CASE STUDY:

Perform a case study by installing and exploring various types of operating systems on a physical or logical (virtual) machine. (Linux Installation).

Instructions to Install Ubuntu Linux 12.04 (LTS) along with Windows

Back Up Your Existing Data!

This is highly recommended that you should take backup of your entire data before start with the installation process.

Obtaining System Installation Media

Download latest Desktop version of Ubuntu from this link:

http://www.ubuntu.com/download/desktop

Booting the Installation System

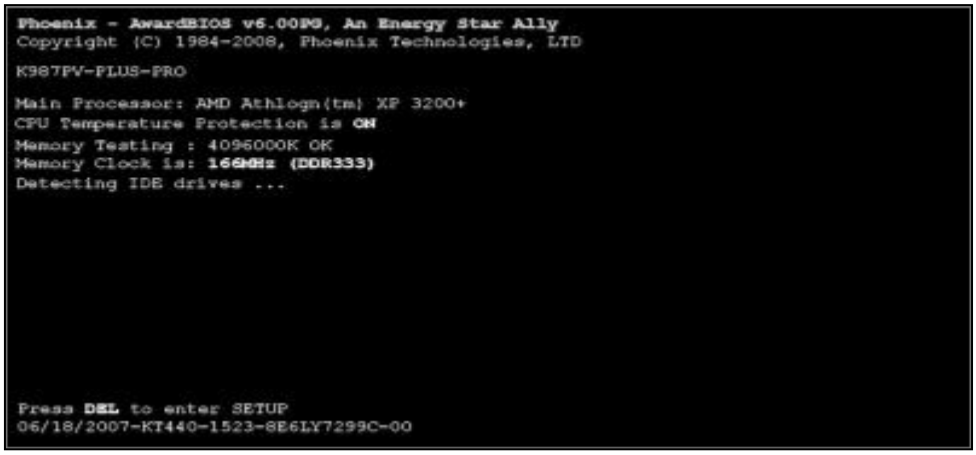
There are several ways to boot the installation system. Some of the very popular ways are , Booting from a CD ROM, Booting from a USB memory stick, and Booting from TFTP.

Here we will learn how to boot installation system using a CD ROM.

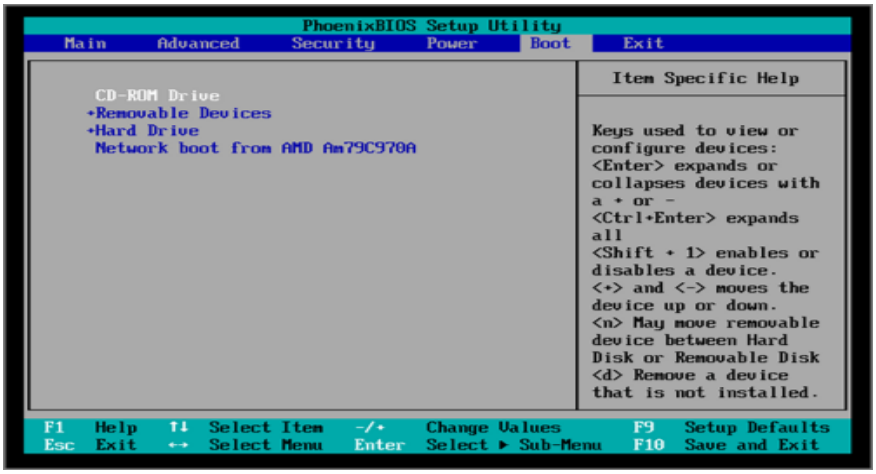
Before booting the installation system, one need to change the boot order and set CD-ROM as first boot device.

Changing the Boot Order of a Computers

As your computer starts, press the DEL, ESC, F1, F2, F8 or F10 during the initial startup screen. Depending on the BIOS manufacturer, a menu may appear. However, consult the hardware documentation for the exact key strokes. In my machine, its DEL key as shown in following screen-shot.



2. Find the Boot option in the setup utility. Its location depends on your BIOS.
   Select the Boot option from the menu, you can now see the options Hard Drive, CD-ROM Drive, Removable Devices Disk etc.
3. Change the boot sequence setting so that the CD-ROM is first. See the list of “Item Specific Help” in right side of the window and find keys which is used to toggle to change the boot sequence.

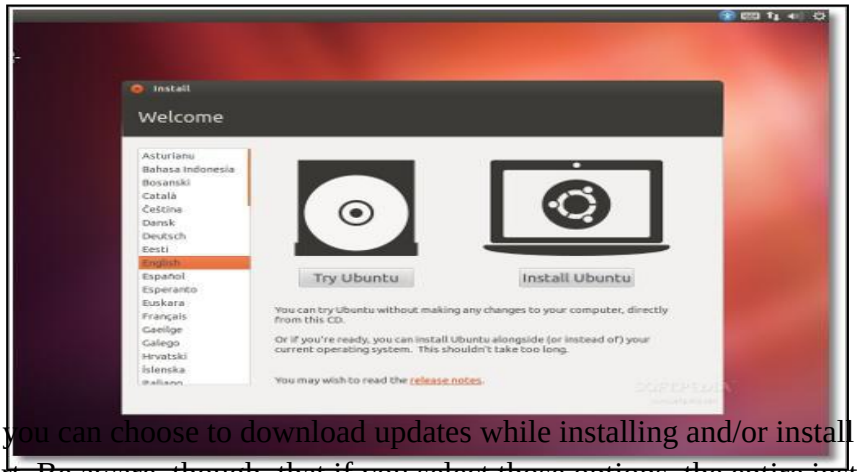


4. Insert the Ubuntu Disk in CD/DVD drive.
5. Save your changes. Instructions on the screen tell you how to save the changes on your computer.

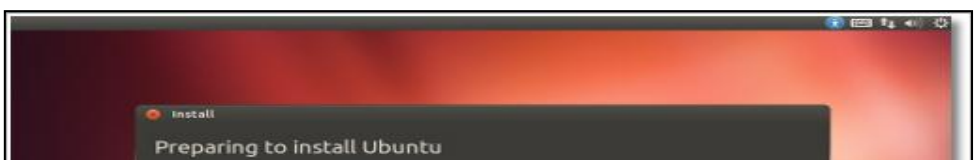
   The computer will restart with the changed settings.

Machine should boot from CD ROM, Wait for the CD to load...

