KLS Vishwanathrao Deshpande Institute of Technology Haliyal– 581329

Operating Systems Laboratory [BCS303]

for

III Semester B.E.

As prescribed by

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI–590014

(For the Academic Year 2025-2026)

Prepared by

Dr. Vijet Swadi

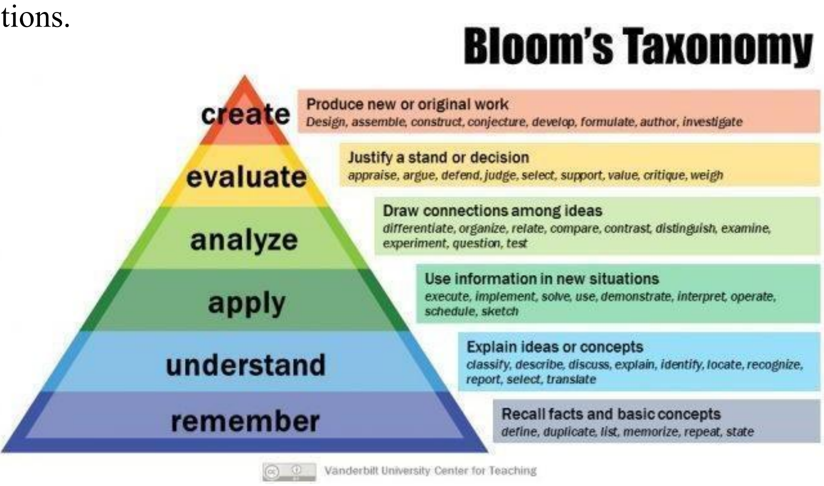
Department of Computer Science & Engineering

Index

PROGRAM OUTCOMES(POs)

Program Outcomes as defined by NBA (PO) 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.


Bloom's Taxonomy
create
Produce new or original work
Design, assemble, construct, conjecture, develop, formulate, author, investigate
evaluate
Justify a stand or decision
appraise, argue, defend, judge, select, support, value, critique, weigh
analyze
Draw connections among ideas
differentiate, organize, relate, compare, contrast, distinguish, examine, experiment, question, test
apply
Use information in new situations
execute, implement, solve, use, demonstrate, interpret, operate, schedule, sketch
understand
Explain ideas or concepts
classify, describe, discuss, explain, identify, locate, recognize, report, select, translate
remember
Recall facts and basic concepts
define, duplicate, list, memorize, repeat, state
Vanderbilt University Center for Teaching

Vision (College)

To nurture talent & enrich society through excellence in technical education, research & innovation.

Mission (College)

- To augment innovative Pedagogy & kindle quest for interdisciplinary learning & to enhance conceptual understanding.
- To build competence, professional ethics & develop entrepreneurial thinking.
- To strengthen Industry Institute Partnership & explore global collaborations.
- To inculcate culture of socially responsible citizenship.
- To focus on Holistic & Sustainable development.

Vision (Dept)

To achieve excellence in technical education, research, innovation in computer science and Engineering by emphasizing on global trending technologies.

Mission (Dept)

- To train students with conceptual understanding through innovative pedagogies.
- To imbibe professional, research and entrepreneurial Skills with commitment to the nation development at large.
- To strengthen the industry institute Interaction.
- To promote life–long learning with a sense of societal & ethical responsibilities.

Program Educational Objectives (PEO)

PEO1	To develop an ability to identify and analyze the requirements of Computer Science and Engineering in design and providing novel engineering solutions..
PEO2	To develop abilities to work in team on multidisciplinary projects with effective communication skills, ethical qualities and leadership roles.
PEO3	To develop abilities for successful Computer Science Engineer and achieve higher career goals.

Program Specific Outcomes (PSO)

PSO 1	To develop ability to model real world problems using appropriate data structure and suitable algorithm in the area of Data Processing, System Engineering, Networking for varying complexity.
PSO 2	To develop an ability to use modern computer languages, environments and platforms in creating innovative career.

