Ex No: 8 BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE

AIM

To implement deadlock avoidance by using Banker’s Algorithm.

ALGORITHM

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.
9. Stop.

#include <stdio.h>

#include <stdlib.h>

void final\_output(int k[][10], int n, int p)

{

int i, j;

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

{

printf("\n");

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

{

printf("%d\t", k[i][j]);

}

}

}

//Banker's Algorithm

void Banker(int A[][10], int N[][10],

int M[10][10], int W[1][10], int \*n, int \*m)

{

int i, j;

printf("\n Enter total number of processes : ");

scanf("%d", n);

printf("\n Enter total number of resources : ");

scanf("%d", m);

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

{

printf("\n Process %d\n", i + 1);

for (j = 0; j < \*m; j++)

{

printf(" Allocation for resource %d : ", j + 1);

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

printf(" Maximum for resource %d : ", j + 1);

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

}

}

printf("\n Available resources : \n");

for (i = 0; i < \*m; i++)

{

printf(" Resource %d : ", i + 1);

scanf("%d", &W[0][i]);

}

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

for (j = 0; j < \*m; j++)

N[i][j] = M[i][j] - A[i][j];

printf("\n \*\*\*\*\*\*\*\*Allocation Matrix\*\*\*\*\*\*\*\*\*\*");

final\_output(A, \*n, \*m);

printf("\n \*\*\*\*\*\*\*\*Maximum Requirement Matrix"

"\*\*\*\*\*\*\*\*\*");

final\_output(M, \*n, \*m);

printf("\n \*\*\*\*\*\*\*\*\*\*\*Need Matrix\*\*\*\*\*\*\*\*\*");

final\_output(N, \*n, \*m);

}

//Safety algorithm

int safety(int A[][10], int N[][10],

int B[1][10], int n, int m, int a[])

{

int i, j, k, x = 0, f1 = 0, f2 = 0;

int F[10], W[1][10];

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

F[i] = 0;

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

W[0][i] = B[0][i];

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

{

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

{

if (F[i] == 0)

{

f2 = 0;

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

{

if (N[i][j] > W[0][j])

f2 = 1;

}

if (f2 == 0 && F[i] == 0)

{

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

W[0][j] += A[i][j];

F[i] = 1;

f1++;

a[x++] = i;

}

}

}

if (f1 == n)

return 1;

}

return 0;

}

//Resource Request algorithm

void request(int A[10][10], int N[10][10], int B[10][10], int pid, int K)

{

int rmat[1][10];

int i;

printf("\n Enter additional request : \n");

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

{

printf(" Request for resource %d : ", i + 1);

scanf("%d", &rmat[0][i]);

}

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

if (rmat[0][i] > N[pid][i])

{

printf("\n \*\*\*\*\*\*Error encountered\*\*\*\*\*\*\n");

exit(0);

}

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

if (rmat[0][i] > B[0][i])

{

printf("\n \*\*\*\*\*\*Resources unavailable\*\*\*\*\*\n");

exit(0);

}

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

{

B[0][i] -= rmat[0][i];

A[pid][i] += rmat[0][i];

N[pid][i] -= rmat[0][i];

}

}

int banker(int A[][10], int N[][10],

int W[1][10], int n, int m)

{

int j, i, a[10];

j = safety(A, N, W, n, m, a);

if (j != 0)

{

printf("\n\n");

printf("\n A safety sequence has been ""detected.\n");

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

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

printf("\n");

return 1;

}

else

{

printf("\n Deadlock has occured.\n");

return 0;

}

}

int main()

{

int All[10][10], Max[10][10], Need[10][10], W[1][10];

int n, m, pid, c, r;

printf("\n \*\*\*\*\*\*\*\*\*DEADLOCK AVOIDANCE USING"

"BANKER'S ALGORITHM\*\*\*\*\*\*\*\*\*\*\*\n");

Banker(All, Need, Max, W, &n, &m);

r = banker(All, Need, W, n, m);

if (r != 0)

{

printf("\n Do you want make an additional"

"request for any of the process ? (1=Yes|0=No)");

scanf("%d", &c);

if (c == 1)

{

printf("\n Enter process number : ");

scanf("%d", &pid);

request(All, Need, W, pid - 1, m);

r = banker(All, Need, W, n, m);

if (r == 0)

{

exit(0);

}

}

}

else

exit(0);

return 0;

}

Result

Thus bankers algorithm for dead lock avoidance was executed successfully.

