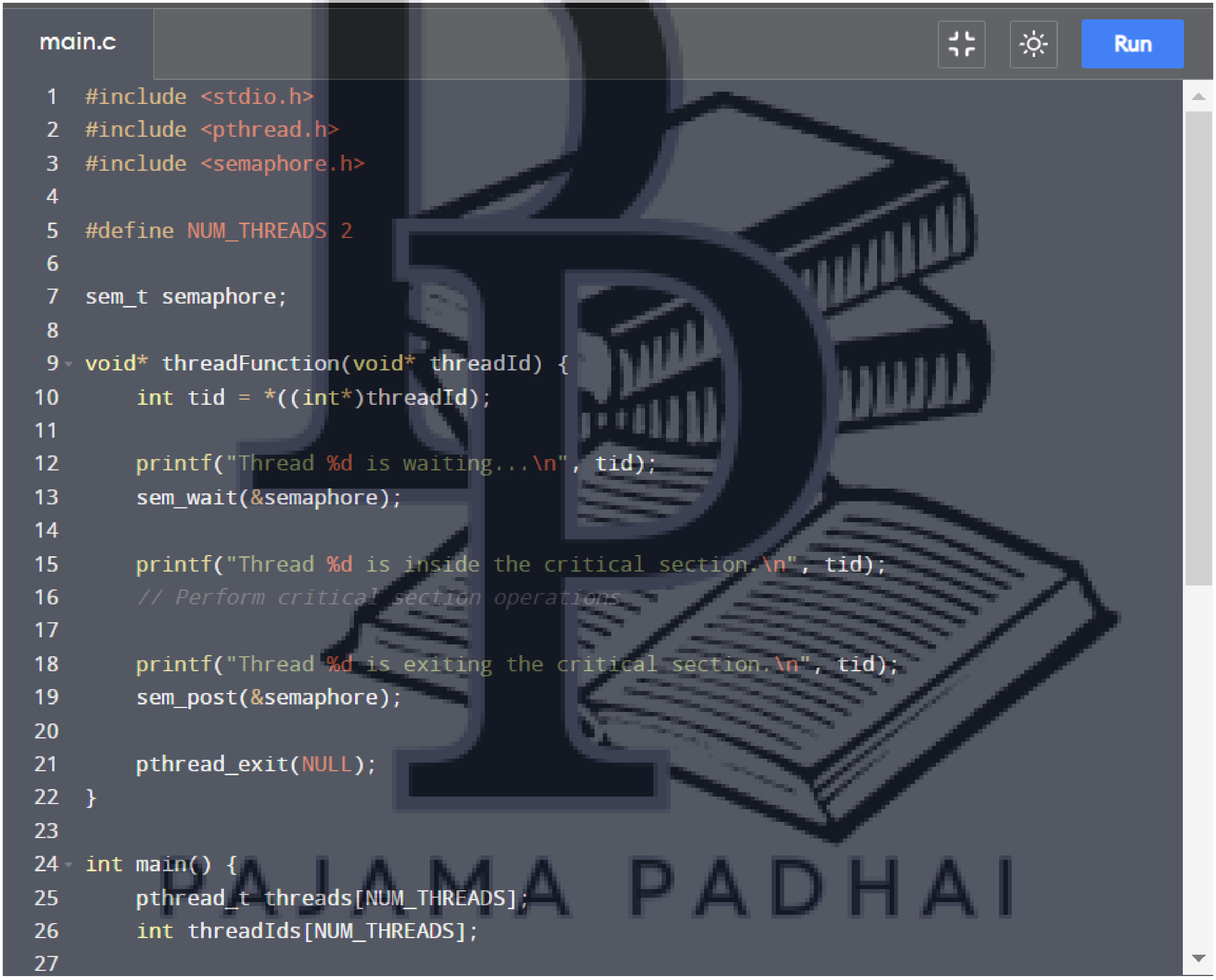
BCSE303P: Operating Systems Lab

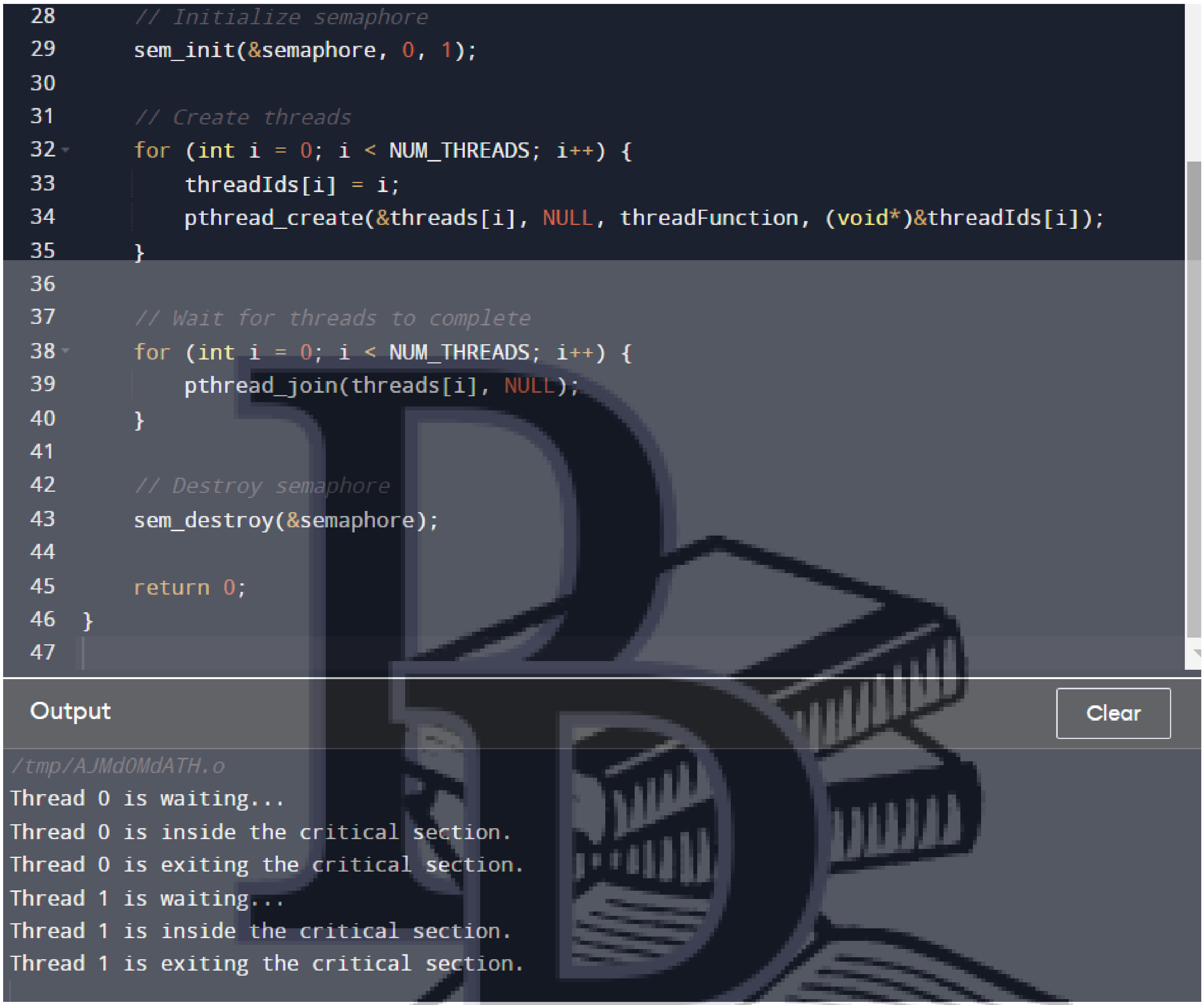
List of Experiments (Programming Languages: C or C++)

3. a. Implement process synchronization using semaphores.
b. Simulation of Banker s algorithm to check whether the given system is in safe state or not. Also check whether addition resource requested can be granted immediately.

