname–a
j)hostname
–displays and set system host name
Syn:$ hostname
k)bc
–stands for „best calculator‟

$bc	$ bc	$ bc	$ bc
10/2*3	scale =1	ibase=2	sqrt(196)
15	2.25+1	obase=16	14 quit
	3.35	11010011	
	quit	89275	
		1010	
		Ā	
		Quit	
$bc	$ bc-l		
for(i=1;i<3;i=i+1)I	scale=2		
1	s(3.14)		
2	0		
3 quit			

FILE MANIPULATION COMMANDS

a) **cat**–this create, view and concatenate files.

Creation:

Syn:$cat>filename

Viewing:

Syn:$cat filename

Add text to an existing file:

Syn:$cat>>filename

Concatenate:

Syn:$catfile1file2>file3
$catfile1file2>>file3 (no over writing of file3)

b) **grep**–used to search a particular word or pattern related to that word from the file.Syn:$grep search word filename
Eg:$grep anu student

c) **rm**–deletes a file from the file systemSyn:$rm filename

d) **touch**–used to create a blank file.
Syn:$touch file names

e) **cp**–copies the files or directories Syn:$cpsource file destination file Eg:$cp student stud

f) **mv**–to rename the file or directorysyn:$mv old file new file
Eg:$mv–i student student list(-i prompt when overwrite)

g) **cut**–it cuts or pickup a given number of character or fields of the file.Syn:$cut<option><filename>
Eg: $cut –c filename
$cut–c1-10emp
$cut–f 3,6emp
$ cut –f 3-6 emp
-c cutting columns
-f cutting fields

h) **head**–displays10 lines from the head(top)of a given fileSyn:$head filename

Eg:$head studentSyn:$head-2student

i) **tail**–displays last 10 lines of the fileSyn:$tail filename
Eg:$tail student
To display the bottom two lines;
Syn:$ tail -2 student

j) **chmod**–used to change the permissions of a file or directory.Syn:$ch mod category operation permission file Where, Category–is the user type
Operation–is used to assign or remove permissionPermission–is the type of

permission
File–are used to assign or remove permission all

Examples:
$chmodu-wx student
Removes write and execute permission for users
$ch modu+rw,g+rwstudent
Assigns read and write permission for users and groups
$chmodg=rwx student
Assigns absolute permission for groups of all read, write and execute permissions
k) **wc**–it counts the number of lines, words, character in a specified file(s)with the options as –l,-w,-c

Category	Operation	Permission
u– users g–group o– others	+assign -remove =assign absolutely	r– read w– write x-execute

Syn: $wc –l filename
$wc –w filename
$wc–c filename

RESULT:
Thus the unix commands was studied succesfully

Ex.No:2B	**SIMPLE SHELL PROGRAMS**

AIM:

To write simple shell programs by using conditional, branching and looping statements.

Write a Shell program to check the given year is leap year or not

ALGORITHM:

SEPT 1: Start the program. STEP 2: Read the value of year.
STEP 3: Calculate „b=expr $y%4".
STEP 4: If the value of b equals 0 then print the year is a leap year
STEP 5: If the value of r not equal to 0 then print the year is not a leap year.

PROGRAM:

```
echo "Enter
the year"read
y
b=`expr
$y % 4`
if [ $b -
eq 0 ]
then
echo "$y is a
leap year"else
echo "$y is not a
leap year"fi
```

RESULT :

Thus the shell programming was developed and output was gained

EX.NO :3	PROGRAM FOR SYSTEM CALLS OF UNIXOPERATING SYSTEM(fork, getpid, exit)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2. STEP 3: Call fork()

system call to create process.STEP 4: If pid==-1, exit.

STEP 5: Ifpid!=-1 , get the process id using getpid().

STEP 6: Print the process id.

STEP 7:Stop the program

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h> // Added for exit()#include <unistd.h>

int main()

{int pid, pid1, pid2;pid = fork();
    if (pid == -1)

    {printf("ERROR IN PROCESS CREATION \n");

        exit(1);}if (pid != 0)

    {pid1 = getpid();

        printf("\nThe parent process ID is %d\n", pid1);

    }else{pid2 = getpid();

        printf("\nThe child process ID is %d\n", pid2);

    }return 0; // Added for good practice}
```

RESULT :

Thus . Program For System Calls Of Unix Operating System Was Developed And Output WasObtained

EX.NO :4A	**Write C Programs To Implement The Various CPU Scheduling Algorithms**

FIRST COME FIRST SERVE

AIM: To write a c program to simulate the CPU scheduling algorithm FirstCome First Serve (FCFS).

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the bursttime Step

4: Set the waiting of the first process as _0‘and its burst time as its turnaroundtime Step

5: for each process in the Ready Q calculate

a). Waiting time (n) = waiting time (n-1) + Burst time (n-1) b).

Turnaround time (n)= waiting time(n)+Burst time(n)

Step 6: Calculate

a) Average waiting time = Total waiting Time / Number of process

b) Average Turnaround time = Total Turnaround Time / Number of processStep 7:

Stop the process

Page 1

SOURCE CODE:

```
#include <stdio.h>
int main()

{int bt[20], wt[20], tat[20], i, n;

    float wtavg = 0, tatavg = 0; // Initialize variables to 0

    printf("\nEnter the number of processes -- ");

    scanf("%d", &n);

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

    {printf("\nEnter Burst Time for Process %d -- ", i);scanf("%d", &bt[i]);
    }wt[0] = 0;

    tat[0] = bt[0];
    for(i = 1; i < n; i++)

    {wt[i] = wt[i-1] + bt[i-1];

        tat[i] = tat[i-1] + bt[i];

        wtavg += wt[i]; // Accumulate waiting time tatavg +=

        tat[i]; // Accumulate turnaround time

    }printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND
    TIME\n");for(i = 0; i < n; i++)
    printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);wtavg /= n; // Calculate
average waiting time
    tatavg /= n; // Calculate average turnaround time
    printf("\nAverage Waiting Time -- %f", wtavg);

    printf("\nAverage Turnaround Time -- %f", tatavg);return 0;

