SL.No	Experiments
1	Develop a C program to implement the Process system calls (fork (), exec(), wait(), create process, terminate process). Click here
2	Simulate the following CPU scheduling algorithms to find turnaround time and waiting time a) FCFS b) SJF c) Round Robin d) Priority Scheduling.
3	Develop a C program to simulate producer-consumer problem using semaphores. Click here
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. Click here
5	Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance. Click here
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. Click here.
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. Click here
10	Develop a C program to simulate SCAN disk scheduling algorithm. Click here

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

SOURCE CODE:

```
#include <stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>

int main()
 {
   pid_t pid=0;
   int status;
   pid=fork();
   if(pid==0)
   {
      printf("I am the child\n");
      sleep(10);
      printf("I am the child , 10 seconds later\n");
   }
   if(pid>0)
   {
      printf("I am the parent, the child is %d\n",pid);
      pid=wait(&status);
      printf("End of the process %d: ",pid);
      if(WIFEXITED(status))
      {
         printf("The process ended with exit(%d)\n",WEXITSTATUS(status));
      }
```

```
        if(WIFSIGNALED(status))
        {
            printf("The process ended with kill-%d\n",WTERMSIG(status));
        }
    }
    if(pid<0)
    {
        perror("In fork(): ");
    }
    exit(0);
}
```

SAMPLE OUTPUT:

```
I am the parent, the child is 3319
I am the child
I am the child , 10 seconds later
End of the process 3319: The process ended with exit(0)
```

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

A)FCFS (First Come First Serve)

<u>SOURCE CODE:</u>

```
#include<stdio.h>
void main()
{
  int bt[15];
  int n;
  printf("Enter the no. of processes: ");
  scanf("%d",&n);
  printf("Enter the burst time of processes: ");
  for(int i=0;i<n;i++)
     scanf("%d",&bt[i]);
  int wt[n];
  wt[0]=0;
  for(int i=1;i<n;i++)
     wt[i]=bt[i-1]+wt[i-1];
  printf("ProcessID   BurstTime   WaitingTime    TurnAroundTime \n");
  float twt=0.0 , ttat=0.0;
  for(int i=0;i<n;i++)
  {
     printf("p%d\t\t%d\t\t%d\t\t%d\n", i, bt[i], wt[i], bt[i]+wt[i]);
     twt+=wt[i];
     ttat+=bt[i]+wt[i];
  }

  printf("\nAvg. Waiting Time: %f\n",twt/n);
  printf("Avg. Turn Around Time: %f\n",ttat/n);
}
```

SAMPLE OUTPUT:

```
Enter the no. of processes: 5

Enter the burst time of processes: 3 2 6 7 1

ProcessID   BurstTime   WaitingTime    TurnAroundTime

p0            3            0             3

p1            2            3             5

p2            6            5             11

p3            7            11            18

p4            1            18            19


Avg. Waiting Time: 7.400000

Avg. Turn Around Time: 11.200000
```

B)SJF (Shortest Job First)

SOURCE CODE:

```
#include<stdio.h>
void swap(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
void main()
{
   int bt[20],p[20],wt[20],i,j,n,twt=0,ttat=0,pos;

   printf("Enter the no. of processes: ");
   scanf("%d",&n);
   printf("Enter the burst time of processes: \n");
   for(int i=0;i<n;i++)
   {
      printf("P%d: ",i+1);
      scanf("%d",&bt[i]);
      p[i]=i+1;
   }

   //sorting in ascending order based on Burst time of the processes
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
          if(bt[j]<bt[pos])
              pos=j;
        }
        swap(&bt[i],&bt[pos]);
        swap(&p[i],&p[pos]);
    }
//calculating waiting time
    wt[0]=0;
    for(i=1;i<n;i++)
        wt[i]=bt[i-1]+wt[i-1];
//display
    printf("ProcessID   BurstTime   WaitingTime    TurnAroundTime \n");
    for(int i=0;i<n;i++)
    {
        printf("p%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], bt[i]+wt[i]);
        twt+=wt[i];
        ttat+=bt[i]+wt[i];
    }

    printf("\nAvg. Waiting Time: %f\n",(float) twt/n);
    printf("Avg. Turn Around Time: %f\n",(float)  ttat/n);
}
```

SAMPLE OUTPUT:

```
Enter the no. of processes: 4
Enter the burst time of processes:
P1: 4
P2: 2
P3: 3
P4: 7
ProcessID   BurstTime   WaitingTime    TurnAroundTime
```

```
p2          2           0            2

p3          3           2           5

p1          4           5           9

p4          7           9           16


Avg. Waiting Time: 4.000000

Avg. Turn Around Time: 8.000000
```

