CS372 Operating System
Lab 3

Part I Producer and consumer (multithread)

- Implement the code from Figures 4.13, 4.14, 4.15

```
import java.util.Date;

public class Factory
{
   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 4.13 The Factory class.

```
import java.util.Date;

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

   public Producer(Channel<Date> queue) {
      this.queue = queue;
   }

   public void run() {
      Date message;

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

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

Figure 4.14 Producer thread.

```
import java.util.Date;

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

   public Consumer(Channel<Date> queue) {
      this.queue = queue;
   }

   public void run() {
      Date message;

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

         // consume an item from the buffer
         message = queue.receive();

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

Figure 4.15 Consumer thread.

Part II Producer and consumer (semaphore)

- Implement the code from Figures 6.9 – 6.14

```
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];

   }

   public void insert(E item) {
      // Figure 6.10
   }

   public E remove() {
      // Figure 6.11
   }
}
```

Figure 6.9 Solution to the bounded-buffer problem using semaphores.

```
// Producers call this method
public void insert(E item) {
    empty.acquire();
    mutex.acquire();

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

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

Figure 6.10 The `insert()` method.

```
// Consumers call this method
public E remove() {
    E item;

    full.acquire();
    mutex.acquire();

    // remove an item from the buffer
    item = buffer[out];
    out = (out + 1) % BUFFER_SIZE;

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

    return item;
}
```

Figure 6.11 The `remove()` method.

```java
import java.util.Date;

public 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 an item & enter it into the buffer
         message = new Date();
         buffer.insert(message);
      }
   }
}
```

Figure 6.12 The producer.

```
import java.util.Date;

public 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();
    }
  }
}
```

Figure 6.13 The consumer.

```
import java.util.Date;

public class Factory
{
   public static void main(String args[]) {
     Buffer<Date> buffer = new BoundedBuffer<Date>();

     // Create the producer and consumer threads
     Thread producer = new Thread(new Producer(buffer));
     Thread consumer = new Thread(new Consumer(buffer));

     producer.start();
     consumer.start();
   }
}
```

Figure 6.14 The `Factory` class.

Part III GUI –Revise Part II

- Add two attributes to the Bounded Buffer Class, producerSleep and consumerSleep, to set the sleep time of producer and consumer.
- Create with get() and set()methods for each.
- Build a GUI that monitors the number of items in the Bounded Buffer and allows the user to change the sleep times for each of the producer and consumer.

Part V Turn in your lab

Zip (1) the source code of Part I, Part II, and Part III and (2) a screen dump of the running of Part I, Part II, and Part III. Turn in the zipped file to Moodle before the deadline.