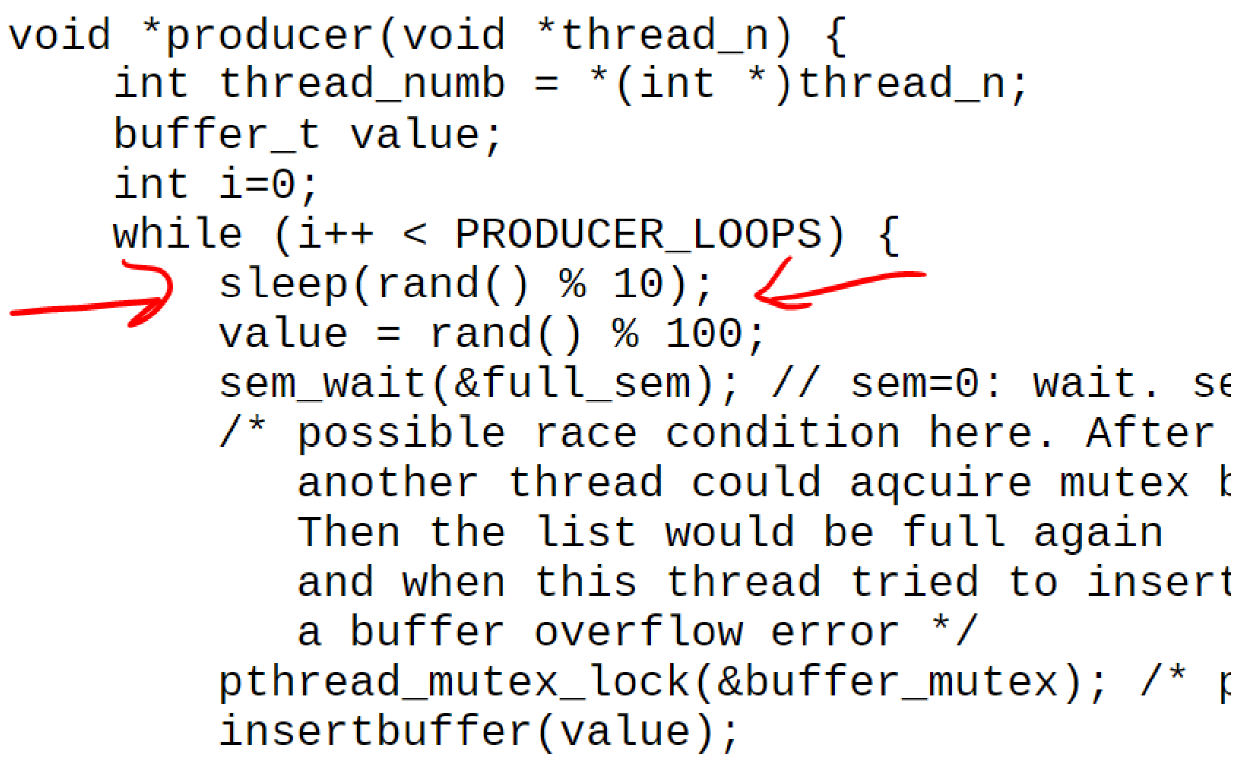
Tyler Brightwell 8:23 PM

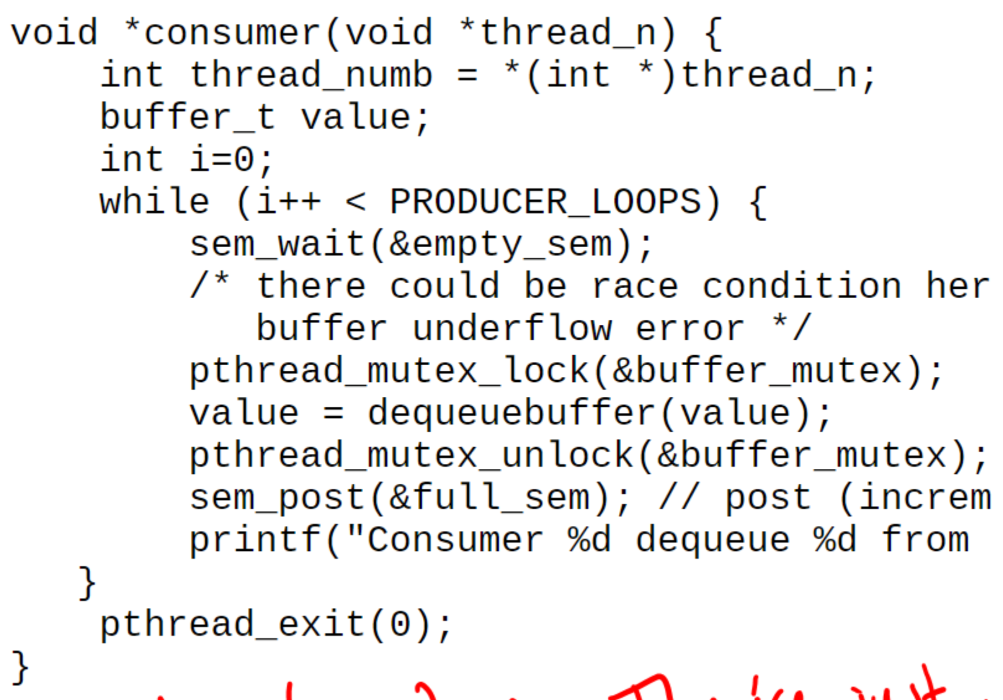
the output of q2 without making changes adds an item and removes it. So producer 0 adds 15 consumer 1 dequeues 15 it. Then producer 2 adds 35 and consumer 3 dequeues 35 it. So do we have to switch the print statements so the producer will produce multiple things and then the consumer will dequeue multiple things afterwards to check if it is lifo or fifo? Because I am confused the way it is now it haha. I can't tell how to check fifo because it seems like it adds and immediately removes it.

Statler 8:32 PM

As the base code is written, Q2 should bound the size of our buffer to 5 elements, but only one producer can access the critical section i.e. store an element to the buffer at a time. Since each producer sleeps randomly between 0 and 10 seconds, you're probably seeing each element being removed by the consumers in real-time. The consumers are just sitting there waiting until something gets stored in the buffer to remove it. (edited)



```
void *producer(void *thread_n) {
    int thread_numb = *(int *)thread_n;
    buffer_t value;
    int i=0;
    while (i++ < PRODUCER_LOOPS) {
        sleep(rand() % 10);
        value = rand() % 100;
        sem_wait(&full_sem); // sem=0: wait. se
        /* possible race condition here. After
           another thread could aqcuire mutex t
           Then the list would be full again
           and when this thread tried to insert
           a buffer overflow error */
        pthread_mutex_lock(&buffer_mutex); /* p
        insertbuffer(value);
```



```
void *consumer(void *thread_n) {
