KNOWLEDGE INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai

Kakapalayam (Po), Salem – 637 504

RECORD NOTE BOOK

REG. NO.

Certified that this is the bonafide record of work done by Selvan/Selvi …… of the …… Semester……Branch during the year …… in the ……..... Laboratory.

Staff – Incharge **Head of the Department**

CONTENTS

Ex.No:01(a)	INSTALLATION OF WINDOWS OPERATING SYSTEM
Date:	

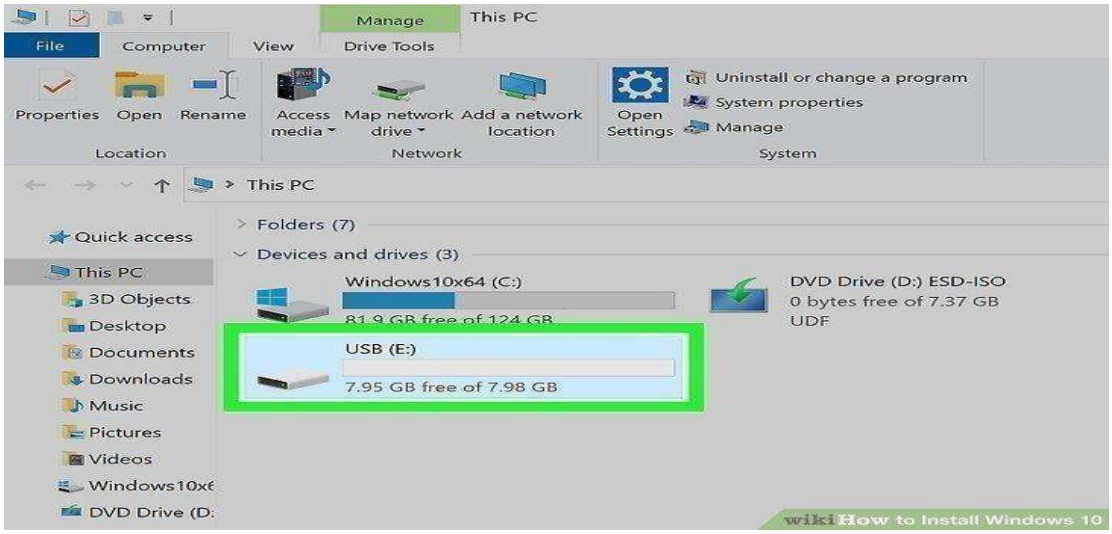
AIM:

To write a step by step process of windows operating system installation.

Installation processes:

Step1: Creating an installation Disc or Drive

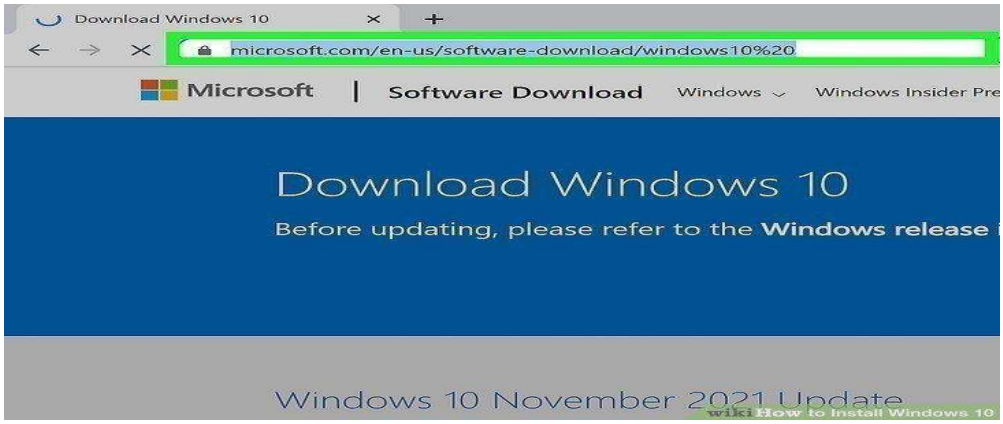
Connect a blank USB flash drive or insert a blank writable DVD. You can install Windows 10 by creating a bootable USB flash drive or DVD that contains the Windows 10 installation files. You'll need a USB flash drive that's at least 8GB, or any blank DVD to get started.



Step-2: Make sure you have the 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

- 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.
- 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.

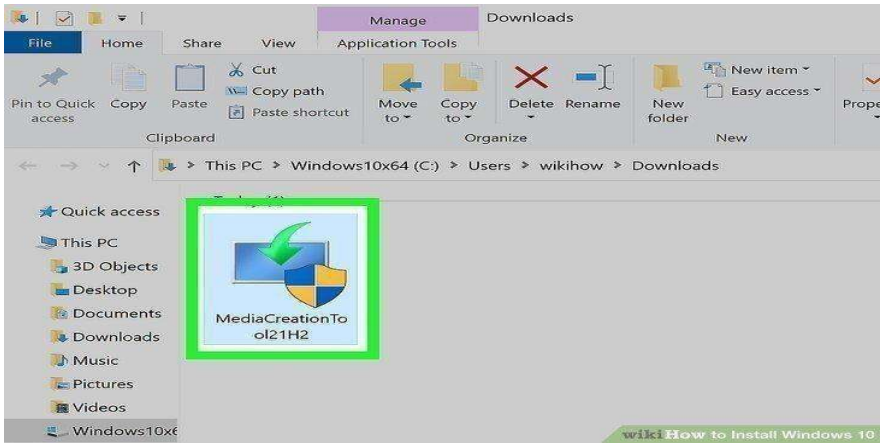


Step:3

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

Step: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).



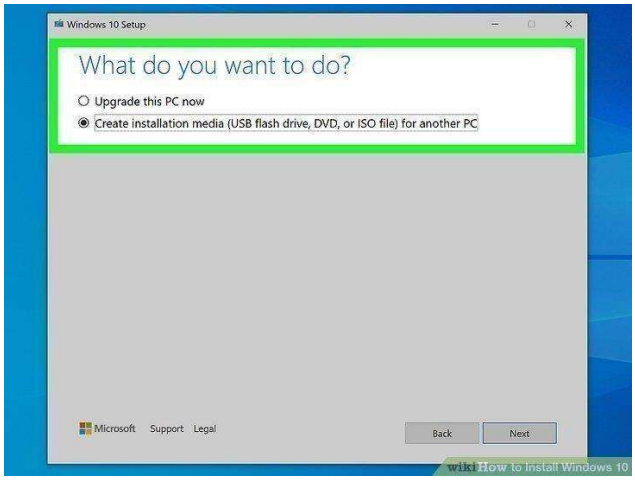
Step:5

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.

