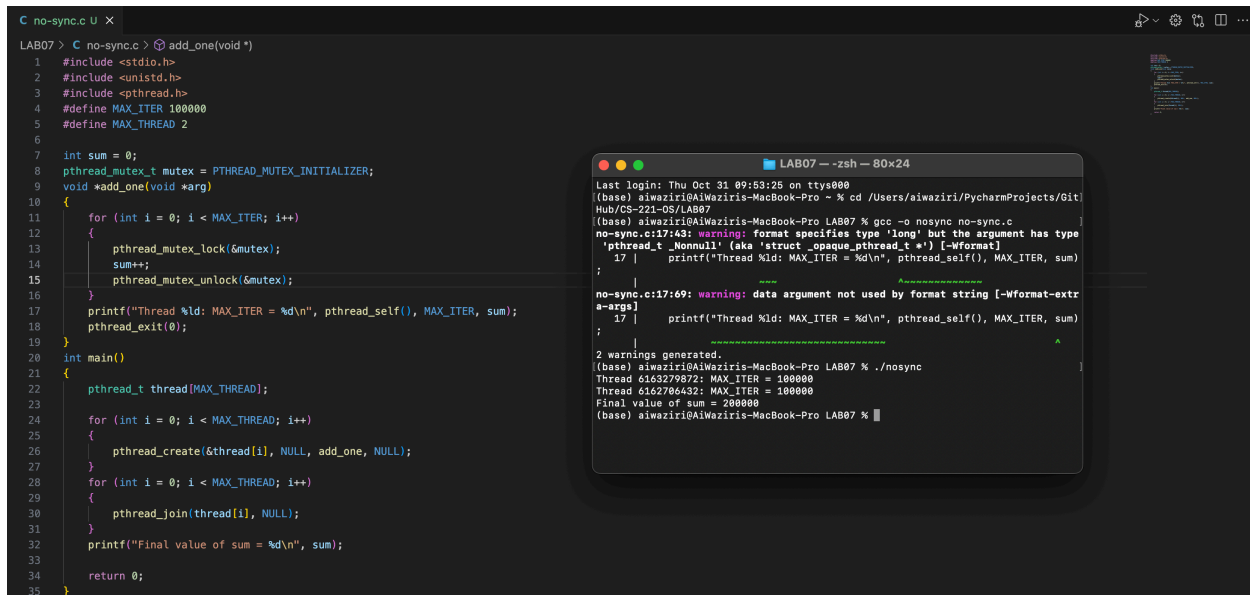


Name: Abubakar Waziri

Id: 4220056

Lab07

Q1:



The screenshot displays a code editor with a C program named `no-sync.c` and its terminal output. The code defines a global variable `sum` and a mutex `mutex`. It creates `MAX_THREAD` threads, each of which increments `sum` `MAX_ITER` times. The program uses `pthread_mutex_lock` and `pthread_mutex_unlock` to ensure mutual exclusion. The terminal output shows the compilation and execution of the program, including a warning about a format string and the final value of `sum`.