}
```

RESULT :

Thus Program For System Calls Of Unix Operating System Was Developed And Output WasObtained.

EX.NO :4B	**WRITE C PROGRAMS TO IMPLEMENT** SHORTEST JOB FIRST:

AIM: To write a program to stimulate the CPU scheduling algorithm Shortest job first (Non-Preemption)

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept theCPU

burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting accordingto

lowest to highest burst time.

Step 5: Set the waiting time of the first process as _0' and its turnaround time asits burst

time.

Step 6: Sort the processes names based on their Burt timeStep 7: For

each process in the ready queue,

calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate

c) Average waiting time = Total waiting Time / Number of process

d)Average Turnaround time = Total Turnaround Time / Number of processStep 9: Stop

the process.

SOURCE CODE :

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

                temp = p[i];
                p[i] = p[k];
                p[k] = temp;
            }}}
    wt[0]  =  wtavg  =  0;
    tat[0] = tatavg = bt[0];
    for(i = 1; i < n; i++) {
        wt[i] = wt[i-1] + bt[i-1];
        tat[i]  =  tat[i-1]  +  bt[i];
        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
    }printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

RESULT : Thus Program For System Calls Of Unix Operating System Was Developed And Output WasObtained.

EX.NO :4C	**WRITE C PROGRAMS TO IMPLEMENT ROUND ROBIN**

AIM: To simulate the CPU scheduling algorithm round-robin

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum(or) time slice

Step 3: For each process in the ready Q, assign the process id and accept theCPU burst time

Step 4: Calculate the no. of time slices for each process where No. of timeslice for process (n) = burst time process (n)/time slice

Step 5: If the burst time is less than the time slice then the no. of time slices =1.

Step 6: Consider the ready queue is a circular Q, calculate

a) Waiting time for process (n) = waiting time of process(n-1)+ burst time ofprocess(n-1) + the time difference in getting the CPU fromprocess(n-1)

b) Turnaround time for process(n) = waiting time of process(n) + burst time ofprocess(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number ofprocess Step8: Stop the process

e) SOURCE CODE :

```
  #include <stdio.h>
  #include <conio.h>
  int main() {
  int i, n, burst_time[10], waiting_time[10], turnaround_time[10], remaining_time[10], time_slice;
  float average_waiting_time = 0, average_turnaround_time = 0;
  clrscr();
printf("Enter the number of processes -- ");
scanf("%d", &n);
```

```
  for (i = 0; i < n; i++) {
    printf("Enter Burst Time for process %d -- ", i
    + 1);scanf("%d", &burst_time[i]);
    remaining_time[i] = burst_time[i];
  }
  printf("Enter the size of time slice -
  - ");scanf("%d", &time_slice);
  int current_time =
  0;while (1) {
    int all_done = 1;
    for (i = 0; i < n; i++) {
      if (remaining_time[i] >
        0) {all_done = 0;
        if (remaining_time[i] <= time_slice) {
          current_time                    +=
          remaining_time[i];
          turnaround_time[i]               =
          current_time;   remaining_time[i]
          = 0;
        } else {
          current_time   +=   time_slice;
          remaining_time[i]            -=
          time_slice;
        }}}
    if (all_done == 1) break;
  }for (i = 0; i < n; i++) {
    waiting_time[i] = turnaround_time[i] -
    burst_time[i];average_waiting_time +=
    waiting_time[i]; average_turnaround_time +=
    turnaround_time[i];
  }
  printf("\nThe Average Turnaround time is -- %f",
  average_turnaround_time / n);printf("\nThe Average Waiting time is --
  %f", average_waiting_time / n); printf("\n\tPROCESS\tBURST
  TIME\tWAITING TIME\tTURNAROUND TIME\n");
  for (i = 0; i < n; i++) {
    printf("\t%d\t%d\t\t%d\t\t%d\n", i+1, burst_time[i], waiting_time[i], turnaround_time[i]);
  }
  getch();
  return 0;
}
```

RESULT : Thus Program For System Calls Of Unix OperatingSystem Was eveloped And Output Was Obtained.

EX.NO :4D	PRIORITY

AIM: To write a c program to simulate the CPU scheduling priority algorithm.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept theCPU burst

time

Step 4: Sort the ready queue according to the priority number.

Step 5: Set the waiting of the first process as ‗0‘ and its burst time as itsturnaround time

Step 6: Arrange the processes based on process priority Step 7:

For each process in the Ready Q calculate Step 8:for each process

in the Ready Q calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)

b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

c) Average waiting time = Total waiting Time / Number of process

d) Average Turnaround time = Total Turnaround Time / Number of processPrint the **result**s

in an order. Step10:

Stop

SOURCE CODE:

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

  for(i=1; i<n; i++) {
    wt[i] = wt[i-1] + bt[i-1];
    tat[i]   =   tat[i-1]   +
    bt[i]; wtavg = wtavg
    +   wt[i];   tatavg   =
    tatavg + tat[i];
  }
  printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND
  TIME");
  for(i=0; i<n; i++) {
    printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ", p[i], pri[i], bt[i], wt[i], tat[i]);
  }
  printf("\nAverage Waiting Time is --- %f", wtavg/n);
  printf("\nAverage Turnaround Time is --- %f", tatavg/n);
  getch();
  return 0;
}
```

RESULT :

Thus C Programs To Implement The Various Cpu Scheduling AlgorithmsWas Developed And Output Was Obtained.

EX.NO :5	**WRITE C PROGRAMS TO IMPLEMENT** IPC USING **SHARED MEMORY**

AIM: To write a c program to implement IPC using shared memory.ALGORITHM:

Step 1: Start the process

Step 2: Declare the segment size Step

3: Create the shared memory

Step 4: Read the data from the shared memoryStep 5:

Write the data to the shared memory Step 6: Edit the

data

Step 7: Stop the process.

