||JAI SRI GURUDEV||
Sri Adichunchanagiri Shikshana Trust(R)

SJB Institute of Technology

Recognized by UGC, New Delhi with 2(F) & 12(B)
(A Constituent Institution of BGS & SJB Group of Institution & Hospitals)
67, BGS Health & Education City, Dr. Vishnuvardhan Road Kengeri,
Bangalore – 560 060

Department of Computer Science and Engineering

OPERATING SYSTEM LAB
[23CSI304]
III SEMESTER – B. E

Staff Name:			
Section:	**A, B, C & D**	**Batch:**	**A1, A2, A3, B1, B2 , B3, C1, C2, C3, D1,D2&D3**

PREFACE

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs conveniently and efficiently.

An operating system is software that manages computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system. A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being application programs.

An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information. The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, a memory management module, I/O programs, and a file system.

Functionalities of Operating System

- **Resource Management:** When parallel accessing happens in the OS means when multiple users are accessing the system the OS works as Resource Manager, Its responsibility is to provide hardware to the user. It decreases the load in the system.
- **Process Management:** It includes various tasks like **scheduling and termination** of the process. It is done with the help of CPU Scheduling Algorithms.
- **Storage Management:** The **file system** mechanism used for the management of the storage. The data is stored in various tracks of Hard disks that are all managed by the storage manager. It included **Hard Disk**.
- **Memory Management:** Refers to the management of primary memory. The operating system has to keep track of how much memory has been used and by whom. It has to decide which process needs memory space and how much. OS also has to allocate and deallocate the memory space.
- **Security/Privacy Management:** Privacy is also provided by the Operating system using passwords so that unauthorized applications can't access programs or data.

SJB INTITUTE OF TECHNOLOGY

Institution's Vision

To become a recognized technical education centre with a global perspective.

Institution's Mission

To provide learning opportunities that foster students ethical values, intelligent development in science & technology and social responsibility so that they become sensible and contributing members of the society.

Department of Information Science and Engineering

Department Vision

We envision our department as a catalyst for developing educated, engaged and employable individuals whose collective energy will be the driving force for prosperity and the quality of life in our diverse world.

Department Mission

Our mission is to provide quality technical education in the field of information technology and to strive for excellence in the education by developing and sharpening the intellectual and human potential for good industry and community.

PROGRAM EDUCATIONAL OBJECTIVES (PEO'S)

Graduates will -

- Possess expertise in problem solving, design and analysis, technical skills for a fruitful career accomplishing professional and social ethics with exposure to modern designing tools and technologies in Information Science and Engineering.

- Excel in communication, teamwork and multiple domains related to engineering issues accomplishing social responsibilities and management skills.

- Outclass in competitive environment through certification courses, gaining leadership qualities and progressive research to become successful entrepreneurs.

PROGRAM SPECIFIC OUTCOMES (PSO'S)

Graduates will be able to -

1. **PSO1:** Apply the Knowledge of Information Science to develop software solutions in current research trends and technology.

2. **PSO2:** Create Social awareness & environmental wisdom along with ethical responsibility to lead a successful career and sustain passion using optimal resources to become an Entrepreneur.

PROGRAM OUTCOMES-PO's

Engineering graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

COURSE OUTCOMES

On successful completion of this course students will be able to,

CO1	Explain the structure and functionality of operating system
CO2	Apply appropriate CPU scheduling algorithms for the given problem.
CO3	Analyze the various techniques for process synchronization and deadlock handling.
CO4	Apply the various techniques for memory management
CO5	Explain file and secondary storage management strategies.
CO6	Describe the need for information protection mechanisms

CO No.	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1												
CO2												
CO3												
CO4												
CO5												

PSO No.	PSO1	PSO2	PSO3	PSO4
PSO1				
PSO2				
PSO3				
PSO4				
PS05				

	Operating System		Semester	3
	Course Code	BCS303	CIE Marks	50
	Teaching Hours/Week(L:T:P:S)	3:0:2:0	SEE Marks	50
	Total Hours of Pedagogy	40 Hours of Theory + 20Hours of Practical	Total Marks	100
	Credits	04	Exam Hours	03
	Examination nature(SEE)	Theory		

Course objectives:

- To Demonstrate the need for OS and different types of OS
- To discuss suitable techniques for management of different resource.
- To demonstrate different APIs/Commands related to processor, memory, and storage and file system management.

Teaching-Learning Process(General Instructions)

Teachers can use the following strategies to accelerate the attainment of the various course outcomes.

1. Lecturer methods (L) need not to be only traditional lecture method, but alternative effective teaching methods could be adopted to attain the outcomes.
2. Use of Video/Animation to explain functioning of various concepts.
3. Encourage collaborative (Group Learning) Learning in the class.
4. Adopt Problem Based Learning (PBL), which fosters students' Analytical skills, develop design thinking skills such as the ability to design, evaluate, generalize, and analyze information rather than simply recall it.
5. Role play for process scheduling.
6. Demonstrate the installation of any one Linux OS on VMware/Virtual Box

Module-1

Introduction to operating systems, System structures: What operating systems do; Computer System organization; Computer System architecture; Operating System structure; Operating System operations; Process management; Memory management; Storage management; Protection and Security; Distributed system; Special-purpose systems; Computing environments.

Operating System Services: User - Operating System interface; System calls; Types of system calls; System programs; Operating system design and implementation; Operating System structure; Virtual machines; Operating System debugging, Operating System generation; System boot.

