CSC 4800 Python Applications Programming

HW/Lab #7 – Producer/Consumer with Multiple Consumers

Due: Monday, February 27, 2017 – Revised Deadline

[Similar to Exercise 4-8: e.g.,
Thread Pools: Instead of a producer thread and a consumer thread, change the code for prodcons.py in Example 4-12 so that you have any number of consumer threads (a *thread pool*) which can process or consume more than one item from the Queue at any given moment.]

HW/Lab 7:

Create a new project similar to Exercise 4-12 Producer-Consumer that still has a single "producer" thread (producing NITEMS=10 items), but allows the creation of multiple "consumer" threads. Modify the code base so that readers and writers have an additional parameter to keep track of their "thread names", with each reader thread having a unique name. The output messages can identify which thread is doing the processing.

While the producer simply loops to produce the 'nloops' items, with multiple consumers you must devise an implementation that allows them to share the 'consuming' operations until the producer has finished and the queue is empty.

Note that it is not sufficient for the consumers to simply check to see if the queue is empty to determine when to stop processing. If the producer task is slow and the consumers are fast, then an empty queue probably means that all the consumers are simply waiting for something to process. Instead, there must be some mechanism for the condition where the producer has finished its work and the queue is now empty.

1. One possible (unacceptable) strategy for this would be to have the producer add additional "None" items to the end of the queue, one for each of the consumers, and then when a consumer gets a "None" value from the queue, it can stop processing and exit its thread. However, this strategy requires that the producer know exactly how many consumers there are, and also that the consumer pool is reliable (nobody crashes) and unchanging in size throughout the execution.

2. A better strategy that you must use for this lab assignment is to use a global Boolean variable that the Producer can modify and the Consumers can examine (e.g., writerfinished), initially False. When the producer has finished its normal operations, it can set writerfinished = True. The consumer threads can check this flag in combination with the status of the queue itself to determine when to stop processing items from the queue and terminate the thread.

   If a consumer thread determines that the producer has not yet finished but the queue is empty, then the readQ( ) implementation can simply sleep for a brief time (e.g., 1 second) and try again (e.g., a polling technique for an empty queue). If the readQ( ) function determines that it really is the end of the prod-cons processing, then it can be modified to return a value "None" to the reader, which can then exit the thread.

   Note that a val = queue.get(False) statement in readQ( ) is a non-blocking operation that either returns the next value from the queue, or throws an exception. You can implement try-except processing to determine when and how to implement the reader polling operations.

Turn in printed assignment in class.

Print out of source code for your modules for this assignment, along with at least 3 sample printouts showing different dynamic interactions, varying number of readers and writer delay times.

Example output: NITEMS = 10, NREADERS = 3, WRITERDELAY = 1 second between items

```
Writer producing object 0 for Q...size now 1
  Reader-0 consumed object 0 from Q... size now 0
  Reader-1 polling empty queue
  Reader-2 polling empty queue
Writer producing object 1 for Q...size now 1
  Reader-1 consumed object 1 from Q... size now 0
  Reader-2 polling empty queue
Writer producing object 2 for Q...size now 1
  Reader-2 consumed object 2 from Q... size now 0
Writer producing object 3 for Q...size now 1
  Reader-0 consumed object 3 from Q... size now 0
  Reader-1 polling empty queue
Writer producing object 4 for Q...size now 1
  Reader-1 consumed object 4 from Q... size now 0
Writer producing object 5 for Q...size now 1
Writer producing object 6 for Q...size now 2
Writer producing object 7 for Q...size now 3
  Reader-0 consumed object 5 from Q... size now 2
  Reader-2 consumed object 6 from Q... size now 1
Writer producing object 8 for Q...size now 2
  Reader-1 consumed object 7 from Q... size now 1
Writer producing object 9 for Q...size now 2
  Reader-0 consumed object 8 from Q... size now 1
  Reader-2 consumed object 9 from Q... size now 0
all DONE
```

Example output: NITEMS = 10, NREADERS = 3, WRITERDELAY = 0.5 second between items

```
Writer producing object 0 for Q...size now 1
  Reader-0 consumed object 0 from Q... size now 0
  Reader-1 polling empty queue
  Reader-2 polling empty queue
Writer producing object 1 for Q...size now 1
  Reader-2 consumed object 1 from Q... size now 0
  Reader-1 polling empty queue
Writer producing object 2 for Q...size now 1
Writer producing object 3 for Q...size now 2
  Reader-1 consumed object 2 from Q... size now 1
Writer producing object 4 for Q...size now 2
Writer producing object 5 for Q...size now 3
  Reader-0 consumed object 3 from Q... size now 2
Writer producing object 6 for Q...size now 3
Writer producing object 7 for Q...size now 4
Writer producing object 8 for Q...size now 5
Writer producing object 9 for Q...size now 6
  Reader-1 consumed object 4 from Q... size now 5
  Reader-0 consumed object 5 from Q... size now 4
  Reader-2 consumed object 6 from Q... size now 3
  Reader-1 consumed object 7 from Q... size now 2
  Reader-1 consumed object 8 from Q... size now 1
  Reader-0 consumed object 9 from Q... size now 0
all DONE
```