

ANIRUDDH NAYAK

4CB22CS012

LAB PROGRAMS

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 ID: %d\n",getpid(),getppid());
        execl("/bin/ps","ps",NULL);
        perror("Exce failed");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("parent process,waiting for the child...\n");
        wait(NULL);
        printf("child process completer.\n");
    }
}
```

```

    }
    return 0;
}

```

OUTPUT:

```

Parent process (PID:24557)
parent process,waiting for the child...
Child process (PID:24558),Parent ID: 24557
PID TTY          TIME CMD
24550 pts/326    00:00:00 dash
    24557 pts/326    00:00:00 x9zZa5swgA.o
    24558 pts/326    00:00:00 ps
child process completer.

```

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

a)FCFS

```

#include <stdio.h>
#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 No. of Processes\n");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        printf("Enter Processes 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 \tBurst \tStart \tTurnaround \tWait\tFinish");
for (int i = 0; i < n; i++)
{
printf("\n%s\t%4d\t%4d\t%4d\t%4d\t%4d\t%4d",
processName[i], arrivalTime[i], burstTime[i], startTime[i],
turnaroundTime[i], waitingTime[i], finishTime[i]);
totWt += waitingTime[i];
totTat += turnaroundTime[i];
}

```

```

}

avgTat = (float)totTat / n;
avgWt = (float)totWt / n;

printf("\nAverage Turnaround Time:%.2f", avgTat);
printf("\nAverage Wait Time:%.2f", avgWt);
}

```

OUTPUT:

```

D:\CODEBLOCKS\oslab\exp2b x + v
Enter No. of Processes
4
Enter Processes Name, Arrival Time and Burst Time:
0
0
6
Enter Processes Name, Arrival Time and Burst Time:
1
1
3
Enter Processes Name, Arrival Time and Burst Time:
2
2
1
Enter Processes Name, Arrival Time and Burst Time:
3
3
4
Process Arrival      Burst   Start   Turaround   Wait   Finish
0           0           6       0           6       0       6
1           1           3       6           8       5       9
2           2           1       9           8       7      10
3           3           4      10          11       7      14
Average Turnaround Time:8.25
Average Wait Time:4.75
Process returned 0 (0x0)   execution time : 19.065 s
Press any key to continue.

```

b)SJF

```

#include <stdio.h>

#include <string.h>

main()
{
int i = 0, processNumber[10], burstTime[10], n, waitTime[10], temp =
0, j, turnaroundTime[10];
float avgWaitTime, avgTat;
printf("\n Enter the no of processes ");
scanf("\n %d", &n);

```

```

printf("\n Enter the burst time of each process");
for (i = 0; i < n; i++)
{
printf("\n p%d", i);
scanf("%d", &burstTime[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;
for (i = 1; i < n; i++)
{
waitTime[i] = burstTime[i - 1] + waitTime[i - 1];
avgWaitTime = avgWaitTime + waitTime[i];
}
printf("\n process no \t burst time\t waiting time \t turn around time\n");
for (i = 0; i < n; i++)
{

```

```

turnaroundTime[i] = burstTime[i] + waitTime[i];
avgTat += turnaroundTime[i];
printf("\n p%d\t\t%d\t\t%d\t\t%d", i, burstTime[i], waitTime[i], turnaroundTime[i]);
}
printf("\n\n\t Average waiting time%f\n\t Average turn around time%f", avgWaitTime,
avgTat);
}

```

OUTPUT:

```

D:\CODEBLOCKS\oslab\exp2\ >
Enter the no of processes 3
Enter the burst time of each process
p0 23
p1 2
p2 3
process no      burst time      waiting time      turn around time
p0              2              0              2
p1              3              2              5
p2             23              5             28
Average waiting time7.000000
Average turn around time35.000000
Process returned 0 (0x0)   execution time : 11.107 s
Press any key to continue.

```

c)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 \t Time");
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 %-10d %-10d %-10d", i + 1, time,
time + j, p[i].burst - p[i].comp);
time += j;
}
}
}
printf("\n\n-----");
printf("\nProcess \t Waiting Time TurnAround Time ");
for (i = 0; i < n; i++)
{
printf("\nProcess # %-12d%-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);
return 0;
}