C) Round Robin

SOURCE CODE:

```
#include<stdio.h>
void main()
{
    int n;
    printf("Enter total no. of processes: ");
    scanf("%d",&n);
    int twt=0,ttat=0,arr_time[20],burst_time[20],temp_burst_time[20];
    for(int i=0;i<n;i++)
    {
        printf("Enter details of the process %d: \n",i+1);
        printf("Arrival Time: ");
        scanf("%d",&arr_time[i]);
        printf("Burst Time: ");
        scanf("%d",&burst_time[i]);
        temp_burst_time[i]=burst_time[i];
    }
    int timeslot;
    printf("Enter time slot: ");
    scanf("%d",&timeslot);
    printf("ProcessID   BurstTime   TurnAroundTime   WaitingTime\n");
    int total=0,counter=0,i,x=n;
    for(i=0;x!=0;)
    {
        if(temp_burst_time[i]<=timeslot && temp_burst_time[i]>0)
        {
            total=total+temp_burst_time[i];
            temp_burst_time[i]=0;
            counter=1;
        }
        else if(temp_burst_time[i]>0)
        {
          temp_burst_time[i]=temp_burst_time[i]-timeslot;
          total=total+timeslot;
        }
        if(temp_burst_time[i]==0 && counter==1)
        {
          x--;
          printf("\nP%d \t\t%d\t\t%d\t\t%d",i+1,burst_time[i],total-arr_time[i],total-
arr_time[i]-burst_time[i]);
          twt=twt+total-arr_time[i]-burst_time[i];
          ttat=ttat+total-arr_time[i];
          counter=0;
        }
        if(i==n-1)
          i=0;
        else if(arr_time[i+1]<=total)
          i++;
        else
          i=0;
    }
    printf("\nAvg. Waiting Time: %f\n",(float) twt/n);
    printf("Avg. Turn Around Time: %f\n",(float)  ttat/n);
}
```

SAMPLE OUTPUT

```
Enter total no. of processes: 3
Enter details of the process 1:
Arrival Time: 0
Burst Time: 5
```

```
Enter details of the process 2:

Arrival Time: 1

Burst Time: 6

Enter details of the process 3:

Arrival Time: 2

Burst Time: 7

Enter time slot: 3

ProcessID   BurstTime   TurnAroundTime   WaitingTime


P1          5           11               6

P2          6           13               7

P3          7           16               9

Avg. Waiting Time: 7.333333

Avg. Turn Around Time: 13.333333
```

D) PRIORITY SCHEDULING

SOURCE CODE:

```
#include<stdio.h>
void swap(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
void main()
{
   int bt[20],priority[20],process[20],wt[20],n,pos,twt=0,ttat=0;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   printf("Enter the Burst Time and Priority value for processes: \n");
   for(int i=0;i<n;i++)
   {
      printf("Process P%d: ",i+1);
      scanf("%d %d",&bt[i],&priority[i]);
      process[i]=i+1;
   }

//sorting in descending order based on priority value
//higher the priority value higher the priority
   for(int i=0;i<n;i++)
   {
      pos=i;
      for(int j=i+1;j<n;j++)
      {
         if(priority[j]>priority[pos])
            pos=j;
      }
      swap(&priority[i],&priority[pos]);
      swap(&process[i],&process[pos]);
      swap(&bt[i],&bt[pos]);
   }
//calculating waiting time
   wt[0]=0;
   for(int i=1;i<n;i++)
      wt[i]=bt[i-1]+wt[i-1];
//display
   printf("ProcessID   BurstTime   WaitingTime    TurnAroundTime \n");
   for(int i=0;i<n;i++)
   {
      printf("p%d\t\t%d\t\t%d\t\t%d\n", process[i], bt[i], wt[i], bt[i]+wt[i]);
      twt+=wt[i];
      ttat+=bt[i]+wt[i];
   }

   printf("\nAvg. Waiting Time: %f\n",(float) twt/n);
   printf("Avg. Turn Around Time: %f\n",(float)  ttat/n);
}
```

SAMPLE OUTPUT:

```
Enter the number of processes: 3

Enter the Burst Time and Priority value for processes:

Process P1: 5 1

Process P2: 3 3

Process P3: 4 2
```

```
ProcessID   BurstTime   WaitingTime    TurnAroundTime

p2            3              0            3

p3            4              3            7

p1            5              7            12


Avg. Waiting Time: 3.333333

Avg. Turn Around Time: 7.333333
```

