
EX:NO:7 Bankers Algorithm for Deadlock avoidance

Aim :

To implement and study Bankers Algorithm for Deadlock Avoidance Problem

Program :

Banker's Algorithm

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

```
int m, n, i, j, al[10][10], max[10][10], av[10], need[10][10], temp, z, y, p, k;  
void main() {
```

```
    printf("\n Enter no of processes : ");  
    scanf("%d", &m); // enter numbers of processes
```

```
    printf("\n Enter no of resources : ");  
    scanf("%d", &n); // enter numbers of resources
```

```
    for (i = 0; i < m; i++) {  
        for (j = 0; j < n; j++) {  
            printf("\n Enter instances for al[%d][%d] = ", i,j); // al[][] matrix is for allocated  
            instances scanf("%d", &al[i][j]);  
            al[i][j]=temp;
```

```
        }  
    }
```

```
    for (i = 0; i < m;  
        i++) {  
        for (j = 0; j < n; j++) {printf("\n Enter instances for max[%d][%d] = ", i,j); // max[][]  
            matrix is for max instances  
            scanf("%d", &max[i][j]);  
        }  
    }
```

```
    for (i = 0; i < n; i++) {  
        printf("\n Available Resource for av[%d] = ",i); // av[] matrix is for available instances  
        scanf("%d", &av[i]);  
    }
```

```

        // Print allocation values
printf("Allocation Values :\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {printf("\t %d", al[i][j]); // printing allocation matrix
}
printf("\n");
}

printf("\n\n");

// Print max values
printf("Max Values :\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
printf("\t %d", max[i][j]); // printing max matrix
}
printf("\n");
}

printf("\n\n");

// Print need values
printf("Need Values :\n");
for (i = 0; i < m; i++) {
for (j = 0; j < n; j++) {
need[i][j] = max[i][j] - al[i][j]; // calculating need matrix
printf("\t %d", need[i][j]); // printing need matrix
}
printf("\n");
}

p = 1; // used for terminating while loop
y = 0;
while (p != 0) {
for (i = 0; i < m; i++) {
z = 0;
for (j = 0; j < n; j++) {
if (need[i][j] <= av[j] &&
(need[i][0] != -1)) { // comparing need with available instance and
// checking if the process is done
// or not
z++; // counter if condition TRUE
}
}
if (z == n) { // if need <= available TRUE for all resources then condition
// is TRUE
for (k = 0; k < n; k++) {
av[k] += al[i][k]; // new work = work + allocated
}

```

```

printf("\n SS process %d", i); // Print the Process
need[i][0] = -1;           // assign -1 if Process done
y++;                       // cont if process done
}

} // end for loop

if (y == m) { // if all done then
p = 0;       // exit while loop
} // end while printf("\n");
}

```

Output :

```

zayedhaque@fedora:~$ ./bankero
Enter no of processes : 5
Enter no of resources : 3
Enter instances for al[0][0] = 1
Enter instances for al[0][1] = 2
Enter instances for al[0][2] = 3
Enter instances for al[1][0] = 4
Enter instances for al[1][1] = 5
Enter instances for al[1][2] = 6
Enter instances for al[2][0] = 7
Enter instances for al[2][1] = 8
Enter instances for al[2][2] = 9
Enter instances for al[3][0] = 1
Enter instances for al[3][1] = 2
Enter instances for al[3][2] = 3
Enter instances for al[4][0] = 4
Enter instances for al[4][1] = 5
Enter instances for al[4][2] = 6
Enter instances for max[0][0] = 7
Enter instances for max[0][1] = 8
Enter instances for max[0][2] = 9
Enter instances for max[1][0] = 1
Enter instances for max[1][1] = 2
Enter instances for max[1][2] = 3
Enter instances for max[2][0] = 4
Enter instances for max[2][1] = 5
Enter instances for max[2][2] = 6
Enter instances for max[3][0] = 7
Enter instances for max[3][1] = 8

```

```

Enter instances for max[1][2] = 3
Enter instances for max[2][0] = 4

```

```

Enter instances for max[1][1] = 2
Enter instances for max[1][2] = 3
Enter instances for max[2][0] = 4
Enter instances for max[2][1] = 5
Enter instances for max[2][2] = 6
Enter instances for max[3][0] = 78
Enter instances for max[3][1] = 8

