

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:

The screenshot shows an online compiler interface for C/C++. The left sidebar contains navigation links: "Welcome, K Sai Krishna", "Create New Project", "My Projects", "Classroom" (highlighted with a red "new" tag), "Learn Programming", "Programming Questions", "Upgrade", and "Logout". The main editor displays a C program in `main.c` using pthreads to manage a shared resource. The code includes comments and uses `pthread_mutex_t` for synchronization. The output window at the bottom shows the execution of two processes, each entering and leaving a critical section to update a shared resource, resulting in a final value of 20. The program finishes with exit code 0.

```
32
33 // Initialize the mutex
34 pthread_mutex_init(&mutex, NULL);
35
36 // Create threads
37 pthread_create(&thread1, NULL, process1, NULL);
38 pthread_create(&thread2, NULL, process2, NULL);
39
40 // Wait for threads to complete
41 pthread_join(thread1, NULL);
42 pthread_join(thread2, NULL);
43
44 // Destroy the mutex
45 pthread_mutex_destroy(&mutex);
46
47 printf("Final value of shared resource: %d\n", shared_resource);
48
49 return 0;
50 }
51
```

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.

about • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
© 2016 - 2024 GDB Online

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.