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

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

int main()
{
  pid_t pid;
  printf("Parent process (PID:%d)\n",getpid());

  pid=fork();
  if(pid<0)
  {
    perror("Fork failed");
    exit(EXIT_FAILURE);
  }
  else if(pid==0)
  {
    printf("Child process (PID:%d),Parent PID:%d\n",getpid(),getppid());
    execl("/bin/ps","ps",NULL);
    perror("Exec failed");
    exit(EXIT_FAILURE);
  }
  else
  {
    printf("Parent process,waiting for the child...\n");
    wait(NULL);
    printf("Child process completed.\n");
  }
  return 0;
}
```

OUTPUT :

```
Parent process (PID:4070)
Parent process,waiting for the child...
Child process (PID:4071),Parent PID:4070
  PID TTY          TIME CMD
 4055 pts/0    00:00:00 bash
 4070 pts/0    00:00:00 a.out
 4071 pts/0    00:00:00 ps
Child process completed.
```

---

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

FCFS

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

int main()
{

    Int n,arrivalTime[20], burstTime[20], startTime[20], finishTime[20],
waitingTime[20], turnaroundTime[20];
    float avgTat,avgWt;
    char processName[20][20];
    printf("Enter the number of processes\n");
    scanf("%d",&n);

    for (int i=0;i<n;i++)
    {
        printf("Enter the Process name ,Arrival Time and Burst Time:");
        scanf("%s%d%d",&processName[i],&arrivalTime[i],&burstTime[i]);
    }
    for (int i=0;i<n;i++)
    {
        if(i==0)
        {
            startTime[i]=arrivalTime[i];
            waitingTime[i]=startTime[i]-arrivalTime[i]

finishTime[i]=startTime[i]+burstTime[i];
            turnaroundTime[i]=finishTime[i]-arrivalTime[i];
        }
        else
        {
            startTime[i]=finishTime[i-1];
            waitingTime[i]=startTime[i]-arrivalTime[i];
            finishTime[i]=startTime[i]+burstTime[i];
            turnaroundTime[i]=finishTime[i]-arrivalTime[i];
            }
            }
```

```
    int totTat=0;
    int totWt=0;
    printf("\nProcess Arrival\t Burst\t Start\t Turnaround\t Wait\t Finish");
    for(int i=0;i<n;i++)
    {

printf("\n%s\t%4d\t%4d\t%4d\t%4d\t\t%4d\t%4d",processName[i],arrivalTime[i]
,burstTime[i],
            startTime[i],turnaroundTime[i],waitingTime[i],finishTime[i]);
        totWt+=waitingTime[i];
        totTat+=turnaroundTime[i];
    }
    avgWt=(float)totWt/n;
    avgTat=(float)totTat/n
    ;

    printf("\n Average Turnaround Time:%2f",avgTat);
    printf("\n Average Wait Time:%2f",avgWt);
}
```

OUTPUT :

```
Enter the number of processes
4
Enter the Process name ,Arrival Time and Burst Time:P0  0  6
Enter the Process name ,Arrival Time and Burst Time:P1  1  3
Enter the Process name ,Arrival Time and Burst Time:P2   2  1
Enter the Process name ,Arrival Time and Burst Time:P3   3  4

Process  Arrival  Burst   Start   Turnaround    Wait    Finish
P0          0        6        0          6              0      6
P1          1        3        6          8              5      9
P2          2        1        9          8              7     10
P3          3        4       10         11              7     14
 Average Turnaround Time:8.250000
Average Wait Time:4.750000
```

SJF

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

int main()
  {
    int i=0, processNumber[10], burstTime[10], waitTime[10],
turnaroundTime[10], n,j,temp=0;
    float avgWaitTime , avgTat;
    printf("Enter the number of processes : ");
    scanf("%d",&n);

    printf("Enter the burst time of each process : \n");
    for(i=0;i<n;i++)
    {
        printf("p%d : ",i);
        scanf("%d",&burstTime[i]);
        processNumber[i] = i;
    }
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(burstTime[i]>burstTime[j])
            {
                temp = burstTime[i];
                burstTime[i] = burstTime[j];
                burstTime[j] = temp;
                temp = processNumber[i];
                processNumber[i] = processNumber[j];
                processNumber[j] = temp;

            }
        }
    }
     waitTime[0] = 0;
    int totalWt=0,totalTat=0;
    for(i=1;i<n;i++)
    {
        waitTime[i] = burstTime[i-1] + waitTime[i-1];
        totalWt += waitTime[i];
}
```

