Project 2

Test No.	Input Filename	Num of Producer	Num of Consumer	Max FIFO Depth
1	Transaction1.txt	1/3/3	3/1/3	5/5/5
2	Transaction2.txt	1/3/3	3/1/3	5/5/5
3	Transaction3.txt	1/3/3	3/1/3	5/5/5

Test Criteria:

*** transaction1:** Producer-sleep time is shorter than consumer-sleep time. For example, for entries (0003, 100, 500) and (0004, 100, 600). As a result, consumer threads process fifo and blocks until producer threads put new data into global fifo object.

*** transaction2:** Producer-sleep time and consumer-sleep time have set up in an increasing order. It makes sure, if producer and consumer threads have similar sleep-times, only one thread will be active because of the Lock object, and two types of threads sync properly with each other.

*** transaction3:** Producer-sleep time is longer than consumer-sleep time. For example, for entries (0003, 1600, 100) and (0004, 2000, 100). As a result, producer threads will be inactive for a while, and consumer threads will process global fifo and wait until producer threads wake up and put new data in the global fifo.

Test No.1 (transaction1.txt):

python project2.py transaction1.txt 1 3 5

Comment: Even though number of consumer is 3, 1 producer reads data from file, because producers sleep fewer seconds than consumer. After one consumer breaks, other consumers reach the specified event, and breaks after 10 seconds.

Starting producer:1

Starting consumer:3

Producer:0001 internalId:1

Consumer:0001 internalId:1

Producer:0002 internalId:2

Producer:0003 internalId:3

Producer:0004 internalId:4

Consumer:0002 internalId:2

Producer:0005 internalId:5

Consumer:0003 internalId:3

Producer:9999 internalId:6

Producer completed

Consumer:0004 internalId:4

Consumer:0005 internalId:5

Consumer:9999 internalId:6

Producer and consumer threads done processing!

python project2.py transaction1.txt 3 1 5

Comment: Here 3 producers and 1 consumer, maxFifoDepth 5. So, producer threads finish reading the whole file way before consumer threads process. After one producer reaches end of file, other producer threads fall back.

Starting producer:3

Producer:0001 internalId:1

Producer:0002 internalId:2

Producer:0003 internalId:3

Starting consumer:1

Consumer:0001 internalId:1

Producer:0004 internalId:4

Producer:0005 internalId:5

Producer:9999 internalId:6

Producer completed

Consumer:0002 internalId:2

Consumer:0003 internalId:3

Consumer:0004 internalId:4

Consumer:0005 internalId:5

Consumer:9999 internalId:6

Producer and consumer threads done processing!

python project2.py transaction1.txt 3 3 5

Comment: Here 3 producer and 3 consumers, maxFifoDepth 5. Since, producer-sleep time is shorter than consumers, they finish reading file before the consumer can process the 2nd entry.

Staring producer:3

Producer:0001 internalId:1

Producer:0002 internalId:2

Producer:0003 internalId:3

Starting consumer:3

Consumer:0001 internalId:1

Producer:0004 internalId:4

Producer:0005 internalId:5

Producer:9999 internalId:6

Producer completed

Consumer:0002 internalId:2

Consumer:0003 internalId:3

Consumer:0004 internalId:4

Consumer:0005 internalId:5

Consumer:9999 internalId:6

Producer and consumer threads done processing!

Test No.2 (transaction2.txt):

python project2.py transaction2.txt 1 3 5

Comment: Here 1 producer and 3 consumers, maxFifoDepth 5. Since, producers and consumers have similar length of sleep and consumer cannot process unless producer reads file, we see producer and consumer threads process concurrently.

