Lancaster University

SCC.211 Operating Systems

Lecture 5 – Classic Coordination Problems

Amit Chopra
School of Computing & Communications, Lancaster University, UK

User Defined Box Sizing

- One **process** reads a number from keyboard and places inside an object of class *BoxDimension*
- Separate **process** extracts the number from *BoxDimension* object, and draws a box of corresponding size

```
public class BoxDimension {
  private int dim = 0;

  public void put(int d) {
    dim = d;
  }
  public int get() {
    return dim;
  }
}
BoxDimension d = new BoxDimension();
```

Process Getter

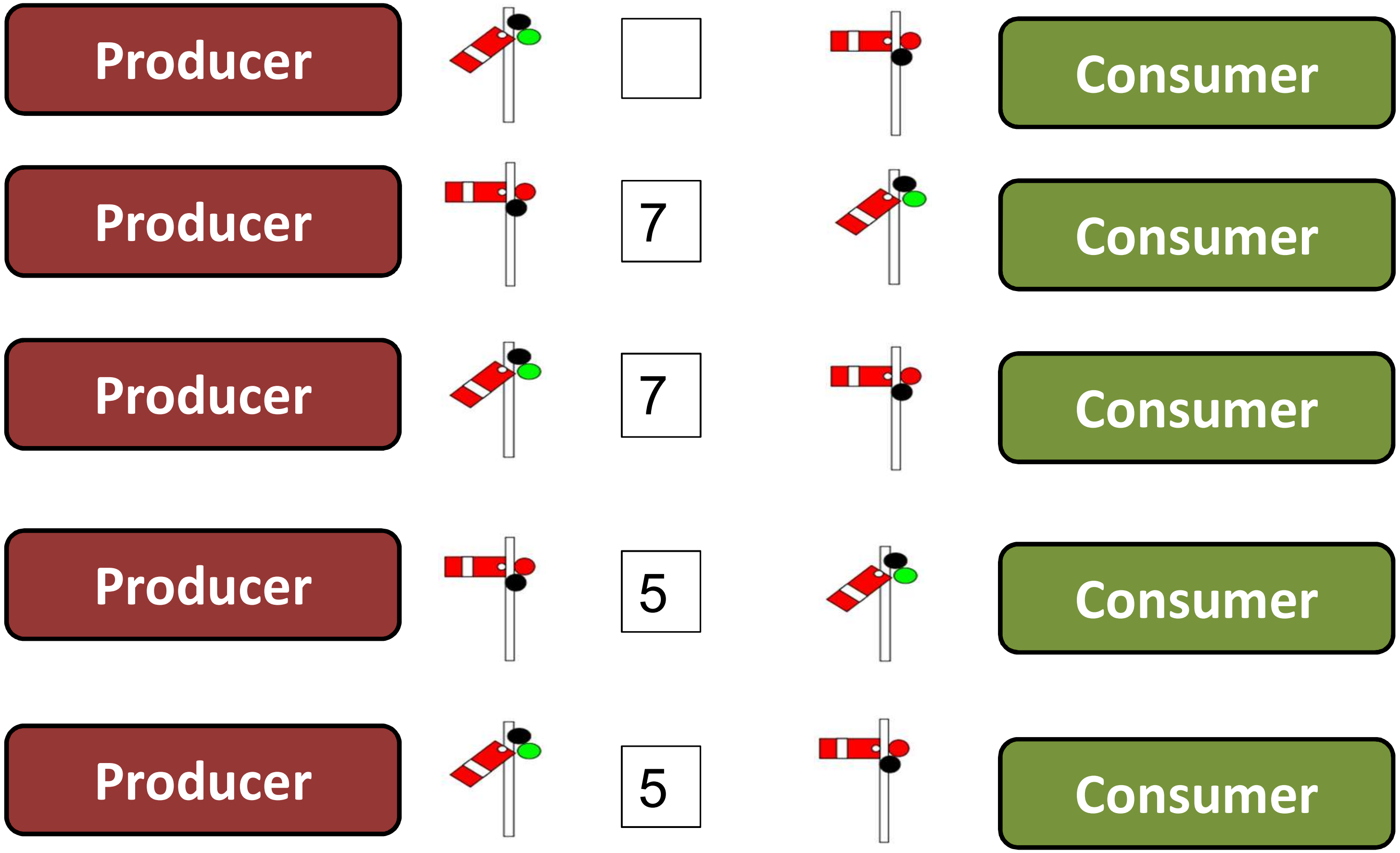
Process Putter

Coordination Problem

- Getter must extract only dimensions that Putter has placed
- Getter must not extract any dimension more than once
- Putter must not place a new dimension until the Getter has extracted the one the Putter placed earlier