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

SOURCE CODE:

```
#include<stdio.h>
#define bufsize 3
void main()
{
   int buffer[bufsize],in=-1,out=-1,produce,consume,choice=0;
   while(choice!=3)
   {
      printf("\n1.Produce\t2.consume\t3.Exit.\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
      switch (choice)
      {
      case 1:
         if((in+1)%bufsize==out)
            printf("Buffer is full\n");
         else
         {
            printf("Enter the value: ");
            scanf("%d",&produce);
            in=(in+1) % bufsize;
            buffer[in]=produce;
            if(out==-1)
               out=0;
         }
         break;
      case 2:
            if(in==-1)
               printf("Buffer is empty\n");
            else
            {
               consume=buffer[out];
               printf("The consumed value is %d\n\n",consume);
```

```
                if(in==out)
                   in=out=-1;
                else
                   out=(out+1)%bufsize;
             }
          break;
       default:
          break;
       }
    }
}
```

SAMPLE OUTPUT:

```
1.Produce     2.consume     3.Exit.

Enter your choice: 1

Enter the value: 12


1.Produce     2.consume     3.Exit.

Enter your choice: 1

Enter the value: 23


1.Produce     2.consume     3.Exit.

Enter your choice: 1

Enter the value: 25


1.Produce     2.consume     3.Exit.

Enter your choice: 1

Buffer is full


1.Produce     2.consume     3.Exit.

Enter your choice: 2

The consumed value is 12
```

```
1.Produce      2.consume      3.Exit.

Enter your choice: 2

The consumed value is 23


1.Produce      2.consume      3.Exit.

Enter your choice: 2

The consumed value is 25


1.Produce      2.consume      3.Exit.

Enter your choice: 2

Buffer is empty


1.Produce      2.consume      3.Exit.

Enter your choice: 3
```

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.

SOURCE CODE:

WRITERS PROCESS:

```
#include<stdio.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<unistd.h>

int main()

{

   int fd;

   char buf[1024];

   char *myfifo="'/myfifo";

   mkfifo(myfifo,0666);

   printf("Run readers process to read the FIFO file\n");

   fd=open(myfifo,O_WRONLY);

   write(fd,"Hii",sizeof("Hi"));

   close(fd);

   unlink(myfifo);

   return 0;

}
```

WRITERS PROCESS:

```
#include<stdio.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>
```

```
#include<unistd.h>
#define MAX_BUF 1024
int main() {
   int fd;
   char *myfifo="./myfifo";
   char buf[MAX_BUF];
   fd=open(myfifo,O_RDONLY);
   read(fd,buf,MAX_BUF);
   printf("Write: %s\n",buf);
   return 0;
}
```

OUTPUT:

5) Develop a C program to simulate Bankers Algorithm for DeadLock Avoidance.

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int max[100][100], alloc[100][100], need[100][100],avail[100];
int n,r;
void input();
void show();
void cal();
int main()

{
   int i,j;
   printf("**********Banker's Algorithm**********");
   input();
   show();
   cal();
   return 0;
}

void input()
{
   int i,j;
   printf("\nEnter the no. of processos:\t");
   scanf("%d",&n);
   printf("Enter the number of resource instances:\t");
   scanf("%d",&r);
   printf("Enter the max matrix:\n");
   for(i=0;i<n;i++)
```

```
    {
        for(j=0;j<r;j++)
            scanf("%d",&max[i][j]);
    }
    printf("Enter the allocation matrix:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
            scanf("%d",&alloc[i][j]);
    }
    printf("Enter the available resource:\n");

    for(j=0;j<r;j++)
        scanf("%d",&avail[j]);
}

void show()
{
    int i,j;
    printf("Process Allocation Max Available\t");
    for(i=0;i<n;i++)
    {
        printf("\nP%d\t",i+1);
        for(j=0;j<r;j++)
        {
            printf("%d ",alloc[i][j]);
        }
        printf("\t ");
        for(j=0;j<r;j++)
        {
            printf("%d ",max[i][j]);
        }
        printf("\t ");
        if(i==0)
        {
            for(j=0;j<r;j++)
                printf("%d ",avail[j]);
        }
    }
}

void cal()
{

    int finish[100],temp, flag=1,k,c1=0;
    int i,j;
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<r;j++)
            need[i][j]=max[i][j]-alloc[i][j];
    }
    printf("\n");

