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

**Aim:**

To design a C program that demonstrates process synchronization using mutex locks to avoid race conditions in a shared resource scenario.

**Algorithm:**

1. **Initialize a Mutex Lock**:
   * The mutex will be used to control access to the shared resource.
2. **Create Threads**:
   * Multiple threads will increment or modify the shared resource (counter) a number of times.
3. **Thread Function**:
   * Each thread will acquire the mutex lock before modifying the shared resource, ensuring exclusive access.
   * Once the modification is complete, the thread will release the mutex lock, allowing other threads to access the resource.
4. **Join Threads**:
   * The main thread will wait for all threads to complete using pthread\_join().

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define NUM\_THREADS 5

pthread\_mutex\_t lock;

int shared\_resource = 0;

void\* thread\_function(void\* arg) {

pthread\_mutex\_lock(&lock);

// Critical section

shared\_resource++;

printf("Thread %d incremented shared resource to %d\n", \*(int\*)arg, shared\_resource);

pthread\_mutex\_unlock(&lock);

return NULL;

}

int main() {

pthread\_t threads[NUM\_THREADS];

int thread\_ids[NUM\_THREADS];

pthread\_mutex\_init(&lock, NULL);

for (int i = 0; i < NUM\_THREADS; i++) {

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, thread\_function, &thread\_ids[i]);

}

for (int i = 0; i < NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

pthread\_mutex\_destroy(&lock);

return 0;

}

**Output:**

****