CO's And PO's Mapping Chart

Subject with code: OPERATING SYSTEMS LABORATORY (BCS303)

Semester: 3rd

AY:2024-25

S.No.	Description	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO 2
1.	Explain the structure and functionality of operating system	1	1												
2.	Apply appropriate CPU scheduling algorithms for the given problem.		2	2										1	
3.	Analyse the various techniques for process synchronization and deadlock handling.		2	2											
4.	Apply the various techniques for memory management		2	2											
5.	Explain file and secondary storage management strategies	2	3												

Degree of compliance Low:1 Medium:2 High:3

Evaluation:

	Particulars	Marks	Total
CIA	Performance	03	**15**
	Journal	10	
	Viva-voce	02	
	Lab IA	10	**10**
	Grand Total		**25**

Mapping of Experiments with CO, PO and PSO

S.No.	Experiment Details	CO	PO	PSO
1	Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)	1	1,2	1
2	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.	2	2,3	2
3	Develop a C program to simulate producer-consumer problem using semaphores.	3	2	1
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.	3	3	2
5	Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.	3	3	1
6	Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.	4	2	2
7	Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU	4	2	1
8	Simulate following File Organization Techniques a) Single level directory b) Two level directory	5	3	1
9	Develop a C program to simulate the Linked file allocation strategies.	5	1,3	2
10.	Develop a C program to simulate SCAN disk scheduling algorithm.	5	3	2

EXPERIMENT WISE LESSON PLAN

Experiment No.1	
Name	fork (), exec(), wait(), create process, terminate process
Objectives	● To Develop a c program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process)
Experiment No.2	
Name	a) FCFS b) SJF c) Round Robin d) Priority Scheduling Algorithms
Objectives	● To Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority.
Experiment No.3	
Name	Producer-Consumer problem
Objectives	● To Develop a C program to simulate producer-consumer problem using semaphores
Experiment No.4	
Name	Inter Process Communication
Objectives	● 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.
Experiment No.5	
Name	Bankers Algorithm for DeadLock Avoidance.
Objectives	● To Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.
Experiment No.6	
Name	Contiguous memory allocation Techniques
Objectives	● To Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.
Experiment No.7	
Name	**Page Replacement algorithms**
Objectives	● To Develop a C program to simulate page replacement algorithms: a) FIFO b) LRU
Experiment No.8	
Name	**File Organization Techniques**
Objectives	● To Simulate following File Organization Techniques a) Single level directory b) Two level directory
Experiment No.9	
Name	**Linked file allocation strategies**
Objectives	● To Develop a C program to simulate the Linked file allocation strategies.
	Experiment No.10
Name	**Disk scheduling algorithm.**
Objectives	● To Develop a C program to simulate SCAN disk scheduling algorithm.

EXPT. NO:1

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

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>  // Include for exit()
#include <sys/wait.h>  // Include for wait()

int main() {
        pid_t pid;
        pid = fork();  /* fork a child process */

        if (pid < 0) {
        fprintf(stderr, "Fork Failed");  /* Error occurred */
        return 1;
        }
        else if (pid == 0) {  /* Child process */
        // Print details in the child process without calling exec or execlp
        printf("Child: pid value = %d\n", pid);
        printf("Child: Process id = %d\n", getpid());
        printf("Child: Parent-Process id = %d\n", getppid());
        printf("Child: Exiting from child\n");
        exit(0);
        }
        else {  /* Parent process */
        wait(NULL);  /* Parent waits for the child to complete */
        printf("Child Complete\n");
        printf("Parent: Waiting for child\n");
        printf("Parent: Process id = %d\n", getpid());
        printf("Parent: Child-Process id = %d\n", pid);
        }

        return 0;
}
```

Output

```
Child: pid value = 0
Child: Process id = 7975
Child: Parent-Process id = 7974
Child: Exiting from child
Child Complete
Parent: Waiting for child
Parent: Process id = 7974
Parent: Child-Process id = 7975
```

EXPT. NO:2

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

a) FIRST COME FIRST SERVE: Source Code

```
#include<stdio.h>
int main()
{
 int bt[20], wt[20], tat[20], i, n;
 float wtavg, tatavg;
 printf("\nEnter the number of processes: ");
 scanf("%d", &n);
 for(i=0;i<n;i++)
 {
   printf("\nEnter Burst Time for Process %d -- ", i);
   scanf("%d", &bt[i]);
 }
 wt[0] = wtavg = 0;
 tat[0] = tatavg = bt[0];
 for(i=1;i<n;i++)
 {
  wt[i] = wt[i-1] +bt[i-1];
  tat[i] = tat[i-1] +bt[i];
  wtavg = wtavg + wt[i];
  tatavg = tatavg + tat[i];
 }
 printf("\t PROCESS \tBURST TIME \t WAITING TIME
             \t TURNAROUND TIME\n");
 for(i=0;i<n;i++)
   printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
 printf("\nAverage Waiting Time -- %f", wtavg/n);
 printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