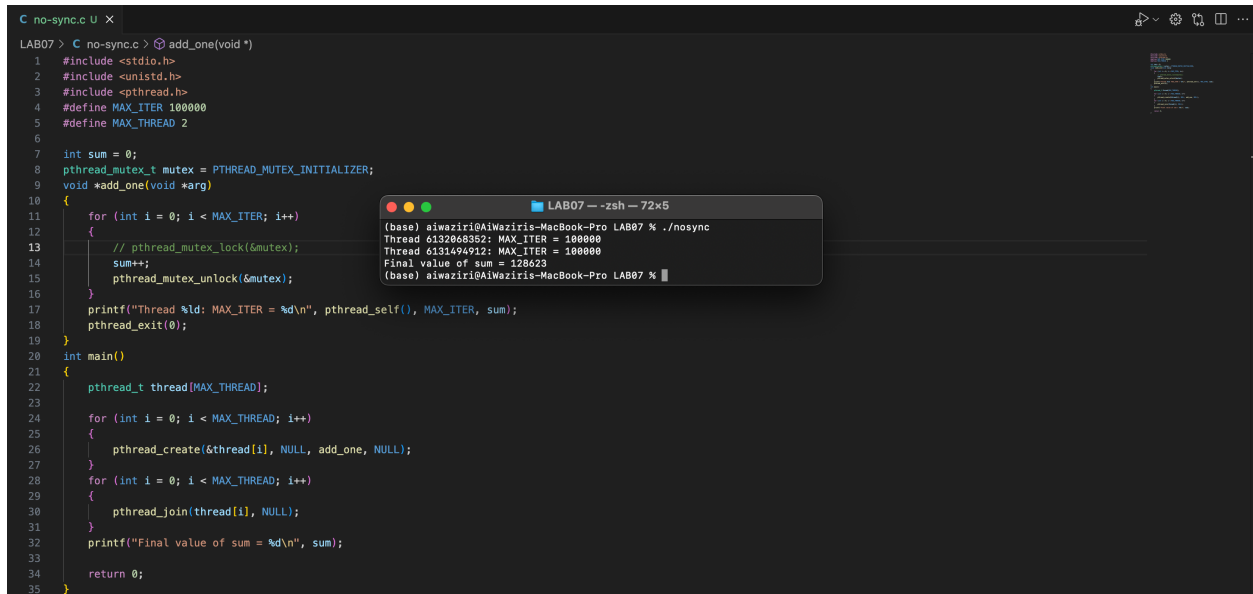
```
LAB07 > C no-sync.c > add_one(void *)
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <pthread.h>
4 #define MAX_ITER 100000
5 #define MAX_THREAD 2
6
7 int sum = 0;
8 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
9 void *add_one(void *arg)
10 {
11     for (int i = 0; i < MAX_ITER; i++)
12     {
13         pthread_mutex_lock(&mutex);
14         sum++;
15         pthread_mutex_unlock(&mutex);
16     }
17     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum);
18     pthread_exit(0);
19 }
20 int main()
21 {
22     pthread_t thread[MAX_THREAD];
23
24     for (int i = 0; i < MAX_THREAD; i++)
25     {
26         pthread_create(&thread[i], NULL, add_one, NULL);
27     }
28     for (int i = 0; i < MAX_THREAD; i++)
29     {
30         pthread_join(thread[i], NULL);
31     }
32     printf("Final value of sum = %d\n", sum);
33
34     return 0;
35 }
```

```
LAB07 -- -zsh -- 80x24
Last login: Thu Oct 31 09:53:25 on ttys000
(base) aiwaziri@AiWaziris-MacBook-Pro ~ % cd /Users/aiwaziri/PycharmProjects/GitHub/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' (aka 'struct _opaque_pthread_t *') [-Wformat]
17 |     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum)
    |                                     ^
    |                                     |
    |                                     int
no-sync.c:17:69: warning: data argument not used by format string [-Wformat-extra-args]
17 |     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum)
    |                                     ^
    |                                     |
    |                                     int
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 no-sync.c U X
LAB07 > C no-sync.c > add_one(void *)
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <pthread.h>
4 #define MAX_ITER 100000
5 #define MAX_THREAD 2
6
7 int sum = 0;
8 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
9 void *add_one(void *arg)
10 {
11     for (int i = 0; i < MAX_ITER; i++)
12     {
13         // pthread_mutex_lock(&mutex);
14         sum++;
15         pthread_mutex_unlock(&mutex);
16     }
17     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum);
18     pthread_exit(0);
19 }
20
21 int main()
22 {
23     pthread_t thread[MAX_THREAD];
24     for (int i = 0; i < MAX_THREAD; i++)
25     {
26         pthread_create(&thread[i], NULL, add_one, NULL);
27     }
28     for (int i = 0; i < MAX_THREAD; i++)
29     {
30         pthread_join(thread[i], NULL);
31     }
32     printf("Final value of sum = %d\n", sum);
33     return 0;
34 }
35 }
```

