Write a C Code to implement the

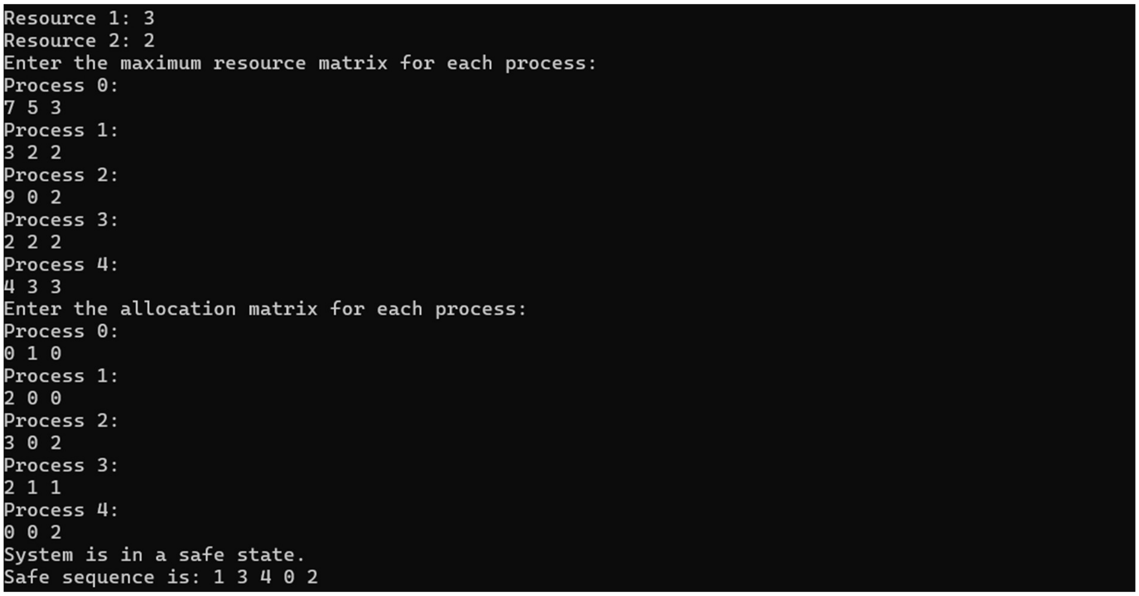
1. Bankers algorithm for deadlock avoidance
2. Deadlock Detection

Bankers algorithm for deadlock avoidance

```
#include <stdio.h>

#include <stdbool.h>


int main() {

    int P, R;


    printf("Enter the number of processes: ");

    scanf("%d", &P);


    printf("Enter the number of resources: ");

    scanf("%d", &R);


    int available[R], maximum[P][R], allocation[P][R], need[P][R];


    printf("Enter the available instances of each resource:\n");

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

        printf("Resource %d: ", i);

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

    }


    printf("Enter the maximum resource matrix for each process:\n");

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

        printf("Process %d:\n", i);

        for (int j = 0; j < R; j++) {

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

        }
```

```
}

printf("Enter the allocation matrix for each process:\n");
for (int i = 0; i < P; i++) {
    printf("Process %d:\n", i);
    for (int j = 0; j < R; j++) {
        scanf("%d", &allocation[i][j]);
    }
}
for (int i = 0; i < P; i++)
    for (int j = 0; j < R; j++)
        need[i][j] = maximum[i][j] - allocation[i][j];
int work[R];
bool finish[P];
int safeSequence[P];
for (int i = 0; i < R; i++)
    work[i] = available[i];
for (int i = 0; i < P; i++)
    finish[i] = false;

int count = 0;
while (count < P) {
    bool found = false;
    for (int p = 0; p < P; p++) {
        if (!finish[p]) {
            int j;
            for (j = 0; j < R; j++)
                if (need[p][j] > work[j])
                    break;
            if (j == R) {
                for (int k = 0; k < R; k++)
```

