Experiment No.7
Process Management: Deadlock A. Write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm
Name Of Student:-Bhagyashri Kaleni Sutar
Roll No:-75
Date of Performance:
Date of Submission:
Marks:
Sign:

Experiment No. 7

Aim: Process Management: Deadlock

Objective:

Write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm

Theory:

It is a banker algorithm used to avoid deadlock and allocate resources safely to each process in the computer system. The 'S-State' examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. In this section, we will learn the Banker's Algorithm in detail. Also, we will solve problems based on the Banker's Algorithm. To understand the Banker's Algorithm first we will see a real word example of it.

Suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

Similarly, it works in an operating system. When a new process is created in a computer system, the process must provide all types of information to the operating system like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

Data Structures for the Banker's Algorithm.

Let n = number of processes, and m = number of resources types.
Available: Vector of length m. If available [j] = k, there are k instances of resource type Rj available
Max: n x m matrix.

If Max [i,j] = k, then process Pi may request at most k instances of resource type Rj
Allocation: n x m matrix.
If Allocation[i,j] = k then Pi is currently allocated k instances of Rj
Need: n x m matrix.
If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task
Need [i,j] = Max[i,j] – Allocation [i,j]

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively.
   Initialize:
   Work = Available
   Finish [i] = false for i = 0, 1, …, n- 1
2. Find an i such that both:
   (a) Finish [i] = false
   (b) Needi ≤ Work
   If no such i exists, go to step 4
3. Work = Work + Allocationi
   Finish[i] = true
   go to step 2
4. If Finish [i] == true for all i, then the system is in a safe state.

Resource-Request Algorithm for Process Pi

Request i = request vector for process Pi . If Requesti [j] = k then process Pi wants k instances of resource type Rj

1. If Requesti ≤ Needi go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim

2. If Requesti ≤ Available, go to step 3. Otherwise Pi must wait, since resources are not available 3. Pretend to allocate requested resources to Pi by modifying the state as follows:
Available = Available – Requesti ;
Allocationi = Allocationi + Requesti ;
Needi = Needi – Requesti ;
1.If safe ⇒ the resources are allocated to Pi
2. If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored.

Program:

```
#include<stdio.h>

int main() {

 /* array will store at most 5 process with 3 resoures if your process or
```

```
 resources is greater than 5 and 3 then increase the size of array */

 int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5], terminate
= 0;

 printf("Enter the number of process and resources");

 scanf("%d %d", & p, & c);

 // p is process and c is diffrent resources

 printf("enter allocation of resource of all process %dx%d matrix", p, c);

 for (i = 0; i < p; i++) {

  for (j = 0; j < c; j++) {

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

  }

 }

 printf("enter the max resource process required %dx%d matrix", p, c);

 for (i = 0; i < p; i++) {

  for (j = 0; j < c; j++) {

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

  }

 }

 printf("enter the  available resource");

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

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


 printf("\n need resources matrix are\n");

 for (i = 0; i < p; i++) {

  for (j = 0; j < c; j++) {

   need[i][j] = max[i][j] - alc[i][j];

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

```
  }
  printf("\n");
 }
 /* once process execute variable done will stop them for again execution */
 for (i = 0; i < p; i++) {
  done[i] = 0;
 }
 while (count < p) {
  for (i = 0; i < p; i++) {
   if (done[i] == 0) {
    for (j = 0; j < c; j++) {
     if (need[i][j] > available[j])
      break;
    }
    //when need matrix is not greater then available matrix then if j==c will true
    if (j == c) {
     safe[count] = i;
     done[i] = 1;
     /* now process get execute release the resources and add them in available resources */
     for (j = 0; j < c; j++) {
      available[j] += alc[i][j];
     }
     count++;
     terminate = 0;
    } else {
     terminate++;
    }
```

```
    }
   }
   if (terminate == (p - 1)) {
    printf("safe sequence does not exist");
    break;
   }

 }
 if (t    for (j = 0; j < c; j++) {



erminate != (p - 1)) {
   printf("\n available resource after completion\n");
   for (i = 0; i < c; i++) {
    printf("%d\t", available[i]);
   }
   printf("\n safe sequence are\n");
   for (i = 0; i < p; i++) {
    printf("p%d\t", safe[i]);
   }
 }

//Enter the number of process and resources4
5
enter allocation of resource of all process 4x5 matrix56
6
4
```

3

2

2

2

1

6

7

8

9

9

8

7

6

5

4

4

5

enter the max resource process required 4x5 matrix3

3

6

8

9

0

7

6

5

4

3

7

8

9

7

6

5

8

9

4

enter the available resource3

5

6

7

8

need resources matrix are

-53	**-3**	**2**	**-2**	**5**
-2	**5**	**5**	**-5**	**-2**
-5	**-2**	**-1**	**0**	**0**
0	**0**	**4**	**5**	**-1**

available resource after completion

75	**27**	**24**	**20**	**20**

safe sequence are

p20	**p20**	**p2**	**p3**

Conclusion:

When can we say that the system is in safe or unsafe state?

The negative numbers in the need matrix indicate that the allocation matrix entries are greater than the maximum matrix entries, which is an error in the input. The program should validate the input to prevent such inconsistencies. Because of the negative numbers, the program results can't be trusted. The code also has a typo. The variable 'terminate' is compared to '(p-1)' within the loop, and then is tested again outside of the loop. If the first test caused the loop to break, the second test will cause the output to be inaccurate.