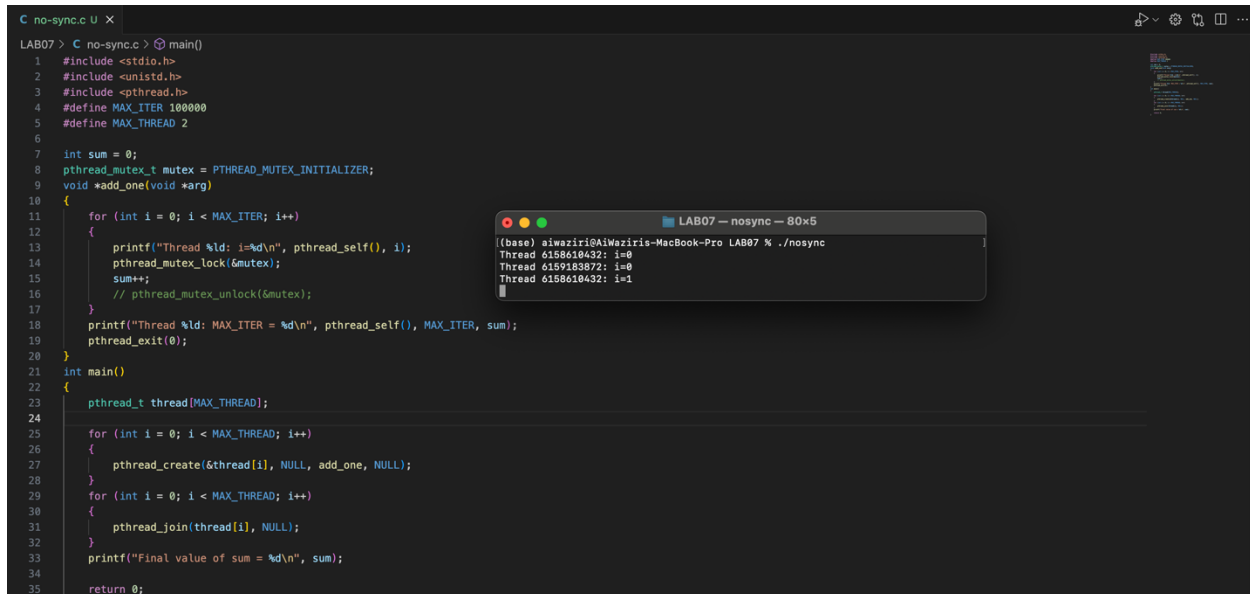
LAB07 -- zsh -- 72x5

(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:



The screenshot shows a C code editor with a file named 'no-sync.c'. The code defines a mutex and a function 'add_one' that increments a global 'sum' variable. In the 'main' function, two threads are created, each running the 'add_one' function. The first thread acquires the mutex, prints its ID, and increments the sum. However, the second thread also prints its ID but then gets stuck because it cannot acquire the mutex, which is still held by the first thread. The terminal window shows the output of the program, displaying the thread IDs and the value of 'sum' after the first increment, but the program hangs and does not reach the final 'printf' statement in 'main'.

```
C no-sync.c U X
LAB07 > C no-sync.c > main()
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <pthread.h>
4 #define MAX_ITER 100000
5 #define MAX_THREAD 2
6
7 int sum = 0;
8 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
9 void *add_one(void *arg)
10 {
11     for (int i = 0; i < MAX_ITER; i++)
12     {
13         printf("Thread %ld: i=%d\n", pthread_self(), i);
14         pthread_mutex_lock(&mutex);
15         sum++;
16         // pthread_mutex_unlock(&mutex);
17     }
18     printf("Thread %ld: MAX_ITER = %d\n", pthread_self(), MAX_ITER, sum);
19     pthread_exit(0);
20 }
21 int main()
22 {
23     pthread_t thread[MAX_THREAD];
24
25     for (int i = 0; i < MAX_THREAD; i++)
26     {
27         pthread_create(&thread[i], NULL, add_one, NULL);
28     }
29     for (int i = 0; i < MAX_THREAD; i++)
30     {
31         pthread_join(thread[i], NULL);
32     }
33     printf("Final value of sum = %d\n", sum);
34
35     return 0;
```

```
LAB07 — nosync — 80x5
(base) aiwaziri@AiWaziris-MacBook-Pro LAB07 % ./nosync
Thread 6158618432: i=0
Thread 6159183872: i=0
Thread 6158618432: 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 bank.c U X
LAB07 > C bank.c > withdraw(int)
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #define MAX_TRANSACTION 10
5
6 int balance = 1000;
7
8 void *deposit(int amount)
9 {
10     printf("+ Thread %ld: Deposit %d\n", pthread_self(), amount);
11     balance += amount;
12 }
13 void *withdraw(int amount)
14 {
15     if (balance >= amount)
16     {
17         printf("- Thread %ld: Withdraw %d\n", pthread_self(), amount);
18         balance -= amount;
19     }
20     else
21     {
22         printf("- Thread %ld: Insufficient funds\n", pthread_self());
23     }
24 }
25 void *transaction(void *arg)
26 {
27     for (int i = 0; i < MAX_TRANSACTION; i++)
28     {
29         int transaction_type = rand() % 2;
30         int amount = rand() % 100;
31         if (transaction_type == 0)
32         {
33             withdraw(amount);
34         }
35         else
```

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.