THENMOZHI R
231901064

Ex. No.: 9
Date: 02-04-2025

DEADLOCK AVOIDANCE

Aim: To find out a safe sequence using Banker's algorithm for deadlock avoidance.

Algorithm:

1. Initialize work=available and finish[i]=false for all values of i
2. Find an i such that both: finish[i]=false and Needi<= work
3. If no such i exists go to step 6
4. Compute work=work+allocationi
5. Assign finish[i] to true and go to step 2
6. If finish[i]==true for all i, then print safe sequence
7. Else print there is no safe sequence

Program Code:

```
#include <stdio.h>
#include <stdbool.h>
#define P 5 // Number of processes
#define R 3 // Number of resources
void findSafeSequence(int processes[], int available[], int max[][R], int allocation[][R]) {
    int need[P][R];
    bool finish[P] = {false};
    int safeSequence[P];
    int work[R];
    // Calculate Need Matrix
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            need[i][j] = max[i][j] - allocation[i][j];
        }
    }
    // Initialize work as available resources
    for (int i = 0; i < R; i++) {
        work[i] = available[i];
    }
    int count = 0;
    while (count < P) {
        bool found = false;
        for (int i = 0; i < P; i++) {
            if (!finish[i]) {
                int j;
```

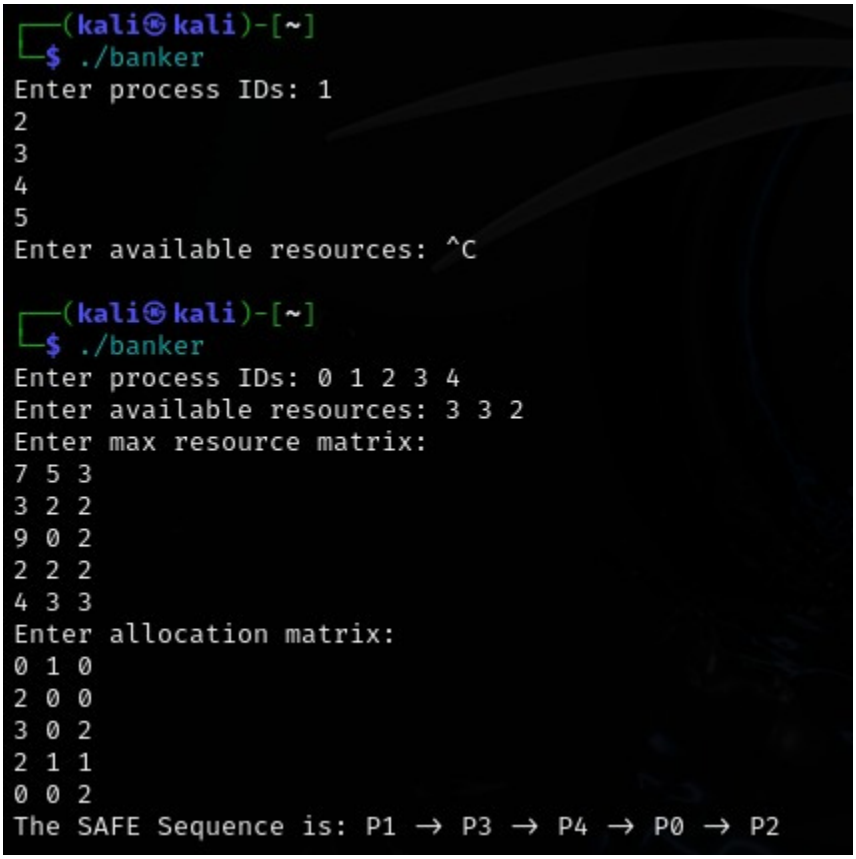
```
            for (j = 0; j < R; j++) {
                if (need[i][j] > work[j]) {
                    break;
                }
            }
            if (j == R) { // If all needs are met
                for (int k = 0; k < R; k++) {
                    work[k] += allocation[i][k];
                }
                safeSequence[count++] = processes[i];
                finish[i] = true;
                found = true;
            }
        }
    }
    if (!found) {
        printf("No safe sequence exists\n");
        return;
    }
}
// Print Safe Sequence
printf("The SAFE Sequence is: ");
for (int i = 0; i < P; i++) {
    printf("P%d", safeSequence[i]);
    if (i < P - 1) printf(" -> ");
}
printf("\n");
}
int main() {
    int processes[P];
    int available[R];
    int max[P][R];
    int allocation[P][R];

    // Get user input
    printf("Enter process IDs: ");
    for (int i = 0; i < P; i++) {
        scanf("%d", &processes[i]);
    }
    printf("Enter available resources: ");
    for (int i = 0; i < R; i++) {
        scanf("%d", &available[i]);
    }
    printf("Enter max resource matrix: \n");
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
```

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

    printf("Enter allocation matrix: \n");
    for (int i = 0; i < P; i++) {
        for (int j = 0; j < R; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }
    findSafeSequence(processes, available, max, allocation);
    return 0;
}
```

OUTPUT:



RESULT:

Hence,safe sequence using Banker’s algorithm for deadlock avoidance has been executed