- Click **Yes** when prompted to allow the installer to run.

Step:6

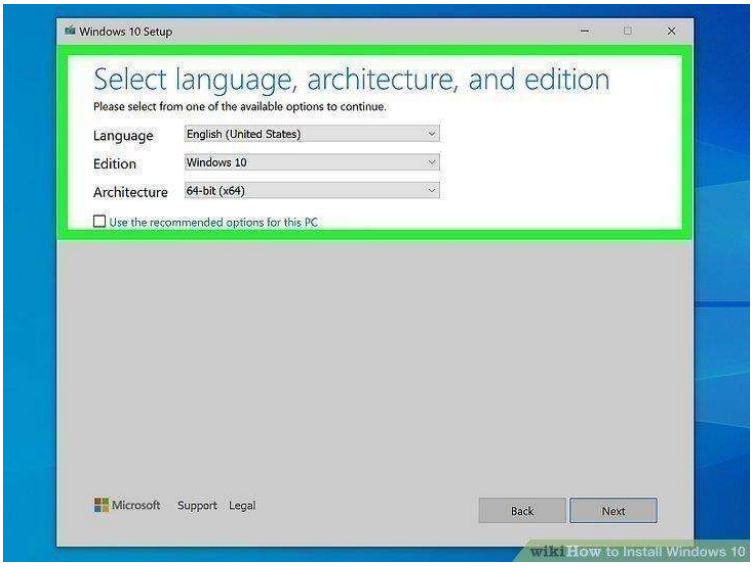
Click Accept to accept the license. It's in the bottom-right corner of the window.



Step-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.

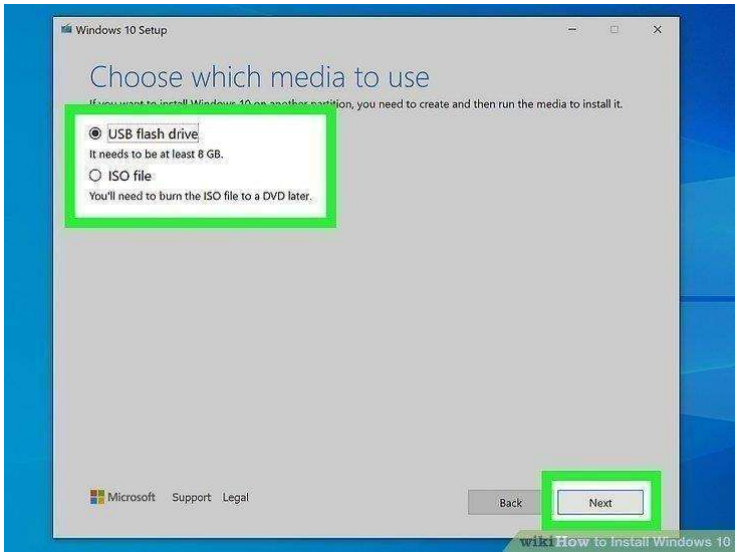
- If you're updating your PC from an earlier version of Windows, select **Upgrade this PC now** instead, and then follow the on-screen instructions to install Windows 10. You're done!



Step-8:

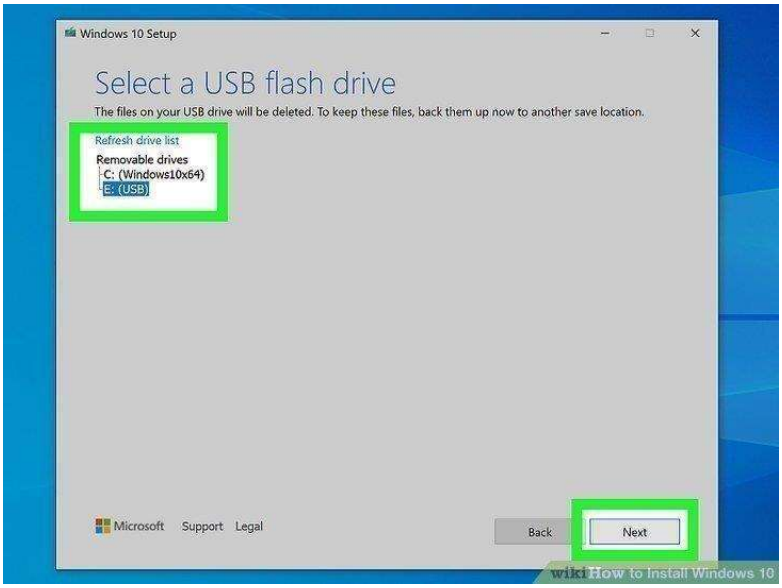
Select your preferences and click Next. If you're installing Windows on the current PC, you can keep the default options. If you need to install on a different PC, make sure you choose the language and edition for which you have a license, and select the architecture (64-bit or 32-bit) that matches the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

- If you're not sure about the architecture, you can choose **Both** from the menu.



Step-09:

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.



Step-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.

Step-11:

Booting from Windows 10 Installation Media

- **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.

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
    - Microsoft Surface Tablets: Press and hold the volume-up button.
    - Samsung and Toshiba: F2
    - Sony: F1, F2, or F3

Step-12:

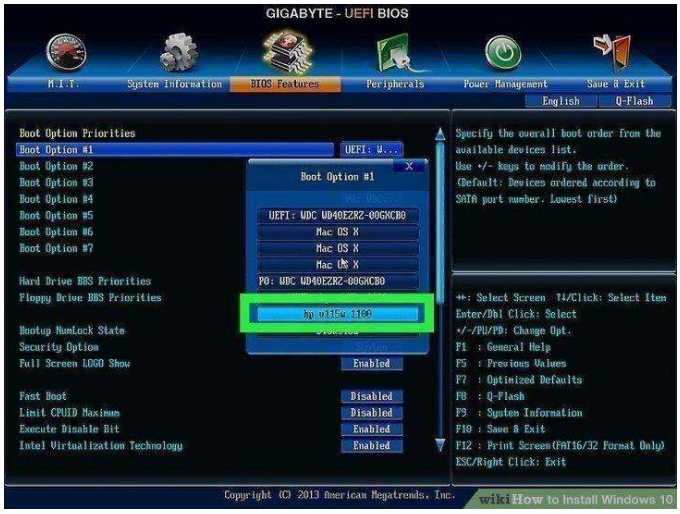
Go to the Boot tab. You'll use the arrow keys to select it.