Textbook 1: Chapter – 1 (1.1-1.12), 2 (2.2-2.11)

Module-2

	Process Management: Process concept; Process scheduling; Operations on processes; Inter process communication **Multi-threaded Programming:** Overview; Multithreading models; Thread Libraries; Threading issues. Process Scheduling: Basic concepts; Scheduling Criteria; Scheduling Algorithms; Thread scheduling; Multiple-processor scheduling, **Textbook 1: Chapter – 3 (3.1-3.4), 4 (4.1-4.4), 5 (5.1 -5.5)**
	Module-3
	Process Synchronization: Synchronization: The critical section problem; Peterson's solution; Synchronization hardware; Semaphores; Classical problems of synchronization; Deadlocks: System model; Deadlock characterization; Methods for handling deadlocks; Deadlock prevention; Deadlock avoidance; Deadlock detection and recovery from deadlock. **Textbook 1: Chapter – 6 (6.1-6.6), 7 (7.1 -7.7)**
	Module-4
	Memory Management: Memory management strategies: Background; Swapping; Contiguous memory allocation; Paging; Structure of page table; Segmentation. **Virtual Memory Management:** Background; Demand paging; Copy-on-write; Page replacement; Allocation of frames; Thrashing. **Textbook 1: Chapter -8 (8.1-8.6), 9 (9.1-9.6)**
	Module-5
	File System, Implementation of File System: File system: File concept; Access methods; Directory and Disk structure; File system mounting; File sharing; Implementing File system: File system structure; File system implementation; Directory implementation; Allocation methods; Free space management. **Secondary Storage Structure, Protection:** Mass storage structures; Disk structure; Disk attachment; Disk scheduling; Disk management; Protection: Goals of protection, Principles of protection, Domain of protection, Access matrix. **Textbook 1: Chapter – 10 (10.1-10.5) ,11 (11.1-11.5),12 (12.1-12.5), 14 (14.1-14.4)**

Course outcome(Course Skill Set)

At the end of the course, the student will be able to:

1. Explain the structure and functionality of operating system
2. Apply appropriate CPU scheduling algorithms for the given problem.
3. Analyse the various techniques for process synchronization and deadlock handling.
4. Apply the various techniques for memory management
5. Explain file and secondary storage management strategies.
6. Describe the need for information protection mechanisms

PRACTICAL COMPONENT OF IPCC(May cover all / major modules)

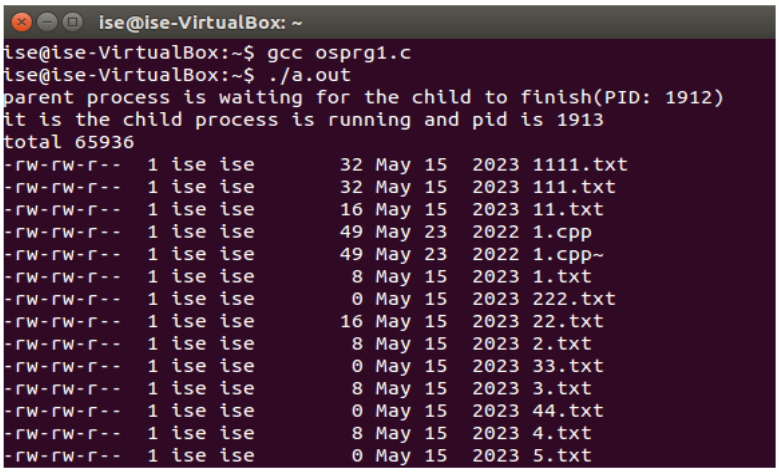
Sl No	Experiments
1	Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)
2	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.
3	Develop a C program to simulate producer-consumer problem using semaphores.
4	Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.
5	Develop a C program to simulate Bankers Algorithm for Deadlock Avoidance.
6	Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.
7	Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU
8	Simulate following File Organization Techniques a) Single level directory b) Two level directory
9	Develop a C program to simulate the Linked file allocation strategies.
10	Develop a C program to simulate SCAN disk scheduling algorithm.

PROGRAM-1

Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
    pid_t pid;
    // Fork a child process
    pid = fork();
     if (pid < 0)
    {
            perror("Fork failed");
            exit(EXIT_FAILURE);
    }
    else if (pid == 0)
    {
            printf("Child process is running (PID: %d)\n",
            getpid());

            execlp("/bin/ls", "ls", "-l", NULL);

            perror("Exec failed");
            exit(EXIT_FAILURE);
    }
    else
    {
            printf("Parent process is waiting for the child
            to finish (PID: %d)\n", getpid());
            wait(NULL);
            printf("Child process has terminated\n");
    }
 return 0;
}
```

OUTPUT



PROGRAM-2

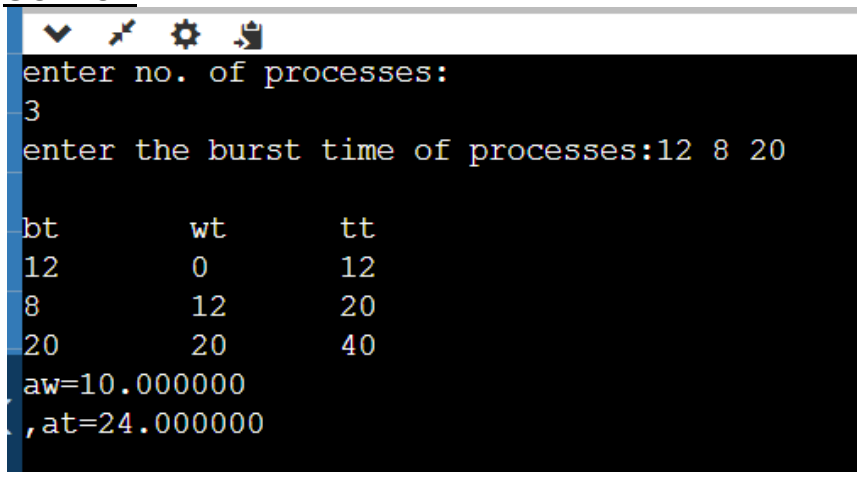
Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.

a) FCFS