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

```
        int c=0;
        for(j=0;j<r;j++)
        {
            if((finish[i]==0)&&(need[i][j]<=avail[j]))
            {
                c++;
                if(c==r)
                {
                    for(k=0;k<r;k++)
                    {
                        avail[k]+=alloc[i][k];
                        finish[i]=1;
                        flag=1;
                    }
                    printf("p%d->",i+1);
                        i=n;
                }
            }
        }
    }
}
for(i=0;i<n;i++)
{
    if(finish[i])
        c1++;
    else
    printf("P%d->",i+1);
}
if(c1==n)
    printf("The system is in safe state");
else
```

```
        printf("System is in dead lock");
}
```

SAMPLE OUTPUT 1:

```
**********Banker's Algorithm**********
Enter the no. of processos:     5
Enter the number of resource instances: 3
Enter the max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter the available resource:
3 3 2
Process Allocation Max Available
P1      0 1 0    7 5 3   3 3 2
P2      2 0 0    3 2 2
P3      3 0 2    9 0 2
P4      2 1 1    2 2 2
P5      0 0 2    4 3 3
p2->p4->p1->p3->p5->The system is in safe state
```

SAMPLE OUTPUT 2:

```
**********Banker's Algorithm**********
Enter the no. of processos:     2
Enter the number of resource instances: 2
Enter the max matrix:
5 3
6 8
Enter the allocation matrix:
1 0
0 0
Enter the available resource:
2 4
Process Allocation Max Available
P1      1 0      5 3     2 4
P2      0 0      6 8
P1->P2->System is in dead lock
```

6) Develop a C program to simulate the following contiguous memory allocation Techniques: a) Worst fit b) Best fit c) First fit.

A) WORST FIT:

SOURCE CODE:

```
#include<stdio.h>
#define max 25
void main()
{
   int frag[max],block[max],file[max],i,j,nb,nf,temp,highest=0;
   static int block_filled[max],file_filled_in[max];
//if array is declared as static int ,all values will be initialized to zero

   printf("\n***** MEMORY MANAGEMENT SCHEME - WORST FIT *****\n");
   printf("Enter the no. of blocks: ");
   scanf("%d",&nb);
   printf("Enter the no. of files: ");
   scanf("%d",&nf);
   printf("Enter the size of blocks:  ");
   for(i=1;i<=nb;i++)
   {
      scanf("%d",&block[i]);
   }
   printf("Enter the size of files:  ");
   for(i=1;i<=nf;i++)
   {
      scanf("%d",&file[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(block_filled[j]!=1)
         {
            temp=block[j]-file[i];
            if(temp>=0 && highest<temp)
            {
               file_filled_in[i]=j;
               highest=temp;
            }
         }
      }
      frag[i]=highest;
      block_filled[file_filled_in[i]]=1;
      highest=0;
   }
   printf("File no: \t File size: \t Block no: \tBlock size: \tFragment:\n");
   for(i=1;i<=nf && file_filled_in[i]!=0;i++)
      printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,file[i],file_filled_in[i],block[file_filled_in[i]],fra
g[i]);
}
```

SAMPLE OUTPUT:

```
***** MEMORY MANAGEMENT SCHEME - WORST FIT *****
Enter the no. of blocks: 3
Enter the no. of files: 3
Enter the size of blocks:  10 5 7
Enter the size of files:  4 6 9
File no:      File size:    Block no:     Block size:    Fragment:
1              4             1             10             6
2              6             3             7              1
```

B) BESTFIT

SOURCE CODE:

```
#include<stdio.h>
#define max 25
void main()
{
   int frag[max],block[max],file[max],i,j,nb,nf,temp,lowest=10000;
   static int block_filled[max],file_filled_in[max];
//if array is declared as static int ,all values will be initialized to zero

   printf("\n***** MEMORY MANAGEMENT SCHEME - BEST FIT *****\n");
   printf("Enter the no. of blocks: ");
   scanf("%d",&nb);
   printf("Enter the no. of files: ");
   scanf("%d",&nf);
   printf("Enter the size of blocks: ");
   for(i=1;i<=nb;i++)
   {
      scanf("%d",&block[i]);
   }
   printf("Enter the size of files: ");
   for(i=1;i<=nf;i++)
   {
      scanf("%d",&file[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(block_filled[j]!=1)
```

```
          {
             temp=block[j]-file[i];
             if(temp>=0 && lowest>temp)
             {
                file_filled_in[i]=j;
                lowest=temp;
             }
          }
       }
       frag[i]=lowest;
       block_filled[file_filled_in[i]]=1;
       lowest=10000;
    }
    printf("File no: \t File size: \t Block no: \tBlock size: \tFragment:\n");
    for(i=1;i<=nf && file_filled_in[i]!=0;i++)
       printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,file[i],file_filled_in[i],block[file_filled_in[i]],fra
g[i]);
}
```