    int thread_numb = *(int *)thread_n;
    buffer_t value;
    int i=0;
    while (i++ < PRODUCER_LOOPS) {
        sem_wait(&empty_sem);
        /* there could be race condition her
           buffer underflow error */
        pthread_mutex_lock(&buffer_mutex);
        value = dequeuebuffer(value);
        pthread_mutex_unlock(&buffer_mutex);
        sem_post(&full_sem); // post (increm
        printf("Consumer %d dequeue %d from
   }
    pthread_exit(0);
}
```

No sleep here. They're just waiting on the producer threads.

A temporary solution would be to comment out the sleep() call in the producer code and decrease the number of consumers to 1.

8:33

This way, we can see the order in which the single consumer removes the element from the buffer. (edited)

Statler 8:39 PM

Just to clarify: The producers aren't storing elements in the buffer fast enough before the consumers remove them.

Tyler Brightwell 8:40 PM

Ok thanks! However, I just did this and it seems the first one produced is the first one to get dequeued by the consumer from the buffer does this mean it is already in a fifo order?

Statler 8:40 PM

You can also make the consumers sleep before attempting to dequeue the buffer.

8:40

This will generally be the case.

8:41

The consumers aren't doing anything but waiting on the producers. They're like an opposing football team waiting to receive the kick-off. (edited)

8:42

Make the consumers sleep for like 20 seconds and then see the result.

8:42

That will give the producers time to fill the buffer.

Tyler Brightwell 8:43 PM

ahh ok I see thanks so much! That really helped! Now I am seeing. Appreciate it!