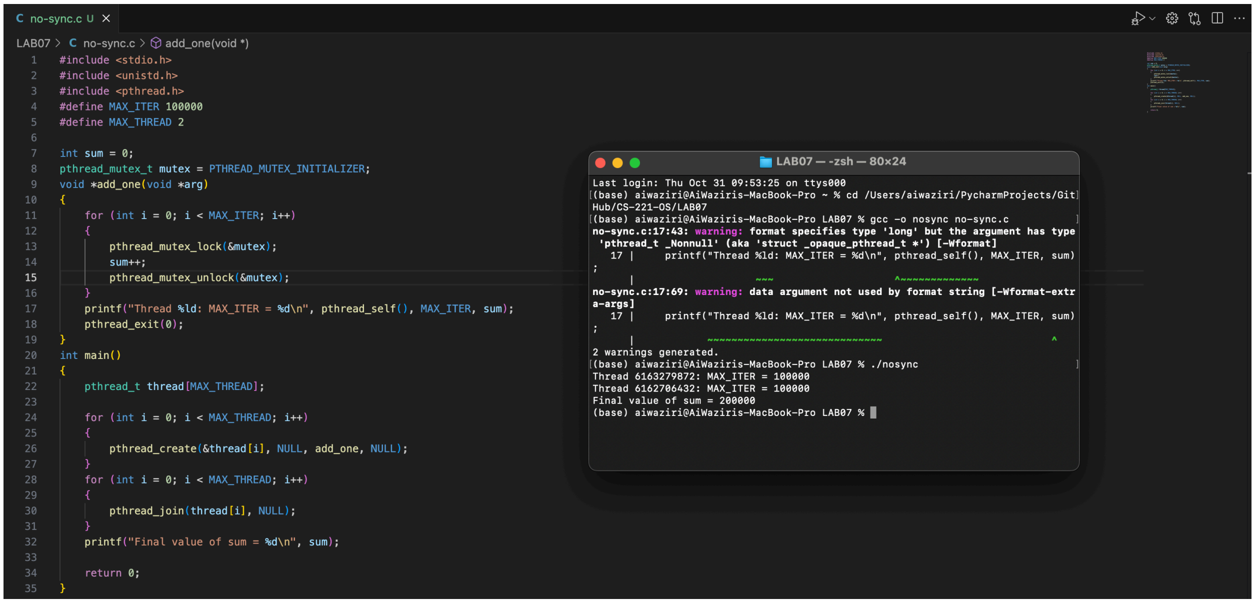
Name: Abubakar Waziri

Id: 4220056

Lab07

Q1:



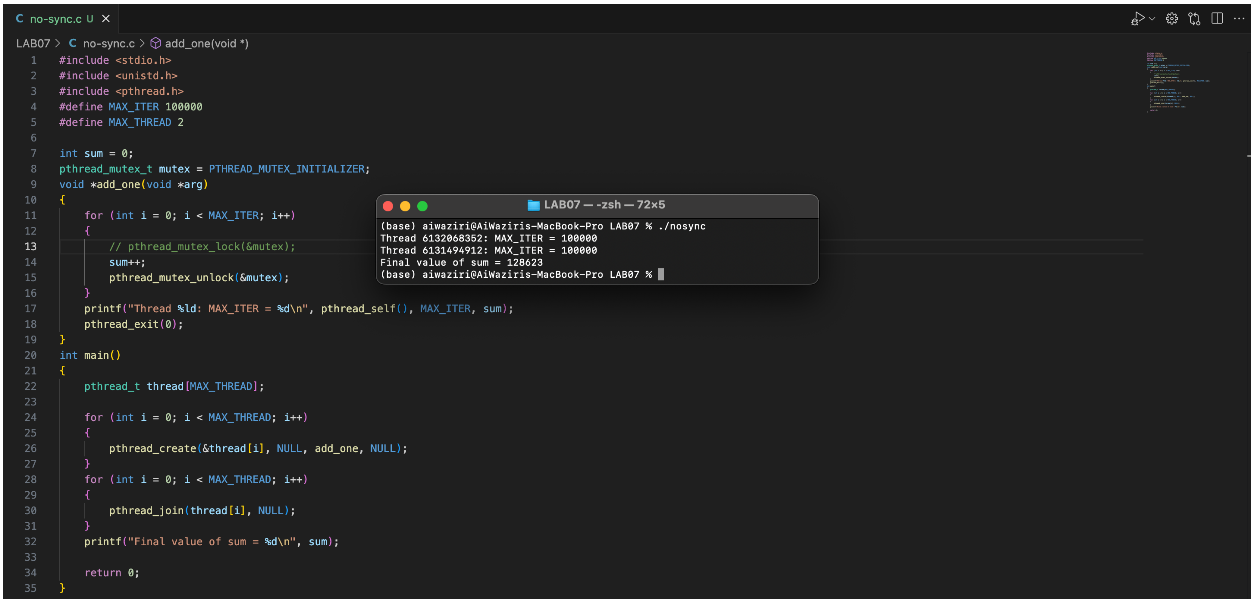
```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define MAX_ITER 100000
#define MAX_THREAD 2

int sum = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void *add_one(void *arg)
{
    for (int i = 0; i < MAX_ITER; i++)
    {
        pthread_mutex_lock(&mutex);
        sum++;
        pthread_mutex_unlock(&mutex);
    }
    printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum);
    pthread_exit(0);
}
int main()
{
    pthread_t thread[MAX_THREAD];

    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_create(&thread[i], NULL, add_one, NULL);
    }
    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_join(thread[i], NULL);
    }
    printf("Final value of sum = %d\n", sum);

    return 0;
}
```

```
Last login: Thu Oct 31 09:53:25 on ttys000
(base) aiwaziri@AiWaziris-MacBook-Pro ~ % cd /Users/aiwaziri/PycharmProjects/Git
Hub/CS-221-OS/LAB07
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % gcc -o nosync no-sync.c
no-sync.c:17:43: warning: format specifies type 'long' but the argument has type
 'pthread_t _Nonnull' (aka 'struct _opaque_pthread_t *') [-Wformat]
   17 |     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum)
;
      |                    ~~~                    ^~~~~~~~~~~~~~
no-sync.c:17:69: warning: data argument not used by format string [-Wformat-extr
a-args]
   17 |     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum)
;
      |            ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                            ^
2 warnings generated.
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % ./nosync
Thread 6163279872: MAX_ITER = 100000
Thread 6162706432: MAX_ITER = 100000
Final value of sum = 200000
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % 
```

Explanation:

- The program creates multiple threads (MAX_THREAD), each of which increments a global variable sum in a loop (MAX_ITER times)
- The pthread_mutex_lock and pthread_mutex_unlock functions are used to ensure that only one thread can increment sum at a time, preventing race conditions.

Q2a:



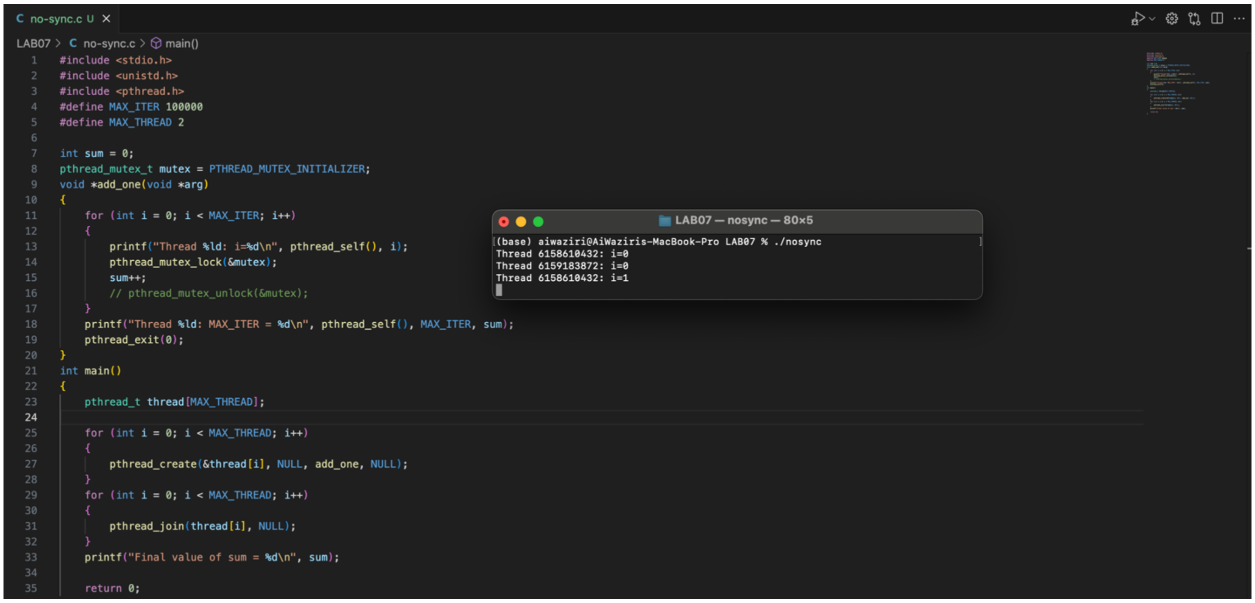
```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define MAX_ITER 100000
#define MAX_THREAD 2

int sum = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void *add_one(void *arg)
{
    for (int i = 0; i < MAX_ITER; i++)
    {
        // pthread_mutex_lock(&mutex);
        sum++;
        pthread_mutex_unlock(&mutex);
    }
    printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum);
    pthread_exit(0);
}
int main()
{
    pthread_t thread[MAX_THREAD];

    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_create(&thread[i], NULL, add_one, NULL);
    }
    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_join(thread[i], NULL);
    }
    printf("Final value of sum = %d\n", sum);

    return 0;
}
```

```
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % ./nosync
Thread 6132068352: MAX_ITER = 100000
Thread 6131494912: MAX_ITER = 100000
Final value of sum = 128623
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % 
```

Explanation of the Output

By commenting out pthread_mutex_lock(&mutex);, the code no longer ensures mutual exclusion when incrementing the sum variable. This means multiple threads can simultaneously access and modify sum, leading to a race condition.

Q2b:



```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#define MAX_ITER 100000
#define MAX_THREAD 2

int sum = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
void *add_one(void *arg)
{
    for (int i = 0; i < MAX_ITER; i++)
    {
        printf("Thread %ld: i=%d\n", pthread_self(), i);
        pthread_mutex_lock(&mutex);
        sum++;
        // pthread_mutex_unlock(&mutex);
    }
    printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum);
    pthread_exit(0);
}
int main()
{
    pthread_t thread[MAX_THREAD];

    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_create(&thread[i], NULL, add_one, NULL);
    }
    for (int i = 0; i < MAX_THREAD; i++)
    {
        pthread_join(thread[i], NULL);
    }
    printf("Final value of sum = %d\n", sum);

    return 0;
```

```
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % ./nosync
Thread 6158610432: i=0
Thread 6159183872: i=0
Thread 6158610432: i=1
```