```
printf("\nProcess no.\tBurst time\tWaiting time\tTurn around time\n");
    for(i=0;i<n;i++)
    {
        turnaroundTime[i] = burstTime[i] + waitTime[i];
        totalTat += turnaroundTime[i];

printf("\np%d\t\t%d\t\t%d\t\t%d",processNumber[i],burstTime[i],waitTime[i],turn
aroundTime[i]);
    }
    avgWaitTime = (float) totalWt / n;
    avgTat = (float) totalTat / n;
    printf("\nAverage waiting time : %0.2f\nAverage turn around time :
%0.2f",avgWaitTime,avgTat);
}
```

OUTPUT:

```
Enter the number of processes : 3
Enter the burst time of each process :
p0 : 24
p1 : 3
p2 : 3

Process no.     Burst time      Waiting time    Turn around time

p1              3               0               3
p2              3               3               6
p0              24               6               30
Average waiting time : 3.00
Average turn around time : 13.00
```

ROUND ROBIN

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

```
                j=p[i].burst-p[i].comp;
                p[i].comp=p[i].burst;
                p[i].f=0;
            }

printf("\nprocess#%-3d\t%-10d%-10d%-10d",i+1,time,time+j,p[i].burst-p[i].com
p);
            time+=j;
        }
    }
  }
  printf("\n\n------------------");
  printf("\nProcess \tWaiting Time\t TurnAround Time");
  for(i=0;i<n;i++)
  {
     printf("\nProcess#%-12d\t%-15d%-15d",i+1,p[i].wait,p[i].wait+p[i].burst);
     totalwait=totalwait+p[i].wait;
     totalturn=totalturn+p[i].wait+p[i].burst;
  }
  printf("\n\nAverage\n-----------"); printf("\nWaiting
  Time:%fms",totalwait/(float)n); printf("\nTurnAround
  Time :%fms\n\n",totalturn/(float)n);
}
```

OUTPUT :

```
Enter The No Of Process:3

Enter The Time Quantum (in ms):2
Enter The Burst Time (in ms)For Process# 1:12
Enter The Burst Time (in ms)For Process# 2:3
Enter The Burst Time (in ms)For Process# 3:2

Order Of Execution

Process Starting Ending Remaining
            Time   Time   Time
process#1      0       2       10
process#2      2       4       1
process#3      4       6       0
```

process#1 6 8 8
process#2 8 9 0
process#1 9 11 6
process#1 11 13 4
process#1 13 15 2
process#1 15 17 0

Process	Waiting Time	TurnAround Time
Process#1	5	17
Process#2	6	9
Process#3	4	6

Average

Waiting Time:5.000000ms

TurnAround Time :10.666667ms

PRIORITY

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

int main()
{
  int
processNo[20],burstTime[20],priority[20],waitTime[20],turnaroundTime[20],i,j,n
,temp;
  int totWaitTime=0,totTat=0;
  float avgWaitTime,avgTat;
  printf("Enter the number of processes---");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
     processNo[i]=i;
     printf("Enter the Burst time and Priority of process %d---",i);
     scanf("%d%d",&burstTime[i],&priority[i]);
  }
  for(i=0;i<n-1;i++)
  {
     for(j=i+1;j<n;j++)
       if(priority[i]>priority[j])
     {
        temp=priority[i];
        priority[i]=priority[j];
        priority[j]=temp;
        temp=burstTime[i];
        burstTime[i]=burstTime[j];
        burstTime[j]=temp;
        temp=processNo[i];
        processNo[i]=processNo[j]
        ; processNo[j]=temp;
        }
        }
  waitTime[0]=0;
    avgTat=turnaroundTime[0]=burstTime[0]
    ; for(i=1;i<n;i++)
    {
```

```
waitTime[i]=waitTime[i-1]+burstTime[i-1];
        turnaroundTime[i]=burstTime[i]+waitTime[i];
    }
    for(i=0;i<n;i++)
    {
        totWaitTime=totWaitTime+waitTime[i];
        totTat=totTat+turnaroundTime[i];
    }
     printf("\nProcess number\tPriority\tBurst Time\tWait Time\tTurnaround
 Time");
    for(i=0;i<n;i++)

 printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d",processNo[i],priority[i],burstTime[i],waitTi
 me[i],turnaroundTime[i]);
        printf("\n Average Waiting time is---%f",(float)totWaitTime/n);
        printf("\n Average Turnaround Time is---%f",(float)totTat/n);
        getch();
}
```