Output

```
Enter the number of processes -- 3

Enter Burst Time for Process 0 -- 24

Enter Burst Time for Process 1 -- 3

Enter Burst Time for Process 2 – 3

    PROCESS    BURST TIME    WAITING TIME    TURNAROUND TIME

        P0         24             0              24
        P1          3            24              27
        P2          3            27              30

Average Waiting Time -- 17.000000
Average Turnaround Time -- 27.000000
```

b) SHORTEST JOB FIRST:

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

 for(i=0;i<n;i++)
    for(k=i+1;k<n;k++)
       if(bt[i]>bt[k])
       {
        temp=bt[i];
        bt[i]=bt[k];
        bt[k]=temp;
        temp=p[i];
        p[i]=p[k];
        p[k]=temp;
       }
       wt[0] = wtavg = 0;
       tat[0] = tatavg = bt[0];
       for(i=1;i<n;i++)
       {
        wt[i] = wt[i-1] +bt[i-1];
        tat[i] = tat[i-1] +bt[i];
        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
       }
 printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
 for(i=0;i<n;i++)
 printf("\n\t P%d \t\t %d \t\t %d\t\t %d", p[i], bt[i], wt[i], tat[i]);
 printf("\nAverage Waiting Time -- %f", wtavg/n);
 printf("\nAverage Turnaround Time -- %f", tatavg/n);
}
```

Output:

```
    Enter the number of processes -- 4
    Enter Burst Time for Process 0 -- 6
    Enter Burst Time for Process 1 -- 8
    Enter Burst Time for Process 2 -- 7
    Enter Burst Time for Process 3 -- 3

  PROCESS    BURST TIME    WAITING TIME    TURNAROUND TIME

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

Average Waiting Time -- 7.000000
Average Turnaround Time -- 13.000000
```

C. Round Robin Scheduling

SOURCE CODE

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

Output:

```
Enter the no of processes – 3
Enter Burst Time for process 1 – 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 – 3
Enter the size of time slice – 3


PROCESS   BURST TIME          WAITING TIME        TURNAROUNDTIME
   1          24                   6                    30
   2          3                    4                    7
   3          3                    7                    10

The Average Turnaround time is – -----15.666667
The Average Waiting time is ------------ 5.666667
```

d). PRIORITY SCHEDULING

ALGORITHM:

```
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as ‗0‘and its burst time as its turnaround time
Step 6: Arrange the processes based on process priority
Step 7: For each process in the Ready Q calculate
Step 8: for each process in the Ready Q calculate
        a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)
        b) Turnaround time (n)= waiting time(n)+Burst time(n)
Step 9: Calculate
        c) Average waiting time = Total waiting Time / Number of process
        d) Average Turnaround time = Total Turnaround Time / Number of process
        Print the results in an order.
Step10: Stop
```

SOURCE CODE:

```
#include<stdio.h>
int main()
{
 int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
 float wtavg,tatavg;
 printf("Enter the number of processes --- ");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
  p[i] = i;
  printf("Enter the Burst Time & Priority of Process %d --- ",i);
  scanf("%d %d",&bt[i], &pri[i]);
 }
 for(i=0;i<n;i++)
    for(k=i+1;k<n;k++)
       if(pri[i] > pri[k])
       {
        temp=p[i];
        p[i]=p[k];
        p[k]=temp;
        temp=bt[i];
        bt[i]=bt[k];
        bt[k]=temp;
        temp=pri[i];
        pri[i]=pri[k];
        pri[k]=temp;
       }
 wtavg = wt[0] = 0;
 tatavg = tat[0] = bt[0];
 for(i=1;i<n;i++)
 {
  wt[i] = wt[i-1] + bt[i-1];
  tat[i] = tat[i-1] + bt[i];
  wtavg = wtavg + wt[i];
  tatavg = tatavg + tat[i];
 }
 printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
 for(i=0;i<n;i++)
     printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
 printf("\nAverage Waiting Time is --- %f",wtavg/n);
 printf("\nAverage Turnaround Time is --- %f",tatavg/n);

}
```

Output

```
Enter the number of processes --- 5
Enter the Burst Time & Priority of Process 0 --- 10   3
Enter the Burst Time & Priority of Process 1 --- 1   1
Enter the Burst Time & Priority of Process 2 --- 2   4
Enter the Burst Time & Priority of Process 3 --- 1   5
Enter the Burst Time & Priority of Process 4 --- 5   2