```
#include<stdio.h>
void main()
{
    int i,j,bt[10],n,wt[10],tt[10],w1=0,t1=0;
    float aw,at;
    printf("enter no. of processes:\n");
    scanf("%d",&n);
    printf("enter the burst time of processes:");
    for(i=0;i<n;i++)
    scanf("%d",&bt[i]);
    for(i=0;i<n;i++)
        {
            wt[0]=0;
            tt[0]=bt[0];
            wt[i+1]=bt[i]+wt[i];
            tt[i+1]=tt[i]+bt[i+1];
            w1=w1+wt[i];
            t1=t1+tt[i];
        }
    aw=w1/n;
    at=t1/n;
    printf("\nbt\t wt\t tt\n");
    for(i=0;i<n;i++)
    printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
    printf("aw=%f\n,at=%f\n",aw,at);
```

```
}
```

OUTPUT



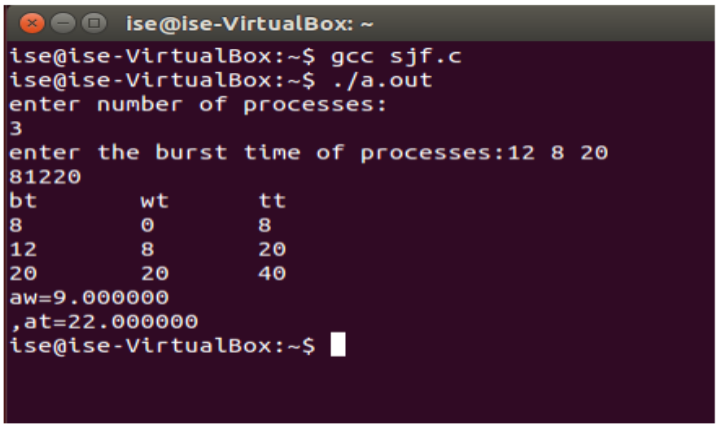
```
enter no. of processes:
3
enter the burst time of processes:12 8 20

bt      wt      tt
12      0       12
8       12      20
20      20      40
aw=10.000000
,at=24.000000
```

b) SJF

```
#include<stdio.h>
void main()
{
    int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0;
    float aw,at;
    printf("enter no. of processes:\n");
    scanf("%d",&n);
    printf("enter the burst time of processes:");
    for(i=0;i<n;i++)
    scanf("%d",&bt[i]);
    for(i=0;i<n;i++)
    {
            for(j=i;j<n;j++)
            if(bt[i]>bt[j])
            {
                    t=bt[i];
                    bt[i]=bt[j];
                    bt[j]=t;
            }
    }
    for(i=0;i<n;i++)
    printf("%d",bt[i]);
    for(i=0;i<n;i++)
    {
            wt[0]=0;
            tt[0]=bt[0];
            wt[i+1]=bt[i]+wt[i];
            tt[i+1]=tt[i]+bt[i+1];
```

```
            w1=w1+wt[i];
            t1=t1+tt[i];
    }
    aw=w1/n;
    at=t1/n;
    printf("\nbt\t wt\t tt\n");
    for(i=0;i<n;i++)
    printf("%d\t %d\t %d\n",bt[i],wt[i],tt[i]);
    printf("aw=%f\n,at=%f\n",aw,at);
}
```

OUTPUT

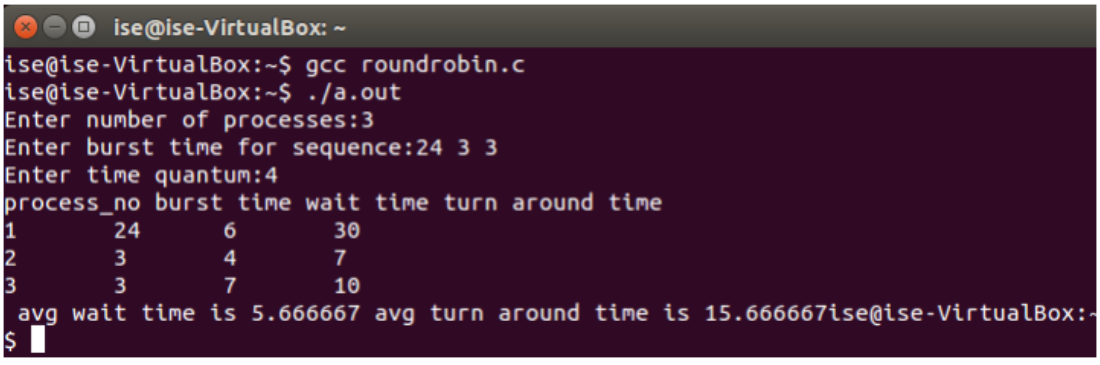


c) Round Robin

```
#include<stdio.h>
void main()
{
    int st[10],bt[10],wt[10],tat[10],n,tq;
    int i,count=0,swt=0,stat=0,temp,sq=0;
    float awt=0.0,atat=0.0;
    printf("Enter number of processes:");
    scanf("%d",&n);
    printf("Enter burst time for sequences:");
    for(i=0;i<n;i++)
    {
            scanf("%d",&bt[i]);
            st[i]=bt[i];
    }
    printf("Enter time quantum:");
    scanf("%d",&tq);
    while(1)
    {
            for(i=0,count=0;i<n;i++)
```

