Test under lock, flag setting and signaling in a critical section

Client thread entry function in bounded-buffer-cond-vars-2.c:

```
    pthread_mutex_lock(&order->lock);
    while(!order->fulfilled){
      pthread_cond_wait(&order->cond_fulfilled, &order->lock);
    }
    pthread_mutex_unlock(&order->lock);
```

Trader thread entry function in bounded-buffer-cond-vars-2.c:

```
    pthread_mutex_lock(&order->lock);
    order->fulfilled = true;
    pthread_cond_signal(&order->cond_fulfilled);
    pthread_mutex_unlock(&order->lock);
```

Condition variables implement "test under lock"; predicate retesting must be performed under lock.

Flag setting and signaling must be in a critical section; if a trader does not use lock-unlock, it can interleave with the execution of a client thread leading to a deadlock:

1) a client tests !order->fulfilled as true and is preempted before calling pthread_cond_wait,
2) a trader sets the fulfilled flag to true and signals cond_fulfilled, which is lost because the client is not yet waiting,
3) the client calls pthread_cond_wait and is deadlocked.

Need for predicate retesting

A signal awakes "at least one" thread (pthread specification) from the queue of waiting threads attached to a condition variable. The following examples demonstrate the need for predicate retesting:

a) after a cond_not_full signal, if there are waiting client threads, at least one client thread is awakened; a non-waiting client thread is allowed to acquire mutex before an awakened waiting client thread reacquires mutex; the former fills the order queue to capacity.

b) two waiting client threads are awakened with a cond_not_full signal; the awakened client thread on top of the thread queue of cond_not_full reacquires mutex, fills the order queue, and releases mutex; the next awakened client thread reacquires mutex, but the order queue is full; the thread is pushed back onto the thread queue of cond_not_full after predicate retesting.