SAMPLE OUTPUT:

```
***** MEMORY MANAGEMENT SCHEME - BEST FIT *****
Enter the no. of blocks: 3
Enter the no. of files: 3
Enter the size of blocks: 10 5 7
Enter the size of files: 4 6 9
File no:      File size:    Block no:    Block size:    Fragment:
1             4             2            5              1
2             6             3            7              1
3             9             1            10             1
```

C) FIRST FIT:

SOURCE CODE:

```
#include<stdio.h>
#define max 25
void main()
{
   int frag[max],block[max],file[max],i,j,nb,nf,temp;
   static int block_filled[max],file_filled_in[max];
//if array is declared as static int ,all values will be initialized to zero
   printf("\n***** MEMORY MANAGEMENT SCHEME - BEST FIT *****\n");
   printf("Enter the no. of blocks: ");
   scanf("%d",&nb);
   printf("Enter the no. of files: ");
   scanf("%d",&nf);
   printf("Enter the size of blocks: ");
   for(i=1;i<=nb;i++)
   {
      scanf("%d",&block[i]);
   }
   printf("Enter the size of files: ");
   for(i=1;i<=nf;i++)
   {
      scanf("%d",&file[i]);
   }
   for(i=1;i<=nf;i++)
   {
      for(j=1;j<=nb;j++)
      {
         if(block_filled[j]!=1)
```

```
          {
             temp=block[j]-file[i];
             if(temp>=0)
             {
                file_filled_in[i]=j;
                break;
             }
          }
       }
       frag[i]=temp;
       block_filled[file_filled_in[i]]=1;
    }
    printf("File no: \t File size: \t Block no: \tBlock size: \tFragment:\n");
    for(i=1;i<=nf && file_filled_in[i]!=0;i++)
       printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,file[i],file_filled_in[i],block[file_filled_in[i]],fra
g[i]);
}
```

SAMPLE OUTPUT:

```
***** MEMORY MANAGEMENT SCHEME - BEST FIT *****
Enter the no. of blocks: 3
Enter the no. of files: 3
Enter the size of blocks: 10 20 10
Enter the size of files: 2 12 3
File no:      File size:    Block no:     Block size:    Fragment:
1              2              1              10             8
2              12             2              20             8
3              3              3              10             7
```

7) Develop a C program to simulate page replacement algorithms:

a) FIFO b) LRU

SOURCE CODE:

```
#include<stdio.h>
#define max_pages 20
#define max_frames 3
void displayPages(int pages[],int n)
{
   printf("Current pages in memory: ");
   for(int i=0;i<n;i++)
      printf("%d ",pages[i]);
   printf("\n");
}

int findPage(int pages[],int n,int target)
{
   for(int i=0;i<n;i++)
   {
      if(pages[i]==target)
         return i;
   }
   return -1;
}

void FIFO(int pageRequests[],int numRequests)
{
   int frames[max_frames];
   int frameIndex=0;
   int pageFaults=0;
   for(int i=0;i<numRequests;i++)
```

```
    {
        printf("Referencing page %d\n",pageRequests[i]);
        displayPages(frames,frameIndex);
        if(findPage(frames,frameIndex,pageRequests[i])==-1)
        {
            if(frameIndex<max_frames)
                frames[frameIndex++]=pageRequests[i];
            else
            {
                for(int j=0;j<max_frames-1;j++)
                    frames[j]=frames[j+1];
                frames[max_frames-1]=pageRequests[i];
            }
            printf("Page %d is not in the memory. Page fault\n",pageRequests[i]);
            pageFaults++;
        }
        else
        {
            printf("Page %d is already in the memory.No page fault\n",pageRequests[i]);
        }

    }
    printf("Total page faults with FIFO: %d\n\n",pageFaults);
}

void LRU(int pageRequests[],int numRequests)
{
    int frames[max_frames];
    int pageFaults=0;
    int used[max_frames];
    for(int j=0;j<max_frames;j++)
    {
        frames[j]=-1;
        used[j]=0;
    }
    for(int i=0;i<numRequests;i++)
    {
        printf("Referencing page %d\n",pageRequests[i]);
        displayPages(frames,max_frames);
        int pageIndex=findPage(frames,max_frames,pageRequests[i]);
        if(pageIndex==-1)
        {
            int replaceIndex=0;
            for(int j=1;j<max_frames;j++)
            {
                if(used[j]<used[replaceIndex])
                    replaceIndex=j;
            }
            frames[replaceIndex]=pageRequests[i];
            used[replaceIndex]=i+1;
            printf("Page %d is not in the memory. Page fault\n",pageRequests[i]);
            pageFaults++;
        }
        else
        {
            used[pageIndex]=i+1;
            printf("Page %d is already in the memory.No page fault\n",pageRequests[i]);
        }
    }
    printf("Total page faults with LRU: %d\n",pageFaults);
}
```

```
int main()
{
   int pageRequests[max_pages]={1,2,3,4,1,2,5,1,2,3,4,5};
   int numRequests=12;
   printf("****Simulating FIFO page replacement algotithm****\n");
   FIFO(pageRequests,numRequests);
   printf("****Simulating FIFO page replacement algotithm****\n");
   LRU(pageRequests,numRequests);
}
```