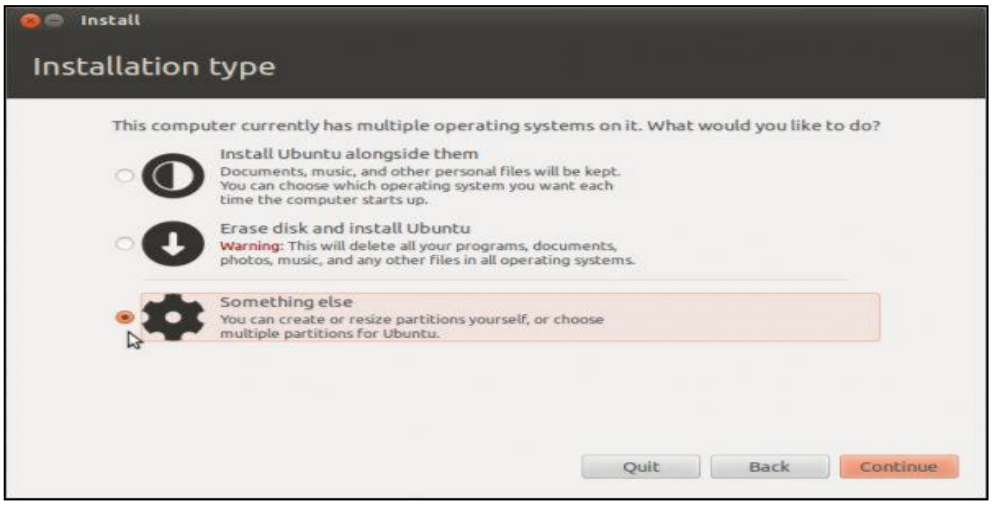
In a few minutes installation wizard will be started. Select your language and click the "Install Ubuntu" button to continue...



Optionally, you can choose to download updates while installing and/or install third party software, such as MP3 support. Be aware, though, that if you select those options, the entire installation process will be longer!



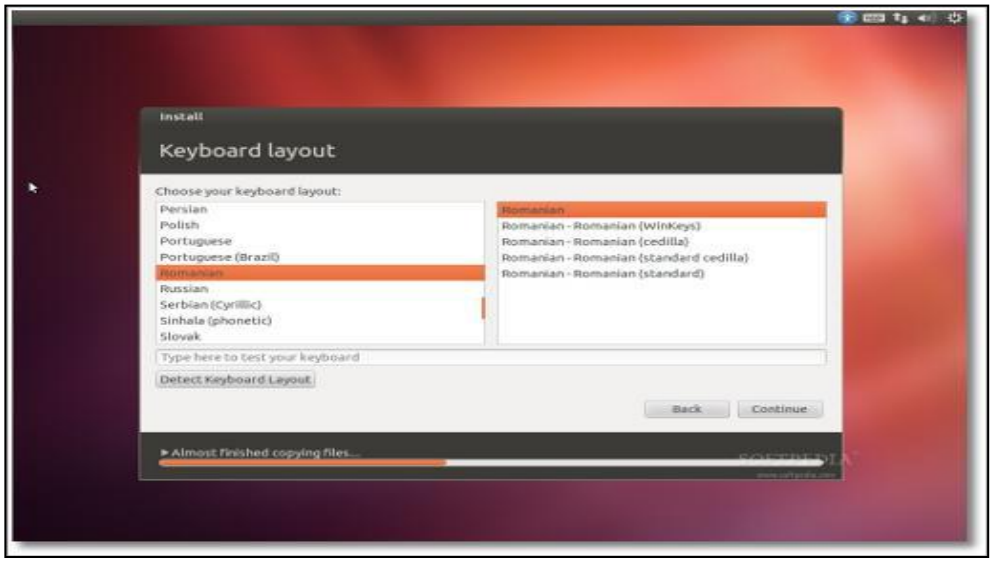
Since we are going to create partitions manually, select **Something else**, then click **Continue**. Keep in mind that even if you do not want to create partitions manually, it is better to select the same option as indicated here. This would insure that the installer will not overwrite your Windows , which will destroy your data. The assumption here is that **sdb** will be used just for Ubuntu 12.04, and that there are no valuable data on it.



Where are you? Select your location and Click the "Continue" button.

Keyboard layout

Select your keyboard layout and UK (English) and Click on “Continue” button.



Who are you?

Fill in the fields with your real name, the name of the computer (automatically generated, but can be overwritten), username, and the password.

Also at this step, there's an option called "Log in automatically." If you check it, you will automatically be logged in to the Ubuntu desktop without giving the password.

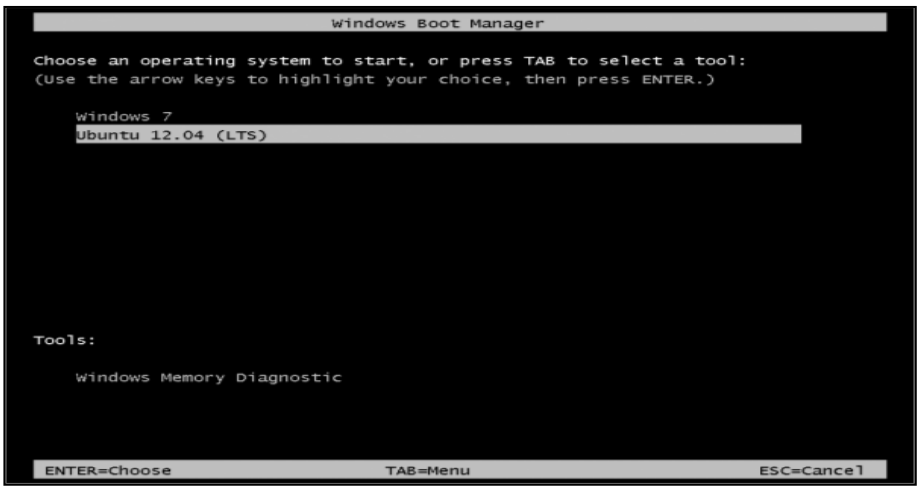
Option "Encrypt my home folder," will encrypt your home folder. Click on the "Continue" button to continue...

Now Ubuntu 12.04 LTS (Precise Pangolin) operating system will be installed.

It will take approximately 10-12 minutes (depending on computer's speed), a pop-up window will appear, notifying you that the installation is complete, and you'll need to restart the computer in order to use the newly installed Ubuntu operating system. Click the "Restart Now" button.

Please remove the CD and press the "Enter" key to reboot. The computer will be restarted. In a few seconds, you should see Windows 7′s boot menu with two entires listed – Windows 7 and Ubuntu 12.04 (LTS).

Then you may choose to boot into Windows 7 or Ubuntu 12.04 using the UP/Down arrow key.



Please select Ubuntu 12.04 (LTS) and press Enter to boot the machine in Ubuntu 12.04 Linux.

Here you can see the users on the machine, Click on the user name and enter the password and press Enter key to login.

We have successfully install and login to Ubuntu 12.04 LTS.

