Name: K.SaiKrishna

Reg-No: 192311106

19. Design a C program to implement process synchronization using mutex locks.

Aim:

The aim of this C program is to demonstrate process synchronization using mutex locks, ensuring that multiple processes do not interfere with each other when accessing shared resources.

Algorithm:

1. Create a mutex lock.
2. Initialize shared resources.
3. Define the critical section.
4. Use `pthread_mutex_lock()` to lock the mutex before accessing the shared resource.
5. Use `pthread_mutex_unlock()` to unlock the mutex after accessing the shared resource.
6. Perform synchronization to avoid race conditions.

Procedure:

1. Create multiple threads (representing processes).
2. Each thread will access a shared resource (e.g., incrementing a counter).
3. Mutex locks will ensure only one thread modifies the resource at a time.

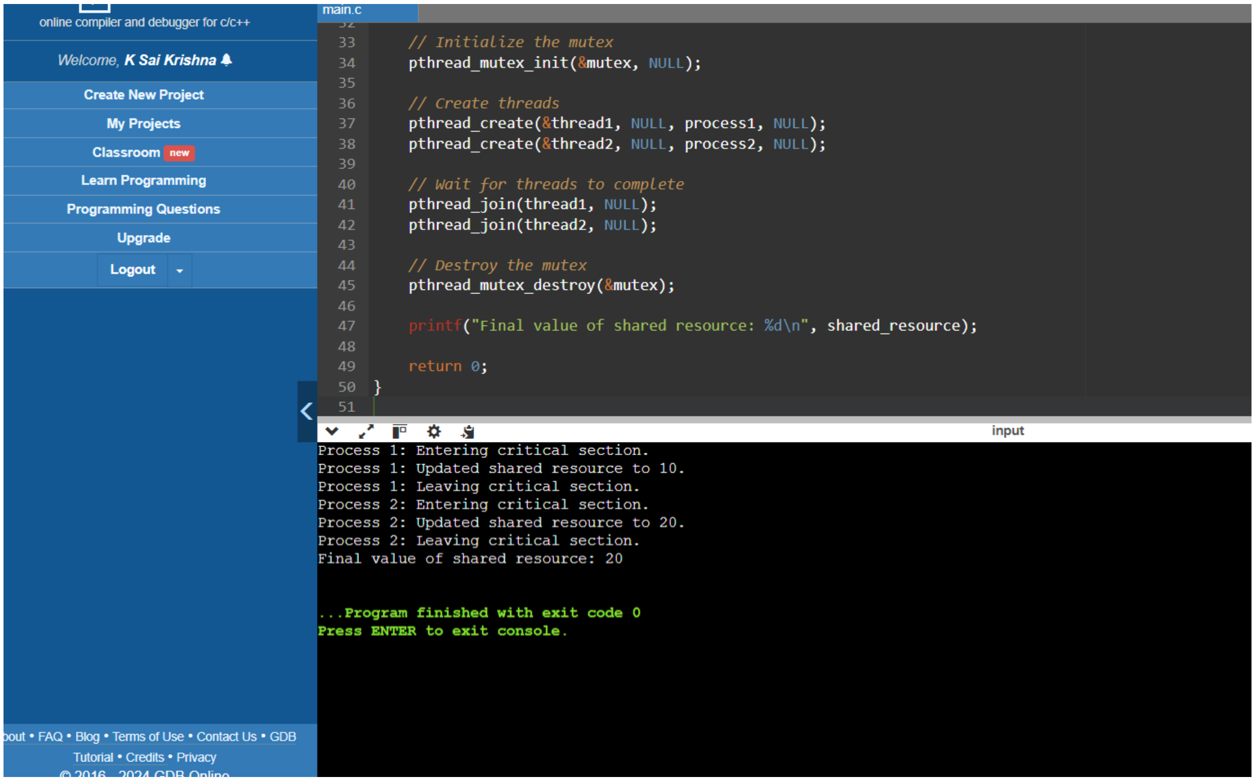
Code:

```
#include <stdio.h>

#include <pthread.h>

pthread_mutex_t mutex;

int shared_resource = 0;



void* increment(void* arg) {

    pthread_mutex_lock(&mutex);

    shared_resource++;

    printf("Shared resource: %d\n", shared_resource);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main() {
    pthread_t threads[5];
    pthread_mutex_init(&mutex, NULL);

    for (int i = 0; i < 5; i++) {
        pthread_create(&threads[i], NULL, increment, NULL);
    }

    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&mutex);
    return 0;
}
```

Result:

The program creates five threads, each incrementing the shared resource. The mutex ensures that only one thread can modify the resource at a time, avoiding race conditions and ensuring that the final value of `shared_resource` is 5.

Output:



```c
    // Initialize the mutex
    pthread_mutex_init(&mutex, NULL);

    // Create threads
    pthread_create(&thread1, NULL, process1, NULL);
    pthread_create(&thread2, NULL, process2, NULL);

    // Wait for threads to complete
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    // Destroy the mutex
    pthread_mutex_destroy(&mutex);

    printf("Final value of shared resource: %d\n", shared_resource);

    return 0;
}
```

```
Process 1: Entering critical section.
Process 1: Updated shared resource to 10.
Process 1: Leaving critical section.
Process 2: Entering critical section.
Process 2: Updated shared resource to 20.
Process 2: Leaving critical section.
Final value of shared resource: 20


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

The program creates five threads, each incrementing the shared resource. The mutex ensures that only one thread can modify the resource at a time, avoiding race conditions and ensuring that the final value of `shared_resource` is 5.