JAIN COLLEGE OF ENGINEERING

(Approved by AICTE, New Delhi & Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Academic Year: 2023-24

LABORATORY MANUAL

SEMESTER : III

SUBJECT : Operating Systems

SUBCODE : BCS 303

NAME: _____

USN: _____

SECTION: _____

BATCH: _____

PROGRAM OUTCOMES

Engineering Graduates will able to:

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, review research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

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.

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.

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

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.

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.

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
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.

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.

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.

INSTITUTE VISION AND MISSION

VISION

To be a university as **a resource of solutions** to **diverse challenges of society** by nurturing innovation, research & entrepreneurship through value based education.

MISSION

M1: To provide work culture that facilitates effective teaching-learning process and lifelong learning skills

M2: To promote innovation, collaboration and leadership through best practices

M3: To foster industry-institute interaction resulting in entrepreneurship skills and employment opportunities

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DEPARTMENT VISION AND MISSION

VISION

To create a centre of excellence in teaching, learning and professional environment in the discipline of Computer Science and Engineering addressing emerging and existing challenges striving to meet the requirement of both industry and society.

MISSION

M1: To provide value-based quality technical education and training with continuous improvement in the discipline of Computer Science and Engineering.

M2: To develop professionals to meet the requirements of industry and society.

M3: To promote innovation, dedication, and hard work with ethically and professionally responsible behaviour.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will be able to solve real-world problems and engineering problems by applying fundamentals and recent technologies in the discipline of Computer Science and Engineering.

PEO2: Graduates will be able to perform well in design and development of software product formulating necessary requirements through effective communication.

PEO3: Graduates will be able to take up higher studies and innovative projects.

PEO4: Graduates will behave in an ethically and professionally responsible manner.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Engineering Graduates will be able to:

PSO1: Identify the Problem, analyze, design, develop, test and implement solution using appropriate tools and technologies.

PSO2: Apply the knowledge to carry out innovative projects and transform ideas into working modules following the professional ethics and engineering principles.

COURSE LEARNING OBJECTIVES

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

COURSE OUTCOMES

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

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.

Course outcomes (Course Skill Set):

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

CO 1. Explain the structure and functionality of operating system

CO 2. Apply appropriate CPU scheduling algorithms for the given problem.

CO 3. Analyse the various techniques for process synchronization and deadlock handling.CO 4. Apply the various techniques for memory management

CO 5. Explain file and secondary storage management strategies.CO 6. Describe the need for information protection mechanisms

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed tohave satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests,each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods

mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second testafter covering 85-90% of the syllabus.

- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theorycomponent of IPCC (that is for **25 marks)**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluationof the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC

Suggested Learning Resources:

Textbooks

1. Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Principles 8th edition,Wiley-India, 2015

Reference Books

1. Ann McHoes Ida M Fylnn, Understanding Operating System, Cengage Learning, 6th Edition
2. D.M Dhamdhere, Operating Systems: A Concept Based Approach 3rd Ed, McGraw- Hill, 2013.
3. P.C.P. Bhatt, An Introduction to Operating Systems: Concepts and Practice 4th Edition, PHI(EEE),2014.
4. William Stallings Operating Systems: Internals and Design Principles, 6th Edition, Pearson.

Table of Content

Program-1

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

```
#include<stdio.h> // printf()
#include<stdlib.h> // exit()
#include<sys/types.h> // pid_t
#include<sys/wait.h> // wait()
#include<unistd.h> // fork

int main(int argc, char **argv)
{
pid_t pid;
pid = fork();
if(pid==0)
{
printf("It is the child process and pid is %d\n",getpid());
int i=0;
for(i=0;i<8;i++)
{
printf("%d\n",i);
}
exit(0);
}
else if(pid > 0)
{
printf("It is the parent process and pid is %d\n",getpid());
int status;
wait(&status);
printf("Child is reaped\n");
}
else
{
printf("Error in forking..\n");
exit(EXIT_FAILURE);
}
return 0;
}
```

Output:

```
It is the parent process and pid is 4498
It is the child process and pid is 4499
0
1
2
3
4
5
6
7
Child is reaped
```

Program-2

Simulate the following CPU scheduling algorithms to find turnaround time and waiting time

a) FCFS

```
#include<stdio.h>
int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20): ");
   scanf("%d",&n);
   printf("\nEnter Process Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]: ",i+1);
      scanf("%d",&bt[i]);
   }
   wt[0]=0;
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
      wt[i]+=bt[j];
   }

   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
   //calculating turnaround time
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
      printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
   }
   avwt/=i;
   avtat/=i;
   printf("\n\nAverage Waiting Time:%d",avwt);
   printf("\nAverage Turnaround Time:%d",avtat);
   return 0;
}
```