```

OUTPUT:


```
D:\CODEBLOCKS\oslab\exp2\ X + v
Enter The No Of Process :3
Enter The Time Quantum (in ms) :4
Enter The Burst Time (in ms) For Process # 1 :24
Enter The Burst Time (in ms) For Process # 2 :3
Enter The Burst Time (in ms) For Process # 3 :3

Order Of Execution

Process Starting Ending Remaining
Time Time Time
process # 1 0 4 20
process # 2 4 7 0
process # 3 7 10 0
process # 1 10 14 16
process # 1 14 18 12
process # 1 18 22 8
process # 1 22 26 4
process # 1 26 30 0

-----
Process Waiting Time TurnAround Time
Process # 1 6 30
Process # 2 4 7
Process # 3 7 10

Average
-----
Waiting Time: 5.666667ms
TurnAround Time : 15.666667ms

Process returned 0 (0x0) execution time : 13.876 s
Press any key to continue.
```

d)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 & 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];
}

```

```

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],
waitTime[i], turnaroundTime[i]);


printf("\nAverage Waiting Time is --- %f", (float)totWaitTime/n);

printf("\nAverage Turnaround Time is --- %f", (float)totTat/n);

}

```

OUTPUT:



```

D:\CODEBLOCKS\oslab\exp2\ x + v
Enter the number of processes --- 5
Enter the Burst Time & Priority of Process 0 --- 10
3
Enter the Burst Time & Priority of Process 1 --- 1
1
Enter the Burst Time & Priority of Process 2 --- 2
4
Enter the Burst Time & Priority of Process 3 --- 1
5
Enter the Burst Time & Priority of Process 4 --- 5
2
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
Process returned 0 (0x0)   execution time : 18.042 s
Press any key to continue.

```

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=3,x=0;

void signal(int *S)

{

    ++S;

    return;

}

```

```

void wait(int *S)
{
    --S;
    return;
}
void producer()
{
    wait(&S);
    ++F;
    --E;
    x++;
    printf("\nproducer produces item %d",x);
    signal(&S);
}
void consumer()
{
    wait(&S);
    --F;
    ++E;
    printf("\nconsumeer consumes item %d",x);
    x--;
    signal(&S);
}
int main()
{
    int n,i;
    printf("\n1.press 1 for producer\n2.press 2 for consumer\n3.press 3 for 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("buffer is full!!");
    }
    break;
case 2:
    if((S==1)&&(F!=0))
    {
        consumer();
    }
    else{
        printf("buffer is empty!!");
    }
    break;
case 3:
    exit(0);
    break;
}
}
}
```

OUTPUT:

```
D:\CODEBLOCKS\oslab\exp3.f  X + -
1.press 1 for producer
2.press 2 for consumer
3.press 3 for 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
buffer is full!!
enter your choice:2

consumer consumes item 3
enter your choice:2

consumer consumes item 2
enter your choice:2

consumer consumes item 1
enter your choice:2
buffer is empty!!
enter your choice:3

Process returned 0 (0x0)   execution time : 17.652 s
Press any key to continue.
```

4.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("\nEnter the no of resources");
    scanf("%d",&r);

    printf("\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 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("\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:\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)
    {
        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 safe state!!\n");
    printf("Safe Sequence:<");
    for(i=0;i<p;i++)
        printf("P%d,",safeSequence[i]);
    printf(">");
    printf("\n");
}
else
    printf("\nThe system is in an unsafe state!!");
}

```

OUTPUT:

```

D:\CODEBLOCKS\oslab\progr...
Enter the no. of processes: 5
Enter the no of resources3
Enter the Max Matrix for each process:
For process 1      7 5 3
For process 2      3 2 2
For process 3      9 0 2
For process 4      2 2 2
For process 5      4 3 3
Enter the allocation for each process:
For process 1:     0 1 0
For process 2:     2 0 0
For process 3:     3 0 2
For process 4:     2 1 1
For process 5:     0 0 2
Enter the Available Resources: 3 3 2
Max matrix:      Allocation matrix:
753              010
322              200
902              302
222              211
433              002
Process 2 runs to completion!

```

```
D:\CODEBLOCKS\oslab\progr. x + v
Max matrix:      Allocation matrix:
753              010
000              000
902              302
222              211
433              002

Process 4 runs to completion!
Max matrix:      Allocation matrix:
753              010
000              000
902              302
000              000
433              002

Process 1 runs to completion!
Max matrix:      Allocation matrix:
000              000
000              000
902              302
000              000
433              002

Process 3 runs to completion!
Max matrix:      Allocation matrix:
000              000
000              000
000              000
000              000
433              002

Process 5 runs to completion!
The system is in safe state!!
Safe Sequence:<P2,P4,P1,P3,P5,>