SAMPLE OUTPUT:

```
****Simulating FIFO page replacement algotithm****
Referencing page 1
Current pages in memory:
Page 1 is not in the memory. Page fault
Referencing page 2
Current pages in memory: 1
Page 2 is not in the memory. Page fault
Referencing page 3
Current pages in memory: 1 2
Page 3 is not in the memory. Page fault
Referencing page 4
Current pages in memory: 1 2 3
Page 4 is not in the memory. Page fault
Referencing page 1
Current pages in memory: 2 3 4
Page 1 is not in the memory. Page fault
Referencing page 2
Current pages in memory: 3 4 1
Page 2 is not in the memory. Page fault
Referencing page 5
```

```
Current pages in memory: 4 1 2
Page 5 is not in the memory. Page fault
Referencing page 1
Current pages in memory: 1 2 5
Page 1 is already in the memory.No page fault
Referencing page 2
Current pages in memory: 1 2 5
Page 2 is already in the memory.No page fault
Referencing page 3
Current pages in memory: 1 2 5
Page 3 is not in the memory. Page fault
Referencing page 4
Current pages in memory: 2 5 3
Page 4 is not in the memory. Page fault
Referencing page 5
Current pages in memory: 5 3 4
Page 5 is already in the memory.No page fault
Total page faults with FIFO: 9

****Simulating FIFO page replacement algotithm****
Referencing page 1
Current pages in memory: -1 -1 -1
Page 1 is not in the memory. Page fault
Referencing page 2
Current pages in memory: 1 -1 -1
Page 2 is not in the memory. Page fault
Referencing page 3
Current pages in memory: 1 2 -1
Page 3 is not in the memory. Page fault
Referencing page 4
Current pages in memory: 1 2 3
```

```
Page 4 is not in the memory. Page fault
Referencing page 1
Current pages in memory: 4 2 3
Page 1 is not in the memory. Page fault
Referencing page 2
Current pages in memory: 4 1 3
Page 2 is not in the memory. Page fault
Referencing page 5
Current pages in memory: 4 1 2
Page 5 is not in the memory. Page fault
Referencing page 1
Current pages in memory: 5 1 2
Page 1 is already in the memory.No page fault
Referencing page 2
Current pages in memory: 5 1 2
Page 2 is already in the memory.No page fault
Referencing page 3
Current pages in memory: 5 1 2
Page 3 is not in the memory. Page fault
Referencing page 4
Current pages in memory: 3 1 2
Page 4 is not in the memory. Page fault
Referencing page 5
Current pages in memory: 3 4 2
Page 5 is not in the memory. Page fault
Total page faults with LRU: 10
```

8) Simulate following File Organization Techniques: a) Single level directory

b) Two level directory.

A)SINGLE LEVEL 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];
   printf("\nEnter name of directory -- ");
   scanf("%s", dir.dname);
   while (1)
   {
      printf("\n\n1. Create File\t2. Delete File\t3. Search File \t 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);
      }
   }
}
```

SAMPLE OUTPUT:

```
Enter name of directory -- TEXT_FILES



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.txt



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.txt



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

Enter the name of the file -- c.txt



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

Enter the name of the file -- b.txt
File b.txt is deleted

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

Enter the name of the file -- c.txt
File c.txt is found

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.txt
File b.txt not found

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

The Files are --        a.txt   c.txt
```

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

B)TWO LEVEL DIRECTORY:

SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct
{
   char dname[10],fname[10][10];
   int fcnt;
}dir[10];

void main()
{
   int i,ch,dcnt=0,k;
   char f[30],d[30];

   while(1)
   {
      printf("\n\n1.Create Directory\t2.Create File\t3.Delete File\n4.Search
File\t5.Display\t6.Exit\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("Diractory created");
         break;
```

```
case 2:
   printf("\nEnter name of the directory -- ");
   scanf("%s",d);
   //searching the diractory name using linear search
   for(i=0;i<dcnt;i++)
   {
      if(strcmp(d,dir[i].dname)==0)
      {
         printf("Enter the 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("Enter name of directory-- ");
   scanf("%s",d);
   for(i=0;i<dcnt;i++)
   {
      if(strcmp(d,dir[i].dname)==0)
      {
         printf("Enter name of 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 jump;
            }
        }
        printf("File %s not found",f);
        goto jump;
    }
  }
  printf("Directory %s not found",d);
  jump:break;
break;

case 4:
  printf("Enter name of directory-- ");
  scanf("%s",d);
  for(i=0;i<dcnt;i++)
  {
    if(strcmp(d,dir[i].dname)==0)
    {
        printf("Enter name of 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 jump1;
```

```
                     }
                  }
                  printf("File %s not found",f);
                  goto jump1;
               }
            }
            printf("Directory %s not found",d);
            jump1:break;
         break;

         case 5:
            if(dcnt==0)
               printf("\nNo Directories");
            else
            {
               printf("Directory\tFiles");
               for(i=0;i<dcnt;i++)
               {
                  printf("\n%s \t",dir[i].dname);
                  for(k=0;k<dir[i].fcnt;k++)
                  {
                     printf("%s  ",dir[i].fname[k]);
                  }
               }
            }
         break;
         default: exit(0);
      }
   }
}
```

SAMPLE 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-- TEXTS

Diractory 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-- VIDEOS

Diractory 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 -- MOVIES

Directory MOVIES not found


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 -- TEXTS

Enter the name of the file-- a.txt

File created


1.Create Directory     2.Create File   3.Delete File
```

```
4.Search File   5.Display       6.Exit
Enter your choice-- 2

Enter name of the directory -- TEXTS
Enter the name of the file-- b.txt
File created

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

Enter name of the directory -- VIDEOS
Enter the name of the file-- dance.mp4
File created

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

Enter name of the directory -- VIDEOS
Enter the name of the file-- song.mp4
File created

1.Create Directory      2.Create File   3.Delete File
4.Search File   5.Display       6.Exit
Enter your choice-- 5
Directory       Files
TEXTS   a.txt  b.txt
VIDEOS  dance.mp4  song.mp4

1.Create Directory      2.Create File   3.Delete File
```

```
4.Search File   5.Display       6.Exit
Enter your choice-- 3
Enter name of directory-- TEXTS
Enter name of file-- b.txt
File b.txt is deleted

1.Create Directory      2.Create File   3.Delete File
4.Search File   5.Display       6.Exit
Enter your choice-- 3
Enter name of directory-- VIDEOS
Enter name of file-- movie.mp4
File movie.mp4 not found

1.Create Directory      2.Create File   3.Delete File
4.Search File   5.Display       6.Exit
Enter your choice-- 4
Enter name of directory-- TEXTS
Enter name of file-- a.txt
File a.txt is found

1.Create Directory      2.Create File   3.Delete File
4.Search File   5.Display       6.Exit
Enter your choice-- 4
Enter name of directory-- VIDEOS
Enter name of file-- movies.mp4
File movies.mp4 not found

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

```
Directory       Files
TEXTS   a.txt
VIDEOS  dance.mp4  song.mp4

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

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

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int f[50], p, i, j, k, a, st, len, c;
   for(i=0;i<50;i++)
      f[i]=0;
   printf("How many blocks that are already allocated: ");
   scanf("%d",&p);
   printf("Enter the block no.s that are already allocated: ");
   for(i=0;i<p;i++)
   {
      scanf("%d",&a);
      f[a]=1;
   }

   while(1)
   {
      printf("Enter the starting index block & length: ");
      scanf("%d %d",&st,&len);
      k=len;
      for(j=st;j<(k+st);j++)
      {
         if(f[j]==0)
         {
            f[j]=1;
            printf("\n%d->%d",j,f[j]);
         }
```

```
            else
            {
               printf("\n%d->file is already allocated",j);
               k++;
            }
         }
         printf("\nDo you want to enter one more file? (yes-1/no-0): ");
         scanf("%d",&c);
         if(c!=1)
            break;
      }
   }
```

SAMPLE OUTPUT:

```
How many blocks that are already allocated: 3

Enter the block no.s that are already allocated: 2 4 7

Enter the starting index block & length: 1 7


1->1

2->file is already allocated

3->1

4->file is already allocated

5->1

6->1

7->file is already allocated

8->1

9->1

10->1

Do you want to enter one more file? (yes-1/no-0): 1

Enter the starting index block & length: 3 8
```

```
3->file is already allocated

4->file is already allocated

5->file is already allocated

6->file is already allocated

7->file is already allocated

8->file is already allocated

9->file is already allocated

10->file is already allocated

11->1

12->1

13->1

14->1

15->1

16->1

17->1

18->1

Do you want to enter one more file? (yes-1/no-0): 0
```

10) Develop a C program to simulate SCAN disk scheduling algorithm.