List Of Experiments (preferred programming language is C)

1. **Write a C programs to implement UNIX system calls and file management.**

a) Fork () System Call:

SOURCE CODE:

```
/* fork system call */
#include<stdio.h>
#include <unistd.h>
#include<sys/types.h>
int main()
{
        int childid;
        if((childid=fork())>0)
        {
                printf("\n i am in the parent process %d",getpid());
                printf("\n i am in the parent process %d\n",getppid());
        }
        else
        {
                printf("\n i am in the child process %d",getpid());
                printf("\n i am in the child process %d",getppid());
        }
}
```

OUTPUT:

```
gedit fork.c
cc fork.c
./a.out

 i am in the parent process 2011
 i am in the parent process 1531

 i am in the child process 2012
 i am in the child process 2011
```

b) Wait () and Exit () System Calls:

SOURCE CODE:

```
// wait() and exit() system calls
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main( )
{
        int i, pid;
        pid=fork( );
        if(pid== -1)
        {
                printf("fork failed");
                exit(0);
        }
        else if(pid==0)
        {
                printf("\n Child process starts");
                for(i=0; i<5; i++)
                {
                        printf("\n Child process %d is called", i);
                }
                printf("\n Child process ends");
        }
        else
        {
                wait(0);
                printf("\n Parent process ends");
        }
        exit(0);
```

}

OUTPUT:

gedit wait.c
cc wait.c
./a.out

Child process starts
Child process 0 is called
Child process 1 is called
Child process 2 is called
Child process 3 is called
Child process 4 is called
Child process ends
Parent process ends

c) Execl () System Call:

SOURCE CODE:

```
// execl() system call.

#include <unistd.h>

 int main(void)

 {

  char *binaryPath = "/bin/ls";

  char *arg1 = "-l";

  char *arg2 = "/home/student/lalitha/dir";

  execl(binaryPath, binaryPath, arg1, arg2, NULL);

  return 0;

}
```

OUTPUT:

```
gedit execl.c
cc execl.c
./a.out
total 0
-rw-rw-r-- 1 student student 0 May  2 09:53 f1
-rw-rw-r-- 1 student student 0 May  2 09:53 f2
-rw-rw-r-- 1 student student 0 May  2 09:53 f3
-rw-rw-r-- 1 student student 0 May  2 09:53 file
```

d) Exclp () System Call:

SOURCE CODE:

```
// execlp() system call.

#include <unistd.h>

int main(void)

{

  char *programName = "ls";

  char *arg1 = "-lh";

  char *arg2 = "/home/student/lalitha/dir";

  execlp(programName, programName, arg1, arg2, NULL);

  return 0;

}
```

OUTPUT:

```
gedit execlp.c
cc execlp.c
./a.out
total 0
-rw-rw-r-- 1 student student 0 May  2 09:53 f1
-rw-rw-r-- 1 student student 0 May  2 09:53 f2
-rw-rw-r-- 1 student student 0 May  2 09:53 f3
-rw-rw-r-- 1 student student 0 May  2 09:53 file
```

e) Execv () System Call:

SOURCE CODE:

```
// execv() system call.

#include <unistd.h>

 int main(void)

 {

        char *binaryPath = "/bin/ls";

        char *args[] = {binaryPath, "-l", "/home/student/lalitha/dir", NULL};

        execv(binaryPath, args);

        return 0;

}
```

OUTPUT:

```
gedit execv.c
cc execv.c
./a.out
total 0
-rw-rw-r-- 1 student student 0 May  2 09:53 f1
-rw-rw-r-- 1 student student 0 May  2 09:53 f2
-rw-rw-r-- 1 student student 0 May  2 09:53 f3
-rw-rw-r-- 1 student student 0 May  2 09:53 file
```

f) Zombie Process:

SOURCE CODE:

```
// Zombie Process
// Child becomes Zombie as Parent is sleeping when Child Process exits.
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
int main()
{
	// Fork returns Process id in Parent Process
	pid_t child_pid = fork();
	// Parent Process
	if(child_pid > 0)
	{
		printf("In Parent Process");
		sleep(10);
	}
	// Child Process
	else
		exit(0);
	return 0;
}
```

OUTPUT:

```
gedit zombie.c
cc zombie.c
./a.out
In Parent Process
```

g) Orphan Process:

SOURCE CODE:

```
// Orphan Process
// Parent Process finishes execution while the Child Process is running.
// The Child Process becomes Orphan.
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        // Create a child Process.
        int pid = fork();
        if (pid > 0)
        {
                printf("\n In Parent Process");
                exit(0);
        }
        // Note that pid is 0 in Child Process and negative if fork() fails.
        else if (pid == 0)
        {
                sleep(10);
                printf("\n In Child Process");
        }
        return 0;
}
```

OUTPUT:

```
gedit orphan.c
cc orphan.c
./a.out

 In Parent Process
 In Child Process
```

h) File Management:

SOURCE CODE:

```
// C Program to implement File Management.
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<fcntl.h>
int main( )
{
        int fd[2];
        char buf1[25]= "just a test\n";
        char buf2[50];
        fd[0]=open("file1", O_RDWR);
        fd[1]=open("file2", O_RDWR);
        write(fd[0], buf1, strlen(buf1));
        printf("\n Enter the text nowâ€¦.");
        gets(buf1);
        write(fd[0], buf1, strlen(buf1));
        lseek(fd[0], SEEK_SET, 0);
        read(fd[0], buf2, sizeof(buf1));
        write(fd[1], buf2, sizeof(buf2));
        close(fd[0]);
        close(fd[1]);
        printf("\n");
        return 0;
}
```

OUTPUT:

```
gedit filemanagement.c
cc filemanagement.c
./a.out

 Enter the text now…. Hello World
cat file1
just a test
Hello World
cat file2
just a test
Hello World
```

i) Directory Hierarchy:

SOURCE CODE:

```
/* Recursively descend a directory hierarchy pointing a file */
#include<stdio.h>
#include<dirent.h>
#include<errno.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc,char *argv[])
{
        struct dirent *direntp; DIR *dirp; if(argc!=2)
        {
                printf("ussage %s directory name \n",argv[0]);
                return 1;
        }
        if((dirp=opendir(argv[1]))==NULL)
        {
                perror("Failed to open directory \n");
                return 1;
        }
        while((direntp=readdir(dirp))!=NULL)
                printf("%s\n",direntp->d_name);
        while((closedir(dirp)==-1)&&(errno==EINTR));
                return 0;
}
```