Output:

```
Enter total number of processes(maximum 20): 4
Enter Process Burst Time:
P[1]: 5
P[2]: 8
P[3]: 2
P[4]: 7
Process     Burst Time  Waiting Time    Turnaround Time
P[1]        5       0       5
P[2]        8       5       13
P[3]        2       13      15
P[4]        7       15      22

Average Waiting Time:8
Average Turnaround Time:13
```

b) SJF

```
#include<stdio.h>
int main()
{
   int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
   float avg_wt,avg_tat;
   printf("Enter number of process: ");
   scanf("%d",&n);
   printf("\nEnter Burst Time:\n");
   for(i=0;i<n;i++)
   {
      printf("p%d: ",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }
   for(i=0;i<n;i++)
   { pos=i;
      for(j=i+1;j<n;j++)
      {       if(bt[j]<bt[pos])
            pos=j;
      }
      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;
      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }
   wt[0]=0;
   for(i=1;i<n;i++)
   { wt[i]=0;
      for(j=0;j<i;j++)
      wt[i]+=bt[j];
      total+=wt[i];
   }
   avg_wt=(float)total/n;
   total=0;
   printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++)
   { tat[i]=bt[i]+wt[i];
      total+=tat[i];
```

```
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
      }
     avg_tat=(float)total/n;
      printf("\n\nAverage Waiting Time=%f",avg_wt);
      printf("\nAverage Turnaround Time=%f\n",avg_tat);
      return 0;
  }
```

Output:

```
Enter number of process: 4
Enter Burst Time:
p1: 5
p2: 4
p3: 8
p4: 2
Process     Burst Time      Waiting Time    Turnaround Time
p4        2         0           2
p2        4         2           6
p1        5         6           11
p3        8         11          19

Average Waiting Time=4.750000
Average Turnaround Time=9.500000
```

c) Round Robin

```
#include<stdio.h>
int main()
{
  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for(time=0,count=0;remain!=0;)
  {
    if(rt[count]<=time_quantum && rt[count]>0)
    {
      time+=rt[count];
      rt[count]=0;
```

```
  flag=1;
 }
 else if(rt[count]>0)

 {
  rt[count]-=time_quantum;
  time+=time_quantum;
 }
 if(rt[count]==0 && flag==1)
 {
  remain--;
  printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
  wait_time+=time-at[count]-bt[count];
  turnaround_time+=time-at[count];
  flag=0;
 }
 if(count==n-1)
  count=0;
 else if(at[count+1]<=time)
  count++;
 else
  count=0;
}
printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

return 0;
 }
```

Output:

```
Enter Total Process:     5
Enter Arrival Time and Burst Time for Process Process Number 1 :4
5
Enter Arrival Time and Burst Time for Process Process Number 2 :2
8
Enter Arrival Time and Burst Time for Process Process Number 3 :6
4
Enter Arrival Time and Burst Time for Process Process Number 4 :1
2
Enter Arrival Time and Burst Time for Process Process Number 5 :0
7
Enter Time Quantum: 2
Process |Turnaround Time|Waiting Time

P[4]    |   11  |   9
P[1]    |   11  |   6
P[3]    |   13  |   9
P[2]    |   21  |   13
P[5]    |   26  |   19

Average Waiting Time= 11.200000
Avg Turnaround Time = 16.400000
```

d) Priority

```
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;
   }

    for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }

      temp=pr[i];
      pr[i]=pr[pos];
      pr[pos]=temp;

      temp=bt[i];
      bt[i]=bt[pos];
      bt[pos]=temp;

      temp=p[i];
      p[i]=p[pos];
      p[pos]=temp;
   }

   wt[0]=0;
   for(i=1;i<n;i++)
   {
      wt[i]=0;

      for(j=0;j<i;j++)
         wt[i]+=bt[j];

      total+=wt[i];
   }
```

```
avg_wt=total/n;
total=0;

printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{

   tat[i]=bt[i]+wt[i];
   total+=tat[i];
   printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=total/n;
printf("\n\nAverage Waiting Time=%d",avg_wt);
printf("\nAverage Turnaround Time=%d\n",avg_tat);

return 0;
}
```

Output:

```
Enter Total Number of Process:5
Enter Burst Time and Priority

P[1]
Burst Time:4
Priority:2
P[2]
Burst Time:8
Priority:1
P[3]
Burst Time:9
Priority:4
P[4]
Burst Time:7
Priority:3
P[5]
Burst Time:7
Priority:5
Process      Burst Time       Waiting Time     Turnaround Time
P[2]           8          0           8
P[1]           4          8           12
P[4]           7          12          19
P[3]           9          19          28
P[5]           7          28          35

Average Waiting Time=13
Average Turnaround Time=20
```