PROCESS     PRIORITY    BURST TIME    WAITING TIME    TURNAROUND TIME
   1           1            1               0                1
   4           2            5               1                6
   0           3           10               6               16
   2           4            2              16               18
   3           5            1              18               19
Average Waiting Time is --- 8.200000
Average Turnaround Time is --- 12.000000
```

EXPT. NO:3
Develop a C program to simulate producer-consumer problem using semaphores.

Producer consumer problem is a synchronization problem. There is a fixed size buffer where the producer produces items and that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#define MAX 5
sem_t in_sem;
sem_t rm_sem;
void * producer ()
{
int count = 0;
while (1)
{
sem_wait(&in_sem);
sleep(rand()%3);
printf ( "Producer inserted at %d\n" , count % MAX + 1);
count++;
sem_post(&rm_sem);
}
}
void * consumer ()
{
int count = 0;
while (1)
{
sem_wait(&rm_sem);
printf ( "Consumer removed at %d\n" , count % MAX + 1);
sleep(rand()%3);
count++;
sem_post(&in_sem);
```

```
        }
        }
        int main ()
        {
        pthread_t threads[2];
        sem_init(&in_sem, 0, MAX);
        sem_init(&rm_sem, 0, 0);
        srand(time( NULL ));
        pthread_create(&threads[0], NULL , producer, NULL );
        pthread_create(&threads[1], NULL , consumer, NULL );
        pthread_join(threads[0], NULL );
        pthread_join(threads[1], NULL );

        sem_destroy(&in_sem);
        sem_destroy(&rm_sem);
}
```

Output:

```
n@ubuntu-VirtualBox:~/Documents/OS/os5$ gcc producer_consumer.c -l
pthread
n@ubuntu-VirtualBox:~/Documents/OS/os5$ ./a.out
Producer inserted at 1
Consumer removed at 1
Producer inserted at 2
Producer inserted at 3
Consumer removed at 2
Consumer removed at 3
Producer inserted at 4
Consumer removed at 4
Producer inserted at 5
Producer inserted at 1
Consumer removed at 5
Producer inserted at 2
Producer inserted at 3
Consumer removed at 1
Producer inserted at 4
Producer inserted at 5
Consumer removed at 2
Consumer removed at 3
Consumer removed at 4
Consumer removed at 5
Producer inserted at 1
Consumer removed at 1
Producer inserted at 2
Consumer removed at 2
Producer inserted at 3
Producer inserted at 4
```

EXPT. NO:4

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

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#define N 10
int process[N] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int r_count = 0;
sem_t rw_sem;
sem_t rc_sem;
void * reader ( void * num)
{
 int n = *( int *)num;
 sem_wait(&rc_sem);
 if (!r_count)
 {
  printf ( "Waiting for reading\n" );
  sem_wait(&rw_sem);
 }
 r_count++;
 sem_post(&rc_sem);
 printf ( "Process %d is reading\n" , n+1);
 sleep(2);
 sem_wait(&rc_sem);
 r_count--;
 if (!r_count)
 {
  printf ( "Finished reading\n" );
  sem_post(&rw_sem);
 }
 sem_post(&rc_sem);
}
void * writer ( void * num)
{
 int n = *( int *)num;
 printf ( "Waiting for writing\n" );
 sem_wait(&rw_sem);
 printf ( "Process %d is writing\n" , n+1);
 sleep(4);
 printf ( "Finished writing\n" );
 sem_post(&rw_sem);
}

int main ()
{
 pthread_t threads[N];
 sem_init(&rw_sem, 0, 1);
 sem_init(&rc_sem, 0, 1);
 srand(time( NULL ));
 int r = rand()%N;
 for ( int i = 0; i < N; i++)
 {
  if (r == i)
  {
   pthread_create(&threads[i], NULL , writer, &process[i]);
   r = rand()%(N-i) + i;
   continue ;
   }
  pthread_create(&threads[i], NULL , reader, &process[i]);
  }
```

```
for ( int i = 0; i < N; i++)
 {
  pthread_join(threads[i], NULL );
 }



 sem_destroy(&rw_sem);
 sem_destroy(&rc_sem);
}
```

Output

```
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ gedit os4.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ gcc os4.c -l pthread
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ ./a.out
Waiting for reading
Process 1 is reading
Process 8 is reading
Process 9 is reading
Process 7 is reading
Process 6 is reading
Process 4 is reading
Process 2 is reading
Waiting for writing
Waiting for writing
Process 10 is reading
Finished reading
Process 3 is writing
Finished writing
Process 5 is writing
Finished writing
```

EXPT. NO:5

Develop a C program to simulate Bankers Algorithm for Deadlock Avoidance

DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed thetotal number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

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

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively,
Work=Available and
Finish[i] =False.
2. Find an i such that both
Finish[i] =False
Need<=Work
If no such I exists go to step 4.
3. work= work + Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state. Resource request algorithm
Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources areavailable.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I;
Allocation I=Allocation +Request I;
Need i=Need i- Request I;
If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.
***11.** end*

SOURCE CODE :

