EXPERIMENT - 5

NAME - Ayush Sanjay Dhangar ROLL NO - 42

CLASS - TY_IT-A BATCH - 2

DATE OF PERFORMANCE - 11/09/2024

Q . BANKERS ALGORITHM -

CODE -

```
#include <stdio.h>

int main() {
    int n, m, i, j, k;

    // Taking number of processes and resources from the user
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter number of resources: ");
    scanf("%d", &m);

    int alloc[n][m], max[n][m], avail[m], total[m], need[n][m];
    int f[n], ans[n], ind = 0;

    // Input the allocation matrix from the user
    printf("Enter the allocation matrix:\n");
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
            printf("P%d, Resource %d: ", i + 1, j + 1);
            scanf("%d", &alloc[i][j]);
        }
    }
```

```
// Input the max matrix from the user
printf("Enter the max matrix:\n");
for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
        printf("P%d, Resource %d: ", i + 1, j + 1);
        scanf("%d", &max[i][j]);
    }
}

// Input the total resources vector from the user
printf("Enter the total resources vector:\n");
for(i = 0; i < m; i++) {
    printf("Resource %d: ", i + 1);
    scanf("%d", &total[i]);
}

// Calculate available resources (Total - Allocation)
for(j = 0; j < m; j++) {
    int sum_alloc = 0;
    for(i = 0; i < n; i++) {
        sum_alloc += alloc[i][j];
    }
    avail[j] = total[j] - sum_alloc;
}

// Initialize finished flag for all processes
for(k = 0; k < n; k++) {
    f[k] = 0;
}
```

```
// Calculate the need matrix
for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
        need[i][j] = max[i][j] - alloc[i][j];
    }
}

// Check if the system is in a safe state using the Banker's Algorithm
int flag = 0;
int y;
for(k = 0; k < n; k++) {
    for(i = 0; i < n; i++) {
        if(f[i] == 0) {
            flag = 0;
            for(j = 0; j < m; j++) {
                if(need[i][j] > avail[j]) {
                    flag = 1;
                    break;
                }
            }
            if(flag == 0) {
                ans[ind++] = i;
                for(y = 0; y < m; y++) {
                    avail[y] += alloc[i][y];
                }
                f[i] = 1;
            }
        }
    }
}
```

```
// Check if all processes are finished
int safe = 1;
for(i = 0; i < n; i++) {
    if(f[i] == 0) {
        safe = 0;
        printf("The system is not in a safe state.\n");
        break;
    }
}

// Output matrices
printf("\nAllocation Matrix:\n");
for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
        printf("%d ", alloc[i][j]);
    }
    printf("\n");
}

printf("\nMax Matrix:\n");
for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
        printf("%d ", max[i][j]);
    }
    printf("\n");
}

printf("\nNeed Matrix:\n");
for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
        printf("%d ", need[i][j]);
```

```
        }
        printf("\n");
    }

    printf("\nAvailable Vector:\n");
    for(i = 0; i < m; i++) {
        printf("%d ", avail[i]);
    }
    printf("\n");

    // If the system is safe, print the safe sequence
    if(safe) {
        printf("\nFollowing is the SAFE Sequence:\n");
        for(i = 0; i < n - 1; i++) {
            printf("P%d -> ", ans[i]);
        }
        printf("P%d\n", ans[n - 1]);
    }

    return 0;
}
```

OUTPUT -

```
Enter number of processes: 5
Enter number of resources: 3
Enter the allocation matrix:
P1, Resource 1: 0
P1, Resource 2: 1
P1, Resource 3: 0
P2, Resource 1: 2
P2, Resource 2: 0
P2, Resource 3: 0
P3, Resource 1: 3
P3, Resource 2: 0
P3, Resource 3: 2
P4, Resource 1: 2
P4, Resource 2: 1
P4, Resource 3: 1
P5, Resource 1: 0
P5, Resource 2: 0
P5, Resource 3: 2
Enter the max matrix:
P1, Resource 1: 7
P1, Resource 2: 5
P1, Resource 3: 3
P2, Resource 1: 3
P2, Resource 2: 2
P2, Resource 3: 2
P3, Resource 1: 9
P3, Resource 2: 0
P3, Resource 3: 2
P4, Resource 1: 2
P4, Resource 2: 2
P4, Resource 3: 2
P5, Resource 1: 4
P5, Resource 2: 3
P5, Resource 3: 3
```

```
Enter the total resources vector:
Resource 1: 10
Resource 2: 5
Resource 3: 7

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

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

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

Available Vector:
10 5 7

Following is the SAFE Sequence:
P1 -> P3 -> P4 -> P0 -> P2
```