Program-3

Develop a C program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
void main()
{
int buffer[10], bufsize, in, out, produce, consume, choice=0;
in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n 1. Produce \t 2. Consume \t3. Exit");
printf("\n Enter your choice: ");
scanf("%d", &choice);
switch(choice) {
case 1: if((in+1)%bufsize==out)
printf("\n Buffer is Full");
else
{
printf("\nEnter the value: ");
scanf("%d", &produce);
buffer[in] = produce;
in = (in+1)%bufsize;
}
break;
case 2: if(in == out)
printf("\nBuffer is Empty");
else
{
consume = buffer[out];
printf("\nThe consumed value is %d", consume);
out = (out+1)%bufsize;
}
break;
} } }
```

Output:

```
1. Produce   2. Consume     3. Exit
 Enter your choice: 2
 Buffer is Empty
 1. Produce      2. Consume     3. Exit
 Enter your choice: 1
 Enter the value: 56
 1. Produce      2. Consume     3. Exit
 Enter your choice: 2
 The consumed value is 56
 1. Produce      2. Consume     3. Exit
 Enter your choice:
```

Program-4

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

/*Writer Process*/

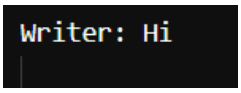
```
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include  <unistd.h>
int main()
{
 int fd;
 char buf[1024];
 /* create the FIFO (named pipe) */
 char * myfifo = "/tmp/myfifo";
 mkfifo(myfifo, 0666);
 printf("Run Reader process to read the FIFO File\n");
 fd = open(myfifo,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

Instructions for Execution:

1. Run Reader Process
2. Then Run Writer Process



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:

```
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>
int main()
{
 int fragments[10], blocks[10], files[10];
 int m, n, number_of_blocks, number_of_files, temp, top = 0;
static int block_arr[10], file_arr[10];
 printf("\nEnter the Total Number of Blocks:\t");
 scanf("%d",&number_of_blocks);
 printf("Enter the Total Number of Files:\t");
 scanf("%d",&number_of_files);
 printf("\nEnter the Size of the Blocks:\n");
 for(m = 0; m < number_of_blocks; m++)
{
printf("Block No.[%d]:\t", m + 1);
scanf("%d", &blocks[m]);
  }
printf("Enter the Size of the Files:\n");
 for(m = 0; m < number_of_files; m++)
{
printf("File No.[%d]:\t", m + 1);
scanf("%d", &files[m]);
 }
  for(m = 0; m < number_of_files; m++)
{
for(n = 0; n < number_of_blocks; n++)
{
    if(block_arr[n] != 1)
    {
        temp = blocks[n] - files[m];
        if(temp >= 0)
        {
            if(top < temp)
            {
                file_arr[m] = n;
                top = temp;
            }
        }
    }
    fragments[m] = top;
    block_arr[file_arr[m]] = 1;
    top = 0;
}
}
printf("\nFile Number\tFile Size\tBlock Number\tBlock Size\tFragment");
for(m = 0; m < number_of_files; m++)
{
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", m, files[m], file_arr[m], blocks[file_arr[m]], fragments[m]);
 }
 printf("\n");
```

```
return 0;
}
```

Output:

```
Enter the Total Number of Blocks:   5
Enter the Total Number of Files:    4
Enter the Size of the Blocks:
Block No.[1]:   5
Block No.[2]:   4
Block No.[3]:   3
Block No.[4]:   6
Block No.[5]:   7
Enter the Size of the Files:
File No.[1]:    1
File No.[2]:    3
File No.[3]:    5
File No.[4]:    3
File Number File Size   Block Number    Block Size  Fragment
0       1       4       7       6
1       3       0       5       0
2       5       0       5       0
3       3       0       5       0
```

b) Best Fit

```
#include<stdio.h>

void main()
{
   int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
   static int barray[20],parray[20];

   printf("\n\t\t\tMemory 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 no.%d:",i);
      scanf("%d",&b[i]);
   }

   printf("\nEnter the size of the processes :-\n");
   for(i=1;i<=np;i++)
   {
      printf("Process no.%d:",i);
      scanf("%d",&p[i]);
   }

   for(i=1;i<=np;i++)
   {
```

```
    for(j=1;j<=nb;j++)
    {
       if(barray[j]!=1)
       {
          temp=b[j]-p[i];
          if(temp>=0)
             if(lowest>temp)
             {
                parray[i]=j;
                lowest=temp;
             }
       }
    }

    fragment[i]=lowest;
    barray[parray[i]]=1;
    lowest=10000;
}

printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
for(i=1;i<=np && parray[i]!=0;i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