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/ipc.h>
#include
<sys/shm.h>
#include
<sys/types.h>
#define SEGSIZE 100
int main(int argc, char
  *argv[]) {int shmid, cntr;
  key_t key;
  char
  *segptr;
  char buff[] = "poooda..";
  key = ftok(".", 's');
  if ((shmid = shmget(key, SEGSIZE, IPC_CREAT | IPC_EXCL |
    0666)) == -1) {if ((shmid = shmget(key, SEGSIZE, 0)) == -1) {
      perror("shmget"
      );exit(1);
    }} else {
    printf("Creating a new shared memory seg
    \n");printf("SHMID:%d\n", shmid);
  }system("ipcs -m");
  if ((segptr = shmat(shmid, 0, 0)) == (char*)
    -1) {perror("shmat");
    exit(1);
  }
  printf("Writing data to shared
  memory...\n");strcpy(segptr, buff);
  printf("DONE\n");
  printf("Reading data from shared
  memory...\n");printf("DATA: %s\n", segptr);
  printf("DONE\n");
  printf("Removing shared memory segment...\n");if (shmctl(shmid, IPC_RMID, 0) == -1) {
    printf("Can't remove shared memory segment\n");
  } else {
    printf("Removed successfully\n");
  }
  return 0;
}
```

RESULT :

Thus Illustrate The Inter Process Communication Strategy Was Developed And Output Was Obtained.

EX.NO :6	**WRITE C PROGRAMS TO IMPLEMENT MUTUAL EXCLUSION BY SEMAPHORE**

AIM :

To implement mutual exclusion by semaphore in c program

ALGORITHM:

1. Include necessary header files:
   stdio.h for standard input/output operations.
   pthread.h for thread-related operations.
   semaphore.h for semaphore functions.
   unistd.h for the sleep function.
2. Declare a semaphore variable mutex.
3. Define the thread function thread:
   The function takes a void* argument and returns a void*.
   It waits for the semaphore using sem_wait(&mutex) to enter the critical section.
   Prints a message indicating that it has entered the thread.
   Sleeps for 4 seconds to simulate some work being done in the critical section.
   Prints a message indicating that it is exiting the thread.
   Signals the semaphore using sem_post(&mutex) to release the lock.
4. In the main function:
   Initialize the semaphore using sem_init(&mutex, 0, 1) with an initial value of 1.
   Create two threads using pthread_create:
   The first thread (t1) calls the thread function.
   Sleep for 2 seconds using sleep(2).
   The second thread (t2) also calls the thread function.
   Wait for both threads to finish using pthread_join.
   Destroy the semaphore using sem_destroy(&mutex).

PROGRAM:

```
#include < stdio.h>
#include < pthread.h>
#include < semaphore.h>
#include < unistd.h>
sem_t mutex;
void* thread(void* arg)
{    //wait    sem_wait(&mutex);
   printf("\nEntered thread\n");
     //critical section
   sleep(4);  //signal
   printf("\n Exit thread\n");
   sem_post(&mutex);
} int main() {    sem_init(&mutex, 0, 1);
   pthread_t t1,t2;
   pthread_create(&t1,NULL,thread,NULL);
   sleep(2);     pthread_create(&t2,NULL,thread,NULL);
   pthread_join(t1,NULL);
   pthread_join(t2,NULL);
   sem_destroy(&mutex);    return 0; }
```

RESULT : Thus Write C Programs To Implement Mutual Exclusion By SEMAPHORE was successfully Executed and Verified

EX.NO :7	WRITE C PROGRAMS TO AVOID DEADLOCK USING BANKER'S ALGORITHM

AIM:

To write a C program to implement banker‟s algorithm for deadlock avoidance.

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and availablematrix.

Step-4: Compare each and every process using the banker‟s algorithm.

Step-5: If the process is in safe state then it is a not a deadlock processotherwise it is a

deadlock process

Step-6: produce the **RESULT** of state of process

Step-7: Stop the program

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_PROCESS 100
#define MAX_RESOURCE 100
int
max[MAX_PROCESS][MAX_RESOURC
E];                                   int
alloc[MAX_PROCESS][MAX_RESOURC
E];                                   int
need[MAX_PROCESS][MAX_RESOURC
E];
int
avail[MAX_RESOURCE]
;int n, r;
void
input(); void
show(); void
cal();
int main() {
   printf("********** Baner's Algo
   ************\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 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\tAllocation\tMax\tAvailable\n");
  for(i = 0; i < n; i++) {
    printf("P%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]);
      }}
    printf("\n");
  }}
void cal() {
  int finish[MAX_PROCESS], safe[MAX_PROCESS], work[MAX_RESOURCE], i, j, k,
  count = 0;for(i = 0; i < n; i++) {
    finish[i] = 0;
  }
  for(i = 0; i < r; i++)
    {work[i] =
    avail[i];
  }while(count < n) {int found = 0;
    for(i = 0; i < n; i++)
      {if(finish[i] ==
      0) {
        int j;
```

```
            for(j = 0; j < r; j++) {
               if(need[i][j] > work[j])
               {
                  break;
               }}
            if(j == r) {
               for(k = 0; k < r; k++) {
                  work[k] += alloc[i][k];
               }
               safe[count++] =
               i;finish[i] = 1;
               found = 1;
            }}}
      if(found == 0)
         {break;
      }}
   if(count == n) {
      printf("\nThe system is in safe state.\n");
      printf("Safe Sequence is:");
      for(i = 0; i < n; i++) {
         printf("P%d ", safe[i] +
         1);
      }}
   else {
      printf("\nSystem is in unsafe state.\n");
   }}
```

RESULT :

Thus C Programs To Avoid Deadlock Using Banker's Algorithm WasDeveloped And Output Was Verified.

EX NO 8	WRITE A C PROGRAM TO IMPLEMENT DEADLOCK DETECTION ALGORITHM

AIM:

To write a C program to implement algorithm for deadlock detection.

ALGORITHM:

Step-1: Start the program.

Step-2: Declare the memory for the process.

Step-3: Read the number of process, resources, allocation matrix and availablematrix.

Step-4: Compare each and every process using the banker"s algorithm.

