ASSIGNMENT 7

1. (i) What is Deadlock in Operating System?

Deadlock is a situation in an operating system where a set of processes are permanently blocked because each process is holding at least one resource and waiting for another resource that is held by some other process in the same set.

In deadlock, none of the processes can proceed and the system comes to a standstill.

Necessary Conditions for Deadlock (Coffman Conditions):

Deadlock occurs if all four conditions hold simultaneously:

1. Mutual Exclusion – Resources cannot be shared.
2. Hold and Wait – A process holds resources while waiting for others.
3. No Preemption – Resources cannot be forcibly taken.
4. Circular Wait – A circular chain of processes exists, each waiting for a resource held by the next.

(ii) Banker's Algorithm for Deadlock Avoidance

Banker's Algorithm is a deadlock avoidance algorithm that checks whether granting a resource request will leave the system in a safe state.

A system is said to be in a safe state if there exists at least one safe sequence of processes such that all processes can complete without deadlock.

Data Structures Used:

- Available[m] – Number of available resources of each type
- Max[n][m] – Maximum demand of each process
- Allocation[n][m] – Resources currently allocated
- Need[n][m] – Remaining resource need
  Need = Max – Allocation

Steps of Banker's Algorithm:

1. Calculate the Need matrix.
2. Initialize Work = Available and Finish[i] = false.
3. Find a process Pi such that:
    - Finish[i] == false
    - Need[i] ≤ Work
4. If found:
    - Work = Work + Allocation[i]
    - Finish[i] = true
    - Add Pi to safe sequence

5. Repeat until all processes finish.
6. If all Finish[i] == true, system is safe.
   Otherwise, unsafe (deadlock may occur).

2. i. Write a program, which will simulate banker's algorithm. Show the safe sequence(If available)

CODE:

```
#include <stdio.h>
int main() {
   int n, m;
   printf("Enter number of processes: ");
   scanf("%d", &n);
   printf("Enter number of resource types: ");
   scanf("%d", &m);
   int alloc[n][m], max[n][m], need[n][m];
   int avail[m], finish[n], safe[n];
   printf("\nEnter Allocation Matrix:\n");
   for(int i = 0; i < n; i++)
      for(int j = 0; j < m; j++)
         scanf("%d", &alloc[i][j]);
   printf("\nEnter Max Matrix:\n");
   for(int i = 0; i < n; i++)
      for(int j = 0; j < m; j++)
         scanf("%d", &max[i][j]);
   printf("\nEnter Available Resources:\n");
   for(int i = 0; i < m; i++)
      scanf("%d", &avail[i]);
   // Need = Max - Allocation
   for(int i = 0; i < n; i++)
      for(int j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];
   for(int i = 0; i < n; i++)
      finish[i] = 0;
   int count = 0;
```

```
    while(count < n) {
        int found = 0;
        for(int i = 0; i < n; i++) {
            if(!finish[i]) {
                int j;
                for(j = 0; j < m; j++)
                    if(need[i][j] > avail[j])
                        break;
                if(j == m) {
                    for(int k = 0; k < m; k++)
                        avail[k] += alloc[i][k];
                    safe[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if(!found) {
            printf("\nSystem is NOT in safe state\n");
            return 0;
        }
    }
    printf("\nSystem is in SAFE state\nSafe Sequence: ");
    for(int i = 0; i < n; i++)
        printf("P%d ", safe[i]);
    return 0;
}
```

OUTPUT:

```
Enter number of processes: 5
Enter number of resource types: 3

Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2


Enter Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3


Enter Available Resources:
3 3 2
```

```
System is in SAFE state
Safe Sequence: P1 P3 P4 P0 P2
```