OUTPUT:

```
gedit direc.c
cc direc.c
./a.out dir
f1
f2
..
.
f3
file
```

2. Write C programs to demonstrate various thread related concepts.
SOURCE CODE:

```
/* Implementing a program using thread */
#include<pthread.h>
#include<stdio.h>
#define NUM_THREADS 3
int je,jo,evensum=0,sumn=0,oddsum=0,evenarr[50],oddarr[50];
void *Even(void *threadid)
{
        int i,n;
        je=0;
        n=(int)threadid;
        for(i=1;i<=n;i++)
        {
                if(i%2==0)
                {
                        evenarr[je]=i;
                        evensum=evensum+i;
                        je++;
                }
        }
}
void *Odd(void *threadid)
{
        int i,n;
        jo=0;
        n=(int)threadid;
        for(i=0;i<=n;i++)
        {
                if(i%2!=0)
                {
                        oddarr[jo]=i;
                        oddsum=oddsum+i;
                        jo++;
                }
        }
}
void *SumN(void *threadid)
{
        int i,n;
        n=(int)threadid;
        for(i=1;i<=n;i++)
        {
                sumn=sumn+i;
        }
}
int main()
{
        pthread_t threads[NUM_THREADS];
        int i,t;
        printf("Enter a number\n");
        scanf("%d",&t);
        pthread_create(&threads[0],NULL,Even,(void *)t);
        pthread_create(&threads[1],NULL,Odd,(void *)t);
        pthread_create(&threads[2],NULL,SumN,(void *)t);
```

```
	for(i=0;i<NUM_THREADS;i++)
	{
		pthread_join(threads[i],NULL);
	}
	printf("The sum of first N natural nos is %d\n",sumn);
	printf("The sum of first N even natural nos is %d\n",evensum);
	printf("The sum of first N odd natural nos is %d\n",oddsum);
	printf("The first N even natural nos is----\n");
	for(i=0;i<je;i++)
		printf("%d\n",evenarr[i]);
	printf("The first N odd natural nos is----\n");
	for(i=0;i<jo;i++)
		printf("%d\n",oddarr[i]);
	pthread_exit(NULL);
}
```

OUTPUT:

```
gedit threadf.c
cc threadf.c -pthread
./a.out
Enter a number
12
The sum of first N natural nos is 78
The sum of first N even natural nos is 42
The sum of first N odd natural nos is 36
The first N even natural nos is----
2
4
6
8
10
12
The first N odd natural nos is----
1
3
5
7
9
11
```

3. Write a C Programs to Simulate the Following CPU Scheduling Algorithms.

a) FCFS CPU Scheduling Algorithm.

SOURCE CODE:

```
/* A program to simulate the FCFS CPU scheduling algorithm */
#include<stdio.h>
#include<string.h>
int main()
{
    int pn[10];
    int at[10],bt[10],st[10],fn[10],tat[10],wt[10],i,n;
    int totwt=0,tottat=0;
    printf("Enter the number of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the Process ID, Arrival Time & Burst Time:");
        scanf("%d%d%d",&pn[i],&at[i],&bt[i]);
    }
    for(i=0;i<n;i++)
    {
        if(i==0)
        {
            st[i]=at[i];
            wt[i]=st[i]-at[i];
            fn[i]=st[i]+bt[i];
            tat[i]=fn[i]-at[i];
        }
        else
        {
            st[i]=fn[i-1];
            wt[i]=st[i]-at[i];
            fn[i]=st[i]+bt[i];
```

```
                tat[i]=fn[i]-at[i];
            }
        }
        printf("\nPID  Arrtime  Burtime  Start  TAT  Finish        Wait");
        for(i=0;i<n;i++)
        {
            printf("\n%d\t%6d\t%6d\t%6d\t%6d\t%6d\t%6d",pn[i],at[i],bt[i],st[i],tat[i],fn[i],wt[i]);
            totwt+=wt[i];
            tottat+=tat[i];
        }
        printf("\nAverage Waiting time:%f",(float)totwt/n);
        printf("\nAverage Turn Around Time:%f",(float)tottat/n);
    }
```

OUTPUT:

gedit fcfs.c

cc fcfs.c

./a.out

Enter the number of processes: 3

Enter the Process Name, Arrival Time & Burst Time: 1 2 3

Enter the Process Name, Arrival Time & Burst Time: 2 5 6

Enter the Process Name, Arrival Time & Burst Time: 3 6 7

PName	Arrtime	Burtime	Start	TAT	Finish
1	2	3	2	3	5
2	5	6	5	6	11
3	6	7	11	12	18

Average Waiting time: 1.666667

Average Turn Around Time: 7.000000

b) SJF CPU Scheduling Algorithm.

SOURCE CODE:

```
/* A program to simulate the SJF CPU scheduling algorithm */
#include<stdio.h>
#include<string.h>
int main()
{
    int i=0,pno[10],bt[10],n,wt[10],temp=0,j,tt[10];
    float sum,at;
    printf("\n Enter the no of process ");
    scanf("\n %d",&n);
    printf("\n Enter the burst time of each process");
    for(i=0;i<n;i++)
    {
        printf("\n p%d",i);
        scanf("%d",&bt[i]);
    }
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(bt[i]>bt[j])
            {
                temp=bt[i];
                bt[i]=bt[j];
                bt[j]=temp;
                temp=pno[i];
                pno[i]=pno[j];
                pno[j]=temp;
            }
        }
    }
```

```
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=bt[i-1]+wt[i-1];
        sum=sum+wt[i];
    }
    printf("\n Process No \t Burst Time\t Waiting Time \t Turn Around Time\n");
    for(i=0;i<n;i++)
    {
        tt[i]=bt[i]+wt[i];
        at+=tt[i];
        printf("\n p%d\t\t%d\t\t%d\t\t%d",i,bt[i],wt[i],tt[i]);
    }
    printf("\n\n\t Average waiting time%f\n\t Average turn around time%f", sum/n, at/n);
}
```

OUTPUT:

gedit sjf.c

cc sjf.c

./a.out

Enter the no of process 5

Enter the burst time of each process

p0 1

p1 5

p2 2

p3 3

p4 4

Process No	Burst Time	Waiting Time	Turn Around Time
p0	1	0	1
p1	2	1	3
p2	3	3	6
p3	4	6	10
p4	5	10	15

Average waiting time 4.000000

Average turn around time 7.000000

c) Round Robin CPU Scheduling Algorithm.

SOURCE CODE:

```
/* A program to simulate the Round Robin CPU scheduling algorithm */
#include<stdio.h>
struct process
{
	int burst,wait,comp,f;
}p[20]={0,0};
int main()
{
	int n,i,j,totalwait=0,totalturn=0,quantum,flag=1,time=0;
	printf("\nEnter The No Of Process:");
	scanf("%d",&n);
	printf("\nEnter The Quantum time (in ms):");
	scanf("%d",&quantum);
	for(i=0;i<n;i++)
	{
		printf("Enter The Burst Time (in ms) For Process #%2d:",i+1);
		scanf("%d",&p[i].burst);
		p[i].f=1;
	}
	printf("\nOrder Of Execution \n");
	printf("\nProcess Starting Ending Remaining");
	printf("\n\t\tTime \tTime \t Time");
	while(flag==1)
	{
		flag=0;
		for(i=0;i<n;i++)
		{
			if(p[i].f==1)
			{
				flag=1;
```

```
                        j=quantum;
                        if((p[i].burst-p[i].comp)>quantum)
                        {
                                p[i].comp+=quantum;
                        }
                        else
                        {
                                p[i].wait=time-p[i].comp;
                                j=p[i].burst-p[i].comp;
                                p[i].comp=p[i].burst;
                                 p[i].f=0;
                        }
                        printf("\nprocess # %-3d %-10d %-10d %-10d", i+1, time, time+j, p[i].burst-
p[i].comp);
                        time+=j;
                }
        }
    }
    printf("\n\n------------------");
    printf("\nProcess \t Waiting Time TurnAround Time ");
    for(i=0;i<n;i++)
    {
        printf("\nProcess # %-12d%-15d%-15d",i+1,p[i].wait,p[i].wait+p[i].burst);
        totalwait=totalwait+p[i].wait;
        totalturn=totalturn+p[i].wait+p[i].burst;
    }
    printf("\n\nAverage\n------------------              ");
    printf("\nWaiting  Time: %fms",totalwait/(float)n);
    printf("\n Turn Around Time: %fms\n\n",totalturn/(float)n);
    return 0;
}
```