3 a)



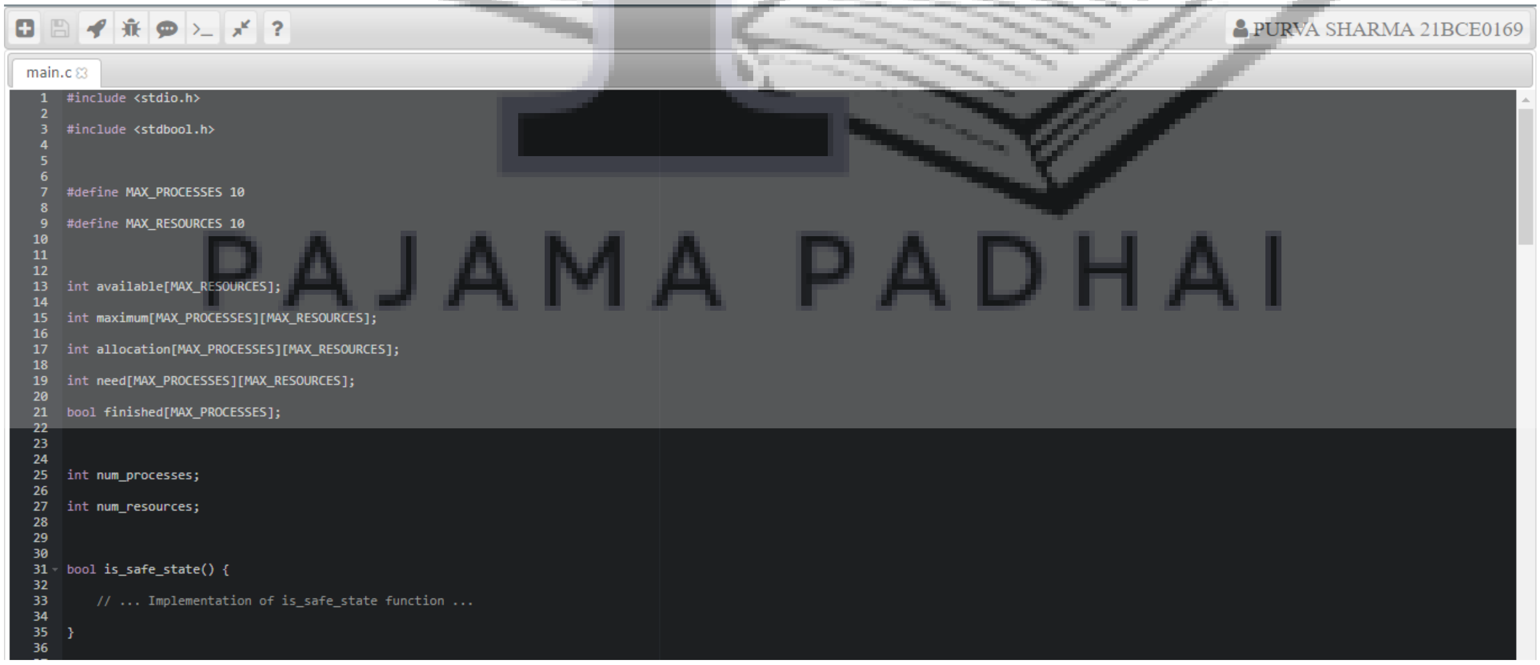
```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define NUM_THREADS 2

sem_t semaphore;

void* threadFunction(void* threadId) {
    int tid = *((int*)threadId);

    printf("Thread %d is waiting...\n", tid);
    sem_wait(&semaphore);

    printf("Thread %d is inside the critical section.\n", tid);
    // Perform critical section operations

    printf("Thread %d is exiting the critical section.\n", tid);
    sem_post(&semaphore);

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int threadIds[NUM_THREADS];