```
#include<stdio.h>
#include<string.h>
void main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
printf("Enter the no. of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");

for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
```

```
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0;
for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A:
for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
if(c==m)
{
printf("All the resources can be allocated to Process %d", i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
else
printf("\n System is in safe mode");
printf("\n The given state is safe state");
}
```

Output

```
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ cc os5.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ ./a.out
Enter the no. of processes and resources:4 3
Enter the claim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2

Available resources are:   6   2   3

Process 2 executed?:y
All the resources can be allocated to Process 3

Available resources are:   8   3   4

Process 3 executed?:y
All the resources can be allocated to Process 4

Available resources are:   8   3   6

Process 4 executed?:y
All the resources can be allocated to Process 1

Available resources are:   9   3   6

Process 1 executed?:y

 System is in safe mode
 The given state is safe state
```

EXPT. NO:6

Develop a C Program to simulate the following contiguous memory allocation techniques.

a) Worst-fit b) Best-fit c) First-fit

DESCRIPTION

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate.

Best-fit strategy chooses the block that is closest in size to the request.
First-fit chooses the first available block that is large enough.
Worst-fit chooses the largest available block.

WORST-FIT

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


    }

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

Output:

```
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ gedit 6.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ cc 6.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ ./a.out

Memory Management Scheme- First Fit
Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:   File_size :   Block_no:   Block_size:   Fragement
1            1             1              5             4
2            4             3              7             3
```

6b. BEST-FIT

```
#include<stdio.h>
#define max 25
void main()
{
 int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
 static int bf[max],ff[max];
 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]);
 }
 printf("Enter the size of the files :-\n");
 for(i=1;i<=nf;i++)
 {
  printf("File %d:",i);
  scanf("%d",&f[i]);
 }
 for(i=1;i<=nf;i++)
 {
  for(j=1;j<=nb;j++)
  {
   if(bf[j]!=1)
   {
     temp=b[j]-f[i];
     if(temp>=0)
        if(lowest>temp)
        {
         ff[i]=j;
         lowest=temp;
        }
    }
   }
  frag[i]=lowest; bf[ff[i]]=1; lowest=10000;
 }
 printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment");
 for(i=1;i<=nf && ff[i]!=0;i++)
 printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

Output

```
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ gedit 6b.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ cc 6b.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File No    File Size    Block No    Block Size    Fragment
1             1             2             2             1
2             4             1             5             1
```

6C. FIRST-FIT

```
#include <stdio.h>
int main() {
    int nb, np, i, j;
    int blockSize[10], processSize[10], allocation[10];

    printf("Enter number of memory blocks: ");
    scanf("%d", &nb);
    printf("Enter size of each block:\n");
    for (i = 0; i < nb; i++) {
        printf("Block %d: ", i + 1);
        scanf("%d", &blockSize[i]);
    }
    printf("\nEnter number of processes: ");
    scanf("%d", &np);
    printf("Enter size of each process:\n");
    for (i = 0; i < np; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d", &processSize[i]);
        allocation[i] = -1;  // initialize all as not allocated
    }

    // First Fit Allocation
    for (i = 0; i < np; i++) {
        for (j = 0; j < nb; j++) {
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j; // allocate block j to process i
                blockSize[j] -= processSize[i]; // reduce block size
                break; // move to next process
            }
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (i = 0; i < np; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
    return 0;
}
```

Output

```
Enter number of memory blocks: 5
Enter size of each block:
Block 1: 100
Block 2: 500
Block 3: 200
Block 4: 300
Block 5: 600

Enter number of processes: 4
Enter size of each process:
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426

Process No.   Process Size   Block No.
1                 212           2
2                 417           5
3                 112           2
4                 426           Not Allocated
```

EXPT. NO:7

Develop a C program to simulate page replacement algorithms.
a) FIFO b) LRU

DESCRIPTION:

Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults.
FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal. LRU

In this algorithm page will be replaced which is least recently used.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the frame Place avail[i]=1 if page is placed in the frame Count page faults
7. Print the results.
8. Stop the process.

