

Operating System

**LAB-13**

**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Roll No \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­\_**

**Marks Obtained \_\_\_\_\_\_\_\_\_\_\_\_**

**Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Deadlock Management Techniques

**Objective:**

Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

Write a C program to simulate disk scheduling algorithms

* FCFS
* SCAN
* C-SCAN

**TASK1:**

**DESCRIPTION**

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Banker’s algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

**PROGRAM**

#include<stdio.h> struct file

{

int all[10]; int max[10]; int need[10]; int flag;

};

void main()

{

struct file f[10]; int fl;

int i, j, k, p, b, n, r, g, cnt=0, id, newr; int avail[10],seq[10];

clrscr();

printf("Enter number of processes -- "); scanf("%d",&n);

printf("Enter number of resources -- "); scanf("%d",&r);

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

{

printf("Enter details for P%d",i); printf("\nEnter allocation\t -- \t"); for(j=0;j<r;j++)

scanf("%d",&f[i].all[j]); printf("Enter Max\t\t -- \t"); for(j=0;j<r;j++)

scanf("%d",&f[i].max[j]);

f[i].flag=0;

}

printf("\nEnter Available Resources\t -- \t"); for(i=0;i<r;i++)

scanf("%d",&avail[i]);

printf("\nEnter New Request Details -- "); printf("\nEnter pid \t -- \t"); scanf("%d",&id);

printf("Enter Request for Resources \t -- \t"); for(i=0;i<r;i++)

{

scanf("%d",&newr); f[id].all[i] += newr;

avail[i]=avail[i] - newr;

}

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

{

for(j=0;j<r;j++)

{

f[i].need[j]=f[i].max[j]-f[i].all[j];if(f[i].need[j]<0)

f[i].need[j]=0;

}

}

cnt=0;

fl=0;

while(cnt!=n)

{

g=0;

for(j=0;j<n;j++)

{

if(f[j].flag==0)

{

b=0;

for(p=0;p<r;p++)

{

if(avail[p]>=f[j].need[p])

b=b+1;

else

b=b-1;

}

if(b==r)

{

printf("\nP%d is visited",j); seq[fl++]=j;

f[j].flag=1;

for(k=0;k<r;k++)

avail[k]=avail[k]+f[j].all[k];

cnt=cnt+1;

printf("(");

for(k=0;k<r;k++)

printf("%3d",avail[k]);

printf(")");

g=1;

}

}

}

if(g==0)

{

printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED"); printf("\n SYSTEM IS IN UNSAFE STATE");

goto y;

}

}

printf("\nSYSTEM IS IN SAFE STATE"); printf("\nThe Safe Sequence is -- ("); for(i=0;i<fl;i++)

printf("P%d ",seq[i]); printf(")");

y:printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");

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

{

printf("P%d\t",i);

for(j=0;j<r;j++)

printf("%6d",f[i].all[j]);

for(j=0;j<r;j++)

printf("%6d",f[i].max[j]);

for(j=0;j<r;j++)

printf("%6d",f[i].need[j]);

printf("\n");

}

getch();