```



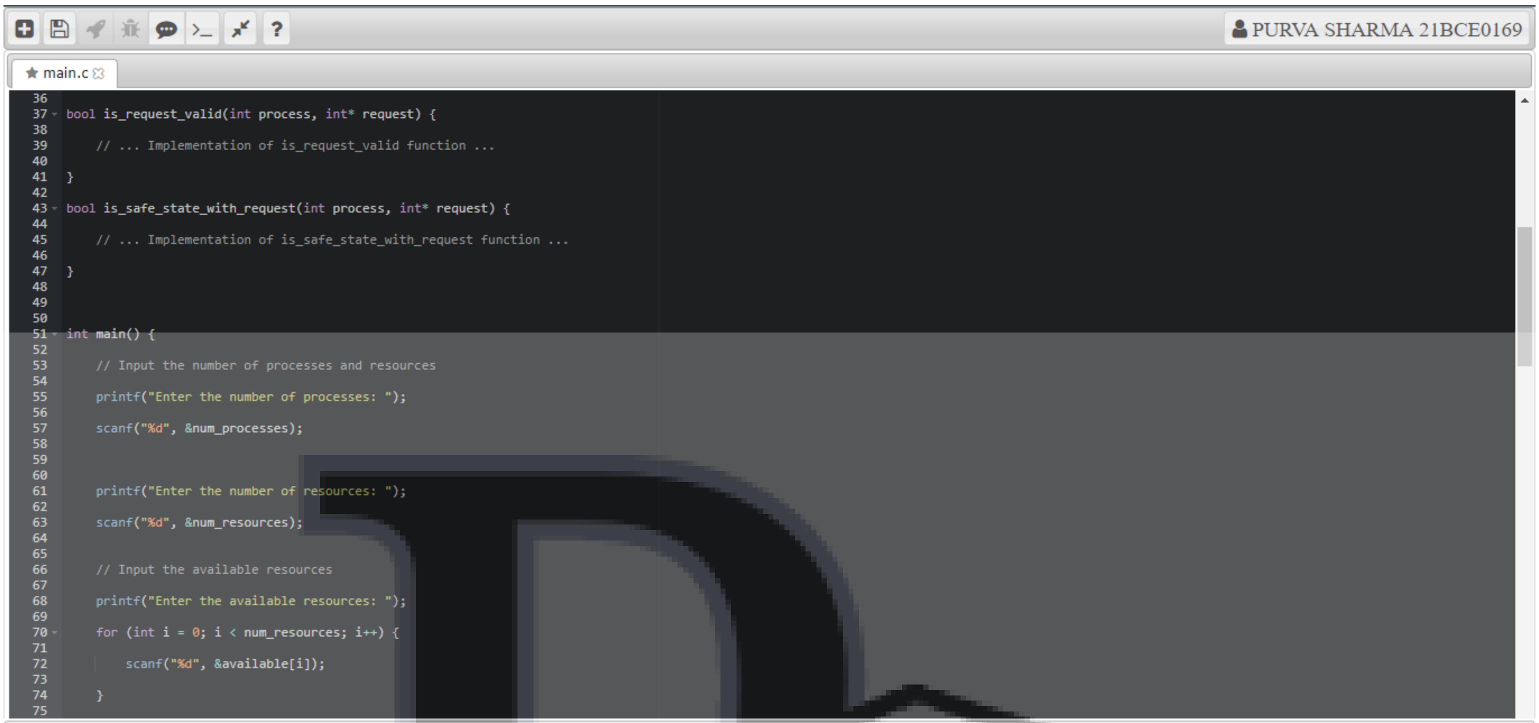
```c
    // Initialize semaphore
    sem_init(&semaphore, 0, 1);

    // Create threads
    for (int i = 0; i < NUM_THREADS; i++) {
        threadIds[i] = i;
        pthread_create(&threads[i], NULL, threadFunction, (void*)&threadIds[i]);
    }

    // Wait for threads to complete
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    // Destroy semaphore
    sem_destroy(&semaphore);

    return 0;
}

```

```
/tmp/AJMd0MdATH.o
Thread 0 is waiting...
Thread 0 is inside the critical section.
Thread 0 is exiting the critical section.
Thread 1 is waiting...
Thread 1 is inside the critical section.
Thread 1 is exiting the critical section.
```