```
Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4
Enter the size of the blocks:-
Block no.1:10
Block no.2:15
Block no.3:5
Block no.4:9
Block no.5:3
Enter the size of the processes :-
Process no.1:1
Process no.2:4
Process no.3:7
Process no.4:12
Process_no  Process_size    Block_no    Block_size  Fragment
1       1       5       3       2
2       4       3       5       1
3       7       4       9       2
4       12      2       15      3
```

c) First Fit

```
#include<stdio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
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) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
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:

```
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 1:2
File 2:5
File_no:    File_size : Block_no:   Block_size: Fragement
1       2       3       7       5
2       5       0       0       0
```

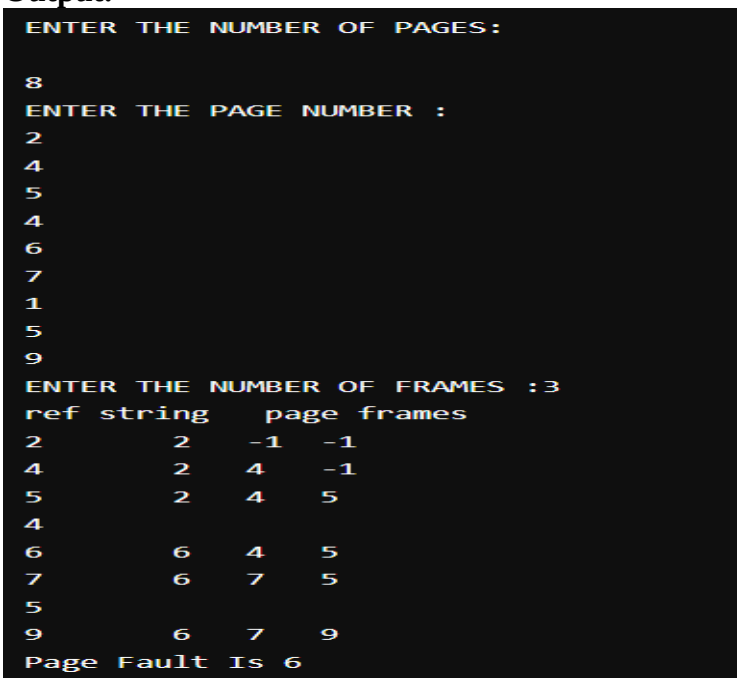
Program-7

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:



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..8
 Enter reference string..
 5
 6
 4
 8
 2
 5
 7
 6
 3
 LRU PAGE REPLACEMENT ALGORITHM
     The given reference string:
……..   5   6   4   8   2   7   6   3

     Reference NO 5->      5  -1  -1
     Reference NO 6->      5   6  -1
     Reference NO 4->      5   6   4
     Reference NO 8->      8   6   4
     Reference NO 2->      8   2   4
     Reference NO 7->      8   2   7
     Reference NO 6->      6   2   7
     Reference NO 3->      6   3   7

     No.of page faults...8
```

Program-8

Simulate following File Organization Techniques

a) Single level directory

```
#include<stdlib.h>
#include<string.h>
#include<stdio.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 \n 4. 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:

```
Enter name of directory -- AI
1. Create File  2. Delete File  3. Search File
 4. Display Files   5. Exit
Enter your choice -- 1
Enter the name of the file -- ab
1. Create File  2. Delete File  3. Search File
 4. Display Files   5. Exit
Enter your choice -- 1
Enter the name of the file -- ac
1. Create File  2. Delete File  3. Search File
 4. Display Files   5. Exit
Enter your choice -- 4
The Files are --    ab  ac

1. Create File  2. Delete File  3. Search File
 4. Display Files   5. Exit
Enter your choice -- 2
Enter the name of the file -- ab
File ab is deleted

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

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

b) Two-Level Directory

```
#include<string.h>
#include<stdlib.h>
#include<stdio.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\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter 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]);
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 -- abc
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 -- xyz
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 -- xyz
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
abc
xyz

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

Program-9

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

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int f[50], p,i, st, len, j, c, k, a;
clrscr();
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);
getch();
}
```

Output:

```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 3 5
Enter index starting block and length: 2 2
2--------->1
3 Block is already allocated
4--------->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 :
4
ENTER THE NO OF REQUEST SEQUENCE :
2
ENTER THE REQUEST SEQUENCE :
1
2
ENTER THE CURRENT HEAD :
1
ENTER THE PRE REQUEST :
2
SEEK SEQUENCE = 1 0 1  2
TOTAL DISTANCE :3
```