Q: Is there a recommended data structure for us to use for our Broker? should we just use a queue and take FIFO

A: a queue would be good

Q: Since there is no actual input file of incoming trade requests, are we to randomly select which type of trade request enters the queue at the beginning of the program (either BTC or ETH)?

If this is the case, I have a follow-up question. Upon calling 'log_request_added()', will RequestType be randomly chosen between 0 and 1? Same case for ConsumerType?

Thanks.

A: The trade requests aren't exactly random, but they ***are*** nondeterministic. The crypto that is produced is dependent on the production time (ie. how long the sleep call takes), the CPU scheduler, how many other processes are accessing the critical section, etc.

The initial logic is that you create all 4 threads (2 producers, 2 consumers) at the same time. Both producers immediately simulate production using a sleep() call. As soon as the sleep() call is done for a producer, they should try and add it to the queue. If there's space on the queue, they should start producing on another item.

Note: Be sure to use a mutex to access to the queue since queues are not thread safe.

Walkthrough video:

Producer, consumer, and main thread must be in separate files, and can add a separate module for broker if we would like

One main thread, 2 producer threads (one for bitcoin and one for Ethereum), 2 consumer threads (one for blockchain X and one for blockchain y), broker queue of trades with a max-capacity of 16, and at most 5 bitcoin requests among those 16 (set up semaphore for enforcing this constraint)

Broker.h

Consumer.h

Producer.h

Constrains:

- 16 max capacity ← semaphore

- 5 bitcoins ← semaphore
- 16 ethereum ← semaphore
- Mutex ← semaphore
- Main wait for consumer to consumer the last item ← semaphore (cannot use busy waiting for accomplishing any of these) **you can use integers to track number of bitcoin and eth requests and how many have been consumed**.

- Use semaphores for producer/consumer synchronization
- Mutually exclusive access we can use pthread lock and unlock but not required, can use semaphores instead
- Integer cannot be used as a flag to ensure order of execution

Production:

- In the production process, produce or create the new item first, then check in there is an available slot in the queue
- For bitcoin, check to see if there are already 5 bitcoin requests in the queue

Consumption:

- Check if there is anything to consume, if so lock and then remove or consume the item

Queue: (FIFO)

- Track what is going on in queue (at any moment we should be able to tell how many total requests are in the queue and how many of those are bitcoin or eth)
- Also track total requests produced throughout whole process for bitcoins and eth in broker structure

Semaphores:

- Binary semaphore for mutex, set initially to one
- For precedence constraint, set initial value to 0
- Controlling num trade requests for particular crypto currency in the queue set initial value to maximum that can have. For bitcoin it would be 5, overall capacity is 16, 0 for unconsumed, don't try to use fancy numbers like -1