Step-5: If the process is in safe state then it is a not a deadlock processotherwise it is a

deadlock process

Step-6: produce the **RESULT** of state of process

Step-7: Stop the program.

```
#include<stdio.h>
#include<conio.h>
int  max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void
input(); void
show(); void
cal();
int main()
{printf("********** Deadlock Detection Algorithm ************\n");input();
   show();
   cal();
   getch();
   return 0;
}void input()
{int i,j;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   printf("Enter the number of resource instances: ");
   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\tAllocation\tMax\t\tAvailable\n");for(i=0;i<n;i++)
   {printf("P%d\t",i+1);for(j=0;j<r;j++)
      {printf("%d ",alloc[i][j]);
      }printf("\t\t"); for(j=0;j<r;j++)
      {printf("%d ",max[i][j]);
      }printf("\t\t");if(i==0)
      {for(j=0;j<r;j++)
         {printf("%d ",avail[j]);
         }}printf("\n");
   }}void cal()
{int finish[100],temp,flag=1,k,c1=0;int dead[100];
   int safe[100];
   int i,j;
   for(i=0;i<n;i++)
   {finish[i]=0;
   }// find need matrixfor(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; break;
               }}}}}
               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: ");for(i=0;i<j;i++)
      {printf("P%d ",dead[i]+1);
      }}else
   {printf("\nNo Deadlock Occurs");
   }}
```

RESULT :

Thus Write A C Program To Implement Deadlock DetectionAlgorithm Wqs Developed And Output Was Obtained.

EX NO 9	WRITE A C PROGRAM C PROGRAM TO IMPLEMENT THREADING

AIM :TO Write C program to implement Threading.

ALGORITHM :

```
STEP 1 : START.

STEP 2 : IMPORT THE THREAD HEADER FILE AND INTIATE
             I=0;

STEP3:  CREATE A FUNCTION AND PRINT VALUE RECEIVED
        BY USING  POINTER IN THE PARAMETER.

STEP 4 : DECLARE ID AND J VALUE IN MAIN FUNCTIONAND CALL THE

         FUNCTION IN MAIN FUNCTION.

STEP 5 : STOP.
```

SOURCE CODE :

```
#include <stdio.h>
#include <string.h>
#include
<pthread.h>
void* foo(void* arg)
{int val = *(int*)arg;
 printf("Value received as argument in starting routine: %i\n",
 val);int* ret = malloc(sizeof(int));
 *ret = val * val;
 pthread_exit(ret)
 ;}int main(void)
{pthread_t id;int j = 2;
 pthread_create(&id, NULL, foo,
 &j);int* ptr;
 pthread_join(id, (void**)&ptr);
 printf("Value received by parent from child: %i\n",
 *ptr);free(ptr);
 return 0;
}Output :Value received as argument in starting routine: 2Value received by parent from child:
4
```

<u>RESULT</u> : Thus program to implement Threading was Developed And Output Was Gained……..

EX NO 10	WRITE A C PROGRAM TO IMPLEMENT THE PAGING TECHNIQUE USING C PROGRAM

AIM: To write a c program to implement Paging technique for memory management.

ALGORITHM:

Step 1: Start the process

Step 2: Declare page number, page table, frame number and process size.Step 3:

Read the process size, total number of pages

Step 4: Read the relative address

Step 5: Calculate the physical addressStep

6: Display the address

Step 7: Stop the process

PROGRAM:

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

RESULT : Thus Implementing Paging Techniques Using C ProgramWas Developed And Output Was Gained.

EX NO 11	WRITE A C PROGRAM TO IMPLEMENT MEMORY ALLOCATION METHODS

WORST FIT

AIM: To Write a C program to simulate the following contiguous memory allocation techniques

ALGORITHM :

1. Define the maximum number of blocks and files that the program can handle using#define.
2. Declare and initialize the necessary variables: b[max] for block sizes, f[max] for file sizes, frag[max] for fragmentation, bf[max] for block flags, and ff[max] for file flags.
3. Get the number of blocks and files from the user.
4. Ask the user to enter the size of each block.
5. Ask the user to enter the size of each file.
6. For each file, find the first block that is large enough to hold it.
7. If a block is found, set the flag for that block to 1 to indicate it is in use.
8. Calculate the fragmentation for that file and store it in frag[].
9. Repeat steps 6 to 8 for all files.
10. Display the results (file number, file size, block number, block size, and fragmentation) using printf().
11. End the program with getch().

PROGRAM

first-FIT

```
#include<stdio.h>
#define max 25
void main()
{int frag[max], b[max], f[max], i, j, nb, nf, temp;static int bf[max], ff[max];
   printf("\n\tMemory Management Scheme - First Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);
   printf("\nEnter the size of the blocks:-\n");
   or(i=1; i<=nb; i++)
   {printf("Block %d:", i);
      scanf("%d", &b[i]);
   }printf("Enter the size of the files :-\n");for(i=1; i<=nf; i++)
   {printf("File %d:", i);
      scanf("%d", &f[i]);
   }for(i=1; i<=nf; i++)
   {for(j=1; j<=nb; j++)
      {if(bf[j]!=1)
         {temp = b[j] - f[i];if(temp >= 0)
            {ff[i] = j;break;
            }}}
```

```
        frag[i] = temp;
        bf[ff[i]] = 1;
    }printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment"); for(i=1; i<=nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

BEST FIT

ALGORITHM :

1. Read the number of blocks and files.
2. Read the sizes of the blocks and files.
3. Initialize an array **bf**to keep track of whether each block is used or not. Set all valuesto 0.
4. For each file, loop through all the blocks to find the block with the smallest amountof fragmentation that can accommodate the file.
5. Keep track of the index of the block with the smallest amount of fragmentation in anarray **ff**.
6. Mark the selected block as used in the **bf**array.
7. Calculate and store the amount of fragmentation in an array **frag**.
8. Print the results.

BEST-FIT