```
            {
                temp=tq;
                if(st[i]==0)
                {
                    count++;
                    continue;
                }
            if(st[i]>tq)
            st[i]=st[i]-tq;
            else
            if(st[i]>=0)
            {
                temp=st[i];
                st[i]=0;
            }
            sq=sq+temp;
            tat[i]=sq;
            }
        if(n==count)
        break;
    }
    for(i=0;i<n;i++)
    {
        wt[i]=tat[i]-bt[i];
        swt=swt+wt[i];
        stat=stat+tat[i];
    }
    awt=(float)swt/n;
    atat=(float)stat/n;
    printf("Process_no Burst time Wait time Turn around time");
    for(i=0;i<n;i++)
    printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);
    printf("\nAvg wait time is %f Avg turn around time is %f",awt,atat);
}
```

OUTPUT

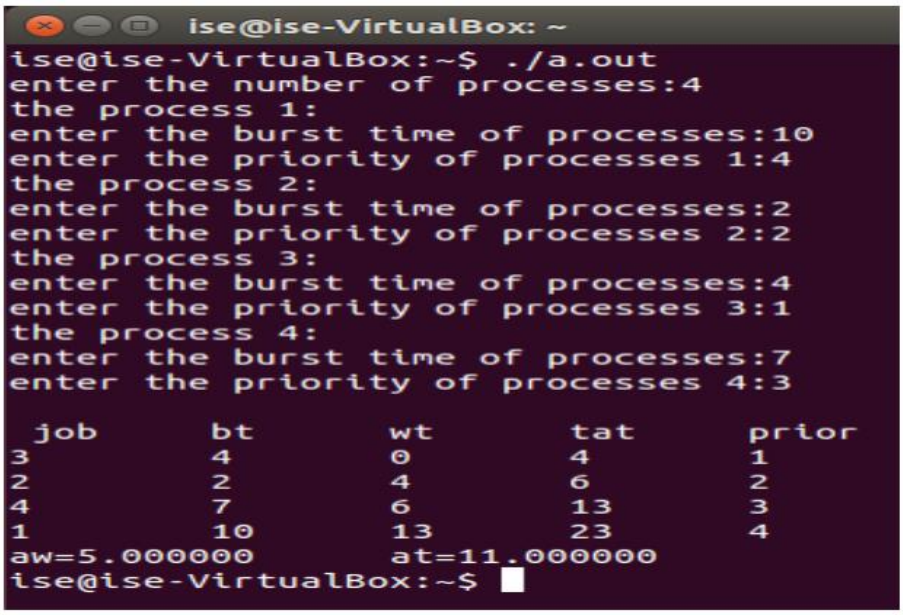


d) Priority

```
#include<stdio.h>
void main()
{
    int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
    float aw,at;
    printf("enter the number of processes:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            printf("The process %d:\n",i+1);
            printf("Enter the burst time of processes:");
            scanf("%d",&bt[i]);
            printf("Enter the priority of processes %d:",i+1);
            scanf("%d",&prior[i]);
            pno[i]=i+1;
    }
    for(i=0;i<n;i++)
    {
            for(j=0;j<n;j++)
                    {
                            if(prior[i]<prior[j])
                        {
                            s=prior[i];
                            prior[i]=prior[j];
                            prior[j]=s;
                            s=bt[i];
                            bt[i]=bt[j];
                            bt[j]=s;
                            s=pno[i];
                            pno[i]=pno[j];
                            pno[j]=s;
                        }
                    }
            }
    for(i=0;i<n;i++)
    {
            wt[0]=0;
            tt[0]=bt[0];
            wt[i+1]=bt[i]+wt[i];
            tt[i+1]=tt[i]+bt[i+1];
            w1=w1+wt[i];
            t1=t1+tt[i];
            aw=w1/n;
```

```
            at=t1/n;
    }
    printf(" \n job \t bt \t wt \t tat \t prior\n");
    for(i=0;i<n;i++)
    printf("%d \t %d \t %d\t %d\t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
    printf("aw=%f \t at=%f \n",aw,at);
}
```

OUTPUT



Program 3

Write a program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
    {
            int n;
            void producer();
            void consumer();
            int wait(int);
            int signal(int);
            printf("\n1.Producer\n2.Consumer\n3.Exit");
            while(1)
            {
                    printf("\nEnter your choice:");
                    scanf("%d",&n);
                    switch(n)
```

```
                {
                        case 1: if((mutex==1)&&(empty!=0))
                                producer();
                                else
                                printf("Buffer is full!!");
                                break;
                        case 2: if((mutex==1)&&(full!=0))
                                consumer();
                                else
                                printf("Buffer is empty!!");
                                break;
                        case 3:
                                exit(0);
                                break;
                    }
            }
            return 0;
    }
int wait(int s)
{
    return (--s);
}
int signal(int s)
{
    return(++s);
}
void producer()
{
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;
    printf("\nProducer produces the item %d",x);
    mutex=signal(mutex);
}
void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}
```

OUTPUT

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
```

Program 4

Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use mkfifo, open, read, write and close APIs in your program.

/*Writer Process*/

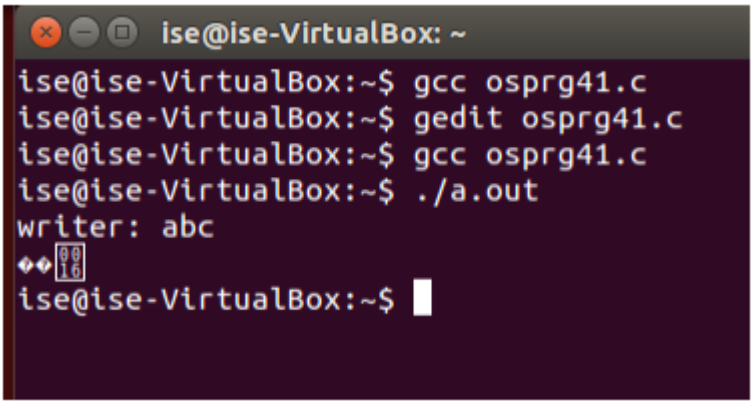
```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int fd;
    char buf[1024];
    /* create the FIFO (named pipe) */
    char * myfifo = "/tmp/myfifo";
    mkfifo(myfifo, 0666);
    printf("Run Reader process to read the FIFO File\n");
    fd = open(myfifo, O_WRONLY);
    write(fd,"Hi", sizeof("Hi"));
    /* write "Hi" to the FIFO */
    close(fd);
```