Starting producer:1

Starting consumer:3

Producer:0001 internalId:1

Consumer:0001 internalId:1

Producer:0002 internalId:2

Consumer:0002 internalId:2

Producer:0003 internalId:3

Consumer:0003 internalId:3

Producer:0004 internalId:4

Consumer:0004 internalId:4

Producer:0005 internalId:5

Consumer:0005 internalId:5

Producer:9999 internalId:6

Producer completed

Consumer:9999 internalId:6

Producer and consumer threads done processing!

python project2.py transaction2.txt 3 1 5

Comment: Here 3 producers and 1 consumer, maxFifoDepth 5. Since producers and consumer sleep times are similar, we see producers can read 3 entries before the first consumer process it. Also, having 3 producers allow them to finish reading file way before consumer can process them.

```
Starting producer:3
Producer:0001 internalId:1
Producer:0002 internalId:2
Producer:0003 internalId:3
Starting consumer:1
Consumer:0001 internalId:1
Producer:0004 internalId:4
Consumer:0002 internalId:2
Producer:0005 internalId:5
Producer:9999 internalId:6
Producer completed
Consumer:0003 internalId:3
Consumer:0004 internalId:4
Consumer:0005 internalId:5
Consumer:9999 internalId:6
Producer and consumer threads done processing!
```

python project2.py transaction2.txt 3 3 5

Comment: Here 3 producers and 3 consumer, maxFifoDepth 5. Since producers and consumer sleep times are similar, it gives us similar result as the previous test. But in this case, the program takes longer because of multiple consumer threads.

```
Starting producer:3
Producer:0001 internalId:1
Producer:0002 internalId:2
Producer:0003 internalId:3
Starting consumer:3
Consumer:0001 internalId:1
Producer:0004 internalId:4
Consumer:0002 internalId:2
Producer:0005 internalId:5
```

Producer:9999 internalId:6

Producer completed

Consumer:0003 internalId:3

Consumer:0004 internalId:4

Consumer:0005 internalId:5

Consumer:9999 internalId:6

Producer and consumer threads done processing!

Test No.3 (transaction3.txt):

python project2.py transaction3.txt 1 3 5

Comment: Here 1 producer and 3 consumers, maxFifoDepth 5. Since in this case, producers sleep longer than consumers, we see that 1 producer thread can fill up only one slot of fifo. Before reading another line, one of the consumer threads process them.

Starting producer:1

Starting consumer:3

Producer:0001 internalId:1

Consumer:0001 internalId:1

Producer:0002 internalId:2

Consumer:0002 internalId:2

Producer:0003 internalId:3

Consumer:0003 internalId:3

Producer:0004 internalId:4

Consumer:0004 internalId:4

Producer:0005 internalId:5

Consumer:0005 internalId:5

Producer:9999 internalId:6

Producer completed

Consumer:9999 internalId:6

Producer and consumer threads done processing!

python project2.py transaction3.txt 3 1 5

Comment: Here 3 producers and 1 consumer, maxFifoDepth 5. In this case, even though producers sleep more, 3 producers can fill up only 4 slots of fifo. Since consumer threads are active than producer, it process before producer can fill up the fifo.

```
Starting producer:3
Producer:0001 internalId:1
Producer:0002 internalId:2
Producer:0003 internalId:3
Starting consumer:1
Consumer:0001 internalId:1
Producer:0004 internalId:4
Consumer:0002 internalId:2
Producer:0005 internalId:5
Producer:9999 internalId:6
Producer completed
Consumer:0003 internalId:3
Consumer:0004 internalId:4
Consumer:0005 internalId:5
Consumer:9999 internalId:6
Producer and consumer threads done processing!
```

python project2.py transaction3.txt 3 3 5

Comment: Here 3 producers and 3 consumers, maxFifoDepth 5. In this case, we get similar output as previous, but this test takes longer to terminate than previous. Because, once one of the consumer threads read the last data in fifo, other consumer threads takes extra time to terminate.

```
Starting producer:3
Producer:0001 internalId:1
Producer:0002 internalId:2
Producer:0003 internalId:3
Starting consumer:3
Consumer:0001 internalId:1
Producer:0004 internalId:4
Consumer:0002 internalId:2
Producer:0005 internalId:5
Producer:9999 internalId:6
Producer completed
```

```
Consumer:0003 internalId:3
Consumer:0004 internalId:4
Consumer:0005 internalId:5
Consumer:9999 internalId:6
Producer and consumer threads done processing!
```