OUTPUT:

gedit rr.c

cc rr.c

./a.out

Enter The No Of Process: 3

Enter The Quantum time (in ms): 5

Enter The Burst Time (in ms) For Process # 1: 25

Enter The Burst Time (in ms) For Process # 2: 30

Enter The Burst Time (in ms) For Process # 3: 54

Order Of Execution

Process	Starting Time	Ending Time	Remaining Time
process # 1	0	5	20
process # 2	5	10	25
process # 3	10	15	49
process # 1	15	20	15
process # 2	20	25	20
process # 3	25	30	44
process # 1	30	35	10
process # 2	35	40	15
process # 3	40	45	39
process # 1	45	50	5
process # 2	50	55	10
process # 3	55	60	34
process # 1	60	65	0
process # 2	65	70	5
process # 3	70	75	29
process # 2	75	80	0
process # 3	80	85	24

process # 3	85	90	19
process # 3	90	95	14
process # 3	95	100	9
process # 3	100	105	4
process # 3	105	109	0

Process	Waiting Time	TurnAround Time
Process # 1	40	65
Process # 2	50	80
Process # 3	55	109

Average

Waiting Time: 48.333333ms

Turn Around Time: 84.666667ms

4. **Write a C Program to Simulate IPC Techniques.**

a) **Pipes.**

Named PIPE.

SOURCE CODE:

// Program 1: Creating FIFO / Named PIPE.

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
int main()
    {
        int res;
        res = mkfifo("fifo1",0777); // Creates a Named PIPE with the name fifo1.
        printf("Named PIPE Created\n");
    }
```

// Program2: A Writing to a FIFO / Named PIPE.

```
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
       int res,n;
    res=open("fifo1",O_WRONLY);
    write(res,"Hello world",100);
    printf("Sender Process %d sent the data\n",getpid());
}
```

// Program 3: Reading from the Named PIPE.

```
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
       int res,n;
    char buffer[100];
```

```
        res=open("fifo1",O_RDONLY);
        n=read(res,buffer,100);
        printf("Reader Process %d Started\n",getpid());
        printf("Data Received by Receiver %d is: %s\n",getpid(), buffer);
}
```

OUTPUT:

```
gedit creating.c
gedit writing.c
gedit reading.c
cc creating.c
./a.out
Named PIPE Created
cc writing.c -o wr
cc reading.c -o rd
./wr & ./rd
[1] 2227
Sender Process 2227 sent the data
Reader Process 2228 Started
Data Received by Receiver 2228 is: Hello world
[1]+  Done                    ./wr
```

Un – Named PIPE.

SOURCE CODE:

```
/* A Program that implements a Unnamed PIPE between parent and child process */
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
#include<string.h>
int main()
{
	int fd[2], nbytes;
	pid_t pid;
	char string[]="Hello World \n";
	char buff[80];
	pipe(fd);
	pid=fork();
	if (pid == -1)
	{
		printf("Fork Error");
		exit(0);
	}
	if(pid == 0)
	{
		close(fd[0]);
		write(fd[1], string,(strlen(string)+1));
		exit(0);
	}
	else
	{
		close(fd[1]);
		nbytes=read(fd[0],buff,sizeof(buff));
		printf("Received String: %s", buff);
```

```
        }

        return (0);

}
```

OUTPUT:

```
gedit pipe.c
cc pipe.c
./a.out
Received String: Hello World
```

b) Message Queues

SOURCE CODE:

// Program 1: Program for IPC using Message Queues to send data to a message queue.

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MAX_TEXT 512   // Maximum length of the message that can be sent allowed.
struct my_msg
{
        long int msg_type;
        char some_text[MAX_TEXT];
};
int main()
{
        int running=1;
        int msgid;
        struct my_msg some_data;
        char buffer[50]; // Array to store user input.
        msgid=msgget((key_t)14534,0666|IPC_CREAT);
        if (msgid == -1) // -1 means the message queue is not created.
        {
                printf("Error in Creating Queue. \n");
                exit(0);
        }

        while(running)
        {
                printf("Enter Some Text:\n");
```

```
            fgets(buffer,50,stdin);
            some_data.msg_type=1;
            strcpy(some_data.some_text,buffer);
            if(msgsnd(msgid,(void *)&some_data, MAX_TEXT,0)==-1) // msgsnd returns -1 if the message is not sent.
            {
                  printf("Msg Not Sent. \n");
            }
            if(strncmp(buffer,"end",3)==0)
            {
                  running=0;
            }
      }
}
```

// Program 2: Program for IPC using Message Queues T]to receive/read message from the above – created message queue.

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
struct my_msg
{
      long int msg_type;
      char some_text[BUFSIZ];
};
int main()
{
      int running=1;
      int msgid;
      struct my_msg some_data;
```

```
        long int msg_to_rec=0;
        msgid=msgget((key_t)14534,0666|IPC_CREAT);
        while(running)
        {
                msgrcv(msgid,(void *)&some_data, BUFSIZ, msg_to_rec, 0);
                printf("Data received: %s\n", some_data.some_text);
                if(strncmp(some_data.some_text, "end",3) == 0)
                {
                        running=0;
                }
        }
         msgctl(msgid,IPC_RMID,0);
 }
```

OUTPUT:

```
gedit msgquesender.c
gedit msgquereceiver.c
cc msgquesender.c -o s
cc msgquereceiver.c -o r
./s
Enter Some Text:
Hello World
Enter Some Text:
end
./r
Data received: Hello World

Data received: end
```

c) Shared Memory.

SOURCE CODE:

// Program 1: A Program for Shared Memory Sender.

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{

        int i;

        void *shared_memory;

        char buff[100];

        int shmid;

        shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT); //Creates shared memory segment with key 2345, having size 1024 bytes. IPC_CREAT is used to create the shared segment if it does not exist. 0666 are the permisions on the shared segment.

        printf("Key of shared memory is %d\n",shmid);

        shared_memory=shmat(shmid,NULL,0); //Process attached to shared memory segment.

        printf("Process attached at %p\n",shared_memory); //This prints the address where the segment is attached with this process.

        printf("Enter some data to write to shared memory\n");

        read(0,buff,100); //Get some input from user.

        strcpy(shared_memory,buff); //Data written to shared memory.

        printf("You wrote: %s\n",(char *)shared_memory);

}
```

// Program 2: A Program for Shared Memory Receiver.