Process returned 0 (0x0)   execution time : 82.945 s
```

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

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

a)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;
```

```
    int n=4;
```

```
    int allocation[n];
```

```
    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("\nProcess No.\tProcess Size\tBlock 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:

```

D:\CODEBLOCKS\oslab\LAB-6 x + v
.....First fit.....
Process No.    Process Size    Block No.
1             212             2
2             417             5
3             112             2
4             426             Not Allocated

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
|

```

b)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;
    int n=4;
    int allocation[n];
    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\tBlock 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");

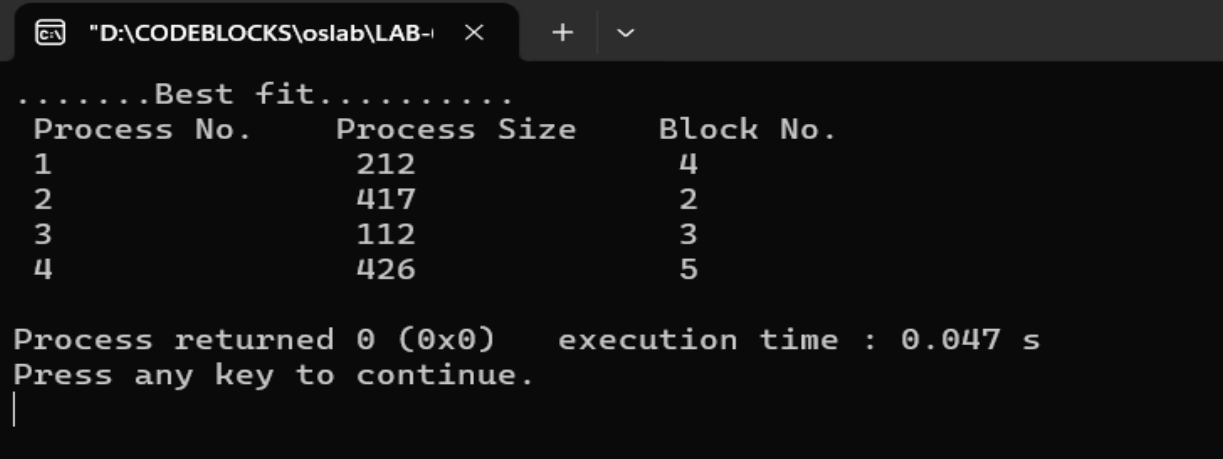
}

return 0;

}

```

OUTPUT:



```

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

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
|

```

c)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;
int n=4;
int allocation[n];
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\tBlock 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 allocation\n");
}
return 0;
}

```

OUTPUT:

```
"D:\CODEBLOCKS\oslab\LAB-1" X + v
.....First fit.....
Process No      Process Size      Block No
1               212                2
2               417                5
3               112                2
4               426                Not allocation

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

6. Develop a C program to simulate page replacement algorithms:

a) FIFO b) LRU

a) 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\n", 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---7a.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("\nNumber of page faults:%d\n",pf);
}

```

OUTPUT:


```
---7a.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
```

```
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

Process returned 25 (0x19)   execution time : 0.043 s
Press any key to continue.
|
```

b) LRU:

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

int fr[3];

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

```

}
}
void main(){
printf("---7b.LRU PAGE REPLACEMENT---");
int
index,k,l,pageFoundFrames=0,pageInserted=0,pageFaults=0,frameSize=3;
int pageRef[12]={1,2,3,4,1,2,5,1,2,3,4,5},i,j,pageStat[3];
for(i=0;i<frameSize;i++){
fr[i]=-1;
}
for(j=0;j<12;j++){
pageFoundFrames=0;
pageInserted=0;
for(i=0;i<3;i++){
if(fr[i]==pageRef[j]){
pageFoundFrames=1;
break;
}
}
if(!pageFoundFrames){
for(i=0;i<frameSize;i++){
if(fr[i]==-1){
fr[i]=pageRef[j];
pageInserted=1;
break;
}
}
if(!pageInserted){
for(i=0;i<frameSize;i++){
pageStat[i]=0;
}
for(k=j-1,l=1;l<=frameSize-1;l++,k--){
for(i=0;i<frameSize;i++){
if(fr[i]==pageRef[k]){

```