- Putter=Producer; put=produce
- Getter=Consumer; extract=consume
- Placing values in a unit buffer (buffer of size 1)

Producer-Consumer (Bounded Buffer) Problem



Producer | | Consumer

Producer | 7 | Consumer

Producer | 7 | Consumer

Producer | 5 | Consumer

Producer | 5 | Consumer

Outline of solution

- Uses two sempahores produce and consume

Producer Thread	Consumer Thread
produce.acquire	*consume.acquire*
populate buffer	*consume buffer*
consume.release	*producer.release*

- How to ensure that consumer does not consume before producer produces?
  - What should the initial values of the semaphores be?

Java Implementation of Producer-Consumer Unit Buffer

```
public class ProducerConsumerUnitBuffer {

	public static void main(String[] args) {

		Semaphore produce = new Semaphore(1);
		Semaphore consume = new Semaphore(0);

		StringBuffer buf = new StringBuffer();

		new Thread(new Producer(produce, consume, buf)).start();
		new Thread(new Consumer(produce, consume, buf)).start();
	}

}
```

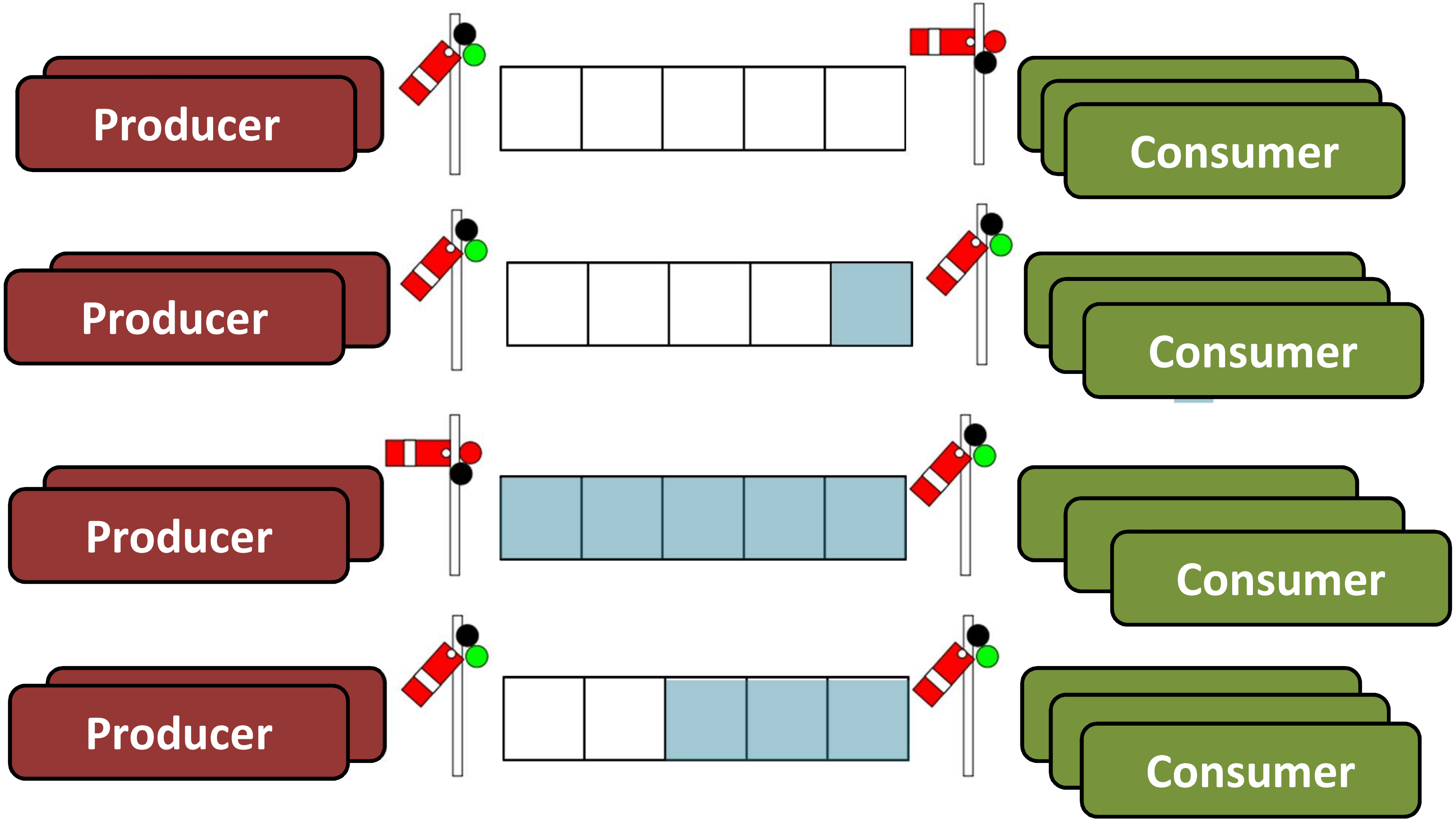
Java Implementation of Producer-Consumer Unit Buffer

```
class Producer implements Runnable {
        Semaphore produce, consume;
        StringBuffer buf;

        public Producer(Semaphore produce, Semaphore consume, StringBuffer buf) {
                this.produce = produce; this.consume = consume; this.buf = buf;
        }

        public void run() {
                while(true) {
                        produce.acquire();
                        buf.delete(0,buf.length());
                        buf.append(System.currentTimeMillis());
                        consume.release();
                }
        }
}
```