```
    unlink(myfifo); /* remove the FIFO */
    return 0;
}
```

/* Reader Process*/

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#define MAX_BUF 1024
int main()
{
    int fd;
    /* A temp FIFO file is not created in reader */
    char *myfifo = "/tmp/myfifo";
    char buf[MAX_BUF];
    /* open, read, and display the message from the FIFO */
    fd = open(myfifo, O_RDONLY);
    read(fd, buf, MAX_BUF);
    printf("Writer: %s\n", buf);
    close(fd);
    return 0;
}
```

OUTPUT



Program 5

Develop a C program to simulate Bankers Algorithm for Deadlock Avoidance.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10],
safeSequence[10];
 int p, r, i, j, process, count;
 count = 0;
 printf("Enter the no of processes : ");
 scanf("%d", &p);
 for(i = 0; i< p; i++)
 completed[i] = 0;
 printf("\n\nEnter the no of resources : ");
 scanf("%d", &r);
 printf("\n\nEnter the Max Matrix for each process : ");
 for(i = 0; i < p; i++)
 {
 printf("\nFor process %d : ", i + 1);
 for(j = 0; j < r; j++)
 scanf("%d", &Max[i][j]);
 }
 printf("\n\nEnter the allocation for each process : ")
for(i = 0; i < p; i++)
 {
 printf("\nFor process %d : ",i + 1);
 for(j = 0; j < r; j++)
 scanf("%d", &alloc[i][j]);
 }
 printf("\n\nEnter the Available Resources : ");
 for(i = 0; i < r; i++)
 scanf("%d", &avail[i]);
 for(i = 0; i < p; i++)
 for(j = 0; j < r; j++)
 need[i][j] = Max[i][j] - alloc[i][j];
 do
 {
 printf("\n Max matrix:\tAllocation matrix:\n");
 for(i = 0; i < p; i++)
 {
 for( j = 0; j < r; j++)
```

```
 printf("%d ", Max[i][j]);
 printf("\t\t");
 for( j = 0; j < r; j++)
 printf("%d ", alloc[i][j]);
 printf("\n");
 }
process = -1;
 for(i = 0; i < p; i++)
 {
 if(completed[i] == 0)//if not completed
 {
 process = i ;
 for(j = 0; j < r; j++)
 {
 if(avail[j] < need[i][j])
 {
 process = -1;
 break;
 }
 }
 }
 if(process != -1)
 break;
 }
 if(process != -1)
 {
 printf("\nProcess %d runs to completion!", process + 1);
 safeSequence[count] = process + 1;
 count++;
 for(j = 0; j < r; j++)
 {
 avail[j] += alloc[process][j];
 alloc[process][j] = 0;
 Max[process][j] = 0;
 completed[process] = 1;
 }
 }
 }
 while(count != p && process != -1);
 if(count == p)
 {
 printf("\nThe system is in a safe state!!\n");
 printf("Safe Sequence : < ");
 for( i = 0; i < p; i++)
 printf("%d ", safeSequence[i]);
 printf(">\n");
```

```
}
else
printf("\nThe system is in an unsafe state!!");
```

OUTPUT 1

```
Enter the no of processes: 5
Enter the no of resources: 3
Enter the Max Matrix 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 the allocation for each process:
For process 1 : 0 1 0
For process 2 : 2 0 0
For process 3 : 3 0 2
For process 4 : 2 1 1
For process 5 : 0 0 2
Enter the Available Resources: 3 3 2
Max matrix: Allocation matrix:
7 5 3 0 1 0
3 2 2 2 0 0
9 0 2 3 0 2
2 2 2 2 1 1
4 3 3 0 0 2
Process 2 runs to completion!
Max matrix: Allocation matrix:
7 5 3 0 1 0
0 0 0 0 0 0
9 0 2 3 0 2
2 2 2 2 1 1
4 3 3 0 0 2
Process 4 runs to completion!
Max matrix: Allocation matrix:
7 5 3 0 1 0
0 0 0 0 0 0
9 0 2 3 0 2
0 0 0 0 0 0
4 3 3 0 0 2
Process 1 runs to completion!
Max matrix: Allocation matrix:
0 0 0 0 0 0
0 0 0 0 0 0
9 0 2 3 0 2
```

0 0 0 0 0 0
4 3 3 0 0 2
Process 3 runs to completion!
Max matrix: Allocation matrix:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
4 3 3 0 0 2
Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence: < 2 4 1 3 5 >

OUTPUT 2

Enter the no of processes: 4
Enter the no of resources: 3
Enter the Max Matrix for each process:
For process 1: 4 4 4
For process 2: 2 3 4
For process 3: 5 4 2
For process 4: 7 4 1
Enter the allocation for each process:
For process 1: 1 0 2
For process 2: 2 0 0
For process 3: 1 3 5
For process 4: 4 5 2
Enter the Available Resources: 1 2 4
Max matrix: Allocation matrix:
4 4 4 1 0 2
2 3 4 2 0 0
5 4 2 1 3 5
7 4 1 4 5 2
The system is in an unsafe state!!

PROGRAM-6

Develop a C program to simulate the following contiguous memory allocation Techniques:

a) Worst fit

```
#include<stdio.h>
void main()
{
int fragment[20], b[20], p[20], i, j, nb, np, temp, highest;
static int barray[20], parray[20];

printf("\nMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks: ");
scanf("%d", &nb);
printf("Enter the number of processes: ");
scanf("%d", &np);
printf("\nEnter the size of the blocks:\n");
for(i = 1; i <= nb; i++) {
printf("Block %d: ", i);
scanf("%d", &b[i]);
barray[i] = 0;
}
printf("\nEnter the size of the processes:\n");
for(i = 1; i <= np; i++) {
printf("Process %d: ", i);
scanf("%d", &p[i]);
parray[i] = 0;
}
for(i = 1; i <= np; i++) {
highest = -1; // Resetting highest to a low value for each new process
for(j = 1; j <= nb; j++) {
if(barray[j] == 0 && b[j] >= p[i]) {
temp = b[j] - p[i];
if(temp > highest) {
parray[i] = j;
highest = temp;
}
}
}
fragment[i] = highest;
barray[parray[i]] = 1;
}
```

