Write a C program to simulate Deadlock Detection in operating systems.

Code:

```
#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 resource types: ");
    scanf("%d", &R);

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

    // Input Allocation Matrix
    printf("Enter the Allocation Matrix (%d x %d):\n", P, R);
    for (int i = 0; i < P; i++) {
        printf("Process P%d: ", i);
        for (int j = 0; j < R; j++) {
            scanf("%d", &allocation[i][j]);
        }
    }

    // Input Request Matrix
    printf("Enter the Request Matrix (%d x %d):\n", P, R);
    for (int i = 0; i < P; i++) {
        printf("Process P%d: ", i);
        for (int j = 0; j < R; j++) {
            scanf("%d", &request[i][j]);
        }
    }

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

    // Initialize Finish array
```

```
for (int i = 0; i < P; i++)
    finish[i] = false;

int work[R];
for (int i = 0; i < R; i++)
    work[i] = available[i];

bool deadlockExists = false;

while (true) {
    bool found = false;
    for (int i = 0; i < P; i++) {
        if (!finish[i]) {
            bool canProceed = true;
            for (int j = 0; j < R; j++) {
                if (request[i][j] > work[j]) {
                    canProceed = false;
                    break;
                }
            }
            if (canProceed) {
                for (int j = 0; j < R; j++)
                    work[j] += allocation[i][j];
                finish[i] = true;
                found = true;
            }
        }
    }
    if (!found)
        break;
}

printf("Deadlocked Processes: ");
for (int i = 0; i < P; i++) {
    if (!finish[i]) {
        printf("P%d ", i);
        deadlockExists = true;
    }
}

if (!deadlockExists)
    printf("System is in safe state.");

printf("\n");
```

```
    return 0;
}
```

Output:

```
Enter the number of processes: 4
Enter the number of resource types: 3
Enter the Allocation Matrix (4 x 3):
Process P0: 1 0 2
Process P1: 2 1 1
Process P2: 1 0 3
Process P3: 1 2 2
Enter the Request Matrix (4 x 3):
Process P0: 0 0 1
Process P1: 1 0 2
Process P2: 0 0 0
Process P3: 3 3 0
Enter the Available Resources (3): 0 0 0
Deadlocked Processes: P3

Process returned 0 (0x0)   execution time : 50.824 s
Press any key to continue.
```