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| } |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***INPUT*** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Enter number of processes | | | | | |  | – | 5 |  |  |  |  |  |
| Enter number of resources | | | | | |  | -- | 3 |  |  |  |  |  |
| Enter details for P0 | | | | |  |  |  |  |  |  |  |  |  |
| Enter allocation | |  |  |  | -- |  | 0 | 1 | | 0 |  |  |  |
| Enter Max |  |  |  |  | -- |  |  | 7 |  | 5 | 3 |  |  |
| Enter details for P1 | | | | |  |  |  |  |  |  |  |  |  |
| Enter allocation | |  |  |  | -- |  | 2 | 0 |  | 0 |  |  |  |
| Enter Max |  |  |  |  | -- |  | 3 | 2 |  | 2 |  |  |  |
| Enter details for P2 | | | | |  |  |  |  |  |  |  |  |  |
| Enter allocation | |  |  |  | -- |  | 3 | 0 |  | 2 |  |  |  |
| Enter Max |  |  |  |  | -- |  | 9 | 0 |  | 2 |  |  |  |
| Enter details for P3 | | | | |  |  |  |  |  |  |  |  |  |
| Enter allocation | |  |  |  | -- |  | 2 | 1 |  | 1 |  |  |  |
| Enter Max |  |  |  |  | -- |  | 2 | 2 |  | 2 |  |  |  |
| Enter details for P4 | | | | |  |  |  |  |  |  |  |  |  |
| Enter allocation | |  |  |  | -- |  | 0 | 0 |  | 2 |  |  |  |
| Enter Max |  |  |  |  | -- |  | 4 | 3 |  | 3 |  |  |  |
| Enter Available Resources -- | | | | | | 3 | 3 | 2 |  |  |  |  |  |
| Enter New Request Details -- | | | | | |  |  |  |  |  |  |  |  |
| Enter pid | -- |  | 1 | |  |  |  |  |  |  |  |  |  |
| Enter Request for Resources | | | | | | -- | 1 |  | 0 | 2 |  |  |  |
| ***OUTPUT*** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| P1 is visited( | 5 | 3 | 2) | |  |  |  |  |  |  |  |  |  |
| P3 is visited( | 7 | 4 | 3) | |  |  |  |  |  |  |  |  |  |
| P4 is visited( | 7 | 4 | 5) | |  |  |  |  |  |  |  |  |  |
| P0 is visited( | 7 | 5 | 5) | |  |  |  |  |  |  |  |  |  |
| P2 is visited( 10 | | 5 | | 7) |  |  |  |  |  |  |  |  |  |
| SYSTEM IS IN SAFE STATE | | | | |  |  |  |  |  |  |  |  |  |
| The Safe Sequence is -- (P1 P3 P4 P0 P2 ) | | | | | | | |  |  |  |  |  |  |
| Process |  |  | Allocation | | |  |  |  | Max |  | Need | |  |
| P0 |  |  | 0 | 1 | 0 |  |  | 7 | 5 | 3 | 7 | 4 | 3 |
| P1 |  |  | 3 | 0 | 2 |  |  | 3 | 2 | 2 | 0 | 2 | 0 |
| P2 |  |  | 3 | 0 | 2 |  |  | 9 | 0 | 2 | 6 | 0 | 0 |
| P3 |  |  | 2 | 1 | 1 |  |  | 2 | 2 | 2 | 0 | 1 | 1 |
| P4 |  |  | 0 | 0 | 2 |  |  | 4 | 3 | 3 | 4 | 3 | 1 |

**TASK2:**

**DESCRIPTION**

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip

**PROGRAM**

**FCFS DISK SCHEDULING ALGORITHM**

#include<stdio.h>

main()

{

int t[20], n, I, j, tohm[20], tot=0; float avhm;

clrscr();

printf(“enter the no.of tracks”); scanf(“%d”,&n);

printf(“enter the tracks to be traversed”); for(i=2;i<n+2;i++)

scanf(“%d”,&t\*i+);

for(i=1;i<n+1;i++)

{

tohm[i]=t[i+1]-t[i];if(tohm[i]<0) tohm[i]=tohm[i]\*(-1);

}

for(i=1;i<n+1;i++)

tot+=tohm[i];

avhm=(float)tot/n;

printf(“Tracks traversed\tDifference between tracks\n”); for(i=1;i<n+1;i++)

printf(“%d\t\t\t%d\n”,t\*i+,tohm\*i+); printf("\nAverage header movements:%f",avhm); getch();