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/shm.h>

#include<string.h>

int main()

{
```

```
        int i;
        void *shared_memory;
        char buff[100];
        int shmid;
        shmid=shmget((key_t)2345, 1024, 0666);
        printf("Key of shared memory is %d\n",shmid);
        shared_memory=shmat(shmid,NULL,0);  // Process attached to shared memory segment.
        printf("Process attached at %p\n",shared_memory);
        printf("Data read from shared memory is: %s\n",(char *)shared_memory);
}
```

OUTPUT:

```
gedit sharedmemorysender.c
gedit sharedmemoryreceiver.c
cc sharedmemorysender.c -o sender
cc sharedmemoryreceiver.c -o receiver
./sender
Key of shared memory is 21495817
Process attached at 0x7fbcd5633000
Enter some data to write to shared memory
Methodist College of Engineering and Technology.
You wrote: Methodist College of Engineering and Technology.
./receiver
Key of shared memory is 21495817
Process attached at 0x7f44306ad000
Data read from shared memory is: Methodist College of Engineering and Technology.
```

5. **Write a C Program to Simulate Classical Problems of Synchronization.**

a) **Readers – Writers Problem.**

SOURCE CODE:

// Program 1: (mesg.h) Readers – Writers Problem Using Massage Passing.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define MKEY1 5543L
#define MKEY2 4354L
#define PERMS 0666
typedef struct
{
        long mtype;
        char mdata[50];
}mesg;
```

// Program 2: (sender1.c) Readers – Writers Problem Using Massage Passing.

```
#include "mesg.h"
mesg msg;
int main()
{
        int mq_id;
        int n;
        if((mq_id=msgget(MKEY1,PERMS|IPC_CREAT))<0)
        {
                printf("Sender: Error creating message");
                exit(1);
        }
        msg.mtype=1111L;
        n=read(0,msg.mdata,50);
        msg.mdata[n]='\0';
```

```
        msgsnd(mq_id,&msg,50,0);
}
```

// Program 3: (receiver1.c) Readers – Writers Problem Using Massage Passing.

```
#include "mesg.h"
mesg msg;
int main()
{
        int mq_id;
        int n;
        if( ( mq_id=msgget(MKEY1, PERMS|IPC_CREAT ) ) < 0)
        {
                printf("receiver: Error opening message");
                exit(1);
        }
        msgrcv(mq_id,&msg,50,1111L,0);
        write(1,msg.mdata,50);
        msgctl(mq_id,IPC_RMID,NULL);

}
```

OUTPUT:

```
gedit mesg.h
gedit sender1.c
gedit receiver1.c
cc mesg.h
cc sender1.c -o rws
cc receiver1.c -o rwr
./rws
Hello World
./rwr
Hello World
```

b) Producers – Consumers Problem.

SOURCE CODE:

// Program 1: (shmem.h) Producer – Consumer Problem Using Shared Memory.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define PERMS 0666
#define mkey 45787
```

// Program 2: (sender.c) Producer – Consumer Problem Using Shared Memory.

```
#include "shmem.h"
int main()
{
    int shmemid,n;
    char *pshmem;
    if((shmemid=shmget(mkey,10,PERMS|IPC_CREAT))<0)
    {
        printf("\n Sender: Cant get shared memory");
        exit(1);
    }
    pshmem=shmat(shmemid,(char *) 0,0);
    if(pshmem < 0)
    {
        perror("Sender: Cant attach shared memory");
        exit(1);
    }
    n=read(0,pshmem,100);
    pshmem[n]='\0';
    exit(0);
}
```

// Program 3: (receiver.c) Producer – Consumer Problem Using Shared Memory.

```
#include "shmem.h"
int main()
{
	int shmemid,n;
	char *pshmem;
	if((shmemid=shmget(mkey,10,PERMS|IPC_CREAT))<0)
	{
		printf("\nReceiver: Error opening shared memory");
		exit(1);
	}
	pshmem=shmat(shmemid,(char *) 0,0);
	if(pshmem < 0)
	{
		printf("\n Receiver : Can't attach shared memory");
		exit(1);
	}
	printf("%s",pshmem);
	shmctl(shmemid,IPC_RMID,NULL);
	exit(0);
}
```

OUTPUT:

```
gedit shmem.h
gedit sender.c
gedit receiver.c
cc shmem.h
cc sender.c -o pcs
cc receiver.c -o pcr
./pcs
Methodist College of Engineering and Technology
./pcr
Methodist College of Engineering and Technology
```

c) Dining Philosophers Problem.

SOURCE CODE:

// Program 1: (smop.h) Dinning – Philosophers Problem Using Semaphores.

```
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#define SEMKEY 3010L
#define PERMS 0666
static struct sembuf lock0[2]={0,0,0, 0,1,0};
static struct sembuf unlock0[1]={0,-1,IPC_NOWAIT};
static struct sembuf lock1[2]={1,0,0,1,1,0};
static struct sembuf unlock1[1]={1,-1,IPC_NOWAIT};
static struct sembuf lock2[2]={2,0,0,2,1,0};
static struct sembuf unlock2[1]={2,-1,IPC_NOWAIT};
int semlock0(int semid)
{
        semop(semid,&lock0[0],2);
}
int semunlock0(int semid)
{
        semop(semid,&unlock0[0],1);
}
int semlock1(int semid)
{
        semop(semid,&lock1[0],2);
}
int semunlock1(int semid)
{
        semop(semid,&unlock1[0],1);
}
int semlock2(int semid)
{
```

```
        semop(semid,&lock2[0],2);
}
int semunlock2(int semid)
{
        semop(semid,&unlock2[0],1);
}
```

// Program 2: (dinning.c) Dinning – Philosophers Problem Using Semaphores.

