Dominic Groshong
May 27, 2019

CS365 Operating System
Lab 3

Part I Producer and consumer (multithread)

Implement the code from Figures 4.13, 4.14, 4.15

```java
import java.util.Date;

public class Factory{
    Run | Debug
    public static void main(String args[]){
        // Create the message queue
        Channel<Date> queue = new MessageQueue<Date>();

        //Create the producer and consumer threads and pass each thread a reference to the MessageQueue object.
        Thread producer = new Thread(new Producer(queue));
        Thread consumer = new Thread(new Consumer(queue));

        // Start the Threads
        producer.start();
        consumer.start();
    }
}
```

Figure 1: First Step was to create the Factory class and setting it up to create a producer and consumer

```java
import java.util.Date;

class Consumer implements Runnable{
    private Channel<Date> queue;

    /**
     * Constructor
     * @param queue pass in a queue from some extrenal call
     */
    public Consumer(Channel<Date> queue){
        this.queue = queue;
    }

    public void run(){
        Date message;

        while(true){
            // nap for awhile
            SleepUtilities.nap();

            // produce and item and enter it into the buffer
            message = queue.receive();

            if(message != null){
                System.out.println("Consumer consumed " + message);
            }
        }
    }
}
```

Figure 2: The next step was to create the Consumer class that gets the queue and receives messages and there by "consuming" them.

```java
import java.util.Date;

class Producer implements Runnable{
    private Channel<Date> queue;

    /**
     * Constructor
     * @param queue pass in a queue from some extrenal call
     */
    public Producer(Channel<Date> queue){
        this.queue = queue;
    }

    public void run(){
        Date message;

        while(true){
            // nap for awhile
            SleepUtilities.nap();

            // produce and item and enter it into the buffer
            message = new Date();
            System.out.println("Producer produced " + message);
            queue.send(message);
        }
    }

}
```

Figure 3: Next I recreated the Producer class which gets the queue and adds a date message to the queue thus "producing" them.

Dominic Groshong
May 27, 2019

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
Produced: Mon May 27 09:58:27 PDT 2019
Consumed: Mon May 27 09:58:27 PDT 2019
Produced: Mon May 27 09:58:28 PDT 2019
Produced: Mon May 27 09:58:30 PDT 2019
Consumed: Mon May 27 09:58:28 PDT 2019
Produced: Mon May 27 09:58:33 PDT 2019
Produced: Mon May 27 09:58:34 PDT 2019
```

Figure 4: Output of the queue version

Part II Producer and consumer (semaphore)

Implement the code from Figures 6.9 – 6.14

First was the creation of the BoundedBuffer class.

```java
/**
 * Interface for the Bounded Buffer
 * @param <E>
 */
interface Buffer<E>{
    public void insert(E item);
    public E remove();
}
```

Figure 5: Created an interface class to be used in the bounded buffer.

```java
import java.util.concurrent.Semaphore;

public class BoundedBuffer<E> implements Buffer<E> {
    private static final int BUFFER_SIZE = 5;
    private E[] buffer;
    private int in, out;
    private Semaphore mutex;
    private Semaphore empty;
    private Semaphore full;

    public BoundedBuffer(){
        // buffer is initially empty
        in = 0;
        out = 0;
        mutex = new Semaphore(1);
        empty = new Semaphore(BUFFER_SIZE);
        full = new Semaphore(0);

        buffer = (E[]) new Object[BUFFER_SIZE];
    }
```

Figure 6: BoundedBuffer 1/2 - Class constructor initializing the buffer with in, out, mutex, empty, and full states.

```java
    public void insert(E item){
        try{
            empty.acquire();
            mutex.acquire();
        }
        catch(InterruptedException e){
            System.out.println(e);
        }


        // add an item to the buffer
        buffer[in] = item;
        in = (in + 1) % BUFFER_SIZE;

        mutex.release();
        full.release();
    }

    public E remove(){
        E item;
        try{
            full.acquire();
            mutex.acquire();
        }
        catch(InterruptedException e){
            System.out.println(e);
        }
        // remove an item from the buffer
        item = buffer[out];
        out = (out + 1) % BUFFER_SIZE;

        mutex.release();
        empty.release();

        return item;
    }
}
```

Figure 7: BoundedBuffer 2/2 - implementation of the insert and remove methods from the interface.

Dominic Groshong
May 27, 2019

```java
import java.util.Date;

public class Factory{
    Run | Debug
    public static void main(String args[]){
        // Create the buffer
        Buffer<Date> buffer = new BoundedBuffer<Date>();

        //Create the producer and consumer threads and pass each thread a reference to the buffer.
        Thread producer = new Thread(new Producer(buffer));
        Thread consumer = new Thread(new Consumer(buffer));

        // Start the Threads
        producer.start();
        consumer.start();
    }
}
```

Figure 8: Updated factory to use the new bounded buffer class

```java
import java.util.Date;

class Consumer implements Runnable{
    private Buffer<Date> buffer;

    public Consumer(Buffer<Date> buffer){
        this.buffer = buffer;
    }

    public void run(){
        Date message;

        while(true){
            // nap for awhile
            SleepUtilities.nap();

            // consume an item from the buffer
            message = (Date)buffer.remove();
            System.out.println("Consumed: " + message);
        }
    }
}
```

Figure 9: Updated the consumer to use the buffer and remove() method for "consuming" items

```java
import java.util.Date;

class Producer implements Runnable{
    private Buffer<Date> buffer;

    public Producer(Buffer<Date> buffer){
        this.buffer = buffer;
    }

    public void run(){
        Date message;

        while(true){
            // nap for awhile
            SleepUtilities.nap();

            // produce and item and enter it into the buffer
            message = new Date();
            buffer.insert(message);
            System.out.println("Produced: " + message);
        }
    }
}
```

Figure 10: Updated the producer to use the buffer and insert() method for "producing" items.

Dominic Groshong
May 27, 2019

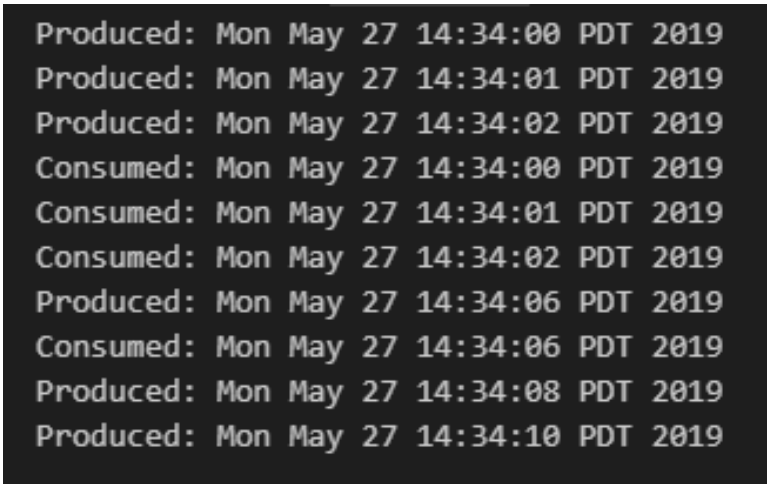

Figure 11: Output of the Bounded Buffer version.