**KASIRALA DURGA-192324073**

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);

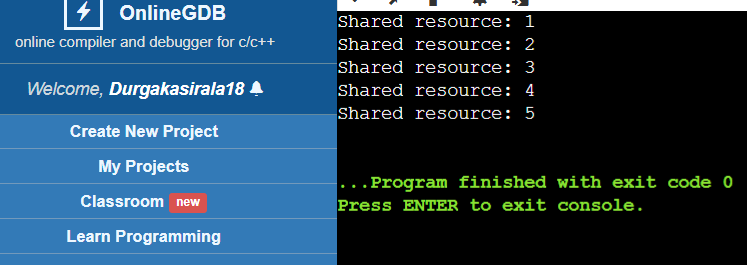
return 0;

}

### 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:**

****