a) FIRST IN FIRST OUT SOURCE CODE

```
#include <stdio.h>

int main() {
   int i, j, n, frames, pages[50], frame[10];
   int pageFaults = 0, next = 0, flag;

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

   printf("Enter the page reference string:\n");
   for (i = 0; i < n; i++)
      scanf("%d", &pages[i]);

   printf("Enter number of frames: ");
   scanf("%d", &frames);
```

```
// Initialize all frames as empty (-1)
for (i = 0; i < frames; i++)
    frame[i] = -1;



printf("\nPage Reference String: ");
for (i = 0; i < n; i++)
    printf("%d ", pages[i]);
printf("\n\nPage\tFrames\t\tPage Fault\n");

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

    // Check if page is already in frame
    for (j = 0; j < frames; j++) {
        if (frame[j] == pages[i]) {
            flag = 1;  // Page found, no fault
            break;
        }
    }

    // Page not found → Replace using FIFO
    if (flag == 0) {
        frame[next] = pages[i];
        next = (next + 1) % frames; // Circular queue
        pageFaults++;
    }

    // Print current status of frames
    printf("%d\t", pages[i]);
    for (j = 0; j < frames; j++) {
        if (frame[j] != -1)
            printf("%d ", frame[j]);
        else
            printf("- ");
    }

    if (flag == 0)
        printf("\t\tYes");
    else
        printf("\t\tNo");
    printf("\n");
}
```

```
    printf("\nTotal Page Faults = %d\n", pageFaults);
    printf("Page Fault Rate = %.2f%%\n", ((float)pageFaults / n) * 100);

    return 0;
}
```

OUTPUT:

```
Enter number of pages: 12
Enter the page reference string:
1 2 3 4 1 2 5 1 2 3 4 5
Enter number of frames: 3

Page Reference String: 1 2 3 4 1 2 5 1 2 3 4 5

Page    Frames          Page Fault
1           1 - -         Yes
2         1 2 -          Yes
3         1 2 3          Yes
4         4 2 3          Yes
1         4 1 3          Yes
2         4 1 2          Yes
5         5 1 2          Yes
1         5 1 2          No
2         5 1 2          No
3         3 1 2          Yes
4         3 4 2          Yes
5         3 4 5          Yes

Total Page Faults = 9
Page Fault Rate = 75.00%
```

b) LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM:

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

SOURCE CODE :

```
#include <stdio.h>
int main() {
    int pages[50], frame[10], count[10];
    int n, frames, pageFaults = 0, i, j, k, min, next, flag1, flag2;

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

    printf("Enter the page reference string:\n");
    for (i = 0; i < n; i++)
        scanf("%d", &pages[i]);

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

    for (i = 0; i < frames; i++) {
        frame[i] = -1;
        count[i] = 0;
    }
    printf("\nPage Reference String: ");
    for (i = 0; i < n; i++)
        printf("%d ", pages[i]);
    printf("\n\nPage\tFrames\t\tPage Fault\n");

    for (i = 0; i < n; i++) {
        flag1 = flag2 = 0;
        // Check if page is already in a frame
        for (j = 0; j < frames; j++) {
            if (frame[j] == pages[i]) {
                count[j] = i + 1;  // Update recent use
                flag1 = flag2 = 1;
                break;
            }
        }
        // If page not found, replace least recently used
        if (flag1 == 0) {
            for (j = 0; j < frames; j++) {
                if (frame[j] == -1) { // Empty frame found
                    frame[j] = pages[i];
                    count[j] = i + 1;
                    flag2 = 1;
                    pageFaults++;
                    break;
                }
            }
        }

        // If no empty frame → replace LRU page
        if (flag2 == 0) {
            min = 0;
            for (j = 1; j < frames; j++) {
                if (count[j] < count[min])
                    min = j;
            }
            frame[min] = pages[i];
            count[min] = i + 1;
            pageFaults++;
        }

        // Print frame status
        printf("%d\t", pages[i]);
        for (k = 0; k < frames; k++) {
            if (frame[k] != -1)
                printf("%d ", frame[k]);
            else
                printf("- ");
        }
        if (flag1 == 1)
            printf("\t\tNo\n");
        else
            printf("\t\tYes\n");
    }

    printf("\nTotal Page Faults = %d\n", pageFaults);
    printf("Page Fault Rate = %.2f%%\n", ((float)pageFaults / n) * 100);
    return 0;
}
```