```
#include<stdio.h>
#include<conio.h>
#define MAX_BLOCKS
25
void main()
{
   int frag[MAX_BLOCKS], blocks[MAX_BLOCKS], files[MAX_BLOCKS], i, j, num_blocks,
num_files,temp, best_block;
   int block_status[MAX_BLOCKS] = {0}; // Initialize all blocks to
   free int file_block[MAX_BLOCKS] = {0}; // Initialize all files to
   unallocated
   clrscr();
   printf("\nEnter the number of blocks: ");
   scanf("%d",  &num_blocks);
   printf("Enter the number of files: ");
   scanf("%d", &num_files);
   printf("\nEnter the size of the blocks:\n");
   for(i = 1; i <= num_blocks; i++)
   {printf("Block %d: ", i);
      scanf("%d", &blocks[i]);
   }printf("Enter the size of the files:\n");for(i = 1; i <= num_files; i++)
   {printf("File %d: ", i);
      scanf("%d", &files[i]);
   }
   for(i = 1; i <= num_files; i++)
   {best_block = -1;
      for(j = 1; j <= num_blocks; j++)
      {if(block_status[j] == 0 && blocks[j] >= files[i])
         {if(best_block == -1 || blocks[j] < blocks[best_block])
```

```
            {best_block = j;
            }}}
        if(best_block != -1)
        {block_status[best_block] = 1;file_block[i] = best_block;
           frag[i] = blocks[best_block] - files[i];
        }}
    printf("\nFile No\tFile Size\tBlock No\tBlock
    Size\tFragment");for(i = 1; i <= num_files; i++)
    {printf("\n%d\t\t%d\t\t", i, files[i]);if(file_block[i] != 0)
        {printf("%d\t\t%d\t\t%d", file_block[i], blocks[file_block[i]], frag[i]);
        }else{
           printf("Not Allocated");
        }}
    getch();
}
```

RESULT :

Thus . Program For System Calls Of Unix OperatingSystem Was Developed And Output Was Obtained.

WORST FIT

ALGORITHM :

1. Start the program.
2. Declare variables and constants, including the maximum number of blocks, files, and fragmentation.
3. Clear the screen and print the memory management scheme's name - Worst Fit.
4. Prompt the user to input the number of blocks and files.
5. Prompt the user to enter the size of each block and file.
6. Loop through each file to allocate a block for it.
7. For each file, loop through each block to find the largest block that can accommodate the file.
8. If a block is not allocated, calculate the remaining space after allocating the file.
9. If the remaining space is greater than or equal to zero, compare the remaining spacewith the highest space found so far.
10. If the remaining space is greater than the current highest space, set the block indexto the current file index, and update the highest space to the remaining space.
11. Store the highest space found as the file's fragmentation value.
12. Mark the block as allocated and update the highest space found to zero.
13. Print the file number, size, block number, block size, and fragmentation value.
14. End the program.

PROGRAM

```
#include<stdio.h>
#include<conio.h
>#define max 25
void main()
{int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;static int bf[max], ff[max];
   clrscr();
   printf("\n\tMemory Management Scheme - Worst Fit");
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);
   printf("\nEnter the size of the blocks:-\n");
   for(i = 1; i <= nb; i++)
   {printf("Block %d:", i);
      scanf("%d",
      &b[i]);bf[i] = 0;
   }
   printf("Enter the size of the files :-\n");
   for(i = 1; i <= nf; i++)
   {printf("File %d:", i);
      scanf("%d", &f[i]);
   }for(i = 1; i <= nf; i++)
   {highest = 0;
      for(j = 1; j <= nb; j++)
      {if(bf[j] != 1) //if bf[j] is not allocated
         {temp = b[j] - f[i];if(temp >= 0)
```

```
                {if(highest < temp)
                   {ff[i] = j;
                      highest = temp;
                   }}}}
         frag[i] =
         highest;bf[ff[i]]
         = 1;
      }printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment"); for(i = 1; i <= nf; i++)
         printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
      getch();
   }
```

RESULT : Thus Program For System Calls Of Unix OperatingSystem Was Developed And Output Was Obtained.

EX NO 12	WRITE C PROGRAMS TO IMPLEMENT THE VARIOUS PAGE REPLACEMENTALGORITHMS

AIM :

TO Write C programs to implement the various Page Replacement Algorithms

ALGORITHM:

1. Start the process

2. Read number of pages n

3. Read number of pages no

4. Read page numbers into an array a[i]

5. Initialize avail[i]=0 .to check page hit

6. Replace the page with circular queue, while re-placing check page availability in the frame

Place avail[i]=1 if page is placed in theframe Count page faults

7. Print the **RESULT**s.

8. Stop the process.

A) FIRST IN FIRST OUT

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void display()
   {int i;
   printf("\n");
   for(i=0;i<3;i++) {
      printf("%d\t",fr[i]);
   }}
void main() {
   int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
   int
   flag1=0,flag2=0,pf=0,frsize=3,top=0;
   clrscr();
   for(i=0;i<3;i++)
      {fr[i] = -1;
   }for(j=0;j<12;j++) {flag1=0; flag2=0;
      for(i=0;i<frsize;i++)
```

```
          { f(fr[i]==page[j])
          {flag1=1;flag2=1;
          break;
          }}
       if(flag1==0) {
          for(i=0;i<frsize;i++) {
             if(fr[i]==-1) {
                fr[i]=page[j]
                ;flag2=1;
                break;
             }}}
       if(flag2==0) {
          fr[top]=page[j];
          top++;
          pf++;
          if(top>=frsize)
          {top=0;
          }}
       display();
   }printf("Number of page faults: %d",pf+frsize);getch();
}
```

B) LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM:

