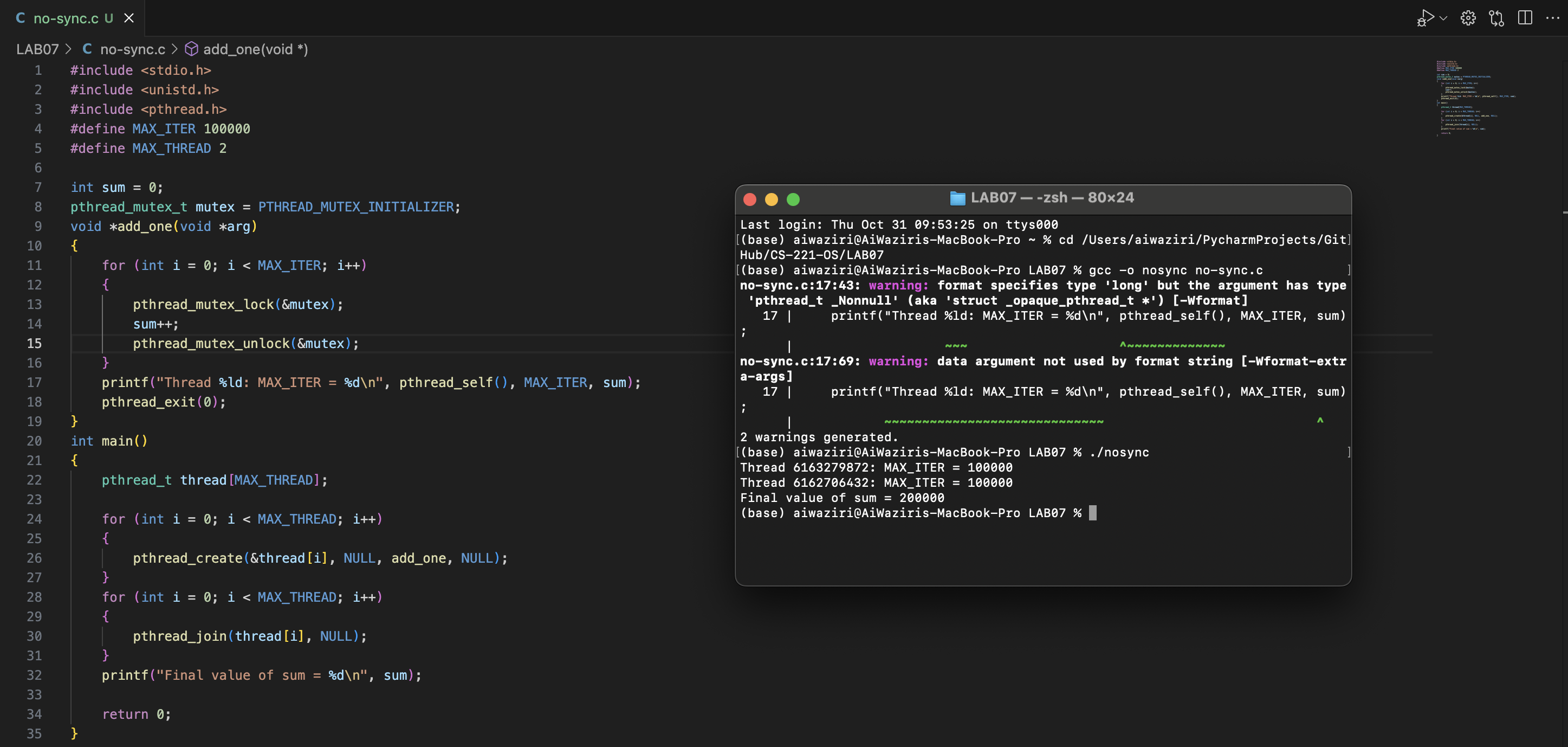
Name: Abubakar Waziri

Id: 4220056

Lab07

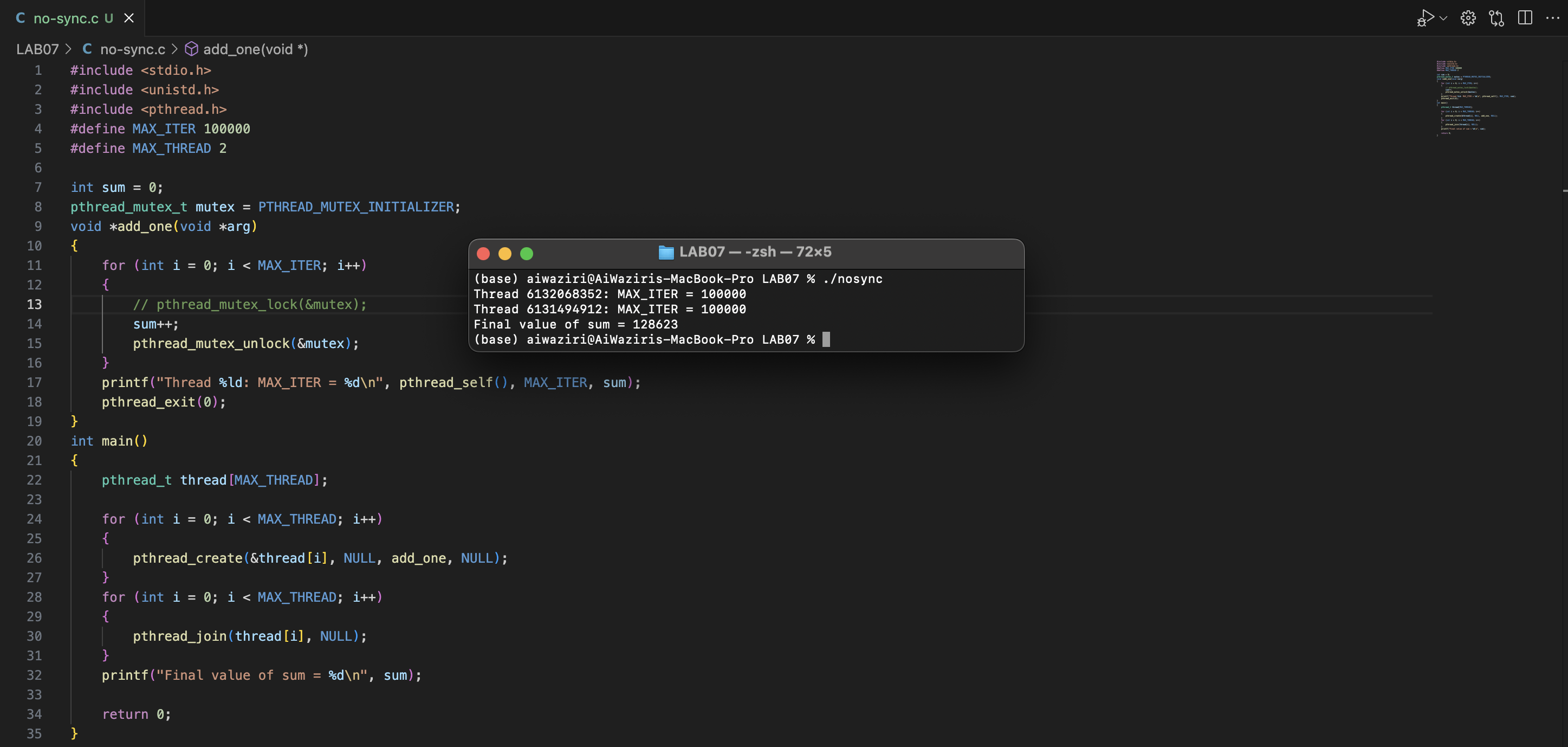
