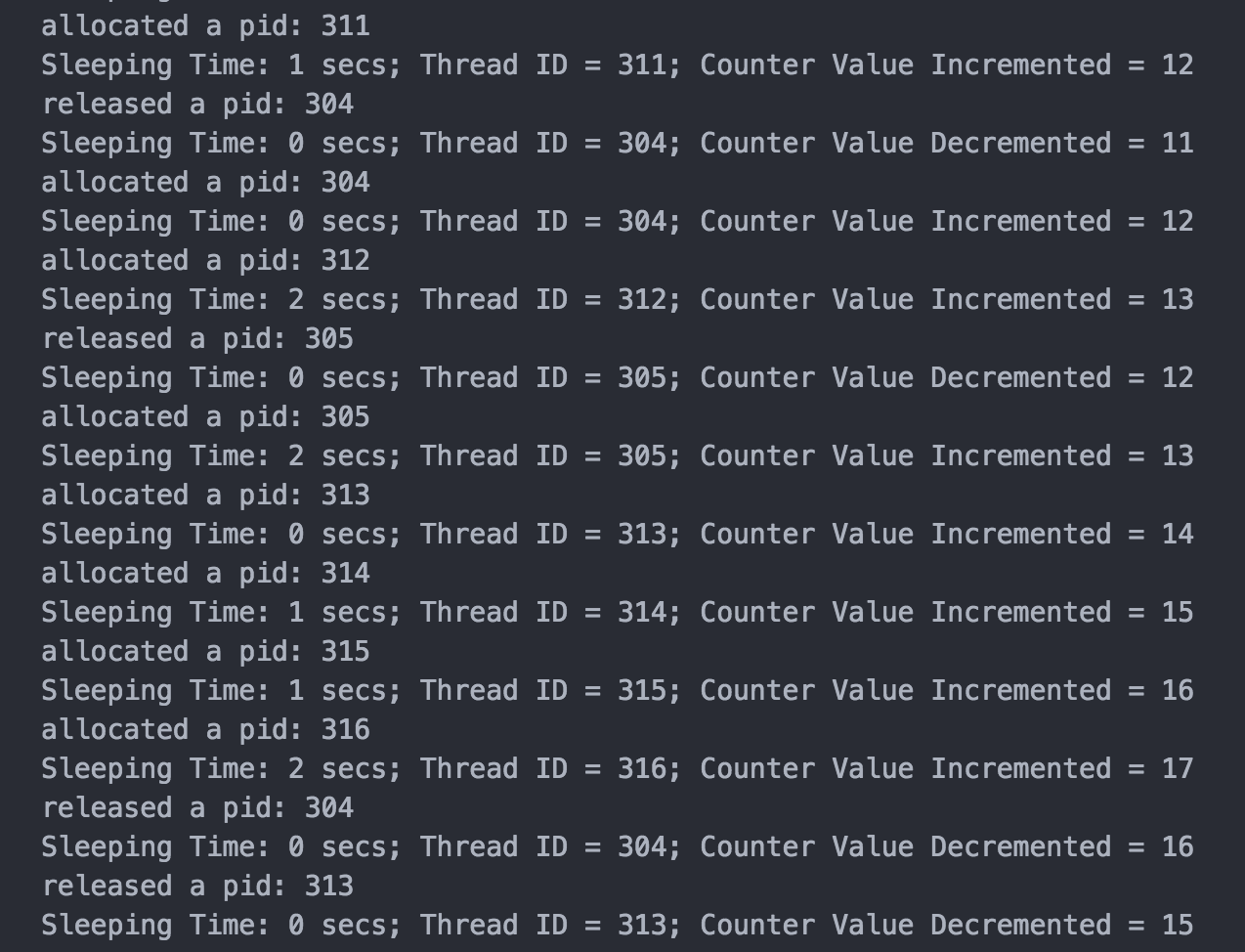


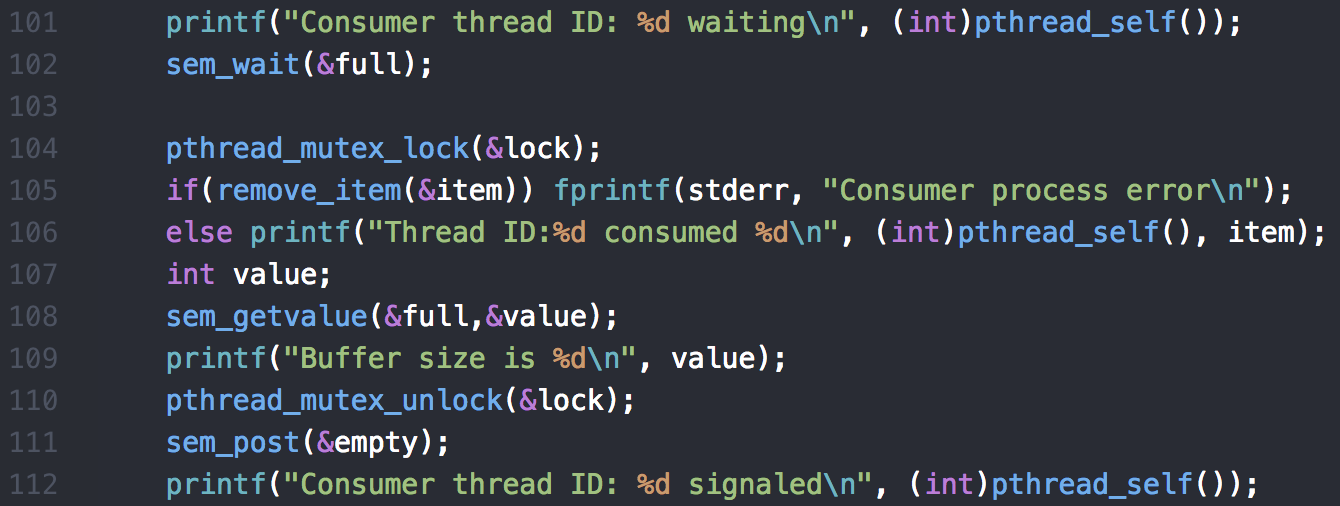
On line 79, the global variable counter is incremented, as well as decremented on line 86, causing the race condition. I placed mutex locks and unlocks around these in order to prevent the race condition.