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

SOURCE CODE :

```
#include <stdio.h>
#include
<conio.h>
int fr[3];
```

```
void display();void main()
{int p[12] = {2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2};
  int i, j, fs[3];
  int index, k, l, flag1 = 0, flag2 = 0, pf = 0,

   frsize =3;clrscr();for (i = 0; i < 3; i++)

  {fr[i] = -1;
  }
  for (j = 0; j < 12; j++)
  {flag1 = 0, flag2 = 0;
    for (i = 0; i < 3; i++)
    {if (fr[i] == p[j])
      {flag1 = 1;
        flag2 =
        1;break;
      }}
    if (flag1 == 0)
    {for (i = 0; i < 3; i++)
      {if (fr[i] == -1)
        {fr[i] = p[j];flag2 = 1; break;
        }}}

    if (flag2 == 0)
    {for (i = 0; i < 3; i++)fs[i] = 0;
      for (k = j - 1, l = 1; l <= frsize - 1; l++, k--)
      {for (i = 0; i < 3; i++)
        {if (fr[i] == p[k])fs[i] = 1;
        }}
        for (i = 0; i < 3; i++)
      {if (fs[i] == 0)index = i;
      }fr[index] = p[j];pf++;
    }display();
  }printf("\n no of page faults :%d", pf + frsize);getch();
}
void display()
{
  int i;
  printf("\n");

  for (i = 0; i < 3;
    i++)printf();
```

C) OPTIMAL

AIM: To implement optimal page replacement technique.

ALGORTHIM:

1. Start Program

2. Read Number Of Pages And Frames

3.Read Each Page Value

4. Search For Page In The Frames

5.If Not Available Allocate Free Frame

6. If No Frames Is Free Repalce The Page With The Page That Is Leastly Used7.Print Page

Number Of Page Faults

8.Stop process.

SOURCE CODE:

/* Program to simulate optimal page replacement */

```
#include <stdio.h>
#include<conio.h>
int fr[10], n, m;
void display();
void main()
{int i, j, page[20], fs[10];
   int max, found = 0, lg[10], index, k, l, flag1 = 0, flag2 = 0, pf =
   0;float pr;
   clrscr();
   printf("Enter length of the reference string: ");
   scanf("%d", &n);
   printf("Enter the reference string: ");
   for (i = 0; i < n; i++)
   scanf("%d", &page[i]);
   printf("Enter no of frames: ");
   scanf("%d", &m);
   for (i = 0; i < m;
      i++)fr[i] = -1;
   pf = m;
   for (j = 0; j < n; j++)
   {flag1 = 0;
      flag2 = 0;
      for (i = 0; i < m; i++)
      {if (fr[i] == page[j])
         {flag1 = 1;
            flag2 =
            1;break;
         }}
      if (flag1 == 0)
      {for (i = 0; i < m; i++)
         {if (fr[i] == -1)
            {fr[i] = page[j];flag2 = 1; break;
            }}}
      if (flag2 == 0)
      {for (i = 0; i < m; i++)lg[i] = 0;
         for (i = 0; i < m; i++)
         {for (k = j + 1; k < n; k++)
            {if (fr[i] == page[k])
               {lg[i] = k - j;break;
               }}}
            found = 0;
        for (i = 0; i < m; i++)
        {if (lg[i] == 0)
          {index = i; found = 1;break;
          }}
        if (found == 0)
        {max = lg[0];index = 0;
          for (i = 0; i < m; i++)
          {if (max < lg[i])
            {max = lg[i];index = i;
            }}}
        fr[index] =
        page[j];pf++;
      }display();}
  printf("Number of page faults : %d\n", pf);
  pr = (float)pf / n * 100;
  printf("Page fault rate = %f \n", pr);
  getch();
}void display()
{int i;
  for (i = 0; i < m; i++)
    printf("%d\t",
    fr[i]);
  printf("\n");
}
```

RESULT :

Thus Write C Programs To Implement The Various Page ReplacementAlgorithms Was Developed And Output Was Gained.

EX NO 13	WRITE C PROGRAMS TO IMPLEMENT THE VARIOUS FILE ORGANIZATIONTECHNIQUES

AIM :

TO Write C programs to Implement the various File Organization Techniques.

SOURCE CODE :

```
#include  <stdio.h>
#include<string.h>
struct directory
{char dname[10]; char fname[10][10];int fcnt;
};
void main()
{int i, ch; char f[30];
   struct directory dir;
   dir.fcnt = 0;
   printf("\nEnter name of directory -- ");
   scanf("%s", dir.dname);
   while (1)
   {printf("\n\n1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5. Exit\nEnter yourchoice
   -- ");scanf("%d", &ch);switch (ch)
      {case 1:
         printf("\nEnter the name of the file -- ");
         scanf("%s", dir.fname[dir.fcnt]);
         dir.fcnt++;
         break;
      case 2:
         printf("\nEnter the name of the file -- ");
         scanf("%s", f);
         for (i = 0; i < dir.fcnt; i++)
         {if (strcmp(f, dir.fname[i]) == 0)
            {printf("File %s is deleted ", f); strcpy(dir.fname[i], dir.fname[dir.fcnt - 1]);dir.fcnt--;
               break;
            }}if (i == dir.fcnt)
            printf("File %s not found",
         f);break;
      case 3:
         printf("\nEnter the name of the file -- ");
         scanf("%s", f);
         for (i = 0; i < dir.fcnt; i++)
         {if (strcmp(f, dir.fname[i]) == 0)
            {printf("File %s is found ", f);break;
            }}
         if (i == dir.fcnt)
            printf("File %s not found",
         f);break;
      case 4:
         if (dir.fcnt == 0)
            printf("\nDirectory Empty");
         else
```

```
        {printf("\nThe Files are -- ");for (i = 0; i < dir.fcnt; i++)
              printf("\t%s", dir.fname[i]);
        }break;default:
        exit(0);
      }}
      }getch();}
```