Java Implementation of Producer-Consumer Unit Buffer

```
class Consumer implements Runnable {
        Semaphore produce, consume;
        StringBuffer buf;

        public Consumer(Semaphore produce, Semaphore consume, StringBuffer buf) {
                this.produce = produce;
                this.consume = consume;
                this.buf = buf;
        }

        public void run() {
                while(true) {
                        consume.acquire();
                        System.out.println(buf);
                        produce.release();
                }
        }
}
```

Producer-Consumer in General (Several producers, consumers, and bounded length buffers)



Conditions

- A Consumer must extract only values that a Producer has placed
- No value should be extracted more than once
- A Producer may place a value only in *free* buffer locations
    - Initially all buffer locations are free
    - Any location that contains a value that has been extracted is free
    - No other location is free

Generalized Producer-Consumer in Java

```java
public class ProducerConsumer {

	public static void main(String[] args) {

		//Size of buffer is 5
		Buffer buf = new Buffer(5);

		Semaphore produce = new Semaphore(5);
		Semaphore consume = new Semaphore(0);

		for(int i=0;i<2;i++) {
			new Thread(new Producer(produce, consume, buf)).start();
			new Thread(new Consumer(produce, consume, buf)).start();
		}
	}
}
```

Generalized Producer-Consumer in Java

```java
class Producer implements Runnable {
        Semaphore produce, consume;
        Buffer buf;

        public Producer(Semaphore produce, Semaphore consume, Buffer buf) {
                this.produce = produce; this.consume = consume; this.buf = buf;
        }

        public void run() {
                while(true) {
                        produce.acquire();
                        buf.put("Job ID:" + System.currentTimeMillis());
                        consume.release();
                }
        }
}
```

Generalized Producer-Consumer in Java

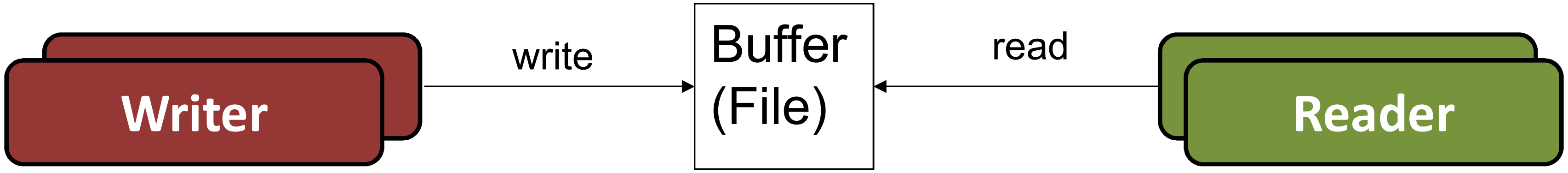
```
class Consumer implements Runnable {
        Semaphore produce,Semaphore consume;
        Buffer buf;

        public Consumer(Semaphore produce, Semaphore consume, Buffer buf){
                this.produce = produce; this.consume = consume; this.buf = buf;
        }

        public void run() {
                while(true) {
                        consume.acquire();
                        System.out.println("Got:" + buf.get());
                        produce.release();
                }
        }
}
```