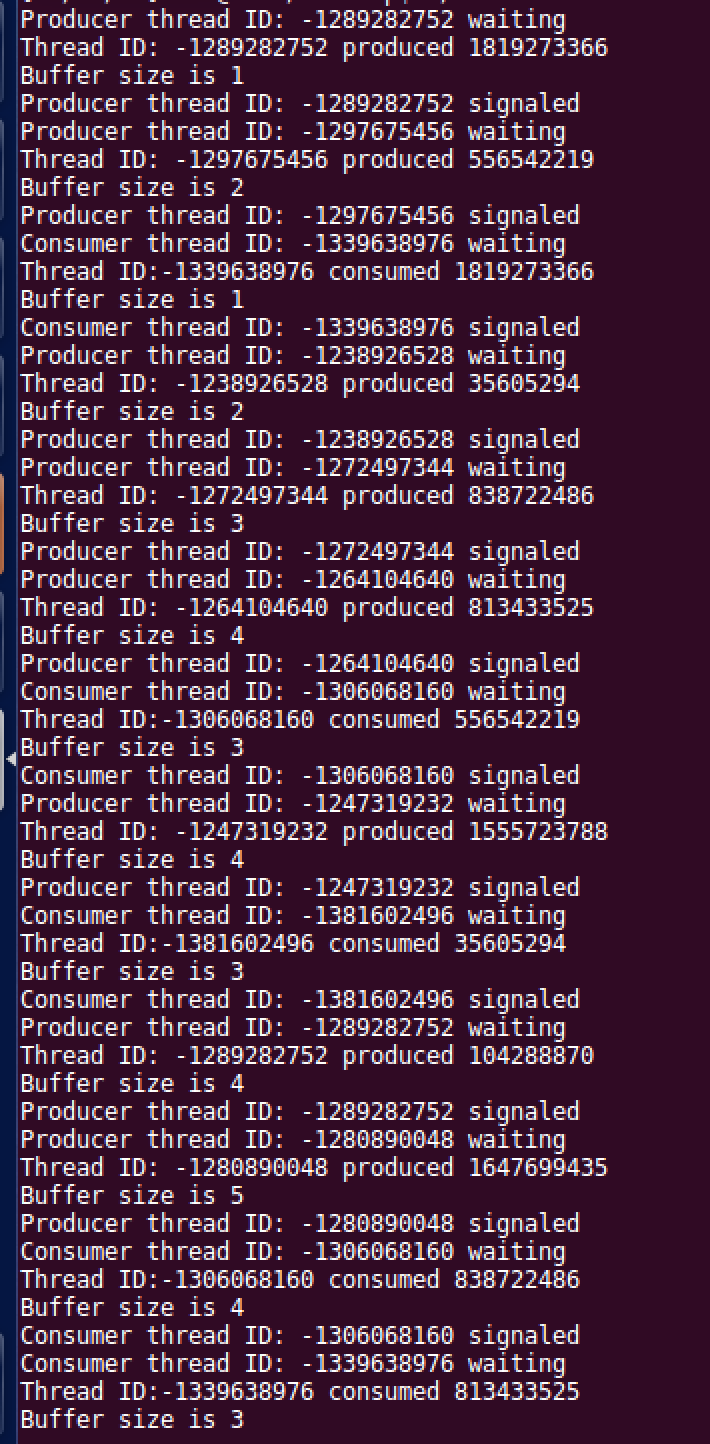
As you can see, whenever a new thread is allocated, the counter value is incremented, and when a thread is releases, the counter is decremented. Because of the mutex lock, only one thread adjusts the counter value at a time, which gives an accurate count of existing threads.



The above code shows the waiting and signaling moments in my producer function. The empty semaphore is waited on in line 74 because for an item to be added to the list, there must be an empty space in the buffer. Then on line 86 the full semaphore is signaled because now that an item is added to the buffer, there is one more full space in the buffer.



This shows the waiting and signaling moments in my consumer function. It does the opposite of the producer function. Waiting on full because if there are no full spots in the buffer, meaning it is empty, then there is nothing to be consumed from it. At the end, it signals empty because there is one more empty spot in the buffer.



In the output, you can see where a producer is waiting on empty, as well as consumer is waiting on full. When a producer thread executes, the buffer size increases, and when a consumer thread executes, the buffer size decreases. The whenever a number is produced or consumed, the number is displayed, being removed on a first in first out bases.