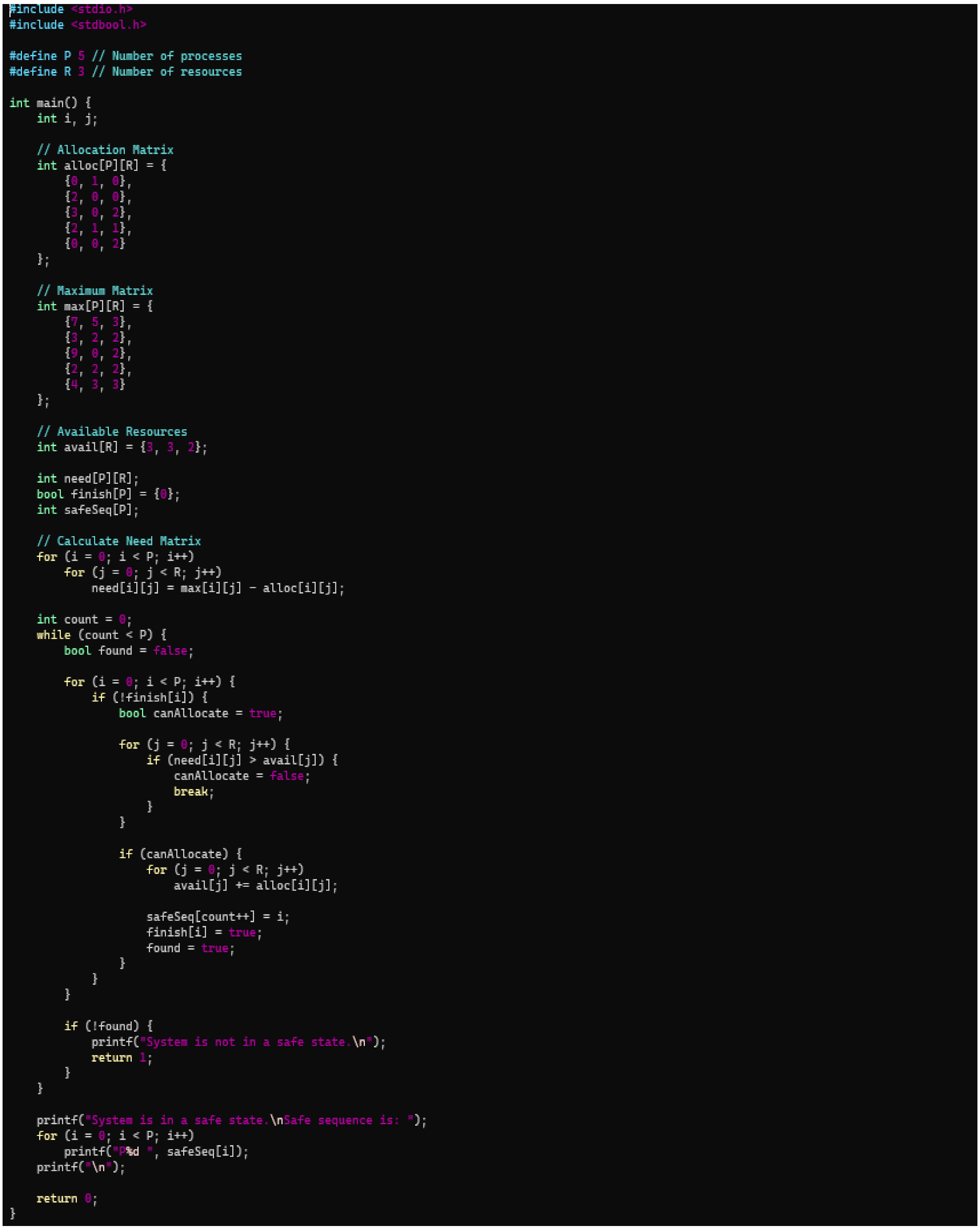
Exp – 9):

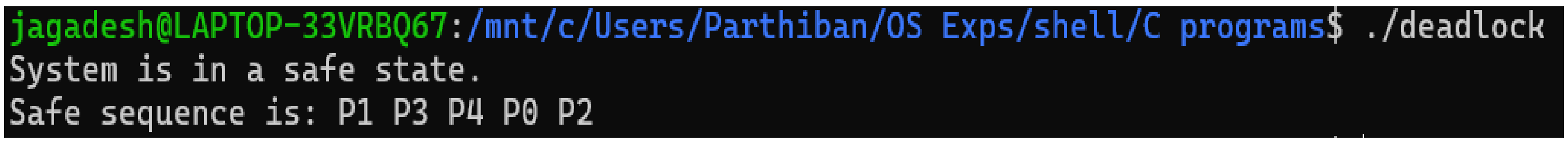
AIM: To find out a safe sequence using Banker's algorithm for deadlock avoidance.

PROGRAM:



```c
#include <stdio.h>
#include <stdbool.h>

#define P 5 // Number of processes
#define R 3 // Number of resources

int main() {
    int i, j;

    // Allocation Matrix
    int alloc[P][R] = {
        {0, 1, 0},
        {2, 0, 0},
        {3, 0, 2},
        {2, 1, 1},
        {0, 0, 2}
    };

    // Maximum Matrix
    int max[P][R] = {
        {7, 5, 3},
        {3, 2, 2},
        {9, 0, 2},
        {2, 2, 2},
        {4, 3, 3}
    };

    // Available Resources
    int avail[R] = {3, 3, 2};

    int need[P][R];
    bool finish[P] = {0};
    int safeSeq[P];

    // Calculate Need Matrix
    for (i = 0; i < P; i++)
        for (j = 0; j < R; j++)
            need[i][j] = max[i][j] - alloc[i][j];

    int count = 0;
    while (count < P) {
        bool found = false;

        for (i = 0; i < P; i++) {
            if (!finish[i]) {
                bool canAllocate = true;

                for (j = 0; j < R; j++) {
                    if (need[i][j] > avail[j]) {
                        canAllocate = false;
                        break;
                    }
                }

                if (canAllocate) {
                    for (j = 0; j < R; j++)
                        avail[j] += alloc[i][j];

                    safeSeq[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }

        if (!found) {
            printf("System is not in a safe state.\n");
            return 1;
        }
    }

    printf("System is in a safe state.\nSafe sequence is: ");
    for (i = 0; i < P; i++)
        printf("P%d ", safeSeq[i]);
    printf("\n");

    return 0;
}
```

OUTPUT:



```
jagadesh@LAPTOP-33VRBQ67:/mnt/c/Users/Parthiban/OS Exps/shell/C programs$ ./deadlock
System is in a safe state.
Safe sequence is: P1 P3 P4 P0 P2
```