```
 printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
 for(i = 1; i <= np; i++) {
 printf("\n%d\t\t%d\t\t", i, p[i]);
 if(parray[i] != 0)
 printf("%d\t\t%d\t\t%d", parray[i], b[parray[i]], fragment[i]);
 else
 printf("Not Allocated");
 }
 printf("\n");
}
```

Output:

```
Memory Management Scheme - Worst Fit
Enter the number of blocks: 5
Enter the number of processes: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the processes:
Process 1: 212
Process 2: 412
Process 3: 112
Process 4: 426

Process no   Process size Block no Block size   Fragment
1               212          5          600          388
2               412          2          500           88
3               112          4          300          188
4               426       Not Allocated
```

b) Best fit

```
#include<stdio.h>
void main()
{
 int fragment[20], b[20], p[20], i, j, nb, np, temp, lowest;
 static int barray[20], parray[20];

 printf("\nMemory Management Scheme - Best Fit");
 printf("\nEnter the number of blocks: ");
```

```
scanf("%d", &nb);
printf("Enter the number of processes: ");
scanf("%d", &np);
printf("\nEnter the size of the blocks:\n");
for(i = 1; i <= nb; i++) {
printf("Block %d: ", i);
scanf("%d", &b[i]);
barray[i] = 0;
}
printf("\nEnter the size of the processes:\n");
for(i = 1; i <= np; i++) {
printf("Process %d: ", i);
scanf("%d", &p[i]);
parray[i] = 0;
}
for(i = 1; i <= np; i++) {
lowest = 9999;
for(j = 1; j <= nb; j++) {
if(barray[j] == 0 && b[j] >= p[i]) {
temp = b[j] - p[i];
if(temp < lowest) {
parray[i] = j;
lowest = temp;
}
}
}
fragment[i] = lowest;
barray[parray[i]] = 1;
}
printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i = 1; i <= np; i++) {
printf("\n%d\t\t%d\t\t", i, p[i]);
if(parray[i] != 0)
printf("%d\t\t%d\t\t%d", parray[i], b[parray[i]], fragment[i]);
else
printf("Not Allocated");
}
printf("\n");
}
```

Output:

Memory Management Scheme - Best Fit
Enter the number of blocks: 5
Enter the number of processes: 4
Enter the size of the blocks:

```
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the processes:
Process 1: 212
Process 2: 417
Process 3: 113
Process 4: 426
```

Process no	Process size	Block no	Block size	Fragment
1	212	4	300	88
2	417	2	500	83
3	113	3	200	87
4	426	5	600	174

c) First fit.

```
#include <stdio.h>
#define max 1000
void main() {
 int frag[max], b[max], f[max], i, j, nb, nf, temp;
 static int bf[max], ff[max];
 printf("\nMemory Management Scheme - First Fit");
 printf("\nEnter the number of blocks: ");
 scanf("%d", &nb);
 printf("Enter the number of files: ");
 scanf("%d", &nf);
 printf("\nEnter the size of the blocks:\n");
 for (i = 1; i <= nb; i++) {
 printf("Block %d: ", i);
 scanf("%d", &b[i]);
 bf[i] = 0;
 }
 printf("Enter the size of the files:\n");
 for (i = 1; i <= nf; i++) {
 printf("File %d: ", i);
 scanf("%d", &f[i]);
 ff[i] = 0;
 }
 for (i = 1; i <= nf; i++) {
 for (j = 1; j <= nb; j++) {
 if (bf[j] == 0 && b[j] >= f[i]) {
 ff[i] = j;
```

```
bf[j] = 1;
break;
}
}
if (ff[i] != 0)
frag[i] = b[ff[i]] - f[i];
}
printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragmentation\n");
for (i = 1; i <= nf; i++) {
if (ff[i] != 0)
printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, f[i], ff[i], b[ff[i]], frag[i]);
else
printf("%d\t\t%d\t\tNot Allocated\n", i, f[i]);
}
}
```

Output:

```
Memory Management Scheme - First Fit
Enter the number of blocks: 5
Enter the number of files: 4
Enter the size of the blocks:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600
Enter the size of the files:
File 1: 212
File 2: 412
File 3: 112
File 4: 426

File no: File size: Block no: Block size: Fragmentation
1        212        2         500         288
2        412        5         600         188
3        112        3         200         88
4        426        Not Allocated
```

PROGRAM-7

Develop a C program to simulate page replacement algorithms:

a) FIFO

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
 printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
 printf("\n ENTER THE PAGE NUMBER :\n");
 for(i=1;i<=n;i++)
 scanf("%d",&a[i]);
 printf("\n ENTER THE NUMBER OF FRAMES :");
 scanf("%d",&no);
for(i=0;i<no;i++)
 frame[i]= -1;
 j=0;
 printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
 {
 printf("%d\t\t",a[i]);
 avail=0;
 for(k=0;k<no;k++)
if(frame[k]==a[i])
 avail=1;
 if (avail==0)
 {
 frame[j]=a[i];
 j=(j+1)%no;
 count++;
 for(k=0;k<no;k++)
 printf("%d\t",frame[k]);
}
 printf("\n");
}
 printf("Page Fault Is %d",count);
return 0;
}
```