OUTPUT :

```
Enter the number of processes---5
Enter the Burst time and Priority of process 0---10 3

Enter the Burst time and Priority of process 1---1 1
Enter the Burst time and Priority of process 2---2 4
Enter the Burst time and Priority of process 3---1 5
Enter the Burst time and Priority of process 4---5 2

Process number   Priority       Burst Time     Wait Time      Turnaround Time
P1                    1              1              0              1
P4                    2              5              1              6
P0                    3              10             6              16
P2                    4              2              16             18
P3                    5              1              18             19
 Average Waiting time is---8.200000
 Average Turnaround Time is---12.000000
```

---

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

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

int S=1;
int F=0;
int E=4,X=0;

void signal(int*S)
{
    ++S;
    return;
}

void wait(int*S)
{
    --S;
    return;
}

void producer()
{
    wait(&S);
    ++F;
    --E;
    X++;
    printf("\n Producer produces item %d",X);
    signal(&S);
}
void consumer()
{
    wait(&S);
    --F;
    ++E;
    printf("\nConsumer consumes the item %d",X); X-
    -;
    signal(&S);
}

int main()
{
```

```
    int n,i;
   printf("\n 1.Press 1 for Producer\n 2.Press 2 for Consumer\n 3.Press 3 to
Exit\n");
   for(i=1;i>0;i++)
   {
      printf("\n Enter your choice---");
      scanf("%d",&n);
      switch (n)
      {
      case 1:
         if ((S==1)&&(E!=0))
         {
            producer();
         }
         else
         {
            printf("\nBuffer is full!!");
         }
         break;
      case 2:

         if ((S==1)&&(F!=0))
         {
            consumer();
         }
         else
         {
            printf("\nBuffer is empty!!");
         }
         break;

      case 3:
         exit(0);
         break;
      }
   }
}
```

OUTPUT :

1.Press 1 for Producer
2.Press 2 for Consumer
3.Press 3 to Exit

Enter your choice---1

Producer produces item 1
Enter your choice---1

Producer produces item 2
Enter your choice---1

Producer produces item 3
Enter your choice---1

Producer produces item 4
Enter your choice---1

Buffer is full!!
Enter your choice---2

Consumer consumes the item 4

Enter your choice---2

Consumer consumes the item 3
Enter your choice---2

Consumer consumes the item 2
Enter your choice---2

Consumer consumes the item 1
Enter your choice---2

Buffer is empty!!
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

writer---

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>

int main()
{
  int fd;
  char buf[1024];
  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"));
  close(fd);
  unlink(myfifo);
  return 0;
}
```

reader---

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
#define MAX_BUF 1024

int main()
{
  int fd;
  char *myfifo="/tmp/myfifo";
  char buf[MAX_BUF];
  fd=open(myfifo,O_RDONLY);
  read(fd,buf,MAX_BUF);
```

```
  printf("Writer:%s\n",buf);
  close(fd);
 return 0;
}
```

OUTPUT

Run Reader process to read FIFO file

Writer:Hi

—----------

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 process--");
scanf("%d",&p);
for(i=0;i<p;i++)
   completed[i]=0;
   printf("Enter the no of resources--");
   scanf("%d",&r);
   printf("\n\nEnter the Max Matrix for each process--");
   for(i=0;i<p;i++)
   {
      printf("\n For process %d:",i+1);
      for(j=0;j<r;j++)
         scanf("%d",&Max[i][j]);
}

   printf("\n\n Enter 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("\nMax Matrix:\t\t\t Allocation matrix:\n");
         for(i=0;i<p;i++)
         {
            for(j=0;j<r;j++)
               printf("%d\t",Max[i][j]);
            printf("\t\t");
            for(j=0;j<r;j++)
               printf("%d\t",alloc[i][j]);
            printf("\n");
         }

         process=-1;
         for(i=0;i<p;i++)
         {
            if(completed[i]==0)
            {
               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("\n Process %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("\n The system is in an unsafe state!!");

}
```

OUTPUT :

```
Enter the no of process--5
Enter the no of resources--3


Enter the Max Matrix for each
 process-- For process 1:7 5 3

 For process 2:3 2 2

 For process 3:9 0 2

 For process 4:2 2 2

 For process 5:4 3 3
```

Enter the allocation for each process--
For process1:0 1 0

For process2:2 0 0

For process3:3 0 2

For process4:2 1 1

For process5:0 0 2

Enter the available resources--3 3 2

Max Matrix:			Allocation matrix:		
7	5	3	0	1	0
3	2	2	2	0	0
9	0	2	3	0	2
2	2	2	2	1	1
4	3	3	0	0	2

Process 2 runs to completion!