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

Step-13:

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



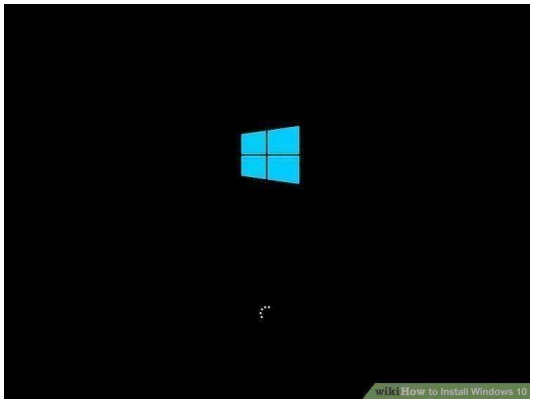
Step-14:

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.

Step-15:

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.

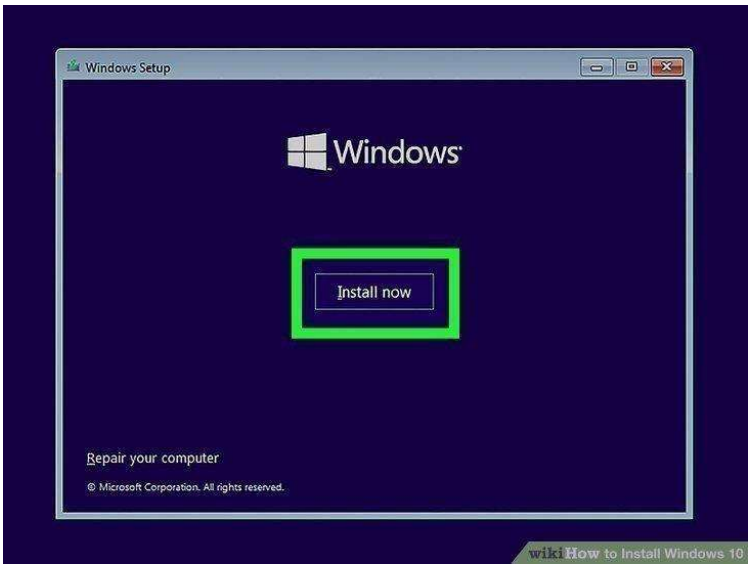


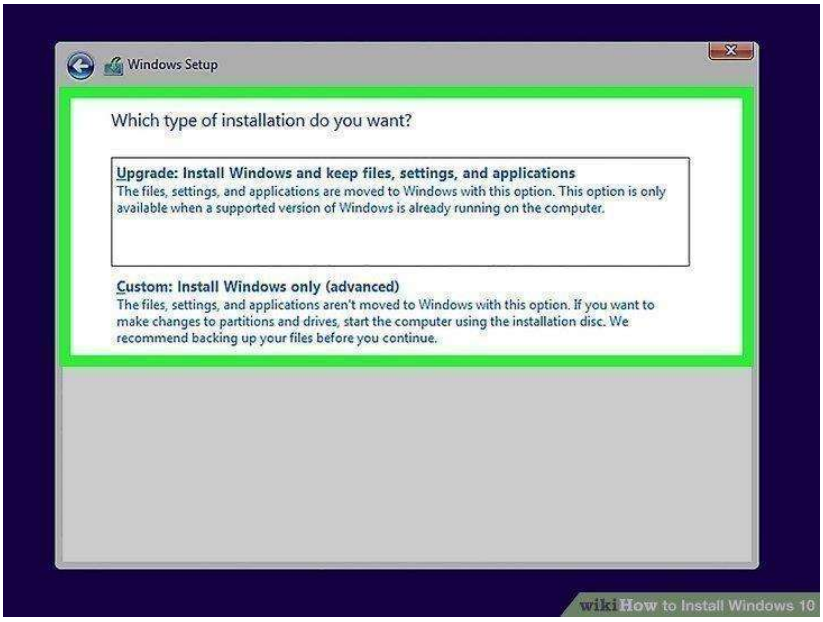
Step-16:

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.

Step-17:

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





Step-18:

Follow the on-screen instructions to install Windows 10. You'll be asked to perform a few tasks, such as connecting to Wi-Fi and choosing some preferences. Once the installation is complete, you'll have a fresh new installation of Windows 10.

- If you're upgrading from an earlier version of Windows, you'll be asked if you want to upgrade the current operating system or do a custom install. If you choose **Upgrade**, you'll preserve existing apps and files.

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result:

Thus the windows operating system was installed and configured successfully.

Ex.No:01(b)	INSTALLATION OF LINUX OPERATING SYSTEM
Date:	

AIM:

To install any guest operating system like LINUX using VMWare.

PROCEDURE:

To create a virtual machine in vCenter Server for each remote desktop that is deployed in a Horizon 8 environment. You must install your Linux distribution on the virtual machine.

Prerequisites

- Verify that your deployment meets the requirements for supporting Linux desktops. See System Requirements for Horizon Agent for Linux.
- Familiarize yourself with the steps for creating virtual machines in vCenter Server and installing guest operating systems. For more information see the *Windows Desktops and Applications in Horizon* document.
- Familiarize yourself with the video memory (vRAM) settings requirements for the monitors you plan to use with the virtual machine. See System Requirements for Horizon Agent for Linux.

1. In vSphere Client, create a virtual machine.
2. Configure custom configuration options.
    a. Right-click the virtual machine and click Edit Settings.
    b. Specify the number of vCPUs and the vMemory size. For the required settings, refer to the following guidelines.
        - If you are preparing the virtual machine for deployment as a single-session virtual desktop pool, follow the guidelines in the installation guide for your Linux distribution.

          For example, Ubuntu 18.04 specifies configuring 2048 MB for vMemory and 2 vCPUs.

        - If you are preparing the virtual machine to serve as a multi-session host for a published desktop or application pool, specify at least 8 vCPUs and 40 GB of vMemory.

Power on the virtual machine and install the required Linux distribution. Note the following considerations for instant-clone desktop pools and multi-session hosts.

Horizon Agent for Linux only supports instant-clone desktop pools created from virtual machines running the following operating systems:

c. Ubuntu 18.04/20.04/22.04

d. RHEL 7.x/8.x/9.x

e. CentOS 7.8/7.9

f. SLED/SLES 12.x/15.x

Only virtual machines running RHEL Workstation 7.8 or later, RHEL Workstation 8.1 or later, RHEL Workstation 9.0 or later, or Ubuntu 18.04/20.04/22.04 can support multi- session published desktop pools and single-session or multi-session application pools.

3. Configure the desktop environment to use for the specific Linux distribution.

   See the Desktop Environment section in System Requirements for Horizon Agent for Linux for additional information.

4. Ensure that the system hostname is resolvable to 127.0.0.1

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

Result

Thus guest operating system linux using VMWare has been successfully installed.