```

pageStat[i]=1;
}
}
}
for(i=0;i<frameSize;i++){
if(pageStat[i]==0){
index=i;
}
}
fr[index]=pageRef[j];
pageFaults++;
}
}
display();
}
printf("\nNumber of page faults : %d\n",pageFaults+frameSize);
}

```

OUTPUT:

```

---7b.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

Process returned 28 (0x1C)   execution time : 0.031 s
Press any key to continue.

```

7. Simulate following file organization techniques

a) Single level directory

b) Two level directory

a) 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 the directory--");
```

```
    scanf("%s",dir.dname);
```

```
    while(1)
```

```
    {
```

```
        printf("\n\n1.Create File\t2.Delete File\t3.Search File\t4.Display Files\t5.Exit\n Enter 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("\n Enter the name of the file--");  
  
scanf("%s",f);  
  
for(i=0;i<dir.fcnt;i++)  
{  
    if(strcmp(f,dir.fname[i])==0)  
    {  
        printf("file%s is deleted",f);  
  
        strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);  
  
        break;  
    }  
}  
  
if(i==dir.fcnt)  
    printf("File %s is not found",f);  
  
else  
    dir.fcnt--;  
  
break;
```

case 3:

```
printf("\n Enter the name of the file--");  
  
scanf("%s",f);  
  
for(i=0;i<dir.fcnt;i++)  
{  
    if(strcmp(f,dir.fname[i])==0)
```

```

        {
            printf("File %s id not 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:

```
D:\OS\8A.exe  X  +  v

The files are-- f1      f2

1.Create File    2.Delete File    3.Search File
4.Display Files  5.Exit
Enter your choice3

Enter the name of the file--f1
File f1 is found

1.Create File    2.Delete File    3.Search File
4.Display Files  5.Exit
Enter your choice2

Enter the name of the file--f1
filef1 is deleted

1.Create File    2.Delete File    3.Search File
4.Display Files  5.Exit
Enter your choice4

The files are-- f2

1.Create File    2.Delete File    3.Search File
4.Display Files  5.Exit
Enter your choice5

Process returned 0 (0x0)    execution time : 39.987 s
Press any key to continue.
|
```

b) Two level directory:

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

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

void main()
{
```

```

int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\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++;
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 ");
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:

```

D:\CODEBLOCKS\oslab\progri  X + v

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

Enter name of directory -- ff
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 -- aa
Directory aa 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 -- ff
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 --2

Enter name of the directory -- ff
Enter name of the file -- bb
File created

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

```

```

D:\CODEBLOCKS\oslab\progri  X + v

File created

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

Directory      Files
ff             aa      bb

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

Enter name of the directory -- ff
Enter name of the file -- aa
File aa is deleted

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

Directory      Files
ff             bb

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 -- ff
Enter the name of the file -- bb
File bb is found

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

```

8. Develop a C to simulate the linked file allocation strategies.

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

```

D:\CODEBLOCKS\oslab\progri
Enter the no. of blocks that are already allocated:10
Enter the index of blocks that are already allocated:1
3
5
7
9
11
13
15
17
19
Enter the index of starting block and its 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)|

```

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

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

int main()
{
    int queue[20], head, n, i, j, seekTime=0, direction, maxTrack;

    printf("enter the number of disk requests:");

    scanf("%d", &n);

    printf("enter the disk request queue:\n");

    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);
        seekTime+=abs(currentTrack-i);
        currentTrack=i;
    }
    printf("0");
    seekTime+=currentTrack;

    for(i=1;i<=maxTrack;i++)
    {
        printf("%d\t",i);
        seekTime+=abs(currentTrack-i);
        currentTrack=i;
    }
}
else
{
    for(i=head;i<=maxTrack;i++)
    {
        printf("%d\t",i);
        seekTime+=abs(currentTrack-i);
        currentTrack=i;
    }
    printf("%d",maxTrack);
    seekTime+=abs(currentTrack-maxTrack);
    for(i=maxTrack-1;i>=0;i--)
    {
        printf("%d",i);
    }
}

```

```

        seekTime+=abs(currentTrack-i);

        currentTrack=i;
    }
}

printf("\n\nTotal track movements:%d\n",seekTime);

getch();
}

```

OUTPUT:

```

D:\CODEBLOCKS\oslab\progr
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):0

Seek Sequence: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        01        2        3        4        5        6        7
8       9      10      11      12      13      14      15      16      17      18      19      20      21      22      23      24      25      26      27
28      29      30      31      32      33      34      35      36      37      38      39      40      41      42      43      44      45      46      47
48      49      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

Total track movements:249
|

```