OUTPUT

```
ENTER THE NUMBER OF PAGES:
20
 ENTER THE PAGE NUMBER: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
 ref string     page frames
7                7 -1 -1
0                7 0 -1
1                 7 0 1
2                 2 0 1
0

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

3
2

1                 0 1 3
2                 0 1 2

0
1
7               7 1 2
0               7 0 2
1               7 0 1
```

Page Fault Is 15

b) LRU

```
#include<stdio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictim();
void main()
{
printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no. of Frames....");
```

```
scanf("%d",&nof);
printf(" Enter no.of reference string..");
scanf("%d",&nor);
printf("\n Enter reference string..");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
printf("\n\t The given reference string:");
printf("\n……..");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
{
frm[i]=-1;
lrucal[i]=0;
} for(i=0;i<10;i++)
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\t Reference NO %d->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++;
else
victim=lruvictim();
pf++;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
recent[ref[i]]=i;
}
printf("\n\n\t No.of page faults...%d",pf);
}
```

```
int lruvictim()
{
int i,j,temp1,temp2;
for(i=0;i<nof;i++)
{
temp1=frm[i];
lrucal[i]=recent[temp1];
}
temp2=lrucal[0];
for(j=1;j<nof;j++)
{
if(temp2>lrucal[j])
temp2=lrucal[j];
}
for(i=0;i<nof;i++)
if(ref[temp2]==frm[i])
return i;
return 0;
}
```

OUTPUT

```
LRU PAGE REPLACEMENT ALGORITHM
 Enter no.of Frames....3
 Enter no.of reference string..20
 Enter reference string.
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
 LRU PAGE REPLACEMENT ALGORITHM
 The given reference string:
…….. 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 Reference NO 7->  7 -1 -1
 Reference NO 0->  7 0 -1
 Reference NO 1->  7 0 1
 Reference NO 2->   2 0 1
 Reference NO 0->
 Reference NO 3->   2 0 3
 Reference NO 0->
 Reference NO 4->   4 0 3
 Reference NO 2->   4 0 2
 Reference NO 3->   4 3 2
 Reference NO 0->   0 3 2
 Reference NO 3->
 Reference NO 2->
 Reference NO 1->   1 3 2
```

```
Reference NO 2->
Reference NO 0->   1 0 2
Reference NO 1->
Reference NO 7->   1 0 7
Reference NO 0->
Reference NO 1->

No. of page faults...12
```

PROGRAM-8

Simulate following File Organization Techniques:
a) Single level directory

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
int i,ch;
char f[30];
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5.
Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;
break;
case 2: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
```

```
{
printf("File %s is deleted ",f);
strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
else
dir.fcnt--;
break;
case 3: printf("\n Enter the name of the file -- ");
scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
printf("File %s is found ", f);
break;
}
}
if(i==dir.fcnt)
printf("File %s not found",f);
break;
case 4: if(dir.fcnt==0)
printf("\n Directory Empty");
else
{
printf("\n The Files are -- ");
for(i=0;i<dir.fcnt;i++)
printf("\t%s",dir.fname[i]);
}
break;
default: exit(0);
}
}
}
```

OUTPUT

```
Enter name of directory -- CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A
```

```
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1
Enter the name of the file -- C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4
The Files are -- A B C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3
Enter the name of the file – ABC
File ABC not found
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 2
Enter the name of the file – B
File B is deleted
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5
```

b) Two level directory

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
void main()
{
int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0;
while(1)
{
printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");
printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\n Enter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
```

```
dcnt++;
printf("Directory created");
break;
case 2: printf("\n Enter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
dir[i].fcnt++;
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);

break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
```

```
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);

goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
}
```

OUTPUT

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1
Enter name of directory -- DIR1
Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1

```
Enter name of directory -- DIR2
Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR1
Enter name of the file -- A2
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2
Enter name of the directory – DIR2
Enter name of the file -- B1
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 5
Directory Files
DIR1 A1 A2
DIR2 B1
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 4
Enter name of the directory – DIR
Directory not found
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 3
Enter name of the directory – DIR1
Enter name of the file -- A2
File A2 is deleted
```

PROGRAM-9

Develop a C program to simulate the Linked file allocation strategies.

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
int f[50], p,i, st, len, j, c, k, a;
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
```

```
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
}
```

OUTPUT

```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 3 5
Enter index starting block and length: 2 4
2-------->1
3 Block is already allocated
4-------->1
```

5 Block is already allocated
6-------->1
7-------->1
Do you want to enter more file (Yes - 1/No - 0)

PROGRAM-10

Develop a C program to simulate SCAN disk scheduling algorithm.

```
#include <stdio.h>
int request[50];
int SIZE;
int pre;
int head;
int uptrack;
int downtrack;
struct max{
int up;
int down;
} kate[50];
int dist(int a, int b){
if (a > b)
return a - b;
return b - a;
}
void sort(int n){
int i, j;
for (i = 0; i < n - 1; i++){
for (j = 0; j < n - i - 1; j++){
if (request[j] > request[j + 1]){
int temp = request[j];
request[j] = request[j + 1];
request[j + 1] = temp;
}
}
}
j = 0;
i = 0;
while (request[i] != head){
kate[j].down = request[i];
j++;
i++;
}
downtrack = j;
i++;
j = 0;
```

```
while (i < n)
{
kate[j].up = request[i];
j++;
i++;
}
uptrack = j;
}
void scan(int n){
int i;
int seekcount = 0;
printf("SEEK SEQUENCE = ");
sort(n);
if (pre < head){
for (i = 0; i < uptrack; i++){
printf("%d ", head);
seekcount = seekcount + dist(head, kate[i].up);
head = kate[i].up;
}
for (i = downtrack - 1; i > 0; i--){
printf("%d ", head);
seekcount = seekcount + dist(head, kate[i].down);
head = kate[i].down;
}
}
else{
for (i = downtrack - 1; i >= 0; i--){
printf("%d ", head);
seekcount = seekcount + dist(head, kate[i].down);
head = kate[i].down;
}
for (i = 0; i < uptrack - 1; i++){
printf("%d ", head);
seekcount = seekcount + dist(head, kate[i].up);
head = kate[i].up;
}
}
printf(" %d\nTOTAL DISTANCE :%d", head, seekcount);
}
int main(){
int n, i;
printf("ENTER THE DISK SIZE :\n");

scanf("%d", &SIZE);
printf("ENTER THE NO OF REQUEST SEQUENCE :\n");
scanf("%d", &n);
```