Ex.No:02	IMPLEMENTATION OF UNIX COMMAND & SHELL PROGRAMMING
Date:	

AIM:

Acquire operating systemskills through UNIX Commands.

UNIX SHELL:

A UNIX shell is a command-line interpreter that provides a user interface for the UNIX operating system. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute or by creating text scripts of one or more such commands

GENERAL COMMANDS:

date	Used to display the current system date andtime.
date +%D	Displays date only
date +%T	Displays time only
date +%Y	Displays the year part of date
date +%H	Displays the hour part of time
cal	Calendar of the current month
cal year	Displays calendar for all months of the specified year
cal month year	Displays calendar for the specified month of the year
who	Login details of all users such as their IP, Terminal No, Username
who am i	Used to display the login details of the user
tty	Used to display the terminal name (or) terminaltype
uname	Displays the Operating System
uname –r	Shows version number of the OS (kernel)
uname –n	Displays domain name of the server
echo "txt"	Displays the given text on the screen
echo $HOME	Displays the user's home directory
bc	Basic calculator. Press Ctrl+d to quit
lp -	file Allows the user to spool a job along with others in a print queue
history	To display the commands used by the user since logon.
Exit	Exit from a process. If shell is the only process then logsout
Clear	Clear the screen

DIRECTORY COMMANDS:

Pwd	Path of the present working directory
Mkdir dir	A directory is created in the name under the current directory
Mkdir dir1 dir2	A number of sub-directories can be created under one stroke
Cd subdir	Change Directory
Cd	To switch to the home directory.
cd ..	To move back to the parent directory
rmdir subdir	Removes an empty sub-directory.

FILE COMMANDS:

cat > filename	To create a file with some contents. At the end press Ctrl+d.(> indicates redirecting output to a file.)
cat filename	Displays the file contents.
cat >> filename	Used to append contents to a file
cp src des	Copy files to given location. If already exists, it will be overwritten
cp –i src des	Warns the user prior to overwriting the destination file
cp –r src des	Copies the entire directory, all its sub-directories and files.
mv old new	To rename an existing file or directory. –i option can also be used
mv f1 f2 f3 dir	To move a group of files to a directory.
rm *	To delete all the files in thedirectory.
rm –r *	Deletes all files and sub-directories
rm –f *	To forcibly remove even write-protectedfiles
ls	Lists all files and subdirectories (blue colored) in sorted manner.
ls name	To check whether a file or directoryexists.
ls name*	Short-hand notation to list out filenames of a specificpattern.
ls –a	Lists all files including hidden files (files beginning with .)
–l	Long listing showing file access rights (read/write/execute-rwx for user/group/others-ugo).
cmp file1 file2	Used to compare two files. Displays nothing if files are empty
wc	Produces the counts of lines (l), words(w), and characters(c).
chmod	Perform file Changes permission for the specified file.
who | wc –l	To terminate the unix session execute the command exit or logout
head	Print the first N number of lines.
tail	Print the last N number of lines.
new file	Splits screen into multiple windows and opens thefile.
finger	Gathers and displays the information about theusers.
grep	Used to search and print specified patterns froma file.
uniq	Fetches one copy of the redundant records writing them tothe standardoutput.
diff	Difference between the two files

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the basics of UNIX commands are studied by executing the commands

Ex.No:3(a)	CPU SCHEDULING ALGORITHMS –
Date:	FIRST COME FIRST SERVER (FCFS)

AIM

Acquire operating system skills through CPU scheduling algorithms.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the variables.

Step 3: Get the number of process and their burst time of each process.

Step 4: Calculate the waiting time and turnaround time of each of the processes

Step 5: Calculate average waiting and average turnaround time of all the process.

Step 6: Display waiting time, turnaround time, average waiting time and turnaround time of the processes

Step 7: Stop the program.

PROGRAM:

```
#include <stdio.h>
void main() {
   int bt[20], wt[20], tat[20], i, n;
   float wtavg = 0.0, tatavg = 0.0;

   printf("\nEnter the number of processes: ");
   scanf("%d", &n);

   for(i = 0; i < n; i++) {
      printf("\nEnter Burst Time for Process %d: ", i + 1);
      scanf("%d", &bt[i]);
   }
   wt[0] = 0;
   tat[0] = bt[0];
   wtavg += wt[0];
```

```
    tatavg += tat[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];
        tatavg += tat[i];
    }
    printf("\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");
    for(i = 0; i < n; i++) {
        printf("\tP%d\t\t%d\t\t%d\t\t%d\n", i + 1, bt[i], wt[i], tat[i]);
    }
    printf("\nAverage Waiting Time: %.2f", wtavg / n);
    printf("\nAverage Turnaround Time: %.2f", tatavg / n);
}
```

INPUT

Enter the number of processes -- 3

Enter Burst Time for Process 0 – 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 – 3

OUTPUT

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

Average Waiting Time-- 17.000000

Average Turnaround Time -- 27.000000

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the C program using FCFS scheduling has been executed successfully.

Ex.No:3(b)	CPU SCHEDULING ALGORITHMS - SHORTEST JOB FIRST (SJF)
Date:	

AIM

Acquire operating system skills through CPU scheduling algorithms.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the variables.
Step 3: Get the number of process and their burst time of each process.
Step 4: Calculate the waiting time and turnaround time of each of the processes
Step 5: Calculate average waiting and average turnaround time of all the process.
Step 6: Display waiting time, turnaround time, average waiting time and turnaround time of the processes
Step 7: Stop the program.

PROGRAM:

```
#include <stdio.h>

void main() {
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float wtavg = 0.0, tatavg = 0.0;

    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]);
    }

    // Sorting burst time and process ID in ascending order of burst time
    for(i = 0; i < n; i++) {
        for(k = i + 1; k < n; k++) {
            if(bt[i] > bt[k]) {
                // Swap burst times
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;

                // Swap corresponding process IDs
                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 += wt[i];
        tatavg += tat[i];
    }
```

```
    printf("\n\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");

    for(i = 0; i < n; i++) {
        printf("\n\tP%d\t\t%d\t\t%d\t\t%d", p[i] + 1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f", wtavg / n);
    printf("\nAverage Turnaround Time: %.2f", tatavg / n);
}
```

INPUT

Enter the number of processes -- 4

Enter Burst Time for

Process 0 ------ 6

Enter Burst Time for

Process 1 ------ 8

Enter Burst Time for

Process 2 ------ 7

Enter Burst Time for

Process 3 ------ 3

OUTPUT

PROCESS

BURST TIMEWAITING TIME TURNAROUND TIME

P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time – -7.000000

Average Turnaround Time-13.000000

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Ex.No:3(c)	CPU SCHEDULING ALGORITHMS - ROUND ROBIN SCHEDULING ALGORITHM
Date:	

AIM:

Acquire programming skills for application development in real-world problem solving.

