## Experiment–4

### BANKER'S ALGORITHM

**Objective** : To Design, develop and implement a C/C++/Java program to implement Banker's algorithm. Assume suitable input required to demonstrate the results.

**//ALGORITHM**

**ALGORITHM**: BankersAlgorithmImplementation
**PURPOSE**: To represent the working of the Bankers Algorithm
**INPUT**: No of Process and Resources
**OUTPUT**: Process state whether safe or unsafe

```
STEP 1 : START
STEP 2 : Initialize, count = 0, k = 0
STEP 3  : Initialize, for i = 1 to p do, comp[i] = 0
          [calculate, available resources of each type]
          for j = 1 to r do,
          total = 0
          avail[j] = 0
         for i = to p do
               total = total + alloc [i][j]
               avail[j] = rsrc[j] – total
        [Initialize, temporary variable work[] to avail[]]
       for j=1 to r do,
               work[j]=avail[j];
               [calculate the need of each process i.e. req[p][r] matrix]
       for i=1 to p do,
               for j = 1 to r do,
                       req[i][j] = claim [i][j] – alloc [i][j]
STEP 4 : Repeat the following until (count!=p && k<2)
          Increment k
          for i =1 to p do,
          for j = 1 to r do,
               if(comp[i] =0) //process Pi is not completed
               if(req[i][j] <= work[j]) //need of Pi is less than available resouces
               work[j] = work[j] + alloc[i][j]
               comp[i] = 1
               alloc[i][j] 0, claim[i][j] = 0
               Increment count
               else break;

STEP 5 : if (count != p)
          Print "system is in an unsafe state"
         else
          Print "system is in a safe state"
STEP 6 : STOP
```

**Program:**

```
#include <stdio.h>

void displayMatrix(int matrix[][10], int rows, int cols) {
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
}

int main()
 {
   int Max[10][10], alloc[10][10], need[10][10], avail[10], completed[10], safeSequence[10];
   int p, r, i, j, process, count = 0;

   // Get the number of processes with validation
   do
 {
      printf("Enter the number of processes (max 10): ");
      scanf("%d", &p);
   } while (p <= 0 || p > 10);

   // Get the number of resources with validation
   do
 {
      printf("Enter the number of resources (max 10): ");
      scanf("%d", &r);
   } while (r <= 0 || r > 10);

   // Initialize completed array
   for (i = 0; i < p; i++)
      completed[i] = 0;

   // Input Max matrix with validation
   printf("Enter the Max Matrix for each process:\n");
   for (i = 0; i < p; i++)
 {
      printf("For process %d: ", i + 1);
```

```
        for (j = 0; j < r; j++)
            scanf("%d", &Max[i][j]);
    }



    // Input allocation matrix with validation
    printf("Enter the allocation for each process:\n");
    for (i = 0; i < p; i++)
{
        printf("For process %d: ", i + 1);
        for (j = 0; j < r; j++)
            scanf("%d", &alloc[i][j]);
    }

    // Input available resources with validation
    printf("Enter the Available Resources:\n");
    for (i = 0; i < r; i++)
        scanf("%d", &avail[i]);

    // Calculate need matrix
    for (i = 0; i < p; i++)
        for (j = 0; j < r; j++)
            need[i][j] = Max[i][j] - alloc[i][j];

    // Display Max, Allocation, and Need matrices
    printf("\nMax Matrix:\n");
    displayMatrix(Max, p, r);

    printf("\nAllocation Matrix:\n");
    displayMatrix(alloc, p, r);

    printf("\nNeed Matrix:\n");
    displayMatrix(need, p, r);

    // Banker's algorithm
    Do
 {
        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);

         // Release resources held by the completed process
         for (j = 0; j < r; j++)
 {
            avail[j] += alloc[process][j];
            alloc[process][j] = 0;
            Max[process][j] = 0;
         }

         // Mark the process as completed outside the loop
         completed[process] = 1;

         // Update safe sequence and increment count
         safeSequence[count] = process + 1;
         count++;
      }
   } while (count != p && process != -1);

   // Display results based on the outcome of the Banker's algorithm
   if (count == p)
{
      printf("\nThe system is in a safe state!!\n");
      printf("Safe Sequence : < ");
      for (i = 0; i < p; i++)
```

```
            printf("%d ", safeSequence[i]);
            printf(">\n");
    }
else
        printf("\nThe system is in an unsafe state!!\n");
        return 0;
}
```

**Input 1:**

**Enter the number of processes (max 10): 3**
**Enter the number of resources (max 10): 4**

**Enter the Max Matrix for each process:**
**For process 1: 7 5 3 2**
**For process 2: 3 2 2 1**
**For process 3: 9 0 2 4**

**Enter the allocation for each process:**
**For process 1: 0 1 0 2**
**For process 2: 2 0 0 1**
**For process 3: 3 3 2 4**

**Enter the Available Resources:**
**1 2 2 1**

**Output 1:**
**Max Matrix:**
**7 5 3 2**
**3 2 2 1**
**9 0 2 4**

**Allocation Matrix:**
**0 1 0 2**
**2 0 0 1**
**3 3 2 4**

**Need Matrix:**
**7 4 3 0**
**1 2 2 0**
**6 0 0 0**

**Process 2 runs to completion!**
**Process 3 runs to completion!**
**Process 1 runs to completion!**

**The system is in a safe state!!**
**Safe Sequence : < 2 3 1 >**

**Input 2:**
**Enter the number of processes (max 10): 3**
**Enter the number of resources (max 10): 4**

**Enter the Max Matrix for each process:**
**For process 1: 7 5 3 2**
**For process 2: 3 2 2 1**
**For process 3: 9 0 2 4**

**Enter the allocation for each process:**
**For process 1: 0 1 0 2**
**For process 2: 2 0 0 1**
**For process 3: 3 3 2 4**

**Enter the Available Resources:**
**0 0 0 0**

**Output 2:**
**Max Matrix:**
**7 5 3 2**
**3 2 2 1**
**9 0 2 4**

**Allocation Matrix:**
**0 1 0 2**
**2 0 0 1**
**3 3 2 4**

**Need Matrix:**
**7 4 3 0**
**1 2 2 0**
**6 0 0 0**

**The system is in an unsafe state!!**