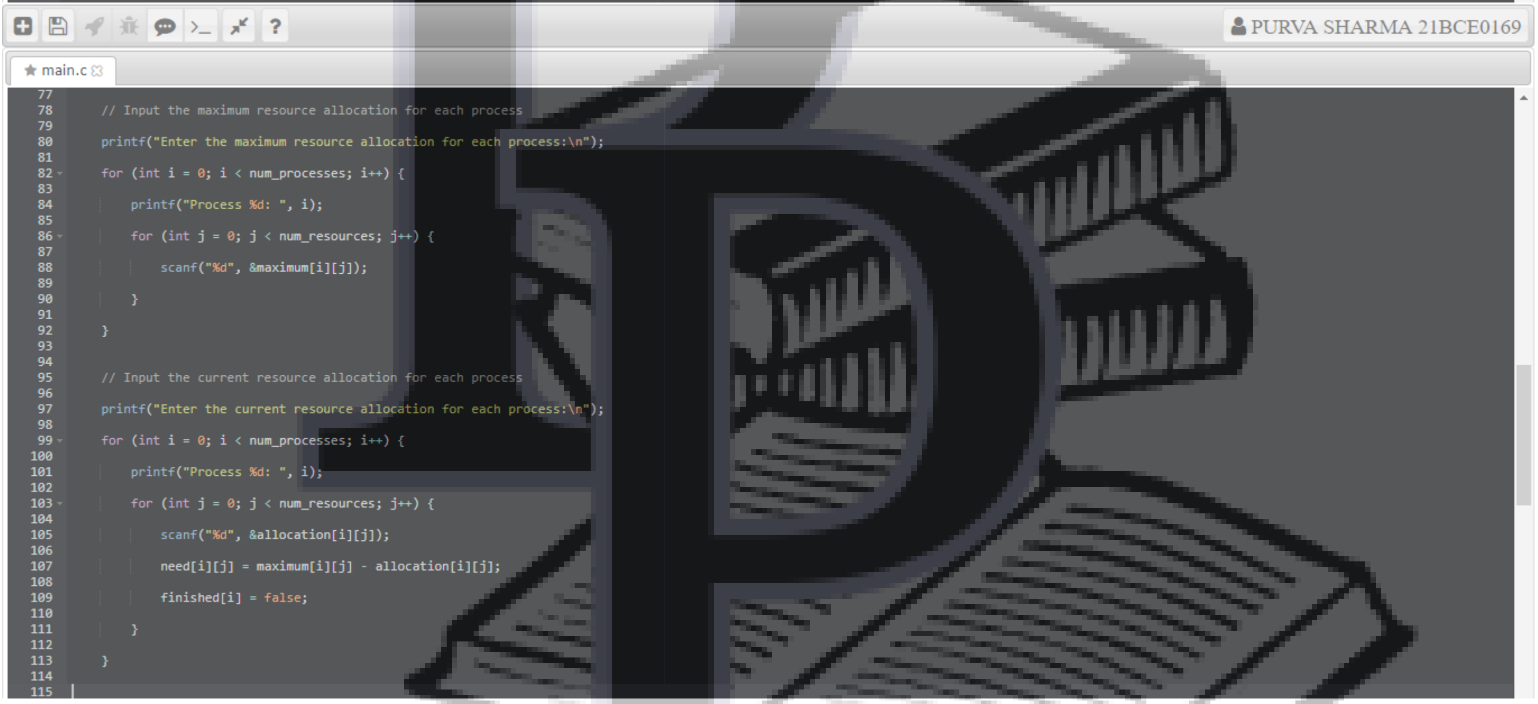
b)