TWO LEVEL DIRECTORY

SOURCE CODE :

```
#include <stdio.h>
#include<string.h>
struct directory
   char name[10];
 charfiles[10][10];
   int file_count;
} dirs[10];
int dir_count = 0;
void create_directory() {
   if (dir_count >= 10) {
      printf("\nMaximum directory limit reached.");
      return;
   }
   printf("\nEnter name of directory: ");
   scanf("%s", dirs[dir_count].name);
   dirs[dir_count].file_count = 0;
   dir_count++;
   printf("\nDirectory created successfully.");
}
void create_file() {
   char dir_name[10], file_name[10];
   printf("\nEnter name of the directory: ");
   scanf("%s", dir_name);
   int dir_index = -1;
   for (int i = 0; i < dir_count; i++) {
      if (strcmp(dir_name, dirs[i].name) == 0)
         {dir_index = i;
         break;
      }}
   if (dir_index == -1) {
      printf("\nDirectory %s not found.", dir_name);
      return;
   }printf("\nEnter name of the file: ");scanf("%s", file_name);
   if (dirs[dir_index].file_count >= 10) {
      printf("\nMaximum file limit reached for directory %s.", dir_name);
      return;
   }
   strcpy(dirs[dir_index].files[dirs[dir_index].file_count], file_name);
   dirs[dir_index].file_count++;
   printf("\nFile created successfully.");
}
void delete_file() {
   char dir_name[10], file_name[10];
```

```
    printf("\nEnter name of the directory: ");
    scanf("%s", dir_name);
    int dir_index = -1;
    for (int i = 0; i < dir_count; i++) {
        if (strcmp(dir_name, dirs[i].name) == 0) {
            dir_index =
            i;break;
        }}if (dir_index == -1) {
        printf("\nDirectory %s not found.", dir_name);
        return;
    }printf("\nEnter name of the file: ");scanf("%s", file_name);
    int file_index = -1;
    for (int i = 0; i < dirs[dir_index].file_count; i++) {
        if (strcmp(file_name, dirs[dir_index].files[i]) == 0)
            {file_index = i;
            break;
        }}
    if (file_index == -1) {
        printf("\nFile %s not found in directory %s.", file_name, dir_name);
        return;
    }printf("\nFile %s deleted successfully from directory %s.", file_name, dir_name);
    dirs[dir_index].file_count--;
    strcpy(dirs[dir_index].files[file_index], dirs[dir_index].files[dirs[dir_index].file_count]);
}void search_file() {
    char dir_name[10], file_name[10];
    printf("\nEnter name of the directory: ");
    scanf("%s", dir_name);
    int dir_index = -1;
    for (int i = 0; i < dir_count; i++) {
        if (strcmp(dir_name, dirs[i].name) == 0)
            {dir_index = i;
            break;
        }}
    if (dir_index == -1) {
        printf("\nDirectory %s not found.", dir_name);
        return;
    }printf("\nEnter name of the file”);
created
```

RESULT :

Thus, Implementation Of various File OrganizationTechniques was Successfully executed and Verified

EX NO 14	WRITE C PROGRAMS TO IMPLEMENTFILE ALLOCATION STRATEGIES USING CPROGRAMS **a. Sequential b. Indexed c. Linked**

AIM: Write a C Program to implement Sequential File Allocation method.
ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches the entire entire memory block until a holewhich is big enough is encountered. It allocates that memory block for therequesting process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest holewhichcanbe allocated to requesting process and allocates if.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and

allocates it to the process.

Step 7: Analyses all the three memory management techniques and display the

bestalgorithm which utilizes the memory resources effectively and efficiently.

Step 8: Stop the program.

Program:

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int n, i, j, b[20], sb[20], t[20], x,
   c[20][20];clrscr();
   printf("Enter number of files: ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Enter number of blocks occupied by file %d: ",
      i+1);scanf("%d", &b[i]);
      printf("Enter the starting block of file %d: ",
      i+1);scanf("%d", &sb[i]);
      t[i] = sb[i];
      for(j=0; j<b[i]; j++)
```

```
        c[i][j] = sb[i++];
    }
    printf("\nFilename\tStart block\tLength\n");
    for(i=0; i<n; i++)
       printf("%d\t\t%d\t\t%d\n", i+1, t[i], b[i]);
    printf("\nEnter file name: ");
    scanf("%d", &x);
    printf("\nFile name: %d",
    x);
    printf("\nLength: %d", b[x-
    1]);printf("\nBlocks occupied:
    "); for(i=0; i<b[x-1]; i++)
       printf("%d ", c[x-
    1][i]);getch();
    return 0;
}
```

INDEXED FILE ALLOCATION

AIM: Write a C Program to implement Indexed File Allocation method.Algorithm:

Step 1: Start.

Step 2: Let n be the size of the buffer Step 3:

check if there are any producer

Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is stored in the bufferStep 6:

If the buffer is full the producer has to wait

Step 7: Check there is any cosumer.If yes check whether the buffer is emptyStep 8: If no

the consumer consumes them from the buffer

Step 9: If the buffer is empty, the consumer has to wait.

Step 10: Repeat checking for the producer and consumer till requiredStep 11:

Terminate the process.

Program:

```
#include<stdio.h>
 #include<conio.h>
void main()
{
    int n, m[20], i, j, sb[20], s[20], b[20][20],
    x;clrscr();
    printf("Enter no. of
    files:");scanf("%d", &n);

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

```
   printf("Enter starting block and size of file%d:", i+1);scanf("%d%d", &sb[i], &s[i]);
      printf("Enter blocks occupied by file%d:",
      i+1);scanf("%d", &m[i]);
      printf("Enter blocks of file%d:",
      i+1);for(j = 0; j < m[i]; j++)
         scanf("%d", &b[i][j]);
   }printf("\nFile\tIndex\tLength\n");for(i = 0; i < n; i++)
      printf("%d\t%d\t%d\n", i+1, sb[i], m[i]);

   printf("\nEnter file name:");
   scanf("%d", &x);
   printf("File   name   is:%d\n",
   x);i = x - 1;
   printf("Index          is:%d\n",
   sb[i]);printf("Block occupied
   are:");for(j = 0; j < m[i]; j++)
      printf("%3d", b[i][j]);

   getch();
}
```

LINKED FILE ALLOCATION

AIM: Write a C Program to implement Linked File Allocation method.

ALGORITHM:

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queueStep 3: Now the new page is inserted at the tail of the queue

Step 4: Create a stack
Step 5: When the page fault occurs replace page present at the bottom of thestack

Step 6: Stop the allocation.