**EXNO 9A) MEMORY MANAGEMENT USING FIRST FIT**

**AIM**

**ALGORITHM**

**First Fit Algorithm**

1. **Read the number of processes and number of the block from the user**
2. **Read the size of each block and the size of all the process requests.**
3. **Start allocating the processes**
4. **Display the results as shown below**
5. **Stop**

**#include <stdio.h>**

**int main()**

**{**

**int a[10], b[10], a1, b1, flags[10], all[10];**

**int i, j;**

**printf("\n\t\t\tMemory Management"**

**" Scheme -"**

**" First Fit\n");**

**for (i = 0; i < 10; i++)**

**{**

**flags[i] = 0;**

**all[i] = -1;**

**}**

**printf("Enter number of blocks: ");**

**scanf("%d", &a1);**

**printf("\nEnter the size of each"**

**" block:\n ");**

**for (i = 0; i < a1; i++)**

**{**

**printf("Block no.%d: ", i);**

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

**}**

**printf("\nEnter no. of "**

**"processes: ");**

**scanf("%d", &b1);**

**printf("\nEnter size of each"**

**" process:\n ");**

**for (i = 0; i < b1; i++)**

**{**

**printf("Process no.%d: ", i);**

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

**}**

**for (i = 0; i < b1; i++)**

**for (j = 0; j < a1; j++)**

**if (flags[j] == 0 && a[j] >= b[i])**

**{**

**all[j] = i;**

**flags[j] = 1;**

**break;**

**}**

**printf("\nBlock no.\tsize\t\t"**

**"process no.\t\tsize");**

**for (i = 0; i < a1; i++)**

**{**

**printf("\n%d\t\t%d\t\t",**

**i + 1, a[i]);**

**if (flags[i] == 1)**

**{**

**printf("%d\t\t\t%d", all[i]**

**+ 1, b[all[i]]);**

**}**

**else**

**printf("Not allocated");**

**}**

**printf("\n");**

**}**

**RESULT:**

**EXNO 9B) MEMORY MANAGEMENT USING BEST FIT**

**AIM**

**ALGORITHM**

**Algorithm**

1. Read the number of processes and number of blocks from the user
2. Get the size of each block and process requests
3. Then select the best memory block
4. Display the result as shown below
5. The fragmentation column will keep track of wasted memory
6. Stop

**#include <stdio.h>**

**int main()**

**{**

**int a[20], b[20], c[20], b1, c1;**

**int i, j, temp;**

**static int barr[20], carr[20];**

**printf("\n\t\t\tMemory Management"**

**" Scheme - Best Fit");**

**printf("\nEnter the number of "**

**"blocks:");**

**scanf("%d", &b1);**

**printf("Enter the number of"**

**" processes:");**

**scanf("%d", &c1);**

**int lowest = 9999;**

**printf("\nEnter the size of the"**

**" blocks:\n");**

**for (i = 1; i <= b1; i++)**

**{**

**printf("Block no.%d:", i);**

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

**}**

**printf("\nEnter the size of"**

**" the processes :\n");**

**for (i = 1; i <= c1; i++)**

**{**

**printf("Process no.%d:", i);**

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

**}**

**for (i = 1; i <= c1; i++)**

**{**

**for (j = 1; j <= b1; j++)**

**{**

**if (barr[j] != 1)**

**{**

**temp = b[j] - c[i];**

**if (temp >= 0)**

**if (lowest > temp)**

**{**

**carr[i] = j;**

**lowest = temp;**

**}**

**}**

**}**

**a[i] = lowest;**

**barr[carr[i]] = 1;**

**lowest = 10000;**

**}**

**printf("\nProcess\_no\tProcess"**

**"\_size\tBlock\_no\t"**

**"Block\_size\tFragment");**

**for (i = 1; i <= c1 && carr[i] != 0; i++)**

**{**

**printf("\n%d\t\t%d\t\t%d\t\t"**

**"%d\t\t%d", i,**

**c[i], carr[i], b[carr[i]], a[i]);**

**}**

**printf("\n");**

**}**

**RESULT**

**EXNO 10A) PAGE REPLACEMENT ALGORITHM USING FIFO**

**AIM**

To write a c program to implement FIFO page replacement algorithm

**ALGORITHM**

1. Start the process

2. Declare the size with respect to page length

3. Check the need of replacement from the page to memory

4. Check the need of replacement from old page to new page in memory

5. Forma queue to hold all pages

6. Insert the page require memory into the queue

7. Check for bad replacement and page fault

8. Get the number of processes to be inserted

9. Display the values

10. Stop the process

#include<stdio.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

            printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

            printf("\n ENTER THE PAGE NUMBER :\n");

            for(i=1;i<=n;i++)

            scanf("%d",&a[i]);

            printf("\n ENTER THE NUMBER OF FRAMES :");

            scanf("%d",&no);

for(i=0;i<no;i++)

            frame[i]= -1;

                        j=0;

                        printf("\tref string\t page frames\n");

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

                        {

                                    printf("%d\t\t",a[i]);

                                    avail=0;

                                    for(k=0;k<no;k++)

if(frame[k]==a[i])

                                                avail=1;

