Programming Assignment 3 Report

- Anthony L Carvalho 6365160

- Thales Moraes 6332398

DESCRIPTION

This program solves the critical section problem for two threads using mutex locks.

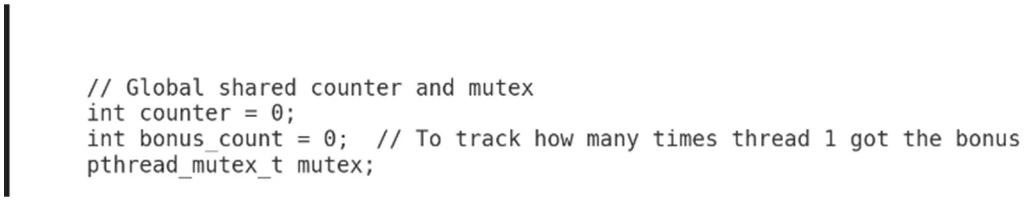
Both threads increment a shared counter up to 2,000,000 times each, but Thread 1

increments the counter by 100 every time counter % 100 == 0 (bonus condition).

The program ensures mutual exclusion, progress, and bounded wait.

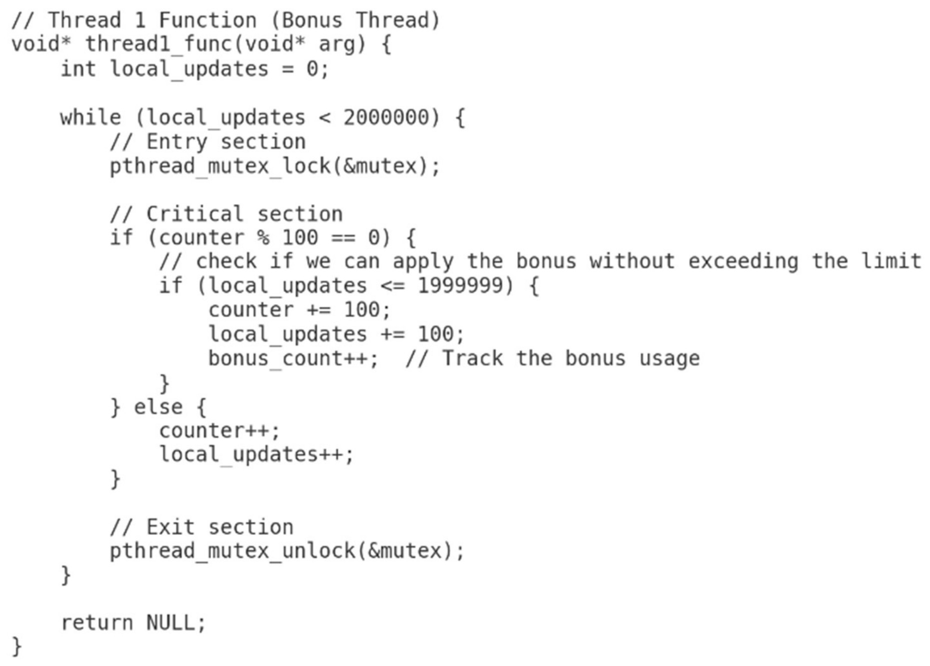
PROGRAM FLOW

The program uses a shared counter and a mutex to ensure that only one thread can access the critical section at a time. A bonus counter tracks how many times Thread 1 is able to apply the bonus increment.



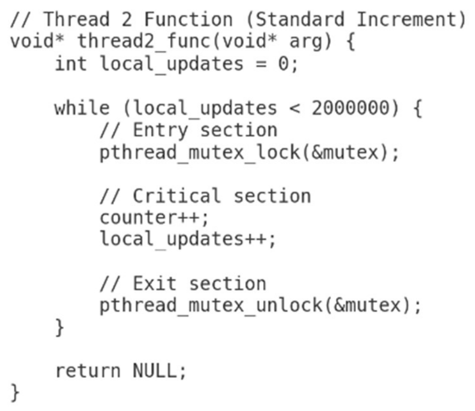
```
// Global shared counter and mutex
int counter = 0;
int bonus_count = 0;  // To track how many times thread 1 got the bonus
pthread_mutex_t mutex;
```

Thread 1 is responsible for checking whether the counter is divisible by 100. If that is true, it increments the counter by 100 and increases the bonus count. The function uses a mutex lock to prevent race conditions and ensures safe access to the shared counter.