SOURCE CODE:

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

void swap(int *a,int *b)
{
   int temp=*a;
   *a=*b;
   *b=temp;
}
int main()
{
   int queue[20];
   int n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],temp1=0,temp2=0;
   float avg;
   printf("Enter the max range of disk\n");
   scanf("%d",&max);
   printf("Enter the initial head position\n");
   scanf("%d",&head);
   printf("Enter the size of queue request\n");
   scanf("%d",&n);
   printf("Enter the queue of disk positions to be read\n");
   for(i=1;i<=n;i++)
   {
      scanf("%d",&temp);
      if(temp>=head)
      {
         queue1[temp1]=temp;
```

```
        temp1++;
    }
    else
    {
        queue2[temp2]=temp;
        temp2++;
    }
}
//sorting queue1 in ascending order
for(i=0;i<temp1-1;i++)
{
    temp=i;
    for(j=i+1;j<temp1;j++)
    {
        if(queue1[temp]>queue1[j])
            temp=j;
    }
    swap(&queue1[temp],&queue1[i]);
}

//sorting queue2 in descending order
for(i=0;i<temp2-1;i++)
{
    temp=i;
    for(j=i+1;j<temp2;j++)
    {
        if(queue2[temp]<queue2[j])
            temp=j;
    }
    swap(&queue2[temp],&queue2[i]);
}
```

```
   //to check whether to move position to greater or lesser position than initial head 
position
   //in other words to decide whether to concatenate 'queue1' or 'queue2' to the 'queue' 
first
   if(abs(head-queue1[0])<=abs(head-queue2[0]))
   {
      queue[0]=head;
      for(i=1,j=0;j<temp1;i++,j++)
         queue[i]=queue1[j];
      queue[i]=max;

      for(i=i+1,j=0;j<temp2;i++,j++)
         queue[i]=queue2[j];
      queue[i]=0;
   }
   else
   {
      queue[0]=head;
      for(i=1,j=0;j<temp2;j++,i++)
         queue[i]=queue2[j];
      queue[i]=0;

      for(i=i+1,j=0;j<temp1;j++,i++)
         queue[i]=queue1[j];

      queue[i]=max;
   }

   for(j=0;j<=n+1;j++)
   {
        diff=abs(queue[j+1]-queue[j]);
        seek+=diff;
        printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);
    }
    printf("Total seek time is %d\n",seek);
    avg=seek/(float)n;
    printf("Average seek time is %f\n",avg);
    return 0;
}
```

SAMPLE OUTPUT 1:

```
Enter the max range of disk
200
Enter the initial head position
50
Enter the size of queue request
8
Enter the queue of disk positions to be read
49 55 60 71 35 40 90 20
Disk head moves from 50 to 49 with seek 1
Disk head moves from 49 to 40 with seek 9
Disk head moves from 40 to 35 with seek 5
Disk head moves from 35 to 20 with seek 15
Disk head moves from 20 to 0 with seek 20
Disk head moves from 0 to 55 with seek 55
Disk head moves from 55 to 60 with seek 5
Disk head moves from 60 to 71 with seek 11
Disk head moves from 71 to 90 with seek 19
Disk head moves from 90 to 200 with seek 110
Total seek time is 250
Average seek time is 31.250000
```

SAMPLE OUTPUT 2:

```
Enter the max range of disk
120
Enter the initial head position
28
Enter the size of queue request
6
Enter the queue of disk positions to be read
45 30 11 23 35 8
Disk head moves from 28 to 30 with seek 2
Disk head moves from 30 to 35 with seek 5
Disk head moves from 35 to 45 with seek 10
Disk head moves from 45 to 120 with seek 75
Disk head moves from 120 to 23 with seek 97
Disk head moves from 23 to 11 with seek 12
Disk head moves from 11 to 8 with seek 3
Disk head moves from 8 to 0 with seek 8
Total seek time is 212
Average seek time is 35.333332
```