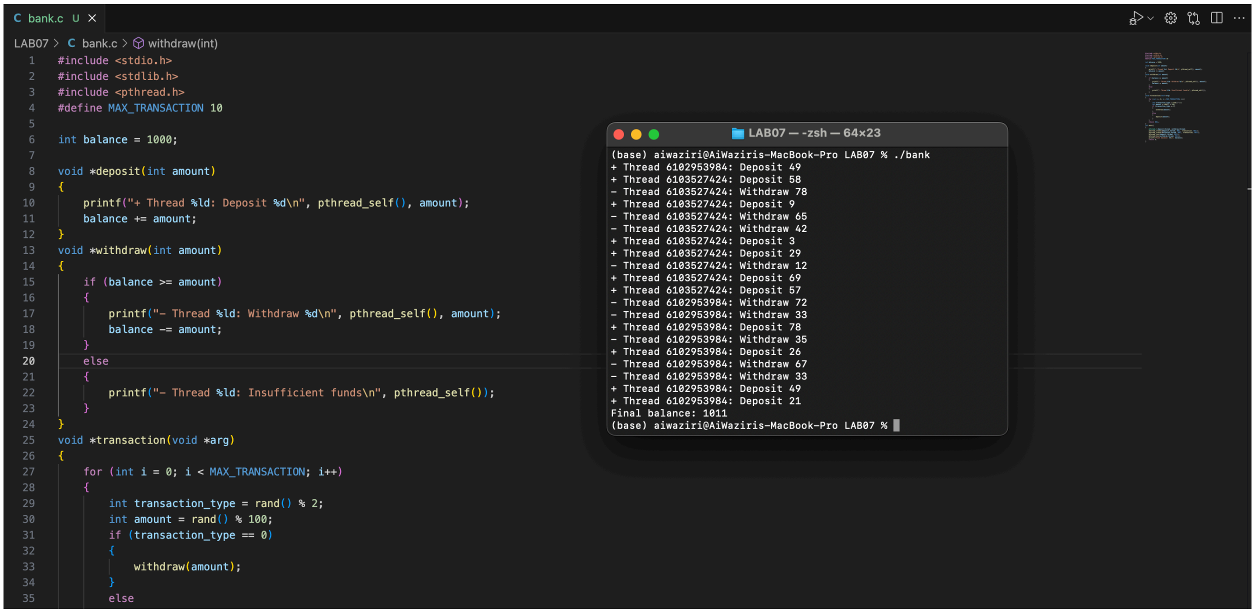
Explanation :

When I commented out pthread_mutex_unlock(&mutex);, the mutex remains locked after the first iteration of the loop in each thread. This causes the following behavior:

1. The first thread to acquire the mutex will print the printf statement, lock the mutex, increment sum, and then get stuck because it never unlocks the mutex.
2. The other thread(s) will print the printf statement before attempting to lock the mutex, but they will get stuck waiting for the mutex to be unlocked, which never happens.

As a result, the program will hang after the first iteration of the loop in the first thread that acquires the mutex. The final printf statement in main will never be reached, and the program will not terminate normally.

Q3:



```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define MAX_TRANSACTION 10

int balance = 1000;

void *deposit(int amount)
{
    printf("+ Thread %ld: Deposit %d\n", pthread_self(), amount);
    balance += amount;
}
void *withdraw(int amount)
{
    if (balance >= amount)
    {
        printf("- Thread %ld: Withdraw %d\n", pthread_self(), amount);
        balance -= amount;
    }
    else
    {
        printf("- Thread %ld: Insufficient funds\n", pthread_self());
    }
}
void *transaction(void *arg)
{
    for (int i = 0; i < MAX_TRANSACTION; i++)
    {
        int transaction_type = rand() % 2;
        int amount = rand() % 100;
        if (transaction_type == 0)
        {
            withdraw(amount);
        }
        else
```

```
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % ./bank
+ Thread 6102953984: Deposit 49
+ Thread 6103527424: Deposit 58
- Thread 6103527424: Withdraw 78
+ Thread 6103527424: Deposit 9
- Thread 6103527424: Withdraw 65
- Thread 6103527424: Withdraw 42
+ Thread 6103527424: Deposit 3
+ Thread 6103527424: Deposit 29
- Thread 6103527424: Withdraw 12
+ Thread 6103527424: Deposit 69
+ Thread 6103527424: Deposit 57
- Thread 6102953984: Withdraw 72
- Thread 6102953984: Withdraw 33
+ Thread 6102953984: Deposit 78
- Thread 6102953984: Withdraw 35
+ Thread 6102953984: Deposit 26
- Thread 6102953984: Withdraw 67
- Thread 6102953984: Withdraw 33
+ Thread 6102953984: Deposit 49
+ Thread 6102953984: Deposit 21
Final balance: 1011
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % 
```

a) Identify the Critical Sections

The critical sections are the parts where the balance variable is accessed and modified. Specifically, these are within the withdraw and deposit functions.

(b) Apply Pthreads Functions to Protect the Critical Sections

To protect the critical sections, we need to use pthread_mutex_lock and pthread_mutex_unlock around the code that modifies the balance variable.