Max Matrix:			Allocation matrix:		
7	5	3	0	1	0
0	0	0	0	0	0
9	0	2	3	0	2
2	2	2	2	1	1
4	3	3	0	0	2

Process 4 runs to completion!

Max Matrix:			Allocation matrix:		
7	5	3	0	1	0
0	0	0	0	0	0
9	0	2	3	0	2
0	0	0	0	0	0
4	3	3	0	0	2

Process 1 runs to completion!

Max Matrix:			Allocation matrix:		
0	0	0	0	0	0
0	0	0	0	0	0
9	0	2	3	0	2
0	0	0	0	0	0
4	3	3	0	0	2

Process 3 runs to completion!

Max Matrix:			Allocation matrix:		
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
4	3	3	0	0	2

Process 5 runs to completion!

The system is in a safe
state!! Safe
Sequence:<24135

—----------

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

WORST FIT

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

int main()
{
    int blockSize[]={100,500,200,300,600};
    int processSize[]={212,417,112,426};
    int m=5;  //no of blocks
    int n=4;  //no of processes
    int allocation[n];
    int i,j;

    for(int i=0;i<n;i++){
        allocation[i]=-1;
        int worstFitdx=-1;
        for(int j=0;j<m;j++){
            if(blockSize[j]>=processSize[i]){
                if(worstFitdx==-1||blockSize[j]>blockSize[worstFitdx]){
                    worstFitdx=j;
                }
            }
        }
        if (worstFitdx!=-1){
            allocation[i]=worstFitdx;
            blockSize[worstFitdx]-=processSize[i];
        }
    }
     printf("......Worst fit....");
     printf("\n Process No.\tProcess Size\t Block No.\n");
     for(int i=0;i<n;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

Worst fit

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

BEST FIT

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

int main()
{
    int blockSize[]={100,500,200,300,600};
    int processSize[]={212,417,112,426};
    int m=5;  //no of blocks
    int n=4;  //no of processes
    int allocation[n];
    int i,j;

    for(int i=0;i<n;i++){
        allocation[i]=-1;
        int bestFitdx=-1;
        for(int j=0;j<m;j++){
            if(blockSize[j]>=processSize[i]){
                if(bestFitdx==-1||blockSize[j]<blockSize[bestFitdx]){
                    bestFitdx=j;
                }

            }
        }
        if(bestFitdx!=-1){ allocation[i]=bestFitdx;
            blockSize[bestFitdx]-=processSize[i];
        }
    }
     printf("......Best fit....");
      printf("\n Process No.\tProcess Size\t Block No.\n");
      for(int i=0;i<n;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

Output:

.....Best fit....

Process No.	Process Size	Block No.
1	212	4
2	417	2
3	112	3
4	426	5

FIRST FIT

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

int main()
{
    int blockSize[]={100,500,200,300,600};
    int processSize[]={212,417,112,426};

    int m=5;  //no of blocks
    int n=4;  //no of
    processes int
    allocation[n];
    int i,j;

    for(int i=0;i<n;i++){
        allocation[i]=-1;
        for(int
        j=0;j<m;j++){
            if(blockSize[j]>processSize[i]){
            allocation[i]=j;
            blockSize[j]-=processSize[i];
            break;
          }
       }
    }

     printf("......First fit....");
     printf("\n Process No.\tProcess Size\t Block No.\n");
     for(int i=0;i<n;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 :

First fit

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

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

FIFO

```
#include <stdio.h>
int fr[3];
int front=0,rear=0;

void enqueue(int page)
{
   fr[rear]=page;
   rear=(rear+1)%3;
}

void display()
{
   int i; printf("\n");
   for(i=0;i<3;i++)
       printf("\t%d",fr[i]);
}

int isPageInFrames(int page)
{
   for(int i=0;i<3;i++)
   {
        if(fr[i]==page)
            return 1;
   }
   return 0;
}

void main()
{
   printf("\n----------FIFO PAGE REPLACEMENT ALGORITHM----------\n");
   int p[12]={1,2,3,4,1,2,5,1,2,3,4,5},i,j,pf=0,frsize=3;

   for(i=0;i<3;i++)
   {
      fr[i]=-1;
   }
   for(j=0;j<12;j++)
{
```

```
        if(!isPageInFrames(p[j]))
        {
            enqueue(p[j]);
            pf++;
        }
        display();
    }
    printf("\n Number of page faults--%d\n",pf);
}
```