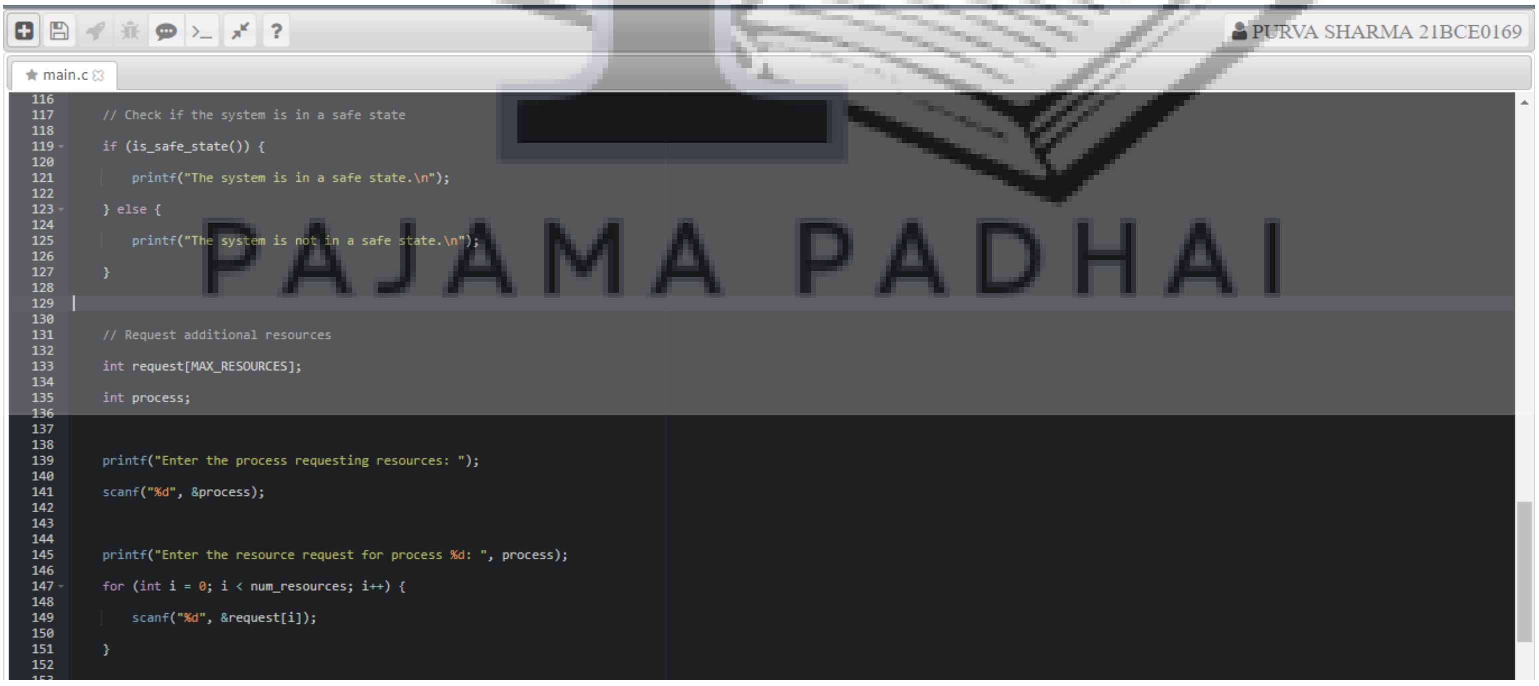
```c
#include <stdio.h>

#include <stdbool.h>



#define MAX_PROCESSES 10

#define MAX_RESOURCES 10



int available[MAX_RESOURCES];

int maximum[MAX_PROCESSES][MAX_RESOURCES];

int allocation[MAX_PROCESSES][MAX_RESOURCES];

int need[MAX_PROCESSES][MAX_RESOURCES];

bool finished[MAX_PROCESSES];



int num_processes;

int num_resources;



bool is_safe_state() {

    // ... Implementation of is_safe_state function ...

}