```
printf("ENTER THE REQUEST SEQUENCE :\n");
for (i = 0; i < n; i++)
scanf("%d", &request[i]);
printf("ENTER THE CURRENT HEAD :\n");
scanf("%d", &head);
request[n] = head;
request[n + 1] = SIZE - 1;
request[n + 2] = 0;
printf("ENTER THE PRE REQUEST :\n");
scanf("%d", &pre);
scan(n + 3);

}
```

OUTPUT

```
ENTER THE DISK SIZE:
200
ENTER THE NO OF REQUEST SEQUENCE:
7
ENTER THE REQUEST SEQUENCE:
82, 170, 43,140, 24,16,190
ENTER THE CURRENT HEAD:
50
ENTER THE PRE REQUEST:
43
SEEK SEQUENCE = 50 82 140 170 190 199 43 24 16
TOTAL DISTANCE: 332

ENTER THE DISK SIZE:
200

ENTER THE NO OF REQUEST SEQUENCE:
8
ENTER THE REQUEST SEQUENCE:
98
183
37
122
14
124
65
67
ENTER THE CURRENT HEAD:
53
ENTER THE PRE REQUEST:
```

124
SEEK SEQUENCE = 53 37 14 0 65 67 98 122 124 183

TOTAL DISTANCE: 236

Viva Questions:

1) What is an operating system?

The operating system is a software program that facilitates computer hardware to communicate and operate with the computer software. It is the most important part of a computer system without it computer is just like a box.

2) What is the main purpose of an operating system?

There are two main purposes of an operating system:

- It is designed to make sure that a computer system performs well by managing its computational activities.
- It provides an environment for the development and execution of programs.

3) What are the different operating systems?

- Batched operating systems
- Distributed operating systems
- Timesharing operating systems
- Multi-programmed operating systems
- Real-time operating systems

4) What is a socket?

A socket is used to make connection between two applications. Endpoints of the connection are called socket.

5) What is a real-time system?

Real-time system is used in the case when rigid-time requirements have been placed on the operation of a processor. It contains a well defined and fixed time constraints.

6) What is kernel?

Kernel is the core and most important part of a computer operating system which provides basic services for all parts of the OS.

7) What do you mean by a process?

An executing program is known as process. There are two types of processes:

- Operating System Processes
- User Processes

8) What are the different states of a process?

A list of different states of process:

- New Process
- Running Process
- Waiting Process
- Ready Process
- Terminated Process

9) What is the difference between micro kernel and macro kernel?

Micro kernel: micro kernel is the kernel which runs minimal performance affecting services for operating system. In macro kernel operating system all other operations are performed by processor.

10) What is context switching?

Transferring the control from one process to other process requires saving the state of the old process and loading the saved state for new process.

11) What are real-time systems?

Real-time systems are used when rigid time requirements have been placed on the operation of a processor. It has well defined and fixed time constraints.

12) What do you mean by a process?

An executing program is known as process. There are two types of processes:

- Operating System Processes
- User Processes

13) What are the advantages of a multiprocessor system?

With an increased number of processors, there is a considerable increase in throughput. It can also save more money because they can share resources. Finally, overall reliability is increased as well.

14) What is time- sharing system?

In a Time-sharing system, the CPU executes multiple jobs by switching among them, also known as multitasking. This process happens so fast that users can interact with each program while it is running.

15) Give some benefits of multithreaded programming.

- There is increased responsiveness to the user
- Resource sharing within the process
- Economy
- Utilization of multiprocessing architecture

16)Explain FCFS?

FCFS stands for First-come, first-served. It is one type of scheduling algorithm. In this scheme, the process that requests the CPU first is allocated the CPU first. Implementation is managed by a FIFO queue.

17)What is fragmentation?

Fragmentation is memory wasted. It can be internal if we are dealing with systems that have fixed-sized allocation units, or external if we are dealing with systems that have variable-sized allocation units.

18) Give an example of a Process State.

- **New State** – means a process is being created
- **Running** – means instructions are being executed
- **Waiting** – means a process is waiting for certain conditions or events to occur
- **Ready** – means a process is waiting for an instruction from the main processor
- **Terminate** – means a process is stopped abruptly

19)Differentiate internal commands from external commands.

Internal commands are built-in commands that are already part of the operating system. External commands are separate file programs that are stored in a separate folder or directory.

20) Operating System assumes responsibility, that of serving as a Control program

- Control program
- Contribute program
- Supplement program
- Supply program

21) To avoid the delay due to manual operation Job sequencing was introduced

- Assistant
- Efficient
- Job sequencing
- Manual

22) System level security is offered by the password in a multi-user environment

- code
- password
- name
- secret code

23) A control program controls the execution of user programs to prevent errors and improper use of the computer.

- control program
- contribute program
- supplement program
- supply program

24) Under DOS, what command will you type when you want to list down the files in a directory, and at the same time pause after every screen output?

- dir /w

- dir /p
- dir /s
- dir /w /p

Answer: d) dir /w /p

25) Explain why Ubuntu is safe and not affected by viruses?

- It does not support malicious e-mails and contents, and before any e-mail is opened by users it will go through many security checks
- Ubuntu uses Linux, which is a super secure O.S system
- Unlike other O.S, countless Linux users can see the code at any time and can fix the problem if there is any
- Malware and viruses are coded to take advantage of the weakness in Windows.