OUTPUT:

```
------------FIFO PAGE REPLACEMENT ALGORITHM----------

        1       -1      -1
        1       2       -1
        1       2       3
        4       2       3
        4       1       3
        4       1       2
        5       1       2
        5       1       2
        5       1       2
        5       3       2
        5       3       4
        5       3       4
 Number of page faults--9
```

LRU

```
#include <stdio.h>

int frames[3];

void displayFrames()
{
    int i;
    printf("\n");
    for (i=0;i<3;i++)
        printf("\t%d",frames[i]);
}
void main()
{
    printf("\n----------LRU PAGE REPLACEMENT----------\n");
    int
index,k,l,pageFoundInFrames=0,pageInserted=0,pageFaults=0,frameSize=
3;
    int pageReferences[12]={1,2,3,4,1,2,5,1,2,3,4,5},i,j,pageStatus[3];
    for(i=0;i<frameSize;i++)
    {
        frames[i]=-1;
    }
    for(j=0;j<12;j++)
    {
        pageFoundInFrames=
        0; pageInserted=0;
        for(i=0;i<frameSize;i+
        +)
        {
            if(frames[i]==pageReferences[j])
            {
                pageFoundInFrames=1;
                break;
            }
        }
        if(!pageFoundInFrames)
        {
```

```
    for(i=0;i<frameSize;i++)
    {
        if(frames[i]==-1)
        {
            frames[i]=pageReferences[j];
            pageInserted=1;
            break;
        }
    }
    if(!pageInserted)
    {
        for(i=0;i<frameSize;i++)
        {
            pageStatus[i]=0;
        }

        for(k=j-1,l=1;l<=frameSize-1;l++,k--)
        {
            for(i=0;i<frameSize;i++)
            {
                if(frames[i]==pageReferences[k])
                {
                    pageStatus[i]=1;
                }
            }
        }
         for(i=0;i<frameSize;i++){
            if(pageStatus[i]==0)
            {
                index=i;
            }
        }
        frames[index]=pageReferences[j];
        pageFaults++;
    }
}

displayFrames();
}
```

```
    printf("\n Number of page faults--%d\n",pageFaults+frameSize);
    }
```

OUTPUT :

```
 LRU PAGE REPLACEMENT

      1      -1     -1
      1      2      -1
      1      2      3
      4      2      3
      4      1      3
      4      1      2
      5      1      2
      5      1      2
      5      1      2
      3      1      2
      3      4      2
      3      4      5
  Number of page faults--10
```

—----------

8. Simulate following File Organization Techniques a) Single level directory b) Two level directory

SINGLE LEVEL DIRECTORY

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

OUTPUT:

```
Enter name of directory--WEATHER


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

Enter the name of the file--SUNNY


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

Enter the name of the file--WINTER


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

The Files are-- SUNNY   WINTER

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

Enter the name of the file--SUNNY
File SUNNY is found

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

4.Display File 5.Exit
Enter your choice--2

Enter the name of the file--WINTER
File WINTER is deleted

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

The Files are-- SUNNY

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

MULTI LEVEL DIRECTORY

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
struct
{
   char dname[10],fname[10][10];
   int fcnt;
}dir[10];
void main()
{
   int i,ch,dcnt,k;
   char f[30],d[30];
   dcnt=0;
   while(1)
   {
      printf("\n\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]);
         dir[i].fcnt++;
         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 found",f);
                    goto jmp;
                }
```

```
            }
            printf("File %s not found",f);
            goto jmp1;
          }
        }
        printf("Directory %s not found",d);
        jmp1:break; case
    5:if(dcnt==0)
    printf("\nNo Directory");
    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-ANIMALS
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--PLANTS
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--ANIMALS
Enter name of the file--CAT

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--PLANTS
Enter name of the file--BANANA

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

Directory Files
ANIMALS CAT
PLANTS BANANA

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--
PLANTS Enter the name of the
file--BANANA File BANANA found

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

Enter name of the directory--PLANTS
Enter name of the file--BANANA
File BANANA is deleted

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

Directory Files
ANIMALS CAT
PLANTS

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.

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

for(i=0;i<50;i++)
f[i]=0;
printf("Enter the number of blocks that are already allocated: ");
scanf("%d",&p);
printf("Enter the index of the blocks that are 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 the number of blocks that are already allocated: 10
Enter the index of the blocks that are already allocated: 1
3
5
7
11
13
15
16
18
19
Enter index starting block and length: 0 5
0-------->1
1 Block is already allocated
2-------->1
3 Block is already allocated
4-------->1
5 Block is already allocated
6-------->1
7 Block is already allocated
8-------->1
```