Q1:



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.

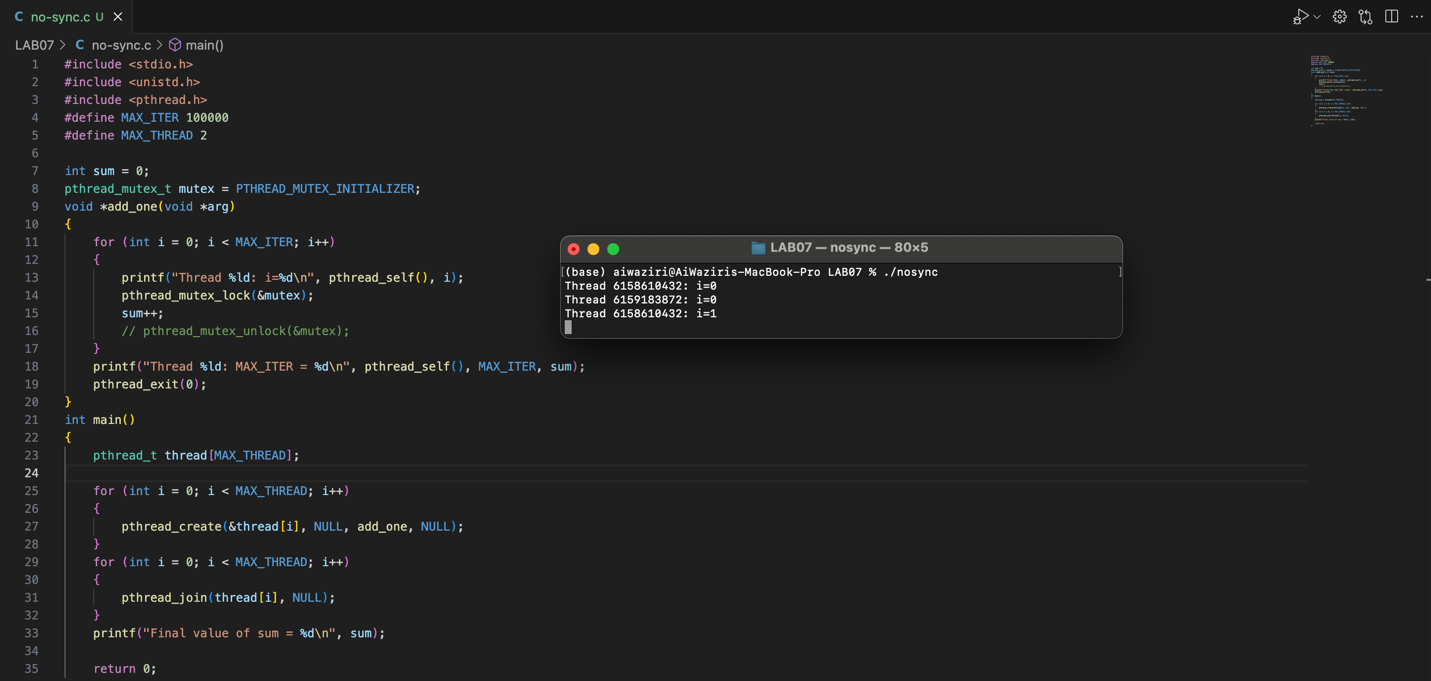
Q2a:



**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:



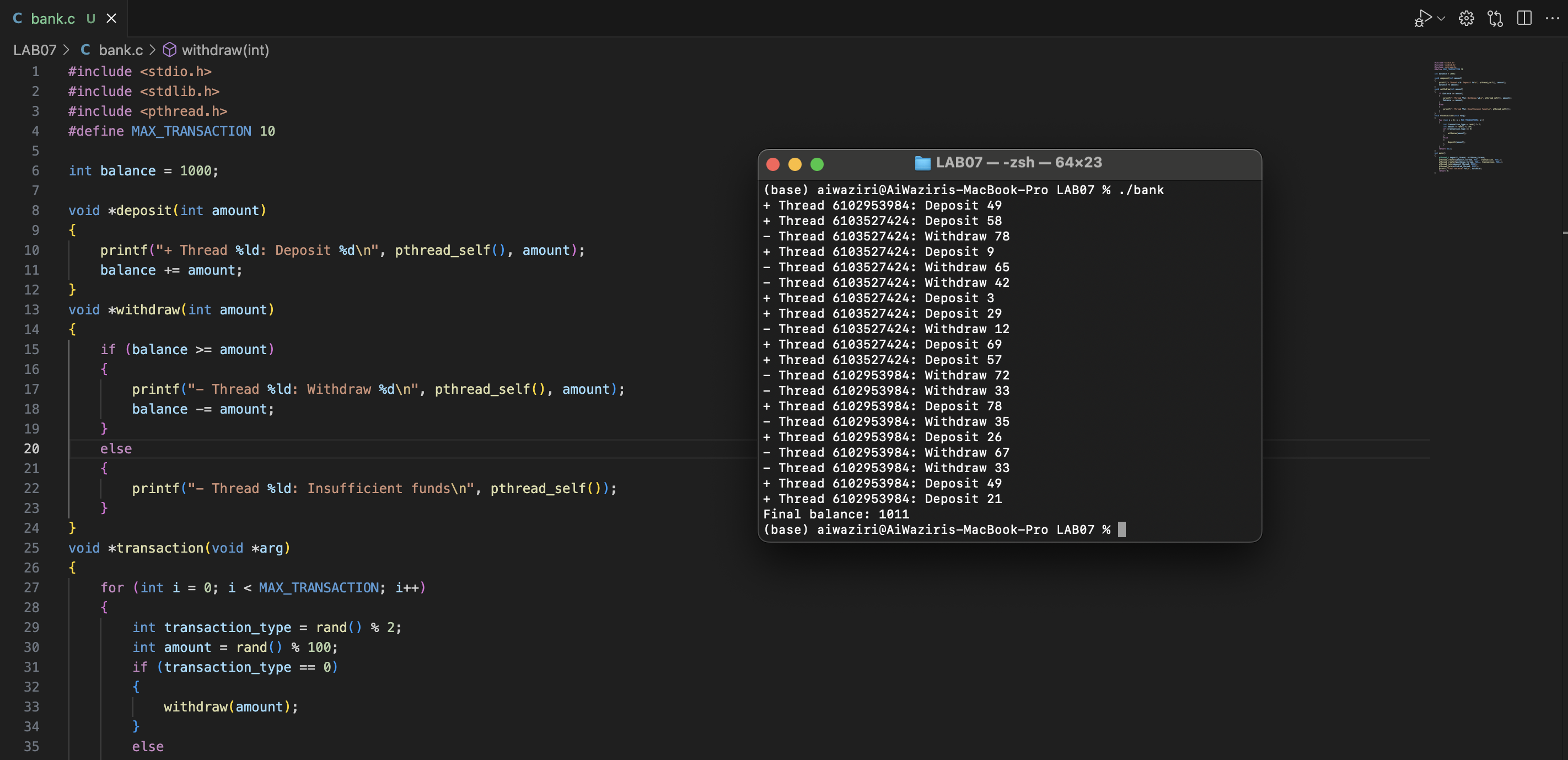
**Explanation :**

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:



**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.