```



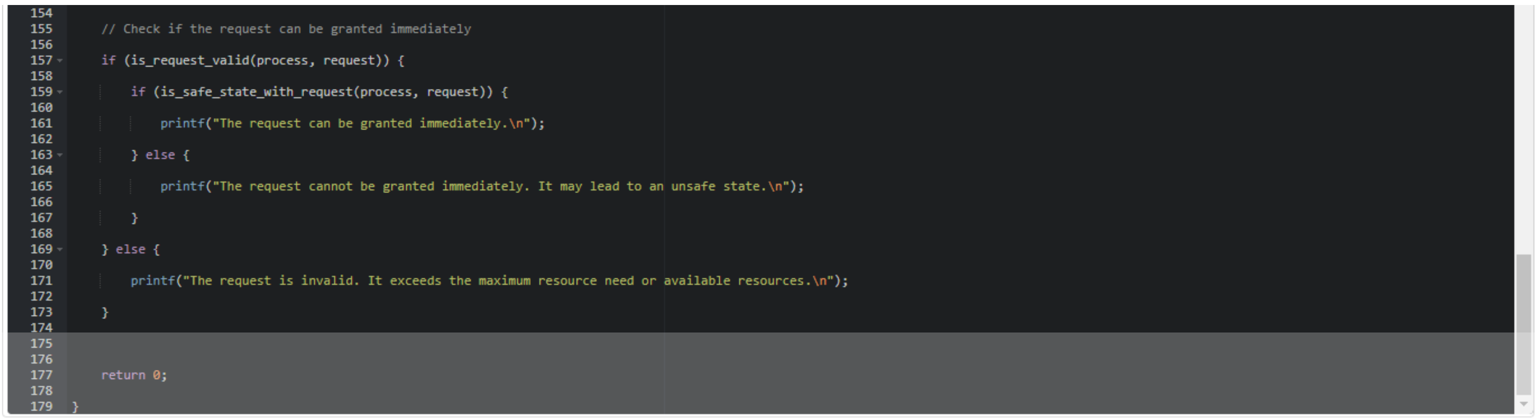
```c
bool is_request_valid(int process, int* request) {
    // ... Implementation of is_request_valid function ...
}

bool is_safe_state_with_request(int process, int* request) {
    // ... Implementation of is_safe_state_with_request function ...
}


int main() {
    // Input the number of processes and resources
    printf("Enter the number of processes: ");
    scanf("%d", &num_processes);

    printf("Enter the number of resources: ");
    scanf("%d", &num_resources);

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



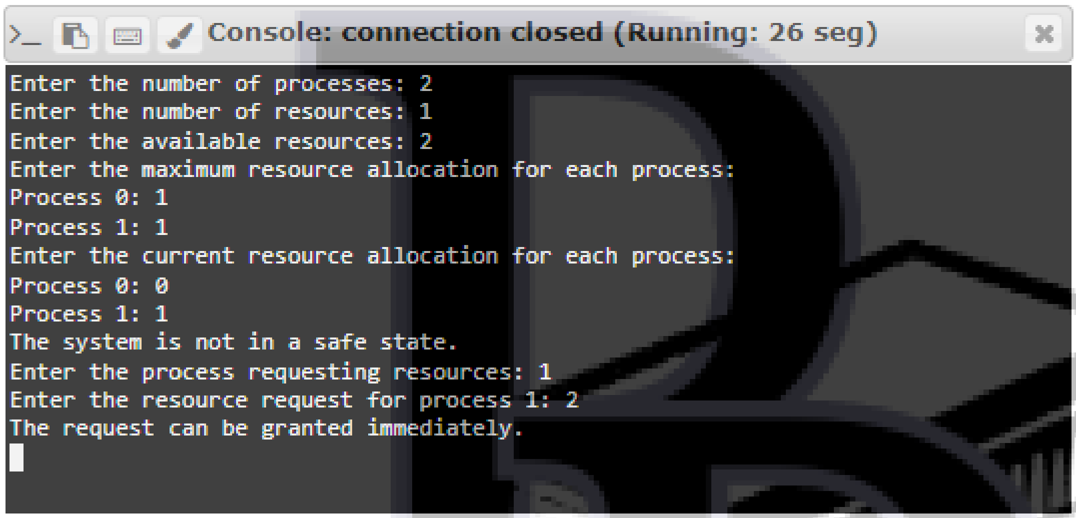
```c
    // Input the maximum resource allocation for each process
    printf("Enter the maximum resource allocation for each process:\n");
    for (int i = 0; i < num_processes; i++) {
        printf("Process %d: ", i);
        for (int j = 0; j < num_resources; j++) {
            scanf("%d", &maximum[i][j]);
        }
    }

    // Input the current resource allocation for each process
    printf("Enter the current resource allocation for each process:\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]);
            need[i][j] = maximum[i][j] - allocation[i][j];
            finished[i] = false;
        }
    }
```



```c
    // Check if the system is in a safe state
    if (is_safe_state()) {
        printf("The system is in a safe state.\n");
    } else {
        printf("The system is not in a safe state.\n");
    }

    // Request additional resources
    int request[MAX_RESOURCES];
    int process;

    printf("Enter the process requesting resources: ");
    scanf("%d", &process);

    printf("Enter the resource request for process %d: ", process);
    for (int i = 0; i < num_resources; i++) {
        scanf("%d", &request[i]);
    }
```



```c

    // Check if the request can be granted immediately

    if (is_request_valid(process, request)) {

        if (is_safe_state_with_request(process, request)) {

            printf("The request can be granted immediately.\n");

        } else {

            printf("The request cannot be granted immediately. It may lead to an unsafe state.\n");

        }

    } else {

        printf("The request is invalid. It exceeds the maximum resource need or available resources.\n");

    }


    return 0;

}
```



```
Enter the number of processes: 2
Enter the number of resources: 1
Enter the available resources: 2
Enter the maximum resource allocation for each process:
Process 0: 1
Process 1: 1
Enter the current resource allocation for each process:
Process 0: 0
Process 1: 1
The system is not in a safe state.
Enter the process requesting resources: 1
Enter the resource request for process 1: 2
The request can be granted immediately.
```