Do you want to enter more file (Yes - 1/No - 0)1
Enter index starting block and length: 10 20
10-------->1
11 Block is already allocated
12-------->1
13 Block is already allocated
14-------->1
15 Block is already allocated
16 Block is already allocated
17-------->1
18 Block is already allocated
19 Block is already allocated
20-------->1
21-------->1

22-------->1
23-------->1
24-------->1
25-------->1
26-------->1
27-------->1
28-------->1
29-------->1
30-------->1
31-------->1
32-------->1
33-------->1
34-------->1
35-------->1
Do you want to enter more file (Yes - 1/No - 0)0

—----------

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

```
#include<stdio.h>
int main() {
int queue[20],head,n,i,j,numMove=0,direction,maxTrack;
printf("enter the number of disk requests:");
scanf("%d",&n);
printf("enter the disk request queue:");
for(i=0;i<n;i++) {
   scanf("%d",&queue[i]);
}
printf("enter the initial head position:");
scanf("%d",&head);
printf("enter the maximum track number:");
scanf("%d",&maxTrack);
printf("enter the direction(0 for left,1 for right):");
scanf("%d",&direction);
printf("\n");
int temp;

for(i=0;i<n-1;i++) {
   for(j=i+1;j<n;j++) {
      if(queue[i]>queue[j]) {
         temp=queue[i];
         queue[i]=queue[j];
         queue[j]=temp;
      }
   }
}
int currentTrack=head;
printf("Seek Sequence:");
if(direction==0) {
   for(i=head;i>=0;i--) { printf("%d\t",i);
      numMove+=abs(currentTrack-i);
      currentTrack=i;
   } printf("0");
   numMove+=currentTrack;
   for(i=1;i<=maxTrack;i++) {
      printf("%d\t",i);
      numMove+=abs(currentTrack-i);
      currentTrack=i;
   }
}
```

```
else {
   for(i=head;i<=maxTrack;i++) {
      printf("%d\t",i);
      numMove+=abs(currentTrack-i);
      currentTrack=i;
   } printf("%d\t",maxTrack);
   numMove+=abs(currentTrack-maxTrack);
   for(i=maxTrack-1;i>=0;i--) {
      printf("%d\t",i);
      numMove+=abs(currentTrack-i);
      currentTrack=i;
   }
}
printf("\nTotal no. of track movements:%d\n",numMove);

getch();
}
```

OUTPUT:

```
enter the number of disk requests:7
enter the disk request queue:82 170 43 140 24 16
190 enter the initial head position:50
enter the maximum track number:199
 enter the direction(0 for left,1 for right):1
```

Seek Sequence: 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85
86 87 88 89 90 91 92 93 94 95 96 97 98
99 100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131
132 133 134 135 136 137 138 139 140 141 142
143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164
165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186
187 188 189 190 191 192 193 194 195 196 197
198 199 199 198 197 196 195 194 193 192 191
190 189 188 187 186 185 184 183 182 181 180
179 178 177 176 175 174 173 172 171 170 169
168 167 166 165 164 163 162 161 160 159 158
157 156 155 154 153 152 151 150 149 148 147
146 145 144 143 142 141 140 139 138 137 136
135 134 133 132 131 130 129 128 127 126 125
124 123 122 121 120 119 118 117 116 115 114
113 112 111 110 109 108 107 106 105 104 103
102 101 100 99 98 97 96 95 94 93 92 91
90 89 88 87 86 85 84 83 82 81 80 79
78 77 76 75 74 73 72 71 70 69 68 67
66 65 64 63 62 61 60 59 58 57 56 55
54 53 52 51 50 49 48 47 46 45 44 43
42 41 40 39 38 37 36 35 34 33 32 31
30 29 28 27 26 25 24 23 22 21 20 19
18 17 16 15 14 13 12 11 10 9 8 7 6 5
4 3 2 1 0
Total no. of track movements:348

—----------