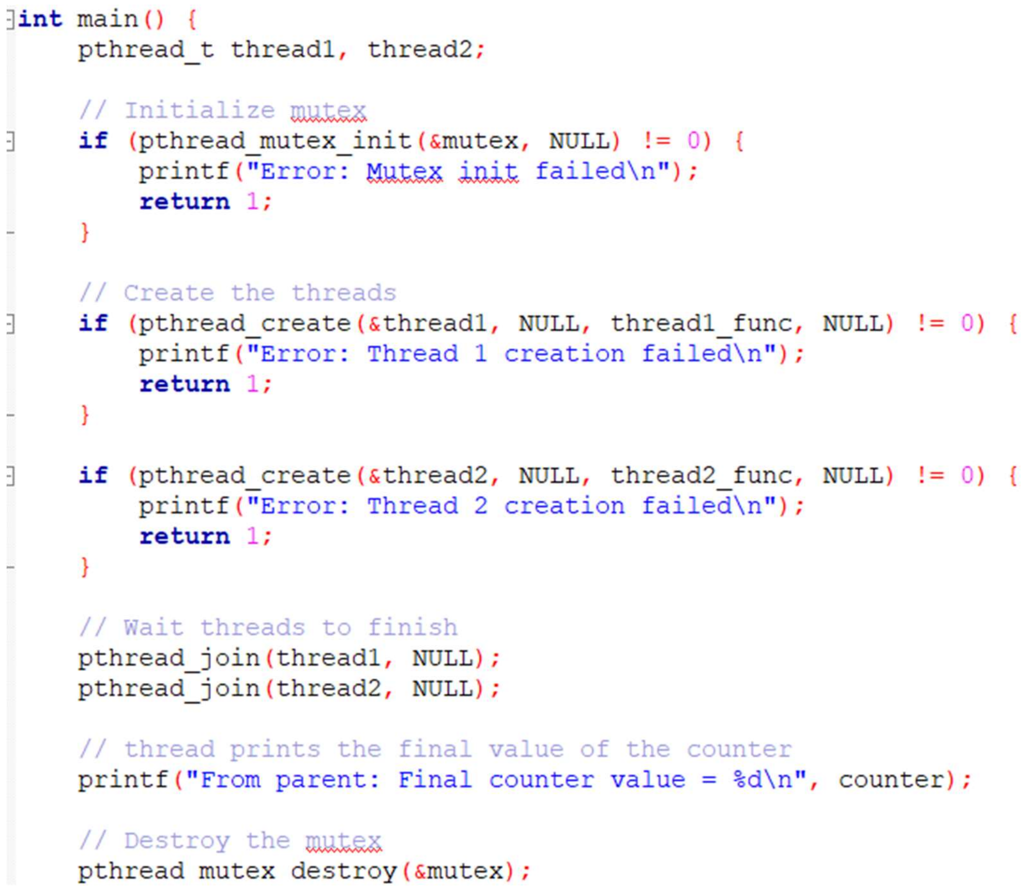
```c
// Thread 1 Function (Bonus Thread)
void* thread1_func(void* arg) {
    int local_updates = 0;

    while (local_updates < 2000000) {
        // Entry section
        pthread_mutex_lock(&mutex);

        // Critical section
        if (counter % 100 == 0) {
            // check if we can apply the bonus without exceeding the limit
            if (local_updates <= 1999999) {
                counter += 100;
                local_updates += 100;
                bonus_count++;  // Track the bonus usage
            }
        } else {
            counter++;
            local_updates++;
        }

        // Exit section
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}
```

Thread 2 simply increments the counter by 1, like Thread 1, but without any additional conditions. It also uses a mutex lock to ensure mutual exclusion when updating the shared counter.



```c
// Thread 2 Function (Standard Increment)
void* thread2_func(void* arg) {
    int local_updates = 0;

    while (local_updates < 2000000) {
        // Entry section
        pthread_mutex_lock(&mutex);

        // Critical section
        counter++;
        local_updates++;

        // Exit section
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}
```

The main function initializes the mutex and creates two threads: Thread 1 and Thread 2. After starting the threads, the program waits for both to complete their tasks using the pthread_join function. Once the threads are finished, the mutex is destroyed, and the final values of the counter and bonus count are printed.