ALGORITHM:

Step 1: Start the program.

Step 2: Declare the variables.

Step 3: Get the number of process and their burst time of each process.

Step 4: Get the quantum time.

Step 5: Schedule the CPU to the first process, after that quantum time is over schedule the second process. This will continue until all processes in the system complete their turn.

Step 6: Draw the chart of the processes as Round Robin manner.

Step 7: Calculate waiting time and turnaround time of each process.

Step 8: Calculate average waiting and average turnaround time of all the process.

Step 9: Display waiting time, turnaround time, average waiting time and turnaround time of the processes.

Step 10: Stop the program.

PROGRAM:

```
#include <stdio.h>

void main() {
   int pt[10][10], a[10][10], at[10], pname[10][10], i, j, n, k = 0, q, sum = 0;
   float avg;

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

   for(i = 0; i < 10; i++) {
      for(j = 0; j < 10; j++) {
         pt[i][j] = 0;
         a[i][j] = 0;
      }
   }

   for(i = 0; i < n; i++) {
      j = 0;
      printf("\n\n Enter the process time for process %d: ", i + 1);
      scanf("%d", &pt[i][j]);
   }

   printf("\n\n Enter the time slice: ");
   scanf("%d", &q);
   printf("\n\n");

   for(j = 0; j < 10; j++) {
      for(i = 0; i < n; i++) {
         a[2 * j][i] = k;
         if((pt[i][j] <= q) && (pt[i][j] != 0)) {
            pt[i][j + 1] = 0;
            printf(" %d P%d %d\n", k, i + 1, k + pt[i][j]);
            k += pt[i][j];
            a[2 * j + 1][i] = k;
         } else if(pt[i][j] != 0) {
            pt[i][j + 1] = pt[i][j] - q;
            printf(" %d P%d %d\n", k, i + 1, (k + q));
            k += q;
            a[2 * j + 1][i] = k;
         } else {
```

```
            a[2 * j + 1][i] = 0;
        }
    }
}

for(i = 0; i < n; i++) {
    sum += a[0][i];
}

for(i = 0; i < n; i++) {
    for(j = 1; j < 10; j++) {
        if((a[j][i] != 0) && (a[j + 1][i] != 0) && ((j + 1) % 2 == 0)) {
            sum += ((a[j + 1][i] - a[j][i]));
        }
    }
}

avg = (float)sum / n;
printf("\n\n Average waiting time = %f msec", avg);
sum = avg = 0;

for(j = 0; j < n; j++) {
    i = 1;
    while(a[i][j] != 0) {
        i += 1;
    }
    sum += a[i - 1][j];
}

avg = (float)sum / n;
printf("\n\n Average turnaround time = %f msec\n\n", avg);
}
{ i+=1;
}
sum+=a[i-1][j];
}
avg=(float)sum/n;
printf("\n\n Average turnaround time = %f msec\n\n",avg);
}
```

OUTPUT:

```
[com39@localhost os]$ gcc ex5b.c
[com39@localhost os]$ ./a.out
Enter the number of processes : 3
Enter the process time for process 1 : 4
Enter the process time for process 2 : 6
Enter the process time for process 3 : 9
Enter the time slice : 2
0 P1 2
2 P2 4
4 P3 6
6 P1 8
8 P2 10
10 P3 12
12 P2 14
14 P3 16
16 P3 18
18 P3 19
Average waiting time = 7.333333 msec
Average turnaround time = 13.666667
msec [com39@localhost os]$
```

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

RESULT:

Thus the C program using Round Robin scheduling has been executed successfully

Ex.No:4	SEMAPHORES
Date:	

AIM:

To write a C program to implement semaphores.

ALGORITHM:

Step 1: Start the program

Step 2: Get the number of jobs from the user

Step3: When job1 is processing, job 2 is also starts processing

Step 4: When job 1 enters critical section, next job starts processing

Step 5: When job1 comes out of the critical section, the other job enters the critical section

Step 6: The above 3 steps are performed for various programs

Step 7: End the program.

PROGRAM:

```
#include<stdio.h>

main() {
   int i, a = 1, h = 2, n;

   printf("\n Enter the number of jobs: ");
   scanf("%d", &n);

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

      if(a == 1) {
         printf("Processing %d!\n", i + 1);
         a++;
      }

      if(h > 1) {
         if(i + 2 <= n) {
            printf("\nProcessing %d!\n", i + 2);
         }

         printf("\nProcess %d enters critical section", i + 1);
         printf("\nProcess %d leaves critical section", i + 1);
      }

      h++;
   }
}}
```

OUTPUT:

Enter the no of jobs 2

processing 1. !

processing 2. !

Process 1 Enters Critical section

Process 1 Leaves Critical section

Process 2 Enters Critical section

Process 2 Leaves Critical section

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

RESULT:

Thus the program to implement semaphores in c was executed and the

Ex.No:5	BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE
Date:	

AIM:

To implement banker's algorithm for Dead Lock Avoidance.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the values of resources and processes.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

Step 5: Check whether it's possible to allocate.

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

Step 7: Else system is not in safety state.

Step 8: If the new request comes then check that the system is insafety.

Step 9: Or not if we allow the request.

Step 10: Stop the program.

