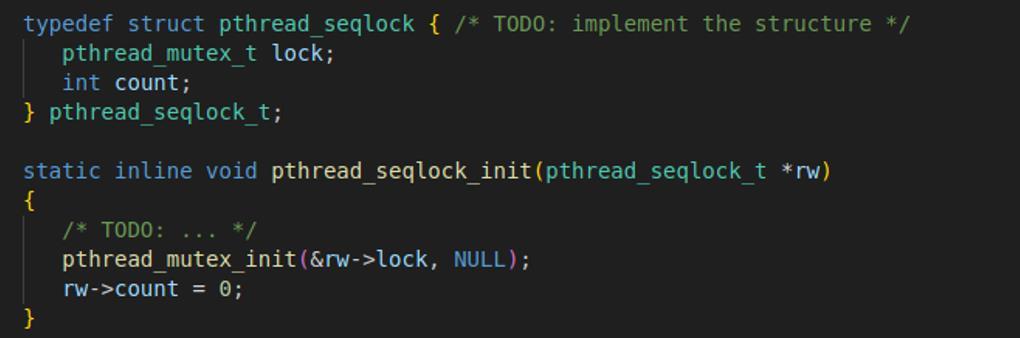
**Problem 1:**

Define a **struct pthread\_seqlock** with the number of variables being read or write status, locking the variable to

ensure data security.

The **pthread\_seqlock\_init** function is the initial initialization of seqlock with the initial state **number = 0**.

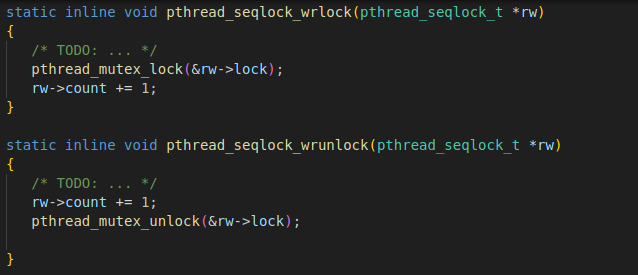


The **pthread\_seqlock\_wrlock** function is used to lock the contention area and allow a process or thread to write.

enter. Update **count + 1** so that count is an odd number indicating the process is writing. After completing the recording process

The **pthread\_seqlock\_wrunlock** function updates the **count + 1** value to an even number indicating a write.

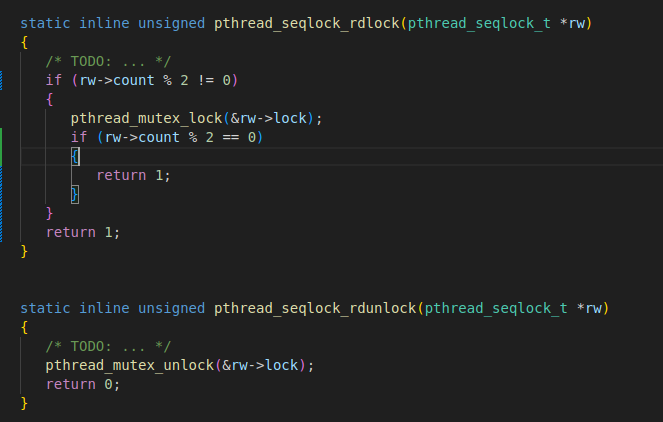
ended and unlocked the disputed zone



The **pthread\_seqlock\_rdlock** function is used to allow multiple threads to read at the same time if **rw->count** is not an odd number.

The process is writing, so you have to wait for the writing process to complete before reading. When the recording process is completed

to rw->count will add up to 1 and unlock the read-in read process. If there is no function write process will return 1.



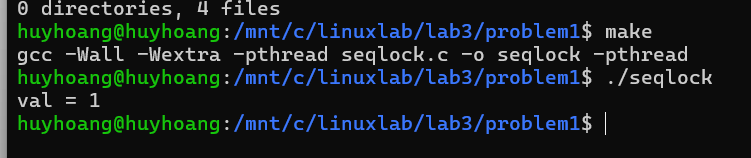
To compile:

**cd problem1**

**make**

To run:

**./seqlock**



**Problem 2:**

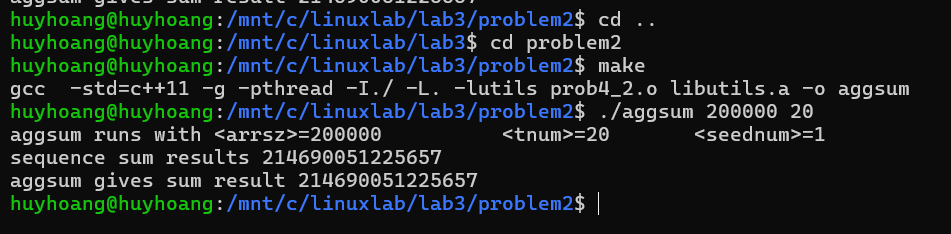
To compile:

**cd problem2**

**make**

To run:

**./aggsum 200000 20**



**Problem 3:**

The logger implementation successfully manages log data in a shared buffer, handles buffer overflow, and periodically flushes the buffer to the screen. The use of mutex locks, semaphores, and separate threads ensures thread safety and efficient log management. Further development could extend the logger's functionality to handle additional events and signals, enhancing its robustness and flexibility in real-world scenarios.

To compile:

**cd problem3**

**make**

To run:

**./problem3**