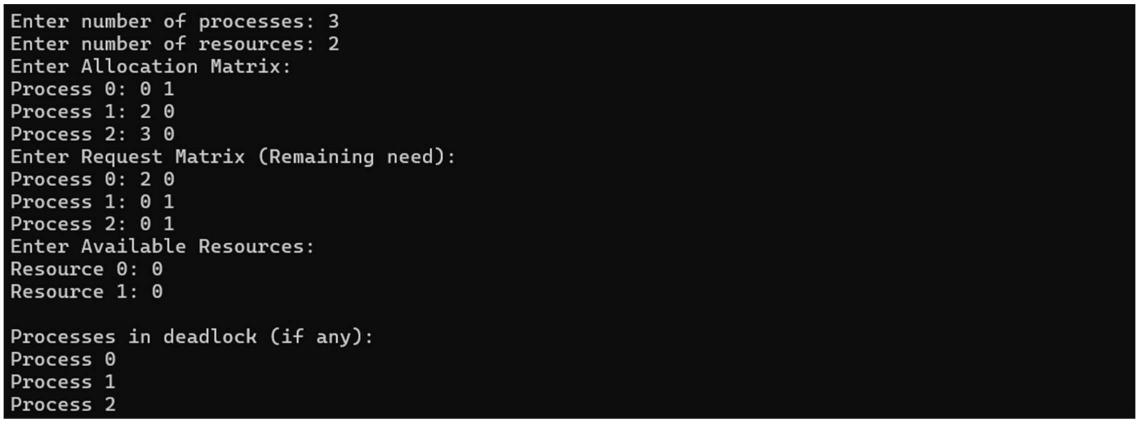
```
                        work[k] += allocation[p][k];
                    safeSequence[count++] = p;
                    finish[p] = true;
                    found = true;
                }
            }
        }
        if (!found) {
            printf("System is not in a safe state.\n");
    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



```
Resource 1: 3
Resource 2: 2
Enter the maximum resource matrix for each process:
Process 0:
7 5 3
Process 1:
3 2 2
Process 2:
9 0 2
Process 3:
2 2 2
Process 4:
4 3 3
Enter the allocation matrix for each process:
Process 0:
0 1 0
Process 1:
2 0 0
Process 2:
3 0 2
Process 3:
2 1 1
Process 4:
0 0 2
System is in a safe state.
Safe sequence is: 1 3 4 0 2
```

Deadlock Detection

```
#include <stdio.h>

#include <stdbool.h>


int main() {

    int P, R;


    printf("Enter number of processes: ");

    scanf("%d", &P);


    printf("Enter number of resources: ");

    scanf("%d", &R);


    int allocation[P][R], request[P][R], available[R];

    bool finish[P];


    printf("Enter Allocation Matrix:\n");

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

        printf("Process %d: ", i);

        for (int j = 0; j < R; j++) {

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

        }

    }


    printf("Enter Request Matrix (Remaining need):\n");

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

        printf("Process %d: ", i);

        for (int j = 0; j < R; j++) {

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

    printf("Enter Available Resources:\n");
    for (int i = 0; i < R; i++) {
        printf("Resource %d: ", i);
        scanf("%d", &available[i]);
    }

    for (int i = 0; i < P; i++) {
        bool zero_allocation = true;
        for (int j = 0; j < R; j++) {
            if (allocation[i][j] != 0) {
                zero_allocation = false;
                break;
            }
        }
        finish[i] = zero_allocation;
    }

    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 (request[i][j] > available[j])
                        break;
                }
                if (j == R) {
                    for (int k = 0; k < R; k++)
                        available[k] += allocation[i][k];
                    finish[i] = true;
                    found = true;
                    count++;
                }
            }
        }
        if (!found)
            break;
    }

    bool deadlock = false;
    printf("\nProcesses in deadlock (if any):\n");
    for (int i = 0; i < P; i++) {
        if (!finish[i]) {
            printf("Process %d\n", i);
            deadlock = true;
        }
    }

    if (!deadlock)
        printf("No deadlock detected. All processes can complete.\n");

    return 0;
}
```



```
Enter number of processes: 3
Enter number of resources: 2
Enter Allocation Matrix:
Process 0: 0 1
Process 1: 2 0
Process 2: 3 0
Enter Request Matrix (Remaining need):
Process 0: 2 0
Process 1: 0 1
Process 2: 0 1
Enter Available Resources:
Resource 0: 0
Resource 1: 0

Processes in deadlock (if any):
Process 0
Process 1
Process 2
```