Step 11: End

PROGRAM:

```
#include <stdio.h>

void main() {
   int allocated[15][15], max[15][15], need[15][15], avail[15], tres[15], work[15], flag[15];
   int pno, rno, i, j, prc, count, total;

   count = 0;

   printf("\nEnter number of processes: ");
   scanf("%d", &pno);

   printf("\nEnter number of resources: ");
   scanf("%d", &rno);

   for(i = 1; i <= pno; i++) {
      flag[i] = 0; // Initialize flags
   }

   printf("\nEnter total numbers of each resource: ");
   for(i = 1; i <= rno; i++) {
      scanf("%d", &tres[i]);
   }

   printf("\nEnter Max resources for each process: ");
   for(i = 1; i <= pno; i++) {
      printf("\nFor process %d: ", i);
      for(j = 1; j <= rno; j++) {
         scanf("%d", &max[i][j]);
      }
   }

   printf("\nEnter allocated resources for each process: ");
   for(i = 1; i <= pno; i++) {
      printf("\nFor process %d: ", i);
      for(j = 1; j <= rno; j++) {
         scanf("%d", &allocated[i][j]);
      }
   }

   // Calculate available resources
   printf("\nAvailable resources:\n");
   for(j = 1; j <= rno; j++) {
      avail[j] = 0;
```

```
    for(i = 1; i <= pno; i++) {
        total += allocated[i][j];
    }
    avail[j] = tres[j] - total;
    work[j] = avail[j];
    printf("%d\t", work[j]);
}

// Start the safe sequence calculation using Banker's Algorithm
do {
    for(i = 1; i <= pno; i++) {
        for(j = 1; j <= rno; j++) {
            need[i][j] = max[i][j] - allocated[i][j];
        }
    }

    printf("\nAllocated Matrix | Max Matrix | Need Matrix\n");
    for(i = 1; i <= pno; i++) {
        printf("\n");
        for(j = 1; j <= rno; j++) {
            printf("%4d", allocated[i][j]);
        }
        printf("|");
        for(j = 1; j <= rno; j++) {
            printf("%4d", max[i][j]);
        }
        printf("|");
        for(j = 1; j <= rno; j++) {
            printf("%4d", need[i][j]);
        }
    }

    prc = 0;

    for(i = 1; i <= pno; i++) {
        if(flag[i] == 0) {
            prc = i;
            for(j = 1; j <= rno; j++) {
                if(work[j] < need[i][j]) {
                    prc = 0;
                    break;
                }
            }
        }
```

```
            break;
        }
    }

    if(prc != 0) {
        printf("\nProcess %d completed", prc);
        count++;
        printf("\nAvailable matrix: ");
        for(j = 1; j <= rno; j++) {
            work[j] += allocated[prc][j];
            allocated[prc][j] = 0;
            max[prc][j] = 0;
            flag[prc] = 1;
            printf("%d ", work[j]);
        }
    }

} while(count != pno && prc != 0);

if(count == pno) {
    printf("\nThe system is in a safe state!!");
} else {
    printf("\nThe system is in an unsafe state!!");
}
}
```

OUTPUT:

Enter number of process: 5

Enter number of resources: 3

Enter total numbers of each resource: 10 5 7

Enter Max resources for each process: for process 1:7 5 3

for process 2:3 2 2

for process 3:9 0 2

for process 4:2 2 2

for process 5:4 3 3

Enter allocated resources for each process: for process 1:0 1 0 for process 2:3 0 2

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

available resources: 2 Allocated matrix Max 3

need 0

0	1	0|	7	5	3|	7	4	3
3	0	2|	3	2	2|	0	2	0
3	0	2|	9	0	2|	6	0	0
2	1	1|	2	2	2|	0	1	1
0	0	2|	4	3	3|	4	3	1

Process 2 completed

Available matrix: 5 3 2 Allocated matrix Max need

0	1	0|	7	5	3|	7	4	3
0	0	0|	0	0	0|	0	0	0
3	0	2|	9	0	2|	6	0	0
2	1	1|	2	2	2|	0	1	1
0	0	2|	4	3	3|	4	3	1

Process 4 completed

Available matrix: 7 4 3 Allocated matrix Max need

0	1	0|	7	5	3|	7	4	3
0	0	0|	0	0	0|	0	0	0
3	0	2|	9	0	2|	6	0	0
0	0	0|	0	0	0|	0	0	0
0	0	2|	4	3	3|	4	3	1

Process 1 completed

Available matrix: 7 5 3 Allocated matrix Max need

0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
3	0	2|	9	0	2|	6	0	0
0	0	0|	0	0	0|	0	0	0
0	0	2|	4	3	3|	4	3	1

Process 3 completed

Available matrix: 10 5 5 Allocated matrix Max need

0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
0	0	0|	0	0	0|	0	0	0
0	0	2|	4	3	3|	4	3	1

Process 5 completed

Available matrix: 10 5 7 The system is in a safe state!!

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the program to implement banker's algorithm for Dead Lock Avoidance was executedand the output was verified.

Ex.No:6	IMPLEMENTATION OF THREADING
Date:	

AIM:

To learn about Threads and synchronization applications.

ALGORITHM:

Step 1: Start the program.

Step 2: Create number of threads.

Step 3: Counter is maintained for the user.

Step 4: Enter the logs of the jobs to start.

Step 5: Stop the program.

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

pthread_t tid[2];
int counter = 0;
pthread_mutex_t lock;

void* doSomething(void *arg)
{
   unsigned long i = 0;

   pthread_mutex_lock(&lock);

   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;

   pthread_mutex_init(&lock, NULL);

   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);

    pthread_mutex_destroy(&lock);

    return 0;
}
```

OUTPUT:

Job 1 started

Job 1 finished

Job 2 started

Job 2 finished

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the C program for threading and synchronization applications was created andexecuted successfully.

Ex.No:7	
Date:	IMPLEMENTATION OF PAGING TECHNIQUE

AIM:

To learn about paging techniques in memory management.

ALGORITHM

Step 1: Start the program.

Step 2: Input the pagesize and initialize memsize of page to 15.

Step 3: Compute number of pages by dividing memsize by pagesize.

Step 4: For the computed pages, give the page frames values.

Step 5: Get the logical address value. Compute frameno and offset.

Step 6: Using the page frameno and offset, Generate the physical address.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>

void main()
{
   int memsize=15;
   int pagesize, nofpage;
   int p[100];
   int frameno, offset;
   int logadd, phyadd;
   int i;
   int choice=0;

   printf("\nYour memsize is %d ", memsize);
   printf("\nEnter page size:");
   scanf("%d", &pagesize);
   nofpage = memsize / pagesize;

   for(i = 0; i < nofpage; i++)
   {
      printf("\nEnter the frame of page%d:", i + 1);
      scanf("%d", &p[i]);
   }

   do{
      printf("\nEnter a logical address:");
      scanf("%d", &logadd);
      frameno = logadd / pagesize;
      offset = logadd % pagesize;
      phyadd = (p[frameno] * pagesize) + offset;
      printf("\nPhysical address is: %d", phyadd);
      printf("\nDo you want to continue(1/0)?:");
      scanf("%d", &choice);
```

```
    } while(choice == 1);
}
```

Output:

Your memsize is 15 Enter page size: 5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address: 3

Physical address is: 13

Do you want to continue (1/0)? : 1

Enter a logical address: 1

Physical address is: 11

Do you want to continue (1/0)? : 0

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

RESULT:

Thus the c program for implementing paging technique of memory management using first fit was executed and the output was verified.

Ex.No:8(a)	IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHMS (LFU)
Date:	

AIM:

To learn about Page Replacement Algorithms.

ALGORITHM

Step 1: Start the program.

Step 2: Input the total length of reference strings, number of frames and individual reference strings from the user.