}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***INPUT*** | | |  |  |  |  |  |  |  |  |  |
| Enter no.of tracks:9 | | |  |  |  |  |  |  |  |  |  |
| Enter track position:55 | | | | 58 | 60 | 70 | 18 | 90 | 150 | 160 | 184 |
| ***OUTPUT*** | | |  |  |  |  |  |  |  |  |  |
| Tracks traversed | | | Difference between tracks | | | |  |  |  |  |  |
| 55 | | |  |  | 45 |  |  |  |  |  |  |
| 58 | | |  |  | 3 |  |  |  |  |  |  |
| 60 | | |  |  | 2 |  |  |  |  |  |  |
| 70 | | |  |  | 10 |  |  |  |  |  |  |
| 18 | | |  |  | 52 |  |  |  |  |  |  |
| 90 | | |  |  | 72 |  |  |  |  |  |  |
|  | Average header movements:30.888889 | | | |
|  |  | | | |
|  |  | | | |
|  |  | | | |
|  |  | | | |

**SCAN DISK SCHEDULING ALGORITHM**

#include<stdio.h>

main()

{

int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;

clrscr();

printf("enter the no of tracks to be traveresed"); scanf("%d'",&n);

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

t[0]=0;t[1]=h; printf("enter the tracks"); for(i=2;i<n+2;i++)

scanf("%d",&t[i]);

for(i=0;i<n+2;i++)

{

for(j=0;j<(n+2)-i-1;j++)

{if(t[j]>t[j+1])

{

temp=t[j];

t[j]=t[j+1];

t[j+1]=temp;

} } }

for(i=0;i<n+2;i++)

if(t[i]==h)

j=i;k=i;

p=0;

while(t[j]!=0)

{

atr[p]=t[j]; j--;

p++;

}

atr[p]=t[j];

for(p=k+1;p<n+2;p++,k++)

atr[p]=t[k+1];

for(j=0;j<n+1;j++)

{

if(atr[j]>atr[j+1]) d[j]=atr[j]-atr[j+1];

else

d[j]=atr[j+1]-atr[j];sum+=d[j];

}

printf("\nAverage header movements:%f",(float)sum/n); getch();

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| } |  |  |  |  |  |  |  |  |  |
| ***INPUT*** |  |  |  |  |  |  |  |  |  |
| Enter no.of tracks:9 |  |  |  |  |  |  |  |  |  |
| Enter track position:55 | | 58 | 60 | 70 | 18 | 90 | 150 | 160 | 184 |
|  |  |  |  |  |  |  |  |  |  |
|  |  | | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

**C-SCAN DISK SCHEDULING ALGORITHM**

#include<stdio.h>

main()

{

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

printf("enter the no of tracks to be traveresed"); scanf("%d'",&n);

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

t[0]=0;t[1]=h; printf("enter total tracks"); scanf("%d",&tot); t[2]=tot-1;

printf("enter the tracks"); for(i=3;i<=n+2;i++)

scanf("%d",&t[i]);

for(i=0;i<=n+2;i++)

for(j=0;j<=(n+2)-i-1;j++)if(t[j]>t[j+1])

{

temp=t[j];

t[j]=t[j+1];

t[j+1]=temp;

}

for(i=0;i<=n+2;i++)

if(t[i]==h)

j=i;break;

p=0; while(t[j]!=tot-1)

{

atr[p]=t[j];

j++;

p++;

}

atr[p]=t[j];

p++;

i=0;

while(p!=(n+3) && t[i]!=t[h])

{

atr[p]=t[i];

i++;

p++;

}

for(j=0;j<n+2;j++)

{

if(atr[j]>atr[j+1]) d[j]=atr[j]-atr[j+1];

else

d[j]=atr[j+1]-atr[j];sum+=d[j];

}

printf("total header movements%d",sum); printf("avg is %f",(float)sum/n);

getch();

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| } |  |  |  |  |  |  |  |  |
| ***INPUT*** |  |  |  |  |  |  |  |  |
| Enter the track position : 55 | 58 | 60 | 70 | 18 | 90 | 150 | 160 | 184 |
| Enter starting position : 100 |  |  |  |  |  |  |  |  |