Thread Communication

There are some use cases where you will want to have your threads communicate with each other. As we mentioned earlier, you can use create an **Event** for this purpose. But a more common method is to use a **Queue**. For our example, we'll actually use both! Let's see what that looks like:

```
import threading

from queue import Queue


def creator(data, q):
    """
    Creates data to be consumed and waits for the consumer
    to finish processing
    """
    print('Creating data and putting it on the queue')
    for item in data:
        evt = threading.Event()
        q.put((item, evt))

        print('Waiting for data to be doubled')
        evt.wait()


def my_consumer(q):
    """
    Consumes some data and works on it

    In this case, all it does is double the input
    """
    while True:
        data, evt = q.get()
        print('data found to be processed: {}'.format(data))
        processed = data * 2
        print(processed)
        evt.set()
        q.task_done()


if __name__ == '__main__':
    q = Queue()
    data = [5, 10, 13, -1]
    thread_one = threading.Thread(target=creator, args=(data, q))
    thread_two = threading.Thread(target=my_consumer, args=(q,))
    thread_one.start()
    thread_two.start()
    thread_two.start()

    q.join()
```

Let’s break this down a bit. First off, we have a creator (AKA a producer) function that we use to create data that we want to work on (or consume). Then we have another function that we use for processing the data that we are calling **my_consumer**. The creator function will use the Queue’s **put** method to put the data into the Queue and the consumer will continually check for more data and process it when it becomes available. The Queue handles all the acquires and releases of the locks so you don’t have to.

In this example, we create a list of values that we want to double. Then we create two threads, one for the creator / producer and one for the consumer. You will note that we pass a Queue object to each thread which is the magic behind how the locks get handled. The queue will have the first thread feed data to the second. When the first puts some data into the queue, it also passes in an Event and then waits for the event to finish. Then in the consumer, the data is processed and when it’s done, it calls the **set** method of the Event which tells the first thread that the second is done processing and it can continue.

The very last line of code call’s the Queue object’s **join** method which tells the Queue to wait for the threads to finish. The first thread ends when it runs out of items to put into the Queue.

Wrapping Up

We covered a lot of material here. You have learned the following:

> - The basics of threading
> - How locking works
> - What Events are and how they can be used
> - How to use a Timer
> - Inter-Thread Communication using Queues / Events

Now that you know how threads are used and what they are good for, I hope you will find many good uses for them in your own code.