Step 3: For each reference string value, check its availability in the page frames. Step 4: If found, increment to next reference string.

Step 5: When the page is not in the frames, increment the number of page fault and remove the page whose page frequency is least in the reference strings.

Step 6: Display the number of faults.

Step 7: Stop the program**.**

PROGRAM

```
#include<stdio.h>

void main() {
   int rs[50], i, j, k, m, f, cntr[20], a[20], min, pf = 0;

   printf("\nEnter number of page references -- ");
   scanf("%d", &m);
   printf("\nEnter the reference strings -- ");
   for(i = 0; i < m; i++)
      scanf("%d", &rs[i]);

   printf("\nEnter the available no. of frames -- ");
   scanf("%d", &f);
   for(i = 0; i < f; i++) {
      cntr[i] = 0;
      a[i] = -1;
   }

   printf("\nThe Page Replacement Process is-- \n");
   for(i = 0; i < m; i++) {
      for(j = 0; j < f; j++) {
         if(rs[i] == a[j]) {
            cntr[j]++;
            break;
         }
      }

      if(j == f) {
         min = 0;
         for(k = 1; k < f; k++) {
            if(cntr[k] < cntr[min])
               min = k;
         }
         a[min] = rs[i];
         cntr[min] = 1;
         pf++;
      }

      for(j = 0; j < f; j++)
         printf("\t%d", a[j]);
      if(j == f)
         printf("\tPF No.-- %d", pf);
   }
```

```
    printf("\n\nTotal number of page faults -- %d", pf);
}
```

Output

Enter number of page references – 10

Enter the reference string --1 2 3 4 5 2 5 2 5 1 Enter the available no. of frames – 3

The Page Replacement Process is –

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

PF No. 8

Total number of pagefaults 8

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the program for Page replacement algorithms for LRU was executed and the output was verified

Ex.No:8(b)	IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHMS (FIFO)
Date:	

AIM:

To learn about Page Replacement Algorithms.

ALGORITHM

Step 1: Start the program.

Step 2: Input the total length of reference strings, number of frames and individualreference strings from the user.

Step 3: For each reference string value, check its availability in the page frames.

Step 4: If found, increment to next reference string.

Step 5: When the page is not in the frames, increment the number of page fault andremove the page that comes first in the FIFO algorithm.

Step 6: Display the number of faults.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>

void main() {
   int i, j, k, f, pf = 0, count = 0, rs[25], m[10], n;

   printf("\nEnter the length of reference string -- ");
   scanf("%d", &n);
   printf("\nEnter the reference string -- ");
   for(i = 0; i < n; i++)
      scanf("%d", &rs[i]);

   printf("\nEnter the number of frames -- ");
   scanf("%d", &f);
   for(i = 0; i < f; i++)
      m[i] = -1;

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

      if(k == f) {
         m[count++] = rs[i];
         pf++;
      }

      for(j = 0; j < f; j++)
         printf("\t%d", m[j]);

      if(k == f)
         printf("\tPF No. %d", pf);

      printf("\n");

      if(count == f)
         count = 0;
   }

   printf("\nThe number of Page Faults using FIFO are -- %d", pf);
}
```

OUTPUT

Enter the length of reference string – 20

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

The Page Replacement Process is –

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

The number of Page Faults using FIFO are-- 15

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the program for Page replacement algorithms for FIFO was executed and the output

Ex.No:8(c)	IMPLEMENTATION OF PAGE REPLACEMENT ALGORITHMS (LRU)
Date:	

AIM:

To learn about Page Replacement Algorithms**.**

ALGORITHM

Step 1: Start the program.

Step 2: Input the total length of reference strings, number of frames and individual reference strings from the user.

Step 3: For each reference string value, check its availability in the page frames.

Step 4: If found, increment to next reference string.

Step 5: When the page is not in the frames, increment the number of page fault and remove the page that is used very least in the reference strings.

Step 6: Display the number of faults.

Step 7: Stop the program.

PROGRAM

```
#include<stdio.h>

void main() {
   int i, j, k, min, rs[25], m[10], count[10], flag[25], n, f, pf = 0, next = 1;

   printf("Enter the length of reference strings -- ");
   scanf("%d", &n);
   printf("Enter the reference strings -- ");
   for(i = 0; i < n; i++) {
      scanf("%d", &rs[i]);
      flag[i] = 0;
   }

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

   for(i = 0; i < f; i++) {
      m[i] = -1; // Initialize frames with -1
      count[i] = 0; // Initialize reference count
   }

   printf("The Page Replacement Process is -- \n");

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

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

      if(!found) {
         // Find the page to replace using the Least Recently Used (LRU) method
         min = 0;
         for(k = 1; k < f; k++) {
            if(count[k] < count[min]) {
               min = k;
            }
         }
```

```
            m[min] = rs[i];
            count[min] = next++;
            pf++;
        }

        // Print current frame content
        for(j = 0; j < f; j++) {
            printf("\t%d", m[j]);
        }

        if(!found) {
            printf("\tPF No. %d", pf);
        }
        printf("\n");
    }

    printf("\nThe number of Page Faults are -- %d", pf);
}
```

OUTPUT

Enter the length of reference string -- 20

Enter the reference string -- 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 Enter the number of frames -- 3

The Page Replacement process is --

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

The number of page faults using LRU are 12

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT:

Thus the program for Page replacement algorithms for LRU was executed and the output

Ex.No:9(a)	IMPLEMENTATION OF FILE ALLOCATION TECHNIQUES
Date:	(SEQUENTIAL FILE ALLOCATION)

AIM:

To learn about File Allocation Techniques.

ALGORITHM:

Step 1: Start the program.

Step 2: Create a structure for building a fileTable.

Step 3: Input enter the number of files, file name, starting block and number ofblocks.

Step 4: Subsequent blocks are allocated one after the other in sequential approach.Search for the file created. if found returns file found. Else returns not found.

Step 5: Display the file allocation details.

Step 6: Stop the program.

PROGRAM:

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

void main() {
    int f[50], i, st, len, j, c, k, count = 0;

    // Initialize all blocks to 0 (unallocated)
    for(i = 0; i < 50; i++) {
        f[i] = 0;
    }

    printf("Files Allocated are : \n");

    while(1) {
        count = 0;
        printf("Enter starting block and length of file: ");
        scanf("%d%d", &st, &len);

        // Check if the file can be allocated
        for(k = st; k < (st + len); k++) {
            if(f[k] == 0) {
                count++;
            }
        }

        // If the file can be allocated
        if(len == count) {
            for(j = st; j < (st + len); j++) {
                if(f[j] == 0) {
                    f[j] = 1; // Allocate block
                    printf("%d\t%d\n", j, f[j]); // Print the allocated block
                }
            }
            if(j != (st + len - 1)) {
                printf("The file is allocated to disk\n");
            }
        } else {
            printf("The file is not allocated \n");
        }

        // Ask if the user wants to enter another file
        printf("Do you want to enter more file (Yes - 1 / No - 0)? ");
```

```
        if(c == 0) {
            exit(0); // Exit the program if no more files need to be allocated
        }
    }

    getch(); // Wait for a key press before exiting
}
```