                                    if (avail==0)

                                    {

                                                frame[j]=a[i];

                                                j=(j+1)%no;

                                                count++;

                                                for(k=0;k<no;k++)

                                                printf("%d\t",frame[k]);

}

                                    printf("\n");

}

                        printf("Page Fault Is %d",count);

                        return 0;

}

**RESULT**

**EXNO 10B) PAGE REPLACEMENT ALGORITHM USING LRU**

**AIM:**

To write a c program to implement LRU page replacement algorithm

**ALGORITHM :**

1. Start the process

2. Declare the size

3. Get the number of pages to be inserted

4. Get the value

5. Declare counter and stack

6. Select the least recently used page by counter value

7. Stack them according the selection.

8.  Display the values

9. Stop the process

**PROGRAM:**

#include<stdio.h>

main()

{

int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];

printf("Enter no of pages:");

scanf("%d",&n);

printf("Enter the reference string:");

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

            scanf("%d",&p[i]);

printf("Enter no of frames:");

scanf("%d",&f);

q[k]=p[k];

printf("\n\t%d\n",q[k]);

c++;

k++;

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

            {

                        c1=0;

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

                        {

                                    if(p[i]!=q[j])

                                    c1++;

                        }

                        if(c1==f)

                        {

                                    c++;

                                    if(k<f)

                                    {

                                                q[k]=p[i];

                                                k++;

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

                                                printf("\t%d",q[j]);

                                                printf("\n");

                                    }

                                    else

                                    {

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

                                                {

                                                            c2[r]=0;

                                                            for(j=i-1;j<n;j--)

                                                            {

                                                            if(q[r]!=p[j])

                                                            c2[r]++;

                                                            else

                                                            break;

                                                }

                                    }

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

                                     b[r]=c2[r];

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

                                    {

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

                                                {

                                                            if(b[r]<b[j])

                                                            {

                                                                        t=b[r];

                                                                        b[r]=b[j];

                                                                        b[j]=t;

                                                            }

                                                }

                                    }

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

                                    {

                                                if(c2[r]==b[0])

                                                q[r]=p[i];

                                                printf("\t%d",q[r]);

                                    }

                                    printf("\n");

                        }

            }

}

printf("\nThe no of page faults is %d",c);

}

**RESULT**

**EX NO 11A) DISK SCHEDULING USING FCFS**

AIM

**ALGORITHM:**  
1. Input the maximum number of cylinders and work queue and its head starting position.  
2. First Come First Serve Scheduling (FCFS) algorithm – The operations are performed in order requested.  
3. There is no reordering of work queue.  
4. Every request is serviced, so there is no starvation.  
5. The seek time is calculated.  
6. Shortest Seek Time First Scheduling (SSTF) algorithm – This algorithm selects the request with the minimum seek time from the current head position.  
7. Since seek time increases with the number of cylinders traversed by the head, SSTF chooses the pending request closest to the current head position.  
8. The seek time is calculated.  
9. SCAN Scheduling algorithm – The disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk.  
10. At the other end, the direction of head movement is reversed, and servicing continues.  
11. The head continuously scans back and forth across the disk.  
12. The seek time is calculated.  
13. Display the seek time and terminate the program

#include<conio.h>

#include<stdio.h>

int main()

{

int i,j,sum=0,n;

int ar[20],tm[20];

int disk

clrscr();

printf("enter number of location\t");

scanf("%d",&n);

printf("enter position of head\t");

scanf("%d",&disk);

printf("enter elements of disk queue\n");

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

{

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

tm[i]=disk-ar[i];

if(tm[i]<0)

{

tm[i]=ar[i]-disk;

}

disk=ar[i];

sum=sum+tm[i];

}

printf("\nmovement of total cylinders %d",sum);

getch();

return 0;

}

RESULT

**EX NO 11B) DISK SCHEDULING USING SCAN**

**AIM**

**Algorithm-**

**1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.**

**2. Let direction represents whether the head is moving towards left or right.**

**3. In the direction in which head is moving service all tracks one by one.**

**4. Calculate the absolute distance of the track from the head.**

**5. Increment the total seek count with this distance.**

**6. Currently serviced track position now becomes the new head position.**

**7. Go to step 3 until we reach at one of the ends of the disk.**

**8. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

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

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

printf("Enter initial head position\n");

scanf("%d",&initial);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");

scanf("%d",&move);

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

{

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

{

if(RQ[j]>RQ[j+1])

{

int temp;

temp=RQ[j];

RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

int index;

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

{

if(initial<RQ[i])

{

index=i;

break;

}

}

if(move==1)

{

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

// last movement for max size

TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);

initial = size-1;

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

}

else

{

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);

initial =0;

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

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

}

printf("Total head movement is %d",TotalHeadMoment);

return 0;

}

RESULT: