Program – 14

Aim – WAP to implement Banker's Algorithm for single resource for single instance for deadlock avoidance.

Source code :

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 10
int main() {
   int n;
   int allocation[MAX], max[MAX], need[MAX], available;
   bool finish[MAX] = {false};
   int safeSequence[MAX];
   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("Enter Allocation (0 or 1) for each process:\n");
   for (int i = 0; i < n; i++)
  scanf("%d", &allocation[i]);
   printf("Enter Maximum need (0 or 1) for each process:\n");
   for (int i = 0; i < n; i++)
  scanf("%d", &max[i]);
  printf("Enter Available instances of the single resource (0 or 1): ");
   scanf("%d", &available);
  for (int i = 0; i < n; i++)
  need[i] = max[i] - allocation[i];
   int count = 0;
   while (count < n) {
      bool found = false;
      for (int i = 0; i < n; i++) {
         if (!finish[i] && need[i] <= available) {
            available += allocation[i];
            safeSequence[count++] = i;
```

```
            finish[i] = true;
            found = true;
        }
    }
    if (!found) {
        printf("\nSystem is not in a safe state (deadlock possible).\n");
        return 0;
    }
  }
  printf("\nSystem is in a safe state.\nSafe sequence: ");
  for (int i = 0; i < n; i++)
    printf("P%d ", safeSequence[i]);
  printf("\n");
  return 0;
}
```

OUTPUT :

```
PS C:\Users\LENOVO\Downloads\OS LAB> cd "c:\Users\LENOVO\Downloads\OS LAB\" ; if ($?) { gcc 14.c -o 14 } ; if ($?) { .\14 }
Enter number of processes: 3
Enter Allocation (0 or 1) for each process:
1 0 0
Enter Maximum need (0 or 1) for each process:
1 1 1
Enter Available instances of the single resource (0 or 1): 1

System is in a safe state.
Safe sequence: P0 P1 P2
PS C:\Users\LENOVO\Downloads\OS LAB>
```

Program – 14(a)

Aim - WAP to implement Banker's Algorithm for multiple resource for multiple instances for deadlock avoidance.

Source code :

```
#include <stdio.h>

#include <stdbool.h>

#define MAX 10

int n, m;

int allocation[MAX][MAX], maxNeed[MAX][MAX], need[MAX][MAX];

int available[MAX];

bool isSafeState(int safeSequence[]) {
   int work[MAX];
   bool finish[MAX] = {false};
   for (int i = 0; i < m; i++)
      work[i] = available[i];
   int count = 0;
   while (count < n) {
      bool found = false;
      for (int i = 0; i < n; i++) {
         if (!finish[i]) {
            bool canAllocate = true;
            for (int j = 0; j < m; j++) {
               if (need[i][j] > work[j]) {
                  canAllocate = false;
                  break;
               }
            }
            if (canAllocate) {
               for (int j = 0; j < m; j++)
                  work[j] += allocation[i][j];
               safeSequence[count++] = i;
```

```
                finish[i] = true;
                found = true;
            }
        }
    }
    if (!found)
        return false;
  }
  return true;
}
int main() {
  printf("Enter number of processes: ");
  scanf("%d", &n);
  printf("Enter number of resource types: ");
  scanf("%d", &m);
  printf("Enter Allocation Matrix (%d x %d):\n", n, m);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      scanf("%d", &allocation[i][j]);
  printf("Enter Maximum Matrix (%d x %d):\n", n, m);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      scanf("%d", &maxNeed[i][j]);
  printf("Enter Available Resources (%d):\n", m);
  for (int j = 0; j < m; j++)
    scanf("%d", &available[j]);
  // Calculate Need Matrix
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      need[i][j] = maxNeed[i][j] - allocation[i][j];
```

```
int safeSequence[MAX];
if (isSafeState(safeSequence)) {
   printf("\nSystem is in a safe state.\nSafe sequence: ");
   for (int i = 0; i < n; i++)
      printf("P%d ", safeSequence[i]);
   printf("\n");
} else {
   printf("\nSystem is NOT in a safe state.\n");
   return 0;
}
// Handle a new request
int reqProcess, request[MAX];
printf("\nDo you want to make a resource request? (1 = Yes, 0 = No): ");
int choice;
scanf("%d", &choice);
if (choice == 1) {
   printf("Enter process number (0 to %d) making the request: ", n - 1);
   scanf("%d", &reqProcess);

   printf("Enter request for each resource (%d values): ", m);
   for (int j = 0; j < m; j++)
      scanf("%d", &request[j]);
   // Check request ≤ need
   bool valid = true;
   for (int j = 0; j < m; j++) {
      if (request[j] > need[reqProcess][j]) {
         valid = false;
         break;
      }
   }
```

```
// Check request ≤ available
if (valid) {
    for (int j = 0; j < m; j++) {
        if (request[j] > available[j]) {
            valid = false;
            break;
        }
    }
}
if (!valid) {
    printf("Request cannot be granted (exceeds need or availability).\n");
} else {
    // Pretend to allocate resources
    for (int j = 0; j < m; j++) {
        available[j] -= request[j];
        allocation[reqProcess][j] += request[j];
        need[reqProcess][j] -= request[j];
    }
    if (isSafeState(safeSequence)) {
        printf("\nRequest can be safely granted.\nSafe sequence: ");
        for (int i = 0; i < n; i++)
            printf("P%d ", safeSequence[i]);
        printf("\n");
    } else {
        // Rollback
        for (int j = 0; j < m; j++) {
            available[j] += request[j];
            allocation[reqProcess][j] -= request[j];
            need[reqProcess][j] += request[j];
        }
        printf("\nRequest leads to unsafe state. Request denied.\n");
```

```
        }
      }
    }
    return 0;
}
```

OUTPUT :

```
PS C:\Users\LENOVO\Downloads\OS LAB> cd "c:\Users\LENOVO\Downloads\OS LAB\" ; if ($?) { gcc banker.c -o banker } ; if ($?) { .\banker }
Enter number of processes: 5
Enter number of resource types: 3
Enter Allocation Matrix (5 x 3):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter Maximum Matrix (5 x 3):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter Available Resources (3):
3 3 2

System is in a safe state.
Safe sequence: P1 P3 P4 P0 P2

Do you want to make a resource request? (1 = Yes, 0 = No): 1
Enter process number (0 to 4) making the request: 1
Enter request for each resource (3 values): 1 0 2

Request can be safely granted.
Safe sequence: P1 P3 P4 P0 P2
PS C:\Users\LENOVO\Downloads\OS LAB>
```

Program – 15

Aim – Write a program to implement algorithm for deadlock detection.

Source code :

```
#include <stdio.h>

#include <stdbool.h>

#define MAX 10

int main() {

   int n, m; // Number of processes and resources

   int allocation[MAX][MAX], request[MAX][MAX], available[MAX];

   int work[MAX];

   bool finish[MAX] = {false};

   printf("Enter number of processes: ");

   scanf("%d", &n);

   printf("Enter number of resources: ");

   scanf("%d", &m);

   printf("\nEnter Allocation Matrix (%d x %d):\n", n, m);

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

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

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

   printf("\nEnter Request Matrix (%d x %d):\n", n, m);

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

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

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

   printf("\nEnter Available Resources (%d):\n", m);

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

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

      work[i] = available[i];

   }

   // Initialize finish[i] = false if allocation[i] != 0

   for (int i = 0; i < n; i++) {
```

```
    bool allocated = false;
    for (int j = 0; j < m; j++) {
        if (allocation[i][j] != 0) {
            allocated = true;
            break;
        }
    }
    finish[i] = !allocated; // true if no resources allocated (can finish)
}
int count = 0;
while (count < n) {
    bool found = false;
    for (int i = 0; i < n; i++) {
        if (!finish[i]) {
            bool canAllocate = true;
            for (int j = 0; j < m; j++) {
                if (request[i][j] > work[j]) {
                    canAllocate = false;
                    break;
                }
            }
            if (canAllocate) {
                for (int j = 0; j < m; j++)
                    work[j] += allocation[i][j];
                finish[i] = true;
                found = true;
                count++;
            }
        }
    }
    if (!found)
```

```
            break;
        }
        // Print result
        bool deadlock = false;
        printf("\nProcesses in deadlock: ");
        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                printf("P%d ", i);
                deadlock = true;
            }
        }
        if (!deadlock)
            printf("None. System is in a safe state.\n");
        else
            printf("\nSystem is in deadlock.\n");
        return 0;
}
```

OUTPUT :

```
PS C:\Users\LENOVO\Downloads\OS LAB> cd "c:\Users\LENOVO\Downloads\OS LAB\" ; if ($?) { gcc 15.c -o 15 } ; if ($?) { .\15 }
Enter number of processes: 4
Enter number of resources: 2

Enter Allocation Matrix (4 x 2):
1 0
0 1
1 0
0 1

Enter Request Matrix (4 x 2):
0 1
1 0
0 1
1 0

Enter Available Resources (2):
0 0

Processes in deadlock: P0 P1 P2 P3
System is in deadlock.
PS C:\Users\LENOVO\Downloads\OS LAB>
```