OUTPUT:

Files Allocated are :

Enter starting block and length of files: 14 3

14 1

15 1

16 1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 1

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 4

The file is not allocated Do you want to enter more file(Yes - 1/No - 0)0

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

RESULT:

Thus the program to implement sequential file allocation was executed and the output was verified

Ex.No:9(b)	IMPLEMENTATION OF FILE ALLOCATION TECHNIQUES
Date:	(INDEXED FILE ALLOCATION)

AIM:

To learn about File Allocation Techniques.

ALGORITHM:

Step 1: Start the program.

Step 2: Create a structure for building a fileTable.

Step 3: Input enter the number of files, file name, starting block and number of blocks.

Step 4: Subsequent blocks are allocated as per the index details of the file in indexed approach. Search for the file created. if found returns file found. Else returns not found.

Step 5: Display the file allocation details.

Step 7: Stop the program.

PROGRAM

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

struct  fileTable  {
   char  name[20];
   int nob;
   int blocks[30];
} ft[30];

void main() {
   int i, j, n;
   char s[20];

   clrscr(); // Clears the screen (usually used in DOS-based systems)

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

   for(i = 0; i < n; i++) {
      printf("\nEnter file name %d: ", i + 1);
      scanf("%s", ft[i].name);

      printf("Enter number of blocks in file %d: ", i + 1);
      scanf("%d", &ft[i].nob);

      printf("Enter the blocks of the file: ");
      for(j = 0; j < ft[i].nob; j++) {
         scanf("%d", &ft[i].blocks[j]);
      }
   }

   printf("\nEnter the file name to be searched: ");
   scanf("%s", s);

   for(i = 0; i < n; i++) {
      if(strcmp(s, ft[i].name) == 0) {
         break;  // Break the loop if the file is found
      }
   }

   if(i == n) {
      printf("\nFile Not Found");
```

```
        printf("\nFILE NAME\tNO OF BLOCKS\tBLOCKS OCCUPIED");
        printf("\n%s\t\t%d\t", ft[i].name, ft[i].nob);

        for(j = 0; j < ft[i].nob; j++) {
            printf("%d, ", ft[i].blocks[j]);
        }
    }

    getch();  // Waits for the user to press a key before exiting
}
```

OUTPUT:

Enter no of files : 2 Enter file 1: A

Enter no of blocks in file 1: 4

Enter the blocks of the file 1: 12 23 9 4 Enter file 2: G

Enter no of blocks in file 2: 5

Enter the blocks of the file 2: 88 77 66 55 44 Enter the file to be searched : G

FILE NAME NO OF BLOCKS BLOCKS OCCUPIED G 5
88,77,66,55,44,

DEPARTMENT OF CSE		
PERFORMANCE	**30**	
OBSERVATION	**30**	
RECORD	**40**	
TOTAL	**100**	

RESULT

Thus the C program for indexed file allocation has been executed successfully.

Ex.No:9(c)	IMPLEMENTATION OF FILE ALLOCATION TECHNIQUES
Date:	(LINKED FILE ALLOCATION)

AIM:

To learn about File Allocation Techniques.

ALGORITHM:

Step 1: Start the program.

Step 2: Create a structure for building a fileTable.

Step 3: Input enter the number of files, file name, starting block and number of blocks.

Step 4: Subsequent blocks are allocated as per pointer value to next block of the file in linked approach. Search for the file created. if found returns file found. Else returns not found.

Step 6: Display the file allocation details.

Step 7: Stop the program.

PROGRAM

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

struct block {
    int bno;
   struct block *next;
};

struct  fileTable  {
   char  name[20];
   int nob;
   struct block *sb;
} ft[30];

void main() {
   int i, j, n;
   char s[20];
   struct block *temp;
   clrscr();
   printf("\n Enter No. of Files: ");
   scanf("%d", &n);

   // Loop through each file
   for(i = 0; i < n; i++) {
      printf("\n Enter File Name %d: ", i + 1);
      scanf("%s", ft[i].name);
      printf("Enter No. of Blocks in File %d: ", i + 1);
      scanf("%d", &ft[i].nob);

      // Allocate memory for the first block of the file
      ft[i].sb = (struct block*)malloc(sizeof(struct block));
      temp = ft[i].sb;

      printf("\n Enter the blocks of the File: ");
      scanf("%d", &temp->bno);
      temp->next = NULL;

      // Loop to create the remaining blocks
      for(j = 1; j < ft[i].nob; j++) {
         temp->next = (struct block*)malloc(sizeof(struct block));
```

```
            temp = temp->next;
            scanf("%d", &temp->bno);
        }
        temp->next = NULL; // End the list of blocks
    }

    // Search for a specific file
    printf("\n Enter the File Name to be Searched: ");
    scanf("%s", s);

    // Search for the file in the table
    for(i = 0; i < n; i++) {
        if(strcmp(s, ft[i].name) == 0) {
            break;
        }
    }

    // If the file is not found
    if(i == n) {
        printf("\n File not Found");
    } else {
        // File found, display the file details
        printf("\n File Name \t No. of Blocks \t Blocks Occupied");
        printf("\n %s \t %d \t", ft[i].name, ft[i].nob);

        // Display the blocks occupied by the file
        temp = ft[i].sb;
        for(j = 0; j < ft[i].nob; j++) {
            printf("%d ->", temp->bno);
            temp = temp->next;
        }
    }

    getch(); // Waits for the user to press a key before exiting
}
```

OUTPUT:

Enter no of files: 2 Enter file 1: A

Enter no of blocks in file1: 4

Enter the blocks of the file 1: 12 23 9 4 Enter file 2: G

Enter no of blocks in file2: 5

Enter the blocks of the file 2: 88 77 66 55 44 Enter the file to be searched: G

FILE NAME NO OF BLOCKS BLOCKS OCCUPIED

G 5 88-> 77-> 66-> 55-> 44->

DEPARTMENT OF CSE		
PERFORMANCE	30	
OBSERVATION	30	
RECORD	40	
TOTAL	100	

RESULT

Thus the C program for linked file allocation has been executed successfully.