Program:

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

struct file{char fname[10];

int start,size,block[10];

}f[10];

main(){

 int i,j,n;

 clrscr();
printf("Enter no. of files:");

scanf("%d",&n);

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

{
printf("Enter file name:"); scanf("%s",&f[i].fname); printf("Enter starting block:");
scanf("%d",&f[i].start);
f[i].block[0]=f[i].start; printf("Enter

no.of blocks:");scanf("%d",&f[i].size);

printf("Enter block numbers:");

for(j=1;j<=f[i].size;j++)

{scanf("%d",&f[i].block[j]);

}}printf("File\tstart\tsize\tblock\n");for(i=0;i<n;i++)
{printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);for(j=1;j<=f[i].size-1;j++)
printf("%d--->",f[i].block[j]);
printf("%d",f[i].block[j]);printf("\n");
}getch();

}
```

RESULT :

Thus, Implement the following File Allocation Strategies using C programs was developed successfully and output was gained.

EX NO 15	WRITE C PROGRAMS FOR THE IMPLEMENTATION OF VARIOUS DISKSCHEDULING ALGORITHMS

AIM :TO Write C programs for the implementation of various disk scheduling algorithms a) FCFS b) SCAN c) C-SCAN

15 A) To Write a C program to simulate disk scheduling algorithms FCFS

PROGRAM :
```
#include<stdio.h>
#include<conio.h>
int main()
{
   int t[20], n, i, j, tohm[20], tot=0;
   float avhm;
   clrscr();
   printf("Enter the number of tracks: ");
   scanf("%d",&n);
   printf("Enter the tracks to be traversed: ");
   for(i=0;i<n;i++)
      scanf("%d",&t[i])
   ;for(i=0;i<n-1;i++)
   {tohm[i]=t[i+1]-t[i];if(tohm[i]<0)
         tohm[i]=tohm[i]*(-1);
   }for(i=0;i<n-1;i++)tot+=tohm[i];
   avhm=(float)tot/n;
   printf("Tracks traversed\tDifference between tracks\n");
   for(i=0;i<n-1;i++)
      printf("%d\t\t\t%d\n",t[i],tohm[i]);
   printf("\nAverage header movements: %f",avhm);
   getch();
   return 0;
}
```

15 B) SCAN DISK SCHEDULING ALGORITHM

```
#include <stdio.h>
#include<conio.h>
main() {
   int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;
   clrscr();
   printf("Enter the no. of tracks to be traversed: ");
   scanf("%d",&n);
   printf("Enter the position of head: ");
   scanf("%d",&h);
   t[0]=0;
   t[1]=h;
   printf("Enter the tracks: ");
   for(i=2;i<n+2;i++)
      scanf("%d",&t[i]);
   // Sorting the tracks in ascending order
   for(i=0;i<n+2;i++) {
      for(j=0;j<(n+2)-i-1;j++) {
          if(t[j]>t[j+1]) {
             temp=t[j];
             t[j]=t[j+1];
             t[j+1]=temp;
          }}}
    // Finding the index of the head position
    for(i=0;i<n+2;i++) {
       if(t[i]==h)
          {j=i;
          k=i;
          break;
       }}p=0;
    while(t[j]!=0) {
       atr[p]=t[j]
       ;j--;
       p++;
    }atr[p]=t[j]; for(p=k+1;p<n+2;p++,k++) {atr[p]=t[k+1];
    }// Calculating the difference between adjacent tracksfor(j=0;j<n+1;j++) {
       if(atr[j]>atr[j+1]) {
          d[j]=atr[j]-atr[j+1];
       } else {
          d[j]=atr[j+1]-atr[j];
       }sum+=d[j];
    }printf("\nAverage header movements: %f",(float)sum/n);getch();
}
```

C) C-SCAN DISK SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<conio.h
>main()
{int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;clrscr();
    printf("Enter the total number of tracks: ");
    scanf("%d",&tot);
    printf("Enter the number of tracks to be traversed: ");
    scanf("%d",&n);
    printf("Enter the position of head: ");
    scanf("%d",&h);
    t[0]=0;
    t[1]=h;
    t[2]=tot-1;
    printf("Enter the tracks: ");
    for(i=3;i<n+3;i++)
    {scanf("%d",&t[i]);
    }for(i=0;i<=n+2;i++)
    {for(j=0;j<=(n+2)-i-1;j++)
       {if(t[j]>t[j+1])
          {temp=t[j]; t[j]=t[j+1]; t[j+1]=temp;
          }}}
    for(i=0;i<=n+2;i++)
    {if(t[i]==h)
       {j=i; break;
       }}p=0;
    while(t[j]!=tot-1)
    {atr[p]=t[j];j++;
       p++;
    }atr[p]=t[j];p++;
    i=0;
    while(p!=(n+3) && t[i]!=t[h])
    {atr[p]=t[i];i++;
       p++;
    }for(j=0;j<n+2;j++)
    {if(atr[j]>atr[j+1])
       {d[j]=atr[j]-atr[j+1];
       }else
       {d[j]=atr[j+1]-atr[j];
       }sum+=d[j];}
    printf("\nTotal header movements: %d",sum);
    printf("\nAverage header movements: %f",(float)sum/n);
    getch();
}
```

RESULT :

Thus Write C programs for the implementation of various disk scheduling algorithms.

EX NO 16	INSTALL ANY GUEST OPERATING SYSTEM USING VMWARE

AIM: To install Kali Linux on Windows using virtual machine software called VM ware.

PROCEDURE :

1 Download and install a virtual machine software such as VMware on your Windows machine.

2 Download the Kali Linux ISO files from the official Kali Linux website.

3 Open the virtual machine software and click on "New" to create a new virtual machine.

4 Follow the instructions to set up the virtual machine, such as specifying the amount of RAM and hard disk space to allocate.

5 When prompted, select the Kali Linux ISO file as the installation media.

6 Complete the Kali Linux installation process within the virtual machine, following the prompts.

7 Once the installation is complete, start the Kali Linux virtual machine from the virtualization software.

8 You should now have Kali Linux running as a virtual machine on your Windows computer.

RESULT :

Thus Installation Of Any Guest Operating System Using VM ware Was Successfully Verified