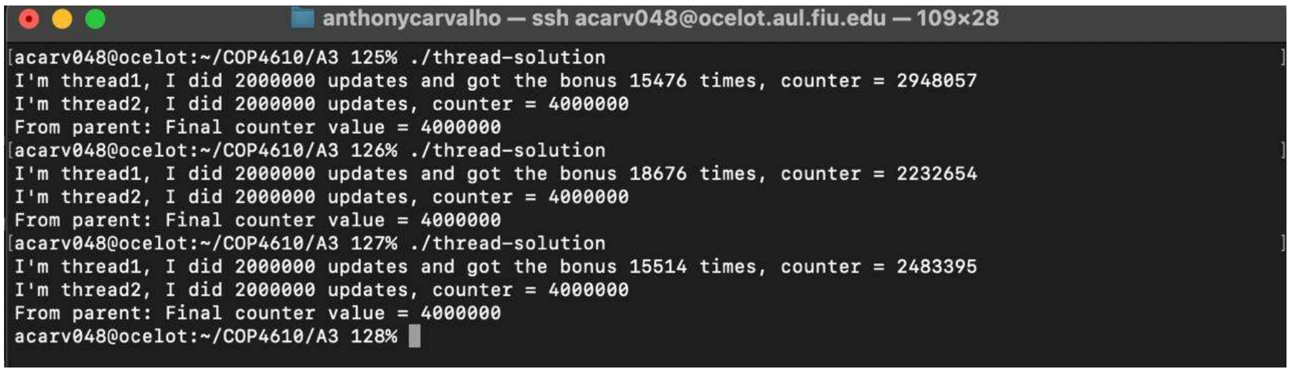


```c
int main() {
    pthread_t thread1, thread2;

    // Initialize mutex
    if (pthread_mutex_init(&mutex, NULL) != 0) {
        printf("Error: Mutex init failed\n");
        return 1;
    }

    // Create the threads
    if (pthread_create(&thread1, NULL, thread1_func, NULL) != 0) {
        printf("Error: Thread 1 creation failed\n");
        return 1;
    }

    if (pthread_create(&thread2, NULL, thread2_func, NULL) != 0) {
        printf("Error: Thread 2 creation failed\n");
        return 1;
    }

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

    // thread prints the final value of the counter
    printf("From parent: Final counter value = %d\n", counter);

    // Destroy the mutex
    pthread mutex destroy(&mutex);
```

Proof: Critical Section Problem Conditions

This C program satisfies all three conditions required for solving the critical section problem:

1. **Mutual Exclusion**: Achieved by using the mutex lock to allow only one thread in the critical section at a time.
2. **Progress**: Achieved as threads always attempt to enter the critical section when it becomes available.
3. **Bounded Wait**: Achieved since threads must wait for the mutex to be unlocked, ensuring no thread is starved or delayed indefinitely.



anthonycarvalho — ssh acarv048@ocelot.aul.fiu.edu — 109×28

```
acarv048@ocelot:~/COP4610/A3 125% ./thread-solution
I'm thread1, I did 2000000 updates and got the bonus 15476 times, counter = 2948057
I'm thread2, I did 2000000 updates, counter = 4000000
From parent: Final counter value = 4000000
acarv048@ocelot:~/COP4610/A3 126% ./thread-solution
I'm thread1, I did 2000000 updates and got the bonus 18676 times, counter = 2232654
I'm thread2, I did 2000000 updates, counter = 4000000
From parent: Final counter value = 4000000
acarv048@ocelot:~/COP4610/A3 127% ./thread-solution
I'm thread1, I did 2000000 updates and got the bonus 15514 times, counter = 2483395
I'm thread2, I did 2000000 updates, counter = 4000000
From parent: Final counter value = 4000000
acarv048@ocelot:~/COP4610/A3 128%
```