OUTPUT:

```
Enter number of pages: 12
Enter the page reference string:
1 2 3 4 1 2 5 1 2 3 4 5
Enter number of frames: 3

Page Reference String: 1 2 3 4 1 2 5 1 2 3 4 5

Page    Frames        Page Fault
1       1 - -         Yes
2       1 2 -          Yes
3       1 2 3          Yes
4       4 2 3          Yes
1       4 1 3          Yes
2       4 1 2          Yes
5       5 1 2          Yes
1       5 1 2          No
2       5 1 2          No
3       3 1 2          Yes
4       3 4 2          Yes
5       3 4 5          Yes

Total Page Faults = 10
Page Fault Rate = 83.33%
```

EXPT. NO:8

Simulate following file organization techniques

a) **Single level directory b) Two level directory**

a) Single level directory

AIM: Program to simulate Single level directory file organization technique.

DESCRIPTION: The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory.

SOURCE CODE :

```
#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\n1. Create File\t2. Delete File\t3. Search File \n4. Display Files\t5. Exit\nEnter your choice -- ");
  scanf("%d",&ch);
 switch(ch)
 {
  case 1: printf("\nEnter the name of the file -- ");
         scanf("%s",dir.fname[dir.fcnt]);
         dir.fcnt++;
  break;
  case 2: printf("\nEnter the name of the file -- ");
         scanf("%s",f);
         for(i=0;i<dir.fcnt;i++)
         {
          if(strcmp(f, dir.fname[i])==0)
         {
          printf("File %s is deleted ",f);



          strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
          break;
         }
        }
       if(i==dir.fcnt)
         printf("File %s not found",f);
       else
         dir.fcnt--;
       break;
  case 3: printf("\nEnter the name of the file-- ");
       scanf("%s",f);
       for(i=0;i<dir.fcnt;i++)
       {
        if(strcmp(f, dir.fname[i])==0)
        {
         printf("File %s is found ", f);
         break;
        }
       }
       if(i==dir.fcnt)
       printf("File %s not found",f);
       break;
 case 4: if(dir.fcnt==0)
         printf("\nDirectory Empty");
       else
       {
        printf("\nThe Files are -- ");
        for(i=0;i<dir.fcnt;i++)
           printf("\t%s",dir.fname[i]);
       }
       break;
default: exit(0);
}
}
}
```

Output

```
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ gedit 8a.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ cc 8a.c
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:~$ ./a.out1

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 -- 3

Enter the name of the file-- B
File B is 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 -- A
File A is deleted

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

The Files are --     B

1. Create File    2. Delete File    3. Search File
4. Display Files    5. Exit
Enter your choice -- 5
ubuntu@ubuntu-HP-Pro-Tower-400-G9-PCI-Desktop-PC:
```

b) Two level directory

Description: In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched.

SOURCE CODE :

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

struct
{
   char dname[10], fname[10][10];
   int fcnt;
} dir[10];

int main()
{
   int i, ch, dcnt = 0, k;
   char f[30], d[30];
   while (1)
   {
      printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
      printf("\n4. Search File\t\t5. Display\t6. Exit");
      printf("\nEnter your choice -- ");
      scanf("%d", &ch);

      switch (ch)
      {
      case 1:
         printf("\nEnter name of directory -- ");
         scanf("%s", dir[dcnt].dname);
         dir[dcnt].fcnt = 0;
         dcnt++;
         printf("Directory created");
         break;

      case 2:
         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", 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 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 Directories");
            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("%s\t", dir[i].fname[k]);
                }
            }
            break;

        case 6:
            exit(0);
        default:
            printf("Invalid choice");
        }
    }
}
```

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 -- A
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 -- A
Enter name of the file -- AA
File created

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

Directory      Files
A              AA

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

Enter name of directory -- B
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 -- B
Enter name of the file -- BB
File created

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

Directory      Files
A              AA
B              BB

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 -- B
Enter name of the file -- BB
File BB is deleted



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

Directory       Files
A               AA
B

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 -- B
Enter name of the file -- BB
File BB not found

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

EXPT. NO:9

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

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each block a pointer is kept to next successive block. Hence, there is no external fragmentation.

ALGORTHIM:

```
Step 1: Start the program.
Step 2: Get the number of files.
Step 3: Get the memory requirement of each file.
Step 4: Allocate the required locations by selecting a location randomly
        q= random(100);
        a) Check whether the selected location is free.
        b) If the location is free allocate and set flag=1 to the allocated locations.
        While allocating next location address, attach it to previous location.
for(i=0;i<n;i++)
{
        for(j=0;j<s[i];j++)
                {
                        q=random(100); if(b[q].flag==0)
                        b[q].flag=1;
                        b[q].fno=j;
                        r[i][j]=q;
                        if(j>0)
                        {
                }
}
p=r[i][j-1]; b[p].next=q;}

Step 5: Print the results file no, length, Blocks allocated.