```

```

zayedhaque@fedora:~
Enter instances for max[1][2] = 3
Enter instances for max[2][0] = 4
Enter instances for max[2][1] = 5
Enter instances for max[2][2] = 6
Enter instances for max[3][0] = 78
Enter instances for max[3][1] = 8
Enter instances for max[3][2] = 9
Enter instances for max[4][0] = 1
Enter instances for max[4][1] = 2
Enter instances for max[4][2] = 3
Available Resource for av[0] = 4
Available Resource for av[1] = 3
Available Resource for av[2] = 4
Allocation Values :
    1    2    3
    4    5    6
    7    8    9
    1    2    3
    4    5    6

Max Values :
    7    8    9
    1    2    3
    4    5    6
    78   8    9
    1    2    3

Need Values :
    6    6    6
   -3   -3   -3
   -3   -3   -3
   77    6    6
   -3   -3   -3

SS process 1
SS process 2
SS process 4

^C
[zayedhaque@fedora ~]$

```

Result :

Successfully implemented the concepts of Banker's Algorithm.

EX:NO:10 FIFO Page Replacement Algorithm

Aim :

To implement FIFO page replacement algorithm

Program :

FIFO Page Replacement Algorithm

```
#include <stdio.h>
int a[100],b[100],i,n,z,f,j,pf,h,temp;
void main(){
    printf("\nEnter the no. of pages : ");          // no. of page
    referencing scanf("%d",&n);
    printf("\nEnter the size of frame : ");        // no. of page
    frames scanf("%d",&f);
    printf("\nEnter the pages value :\n");          // values of page
    referencing for(i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    for(i=0;i<f;i++)                                // assign values of
    page frames 1 innitally
    b[i]=-1;
    }
    i=0;j=0;h=0; // i , j used for loop, h for hit count all initialized to 0
    while(i<n){
        if(b[i]==-1 && i<f){ // when frames are empty so for starting
            enqueue b[i]=a[i];
            pf++;                // page fault counter
        }
        else
        {
            z=0
            ;
            for(j=0;j<f;j++){
                if(b[j]==a[i]){ // to check if value already
                    present h++; // hit counter
                }
                else{
                    z++; // if not hit count increment
                }
            }
            if(z==f){ // if no value
                matched pf++; // page
                fault counter for(j=0;j<f-1;j++){ // shifting
                    values
                    temp=b[j];
                    b[j]=b[j+1]
                    ;
                    b[j+1]=tem
                    p;
                }
            }
        }
        i++;
    }
}
```

values

```
        temp=b[j];
        b[j]=b[j+1]
        ;
        b[j+1]=tem
        p;
    }
    b[f-1]=a[i];          // insert new values
}

} // end else
printf("\n Current Frame:  %d \t %d \t %d \n",b[2],b[1],b[0]);    // frames value for
every iteration
i++;
} //end while
printf("\n frame at the end :");
for(i=0;i<f;i++){
    printf("\n b[%d] = %d",i,b[i]);          // frame values at the end
}
printf("\n Page Fault = %d ",pf);          // no. of page
faults printf("\n Hit = %d ",h);          //
no. of hitS
printf("\n");

}
```

Output :



```
sayed@sayed-virtual-machine: ~/Desktop/OS/EXP 10
Page Fault = 5
Hit = 9
sayed@sayed-virtual-machine: ~/Desktop/OS/EXP 10 $ ./or1.c
Enter the no. of pages : 15
Enter the size of frame : 3
```

Output :

```
xyy@xyy-virtual-machine: ~/Desktop/OS/EXP 10
Page Fault = 9
Hit = 8
xyy@xyy-virtual-machine: ~/Desktop/OS/EXP 10$ ./fifo.c
Enter the no. of pages : 15
Enter the size of frame : 3
Enter the pages value :
3
8
2
5
9
1
4
3
8
6
8
6
10
1
5

Current Frame: -1    -1    3
Current Frame: -1    8    3
Current Frame: 2    8    3
Current Frame: 2    8    3
Current Frame: 2    8    4
Current Frame: 3    2    8
Current Frame: 9    3    2
Current Frame: 9    3    2
Current Frame: 1    9    3
Current Frame: 1    9    3
Current Frame: 3    9    3
Current Frame: 3    9    3
Current Frame: 6    1    9
Current Frame: 6    1    3
Current Frame: 6    6    1

Frame at the end :
h[0] = 1
h[1] = 8
h[2] = 8
h[3] = 9
h[4] = 9
Page Fault = 10
Hit = 8
```

Result :

Successfully implemented page replacement using FIFO algorithm