```
#include <stdio.h>
#include "smop.h"
int main()
{
      int pno, semid,tmp;
      semid=semget(SEMKEY,3,PERMS|IPC_CREAT);
      printf("\n\t Enter Philosopher Number(0-2): ");
      scanf("%d",&pno);
      switch(pno)
      {
            case 0: printf("\n\t Philosopher %d thinking...",pno);
                  printf("\n\t Enter 0 to start eating...");
                  scanf("%d",&tmp);
                  if(tmp==0)
                  {
                        semlock0(semid);
                        semlock1(semid);
                        printf("\n\t Philosopher %d eating...",pno);
                  }
                  else
                  {
                        printf("wrong choice \n");
                        break;
                  }
                    printf("\n\t Press 1 to stop eating ...");
```

```
  scanf("%d",&tmp);
        if(tmp==1)
        {
              semunlock0(semid);
              semunlock1(semid);
        printf("\n\tPhilosopher %d thinking...",pno);
        }
        else
        {
              fflush(stdout);
              printf("wrong choice");
              break;
        }
        break;
case 1: printf("\n\t Philosopher %d thinking...",pno);
        printf("\n\t Enter 0 to start eating...");
        scanf("%d",&tmp);
        if(tmp==0)
        {
              semlock1(semid);
              semlock2(semid);
              printf("\n\t Philosopher %d eating...",pno);
        }
        else
        {
              printf("wrong choice \n");
              break;
        }
  printf("\n\t Press 1 to stop eating...");
  scanf("%d",&tmp);

        if(tmp==1)
```

```
        {
                semunlock1(semid);
                semunlock2(semid);
        printf("\n\t Philosopher %d thinking...",pno);
        }
        else
        {
                fflush(stdout);
                printf("wrong choice \n");
                break;
        }
        break;
case 2: printf("\n\t Philosopher %d thinking...",pno);
        printf("\n\t Enter 0 to start eating...");
        scanf("%d",&tmp);
        if(tmp==0)
        {
                semlock2(semid);
                semlock0(semid);
                printf("\n\t Philosopher %d eating...",pno);
        }
        else
        {
                printf("wrong choice \n");
                break;
        }
          printf("\n\t Press 1 to stop eating...");
          scanf("%d",&tmp);

        if(tmp==1)
        {
                semunlock2(semid);
```

```
                semunlock0(semid);
            printf("\n\t Philosopher %d thinking... \n",pno);
            }
            else
            {
                fflush(stdout);
                printf("wrong choice \n");
                break;
            }
            break;
    }
    semctl(semid,3,IPC_RMID,NULL);
}
```

OUTPUT:

```
gedit smop.h
gedit dinning.c
cc smop.h
cc dinning.c -o dps
./dps

    Enter Philosopher Number(0-2): 2

    Philosopher 2 thinking...
    Enter 0 to start eating...0

    Philosopher 2 eating...
    Press 1 to stop eating...1

    Philosopher 2 thinking...
```

6. Write a C Program to Simulate Bangers Algorithm for Deadlock Avoidance.

SOURCE CODE:

```
/* Bankers algorithm for Deadlock Avoidance */
#include<stdio.h>
struct process
{
    int allocation[3];
    int max[3];
    int need[3];
    int finish;
}p[10];
int main()
{
    int n,i,I,j,avail[3],work[3],flag,count=0,sequence[10],k=0;
    printf("\nEnter the number of process:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            printf("\nEnter the %dth process allocated resources:",i);
            scanf("%d%d%d",&p[i].allocation[0],&p[i].allocation[1],&p[i].allocation[2]);
            printf("\nEnter the %dth process maximum resources:",i);
            scanf("%d%d%d",&p[i].max[0],&p[i].max[1],&p[i].max[2]);
            p[i].finish=0;
            p[i].need[0]=p[i].max[0]-p[i].allocation[0];
            p[i].need[1]=p[i].max[1]-p[i].allocation[1];
            p[i].need[2]=p[i].max[2]-p[i].allocation[2];
    }
    printf("\nEnter the available vector:");
    scanf("%d%d%d",&avail[0],&avail[1],&avail[2]);
    for(i=0;i<3;i++)
            work[i]=avail[i];
    while(count!=n)
```

```
    {
        count=0;
        for(i=0;i<n;i++)
        {
            flag=1;
            if(p[i].finish==0)
            if(p[i].need[0]<=work[0])
            if(p[i].need[1]<=work[1])
            if(p[i].need[2]<=work[2])
            {
                for(j=0;j<3;j++)
                    work[j]+=p[i].allocation[j];
                p[i].finish=1;
                sequence[k++]=i;
                flag=0;
            }
            if(flag==1)
                count++;
        }
    }
    count=0;
    for(i=0;i<n;i++)
        if(p[i].finish==1)
            count++;
    printf("\n The safe sequence is:\t");
    if(count++==n)
        for(i=0;i<k;i++)
            printf("%d\n",sequence[i]);
    else
        printf("SYSTEM IS NOT IN A SAFE STATE \n\n");
    return 0;
}
```

OUTPUT:

gedit bankersavoidance.c

cc bankersavoidance.c

./a.out

Enter the number of process:3

Enter the 0th process allocated resources:1 2 3

Enter the 0th process maximum resources:4 5 6

Enter the 1th process allocated resources:3 4 5

Enter the 1th process maximum resources:6 7 8

Enter the 2th process allocated resources:1 2 3

Enter the 2th process maximum resources:3 4 5

Enter the available vector:10 12 11

The safe sequence is: 0 1 2

7. Write a C Program to Simulate Bangers Algorithm for Deadlock Detection.

SOURCE CODE:

```
/* Bankers algorithm for Deadlock detection */
#include<stdio.h>
void main()
{
    int found,flag,l,p[4][5],tp,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
    printf("enter total no of processes: \n");
    scanf("%d",&tp);
    printf("enter chain matrix: \n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
        {
            scanf("%d",&c[i][j]);
        }
    }
    printf("enter allocation matrix: \n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
        {
            scanf("%d",&p[i][j]);
        }
    }
    printf("enter resource vector: \n");
    for(i=0;i<5;i++)
    {
        scanf("%d",&r[i]);
    }
    printf("enter availability vector: \n");
    for(i=0;i<5;i++)
```

```
{
        scanf("%d",&a[i]);
        temp[i]=a[i];
}
for(i=0;i<4;i++)
{
        sum=0;
        for(j=0;j<5;j++)
        {
                sum+=p[i][j];
        }
        if(sum==0)
        {
                m[k]=i;
                k++;
        }
}
for(i=0;i<4;i++)
{
        for(l=1;l<k;l++)
        {
                if(i!=m[l])
                {
                        flag=1;
                        for(j=0;j<5;j++)
                        {
                                if(c[i][j]>temp[j])
                                {
                                        flag=0;
                                        break;
                                }
                        }
```

```
                }
        }
        if(flag==1)
        {
                m[k]=i;
                k++;
                for(j=0;j<5;j++)
                        temp[j]+=p[i][j];
        }
    }
    printf("deadlock causing processes are: \n");
    for(j=0;j<tp;j++)
    {
                found=0;
                for(i=1;i<k;i++)
                {
                if(j==m[i])
                        found=1;
                }
                if(found==0)
                        printf("%d\t",j);
    }
}
```

OUTPUT:

gedit bankersdetection.c

cc bankersdetection.c

./a.out

```
enter total no of processes:
4
enter chain matrix:
0	1	0	0	1
0	0	1	0	1
0	0	0	0	1
1	0	1	0	1
enter allocation matrix:
1	0	1	1	0
1	1	0	0	0
0	0	0	1	0
0	0	0	0	0
enter resource vector:
2	1	1	2	1
enter availability vector:
0	0	0	0	1
deadlock causing processes are:
0	1
```

8. **Write a C Program to Simulate all Page Replacement Algorithms.**

a) **FIFO Page Replacement Algorithm.**

SOURCE CODE:

```
/* A program to simulate FIFO Page Replacement Algorithm */
#include<stdio.h>
int main()
{
	int a[5],b[20],n,p=0,q=0,m=0,h,k,i,q1=1;
	char f='F';
	printf("Enter the Number of Pages:");
	scanf("%d",&n);
	printf("Enter %d Page Numbers:",n);
	for(i=0;i<n;i++)
	scanf("%d",&b[i]);
	for(i=0;i<n;i++)
	{
		if(p==0)
		{
			if(q>=3)
			q=0;
			a[q]=b[i];
			q++;
			if(q1<3)
			{
				q1=q;
			}
		}
		printf("\n%d",b[i]);
		printf("\t");
		for(h=0;h<q1;h++)
			printf("%d",a[h]);
		if((p==0)&&(q<=3))
		{
			printf("-->%c",f);
			m++;
		}
		p=0;
		for(k=0;k<q1;k++)
		{
			if(b[i+1]==a[k])
			p=1;
		}
	}
	printf("\nNo of faults:%d",m);
}
```

OUTPUT:

```
gedit fifo.c
cc fifo.c
./a.out

Enter the Number of Pages: 12
Enter 12 Page Numbers:
2       3       2       1       5       2       4       5       3       2       5       2

2       2-->F
3       23-->F
2       23
1       231-->F
5       531-->F
2       521-->F
4       524-->F
5       524
3       324-->F
2       324
5       354-->F
2       352-->F

No of faults: 9
```

b) LRU Page Replacement Algorithm.

SOURCE CODE:

```
/* A program to simulate LRU Page Replacement Algorithm */
#include<stdio.h>
int main()
{
	int a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u,n;
	char f='F';
	printf("Enter the number of pages:");
	scanf("%d",&n);
	printf("Enter %d Page Numbers:",n);
	for(i=0;i<n;i++)
	scanf("%d",&b[i]);
	for(i=0;i<n;i++)
	{
		if(p==0)
		{
			if(q>=3)
				q=0;
			a[q]=b[i];
			q++;
			if(q1<3)
			{
				q1=q;
			}
		}
		printf("\n%d",b[i]);
		printf("\t");
		for(h=0;h<q1;h++)
			printf("%d",a[h]);
		if((p==0)&&(q<=3))
		{
			printf("-->%c",f);
			m++;
		}
		p=0;
		if(q1==3)
		{
			for(k=0;k<q1;k++)
			{
				if(b[i+1]==a[k])
					p=1;
			}
			for(j=0;j<q1;j++)
			{
				u=0;
				k=i;
				while(k>=(i-1)&&(k>=0))
				{
					if(b[k]==a[j])
					u++;
					k--;
				}
				if(u==0)
				q=j;
```

```
                        }
                }
                else
                {
                        for(k=0;k<q;k++)
                        {
                                if(b[i+1]==a[k])
                                        p=1;
                        }
                }
        }
        printf("\nNo of faults:%d",m);
}
```

OUTPUT:

```
gedit lru.c
cc lru.c
./a.out

Enter the number of pages: 12
Enter 12 Page Numbers:
2       3       2       1       5       2       4       5       3       2       5       2

2       2-->F
3       23-->F
2       23
1       231-->F
5       251-->F
2       251
4       254-->F
5       254
3       354-->F
2       352-->F
5       352
2       352

No of faults: 7
```

9. **Write C programs to simulate implementation of Disk Scheduling Algorithms.**

a) FCFS Disk Scheduling Algorithms.

SOURCE CODE:

```
// C programs to simulate implementation of FCFS Disk Scheduling Algorithms.
#include<stdio.h>
int main()
{
        int queue[20],n,head,i,j,k,seek=0,max,diff;
        float avg;
        printf("Enter the max range of disk\n");
        scanf("%d",&max);
        printf("Enter the size of queue request\n");
        scanf("%d",&n);
        printf("Enter the queue of disk positions to be read\n");
        for(i=1;i<=n;i++)
                scanf("%d",&queue[i]);
        printf("Enter the initial head position\n");
        scanf("%d",&head);
        queue[0]=head;
        for(j=0;j<=n-1;j++)
        {
                diff=abs(queue[j+1]-queue[j]);
                seek+=diff;
                printf("Disk head moves from %d to %d with
                seek %d\n",queue[j],queue[j+1],diff);
        }
        printf("Total seek time is %d\n",seek);
        avg=seek/(float)n;
        printf("Average seek time is %f\n",avg);
        return 0;
}
```

OUTPUT:

```
Enter the max range of disk
200
Enter the size of queue request
8
Enter the queue of disk positions to be read
90      120     35      122     38      128     65      68
Enter the initial head position
50
Disk head moves from 50 to 90 with seek
40
Disk head moves from 90 to 120 with seek
30
Disk head moves from 120 to 35 with seek
85
Disk head moves from 35 to 122 with seek
87
Disk head moves from 122 to 38 with seek
84
Disk head moves from 38 to 128 with seek
90
Disk head moves from 128 to 65 with seek
```

63
Disk head moves from 65 to 68 with seek
3
Total seek time is 482
Average seek time is 60.250000

b)SSTF Disk Scheduling Algorithms.

// **SSTF Disk Scheduling Algorithms.**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
	int queue[100],t[100],head,seek=0,n,i,j,temp;
	float avg;
	printf("*** SSTF Disk Scheduling Algorithm ***\n");
	printf("Enter the size of Queue\t");
	scanf("%d",&n);
	printf("Enter the Queue\t");
	for(i=0;i<n;i++)
	{
		scanf("%d",&queue[i]);
	}
	printf("Enter the initial head position\t");
	scanf("%d",&head);
	for(i=1;i<n;i++)
		t[i]=abs(head-queue[i]);
	for(i=0;i<n;i++)
	{
		for(j=i+1;j<n;j++)
		{
			if(t[i]>t[j])
			{
				temp=t[i];
				t[i]=t[j];
				t[j]=temp;
				temp=queue[i];
				queue[i]=queue[j];
				queue[j]=temp;
			}
		}
	}
	for(i=1;i<n-1;i++)
	{
		seek=seek+abs(head-queue[i]);
		head=queue[i];
	}
	printf("\nTotal Seek Time is%d\t",seek);
	avg=seek/(float)n;
	printf("\nAverage Seek Time is %f\t",avg);
	return 0;
}
```

OUTPUT:

```
*** SSTF Disk Scheduling Algorithm ***
Enter the size of Queue 5
Enter the Queue 10 17 2 15 4
Enter the initial head position 3
Total Seek Time is14
Average Seek Time is 2.800000
```