Step 6: Stop the program
```

SOURCE CODE :

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

int main()
{
   int f[50], p, i, j, k, a, st, len, n, c;

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

   printf("Enter how many blocks are already allocated: ");
   scanf("%d", &p);
```

```
    printf("Enter the block numbers that are already allocated: ");

    for (i = 0; i < p; i++)
    {
        scanf("%d", &a);
        if (a >= 0 && a < 50)
            f[a] = 1;
        else
            printf("Invalid block number %d (ignored)\n", a);
    }

X:
    printf("\nEnter the starting index block and length: ");
    scanf("%d%d", &st, &len);

    // Input validation
    if (st < 0 || st >= 50)
    {
        printf("Invalid starting block!\n");
        goto X;
    }
    if (st + len > 50)
    {
        printf("Error: Block range exceeds disk size (50 blocks)\n");
        goto X;
    }

    k = len;

    for (j = st; j < st + k; j++)
    {
        if (f[j] == 0)
        {
            f[j] = 1;
            printf("\n%d -> allocated", j);
        }
        else
        {
            printf("\n%d -> block already allocated", j);
            // Try next block, but ensure we don’t go out of range
            if (st + k < 50)
                k++;
        }
    }

    printf("\nDo you want to enter one more file? (yes-1 / no-0): ");
    scanf("%d", &c);

 if (c == 1)
        goto X;
    else
        exit(0);
}
```

OUTPUT:

```
        Enter how many blocks are already allocated: 3
    Enter the block numbers that are already allocated: 2 5 8


    Enter the starting index block and length: 3 4


    3 -> allocated
    4 -> allocated
    5 -> block already allocated
    6 -> allocated
    7 -> allocated
    Do you want to enter one more file? (yes-1 / no-0): 0
```

EXPT. NO: 10

Develop a C program to simulate SCAN disk scheduling algorithm.

DESCRIPTION

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling.

SCAN algorithm: The disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

```
#include <stdio.h>

int main()
{
   int t[20], d[20], h, i, j, n, temp, atr[20], tot, p, sum = 0;
   int pos = 0;

   printf("Enter the number of tracks to be traversed: ");
   scanf("%d", &n);

   printf("Enter the position of head: ");
   scanf("%d", &h);

   t[0] = 0; // Start track (beginning of disk)
   printf("Enter the track numbers: ");
   for (i = 1; i <= n; i++)
      scanf("%d", &t[i]);

   t[n + 1] = h; // Add head position at the end for sorting

   // Sort the tracks
   for (i = 0; i < n + 2; i++)
   {
      for (j = 0; j < (n + 1) - i; j++)
      {
         if (t[j] > t[j + 1])
         {
            temp = t[j];
            t[j] = t[j + 1];
            t[j + 1] = temp;
         }
      }
   }
```

```
  // Find index of head in sorted array
    for (i = 0; i < n + 2; i++)
    {
        if (t[i] == h)
        {
            pos = i;
            break;
        }
    }

    // Arrange the track sequence for SCAN (towards 0 first, then end)
    int p1 = 0;
    for (i = pos; i >= 0; i--)
        atr[p1++] = t[i];
    for (i = pos + 1; i < n + 2; i++)
        atr[p1++] = t[i];

    // Calculate total head movement
    for (i = 0; i < p1 - 1; i++)
    {
        if (atr[i] > atr[i + 1])
            d[i] = atr[i] - atr[i + 1];
        else
            d[i] = atr[i + 1] - atr[i];
        sum += d[i];
    }

    printf("\nOrder of tracks serviced:\n");
    for (i = 0; i < p1; i++)
        printf("%d ", atr[i]);

    printf("\nTotal head movement: %d", sum);
    printf("\nAverage head movement: %.2f\n", (float)sum / n);

    return 0;
}
```

OUTPUT

```
Enter the number of tracks to be traversed: 5

Enter the position of head: 50

Enter the track numbers: 10 20 30 70 90


Order of tracks serviced:

50 30 20 10 0 70 90

Total head movement: 140

Average head movement: 28.00
```