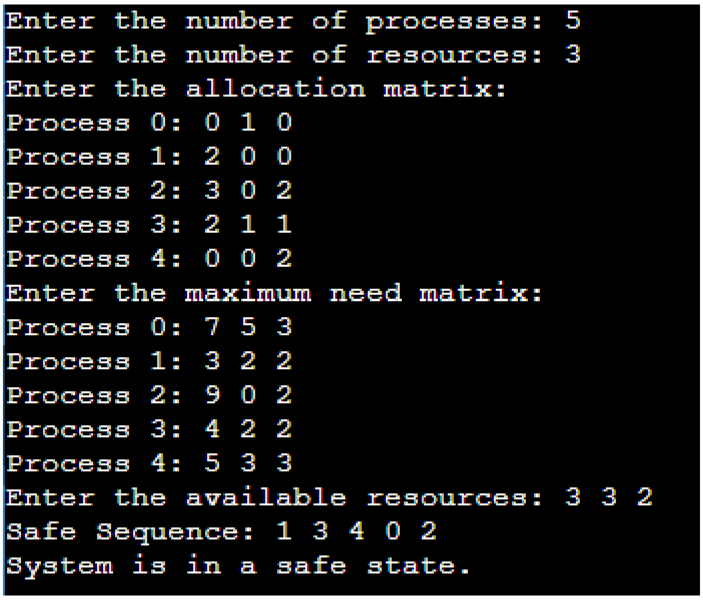
## Practical no 6

**Program :-**

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10
int allocation[MAX_PROCESSES][MAX_RESOURCES];
int max_need[MAX_PROCESSES][MAX_RESOURCES];
int available[MAX_RESOURCES];
int need[MAX_PROCESSES][MAX_RESOURCES];
int finish[MAX_PROCESSES];
int num_processes, num_resources;
// Function prototypes
void calculate_need();
int is_safe();
void request_resources(int process_id, int request[]);
int main() {
   printf("Enter the number of processes: ");
   scanf("%d", &num_processes);
   printf("Enter the number of resources: ");
   scanf("%d", &num_resources);
   // Input allocation matrix
   printf("Enter the allocation matrix:\n");
   for (int i = 0; i < num_processes; ++i) {
      printf("Process %d: ", i);
      for (int j = 0; j < num_resources; ++j) {
         scanf("%d", &allocation[i][j]);
      }
   }
   // Input max_need matrix
   printf("Enter the maximum need matrix:\n");
   for (int i = 0; i < num_processes; ++i) {
      printf("Process %d: ", i);
      for (int j = 0; j < num_resources; ++j) {
         scanf("%d", &max_need[i][j]);
      }
   }
   // Input available resources
   printf("Enter the available resources: ");
   for (int i = 0; i < num_resources; ++i) {
      scanf("%d", &available[i]);
   }
   // Calculate need matrix
   calculate_need();
   // Check if the system is in a safe state
   if (is_safe()) {
      printf("System is in a safe state.\n");
   } else {
      printf("System is in an unsafe state.\n");
```

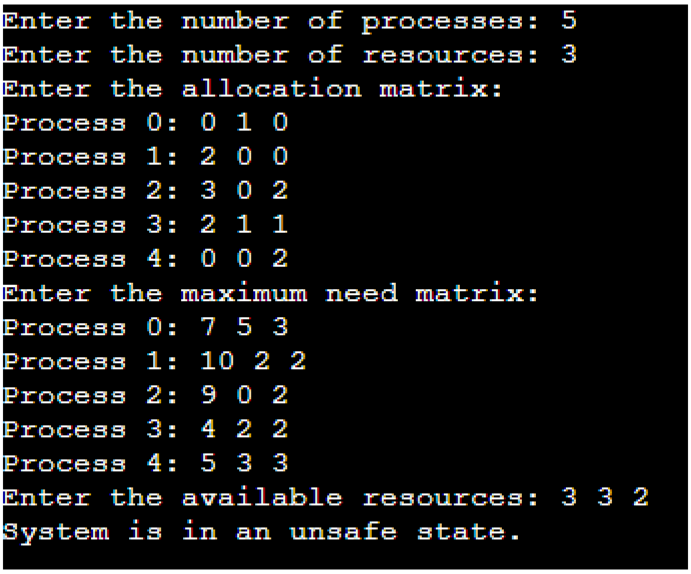
```
    }
    return 0;
}
// Calculate the need matrix
void calculate_need() {
    for (int i = 0; i < num_processes; ++i) {
        for (int j = 0; j < num_resources; ++j) {
            need[i][j] = max_need[i][j] - allocation[i][j];
        }
    }
}
// Check if the system is in a safe state
int is_safe() {
    int work[MAX_RESOURCES];
    int safe_sequence[MAX_PROCESSES];
    int count = 0;
    for (int i = 0; i < num_resources; ++i) {
        work[i] = available[i];
    }
    for (int i = 0; i < num_processes; ++i) {
        finish[i] = 0;
    }
    while (count < num_processes) {
        int found = 0;
        for (int i = 0; i < num_processes; ++i) {
            if (finish[i] == 0) {
                int j;
                for (j = 0; j < num_resources; ++j) {
                    if (need[i][j] > work[j]) {
                        break;
                    }
                }
                if (j == num_resources) {
                    for (int k = 0; k < num_resources; ++k) {
                        work[k] += allocation[i][k];
                    }
                    safe_sequence[count++] = i;
                    finish[i] = 1;
                    found = 1;
                }
            }
        }
        if (!found) {
            break;
        }
    }
    if (count == num_processes) {
        printf("Safe Sequence:");
        for (int i = 0; i < num_processes; ++i) {
            printf(" %d", safe_sequence[i]);
        }
        printf("\n");
        return 1; // System is in safe state
    } else {
        return 0; // System is in unsafe state
    }
}
// Process requests for resources
void request_resources(int process_id, int request[]) {
    // Check if request is within need
    for (int i = 0; i < num_resources; ++i) {
        if (request[i] > need[process_id][i]) {
            printf("Error: Request exceeds maximum need.\n");
            return;
        }
        if (request[i] > available[i]) {
            printf("Error: Request exceeds available resources.\n");
            return;
        }
    }
    // Try to allocate resources
    for (int i = 0; i < num_resources; ++i) {
        available[i] -= request[i];
        allocation[process_id][i] += request[i];
        need[process_id][i] -= request[i];
    }
    // Check if system is in a safe state after allocation
    if (is_safe()) {
        printf("Request granted.\n");
    } else {
        printf("Request denied. System would be in an unsafe state.\n");
        // Rollback allocation
        for (int i = 0; i < num_resources; ++i) {
            available[i] += request[i];
            allocation[process_id][i] -= request[i];
            need[process_id][i] += request[i];
        }
    }
}
```

**Output :-**



```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation matrix:
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
Enter the maximum need matrix:
Process 0: 7 5 3
Process 1: 3 2 2
Process 2: 9 0 2
Process 3: 4 2 2
Process 4: 5 3 3
Enter the available resources: 3 3 2
Safe Sequence: 1 3 4 0 2
System is in a safe state.
```

**Deadlock occurred**



```
Enter the number of processes: 5
Enter the number of resources: 3
Enter the allocation matrix:
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
Enter the maximum need matrix:
Process 0: 7 5 3
Process 1: 10 2 2
Process 2: 9 0 2
Process 3: 4 2 2
Process 4: 5 3 3
Enter the available resources: 3 3 2
System is in an unsafe state.
```