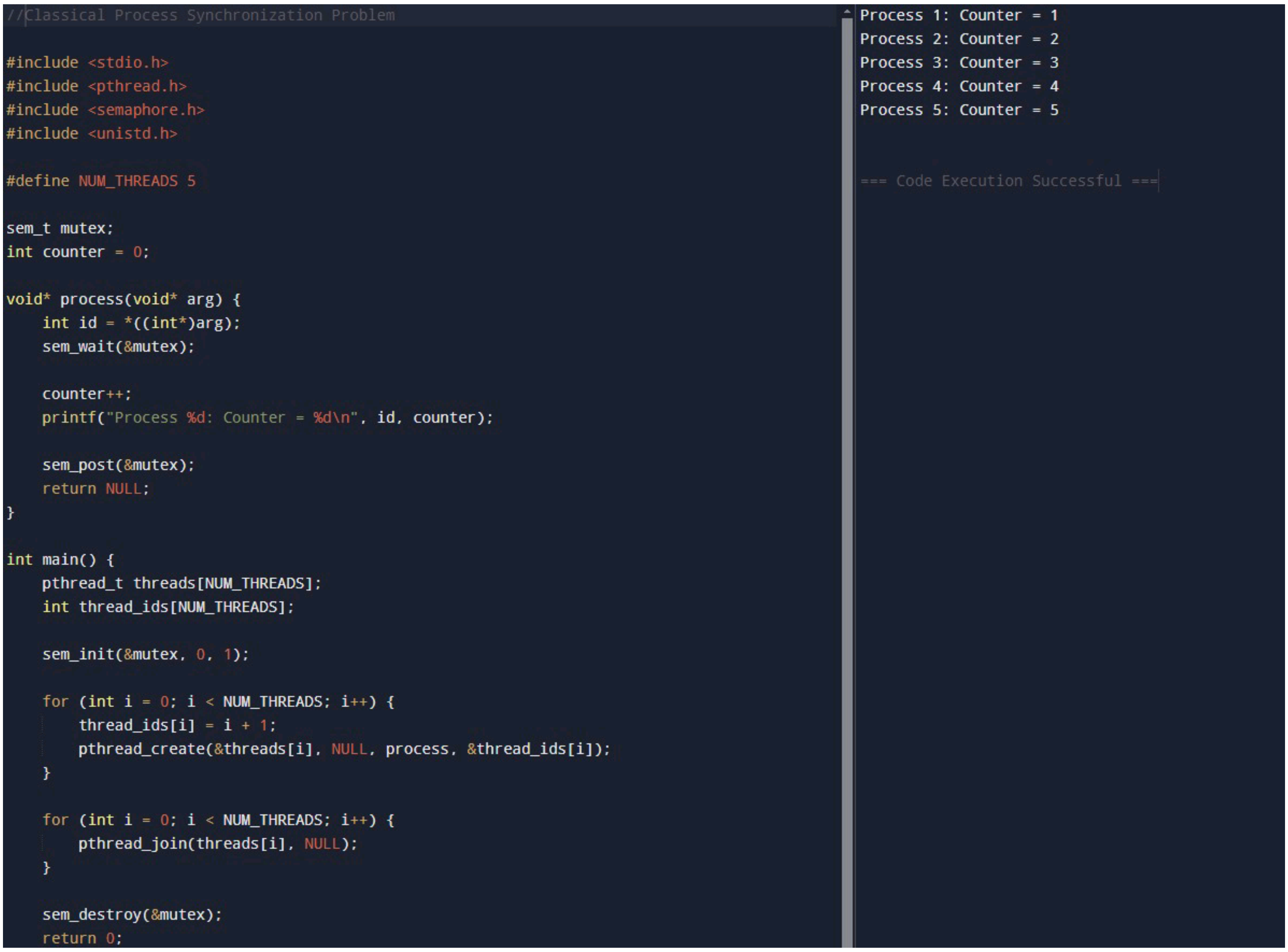
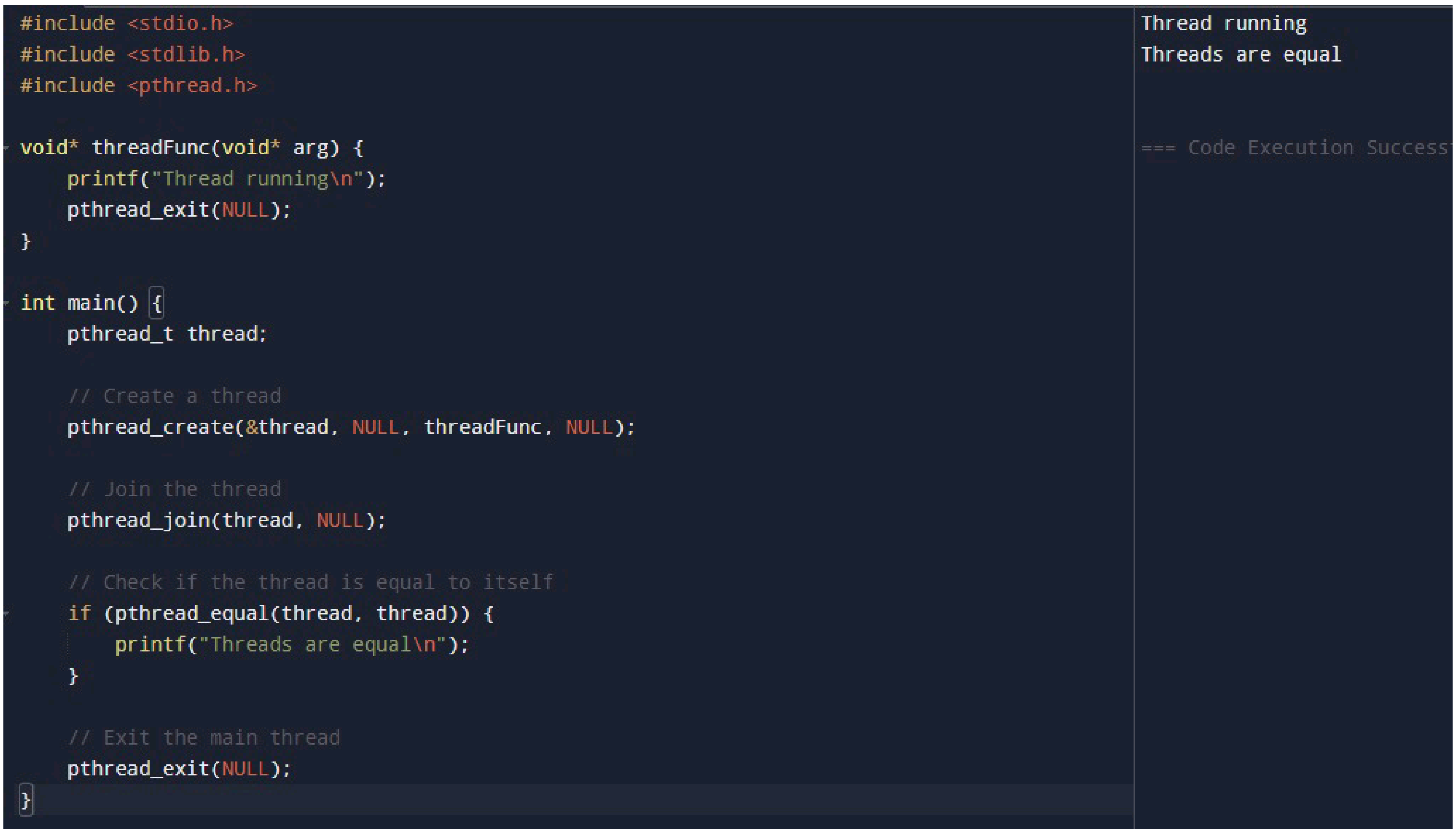


```c
//Classical Process Synchronization Problem

#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define NUM_THREADS 5

sem_t mutex;
int counter = 0;

void* process(void* arg) {
    int id = *((int*)arg);
    sem_wait(&mutex);

    counter++;
    printf("Process %d: Counter = %d\n", id, counter);

    sem_post(&mutex);
    return NULL;
}

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

    sem_init(&mutex, 0, 1);

    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i + 1;
        pthread_create(&threads[i], NULL, process, &thread_ids[i]);
    }

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

    sem_destroy(&mutex);
    return 0;
```

```
Process 1: Counter = 1
Process 2: Counter = 2
Process 3: Counter = 3
Process 4: Counter = 4
Process 5: Counter = 5


=== Code Execution Successful ===
```



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

void* threadFunc(void* arg) {
    printf("Thread running\n");
    pthread_exit(NULL);
}

int main() {
    pthread_t thread;

    // Create a thread
    pthread_create(&thread, NULL, threadFunc, NULL);

    // Join the thread
    pthread_join(thread, NULL);

    // Check if the thread is equal to itself
    if (pthread_equal(thread, thread)) {
        printf("Threads are equal\n");
    }

    // Exit the main thread
    pthread_exit(NULL);
}
```

```
Thread running
Threads are equal


=== Code Execution Success
```