Generalized Producer-Consumer in Java

```
class Buffer
        String[] values; boolean[] availableWrite;

        public Buffer(int bufSize)
                values = new String[bufSize];
                availableWrite = new boolean[bufSize];
                for(int i = 0; i<bufSize;i++)
                        availableWrite[i] = true;

        synchronized void put(String s)
                for(int i = 0; i < availableWrite.length; i++)
                        if(availableWrite[i] == true)
                                values[i] = s;
                                availableWrite[i] = false;
                                break;

        synchronized String get()
                for(int i = 0; i < availableWrite.length; i++)
                        if(availableWrite[i] == false)
                                availableWrite[i] = true;
                                return values[i];
```

Readers-Writers Problem

- A buffer (“file”) that several threads are accessing simultaneously
  - Buffer holds a single value (the file)
  - Writers write values to the buffer (they may also read the buffer)
  - Readers may only read from the buffer
- Ensure that any buffer value read by a thread is a value written by some writer!
  - Unlike Producer-Consumer, a value may be overwritten by a Writer without ever having been read by a Reader
  - Unlike Producer-Consumer, the same value can be read by many Readers and more than once by a Reader



Constraints that Follow from the Requirement

• Thread	• Thread	• Access to Buffer
• Writer	• Writer	• Not allowed
• Writer	• Reader	• Not allowed
• Reader	• Reader	• Allowed

- Writer must have exclusive access to the buffer
- Any number of readers may be concurrent

Solution Outline

- Two semaphores, *write* and *read*, initialized to 1
- A count of the number of Readers numReaders, *initialized to 0*

```
Writer:

Write.acquire

Write to buffer

Write.release
```

```
Reader:
numReaders++
If numReaders==1// self is first Reader,
        write.acquire

Read from buffer

numReaders--
If numReaders==0 //self is last Reader
        write.release
```

Solution Outline

```
Writer:

write.acquire

Write to buffer

write.release
```

```
Reader:
numReaders++
If numReaders==1// self is first Reader,
        write.acquire

Read from buffer

numReaders--
If numReaders==0 //self is last Reader
        write.release
```

Problem: Multiple Readers modifying *numReaders* concurrently

Solution Outline

```
Writer:

write.acquire

Write to buffer

write.release
```

```
Reader:
read.acquire
numReaders++
If numReaders==1// self is first Reader,
        write.acquire
read.release

Read from buffer
read.acquire
numReaders--
If numReaders==0 //self is last Reader
        write.release
read.release
```

Firing up the Threads

```
public class ReaderWriter {

      public static int numReaders = 0;

      public static void main(String[] args) {
            Semaphore write = new Semaphore(1);
            Semaphore read = new Semaphore(1);
            StringBuffer buf = new StringBuffer("Initial");

            for(int i=0;i<2;i++) {
                  new Thread(new Reader(write, read, buf)).start();
                  new Thread(new Writer(write, read, buf)).start();
            }
      }
}
```

Writer

```
public void run() {
      while(true) {
        write.acquire();
        buf.put("Current system time is" + System.currentTimeMillis());
        write.release();
      }
}
```

Reader

```
public void run() {
        while(true) {
                read.acquire();
                ReaderWriter.numReaders++;
                if(1==ReaderWriter.numReaders)
                        write.acquire();
                read.release();

                System.out.println(this + "read:" + buf);

                read.acquire();
                ReaderWriter.numReaders--;
                if(0==ReaderWriter.numReaders)
                        write.release();
                read.release();
        }
}
```

Code does not meet Requirement as we stated it

1. (As stated earlier) Ensure that any buffer value read by a thread is a value written by some writer!
2. (Should Be) Ensure that any buffer value read by a thread is either the initial value or a value written by some writer!

- 2 is normally requirement for Readers-Writers
    - Illustrates the subtlety of implementing multithreaded programs
- How would you modify code so that 1 is met?

Summary

- Semaphores can be used for sophisticated coordination (signalling) between threads
- Producer-Consumer is an archetypal problem that illustrates coordination between threads
  - Generally, many producers and many consumers operating over a buffer of size n.
- Readers-Writers, another archetypal problem, may sound like Producer-Consumer but is different
  - A write must be exclusive to all other activity
  - Reads may be concurrent
- Important to think about the synchronization conditions and then write code
- Difficult to be sure via experimentation whether your implementation is correct
  - Need for formal methods
- Semaphore can have an initial value of 0