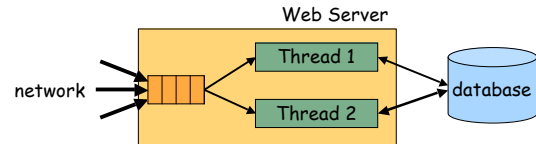


Concurrency

CSCI 334
Stephen Freund

Concurrency

- Benefits
 - Speed
 - Availability
 - Distribution
- Challenges
 - Hard to write
 - Not always possible (circuit evaluation)
 - Specifics
 - communication: send/receive info
 - synchronization: wait for another process
 - atomicity: don't stop in middle



Basic Question for Us

- How can programming languages make concurrent and distributed programming easier?

What Could Languages Provide?

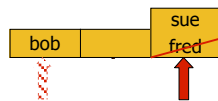
- Concurrent execution primitive
 - parallel "for" loops (Fortran)
 - support for spawning/managing processes.
- Concurrency Control
 - mutual exclusion
 - semaphors
 - monitors
- Communication Abstractions
 - message passing

Basic Issue: Race Conditions

- Sample action

```
void sign_up(person) {
    number = number + 1;
    list[number] = person;
}
```
- Problem with parallel execution

```
sign_up(fred) || sign_up(sue);
```



Conflicts

- Critical Section
 - coded in which process may access shared resource
- Race Condition
 - inconsistent behavior if two actions are interleaved
- Mutual Exclusion
 - allow only one process in critical section
 - process may need to wait for another to exit crit. section
- Deadlock
 - occurs when no process can proceed

Locks and Waiting

<initialize concurrency control>

Thread 1:

```
<wait>
sign_up(fred); // critical section
<signal>
```

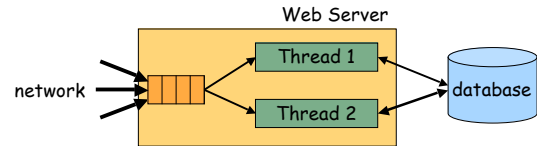
Thread 2:

```
<wait>
sign_up(sue); // critical section
<signal>
```

Need atomic operations to implement wait

Producer-Consumer Buffers

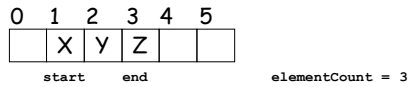
- Buffer with finite size
 - Producers add values to it
 - Consumers remove values from it
- Used "everywhere"
 - buffer messages on network, OS events, events in simulation, messages between threads...



Java Buffer

```
public class Buffer<T> {
```

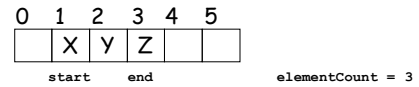
```
    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```



Java Buffer

```
public class Buffer<T> {
```

```
    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```

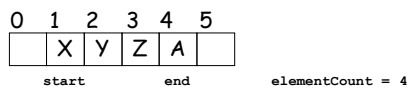


```
b.insert("A");
```

Java Buffer

```
public class Buffer<T> {
```

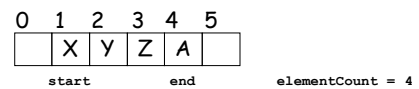
```
    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```



Java Buffer

```
public class Buffer<T> {
```

```
    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
```

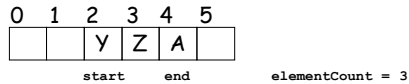


```
s = b.remove();
```

Java Buffer

```
public class Buffer<T> {
```

```
    private T[] elementData;  
    private int elementCount;  
    private int start;  
    private int end;
```



Unsafe Buffer Ops

```
public void insert(T t) {  
    assert (elementCount < elementData.length);  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
}
```

```
public T delete() {  
    assert (elementCount == 0);  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    return elem;  
}
```

Consumers

```
class Consumer extends Thread {  
    private final Buffer<Character> buffer;  
  
    public Consumer(Buffer<Character> b) {  
        buffer = b;  
    }  
  
    public void run() {  
        while (true) {  
            char c = buffer.delete();  
            System.out.print(c);  
        }  
    }  
}
```

Producers

```
class Producer extends Thread {  
    private final Buffer<Character> buffer;  
  
    public Producer(Buffer<Character> b) {  
        buffer = b;  
    }  
  
    public void run() {  
        while (moreData()) {  
            char c = next();  
            buffer.insert(c);  
        }  
    }  
}
```

Using Buffers

```
class Example {  
    public static void main(String[] args) {  
        Buffer<String> buffer = new Buffer<String>(5);  
        Producer prod = new Producer(buffer);  
        Consumer cons = new Consumer(buffer);  
        prod.start();  
        cons.start();  
    }  
}
```

Semaphors

- S = CreateSem(n);
 - new semaphore with value n
- S.wait();
 - if S > 0 then S := S - 1
 - if S == 0 suspend thread and add to wait queue
- S.signal();
 - if process waiting, then wake up
 - else S := S + 1

Identify Critical Sections

```
public void insert(T t) {  
  
    assert (elementCount < elementData.length);  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
  
}  
  
public T delete() {  
  
    assert (elementCount == 0);  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    return elem;  
  
}
```

```
mutex = CreateSem(1);
```

```
public void insert(T t) {  
  
    mutex.wait();  
    assert (elementCount < elementData.length);  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
    mutex.signal();  
  
}  
  
public T delete() {  
  
    mutex.wait();  
    assert (elementCount == 0);  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    mutex.signal();  
    return elem;  
  
}
```

```
mutex = CreateSem(1);  
nonEmpty = CreateSem(0);
```

```
public void insert(T t) {  
  
    mutex.wait();  
    assert (elementCount < elementData.length);  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
    mutex.signal();  
    nonEmpty.signal();  
  
}  
  
public T delete() {  
    nonEmpty.wait();  
    mutex.wait();  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    mutex.signal();  
    return elem;  
  
}
```

```
mutex = CreateSem(1);  
nonEmpty = CreateSem(0);  
nonFull = CreateSem(elementData.length);
```

```
public void insert(T t) {  
    nonFull.wait();  
    mutex.wait();  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
    mutex.signal();  
    nonEmpty.signal();  
}  
  
public T delete() {  
    nonEmpty.wait();  
    mutex.wait();  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    mutex.signal();  
    nonFull.signal();  
    return elem;  
}
```

Broken Version 0...

```
public void insert(T t) {  
  
    mutex.wait();  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
    mutex.signal();  
    nonEmpty.signal();  
  
}  
  
public T delete() {  
    nonEmpty.wait();  
    mutex.wait();  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    mutex.signal();  
    nonFull.signal();  
    return elem;  
  
}
```

Broken Version 1...

```
public void insert(T t) {  
    nonFull.wait();  
    mutex.wait();  
    end = (end + 1) % elementData.length;  
    elementData[end] = t;  
    elementCount++;  
    nonEmpty.signal();  
}  
  
public T delete() {  
    nonEmpty.wait();  
    mutex.wait();  
    T elem = elementData[start];  
    start = (start + 1) % elementData.length;  
    elementCount--;  
    mutex.signal();  
    nonFull.signal();  
    return elem;  
}
```

Broken Version 2...

```
public void insert(T t) {
    mutex.wait();
    nonFull.wait();
    end = (end + 1) % elementData.length;
    elementData[end] = t;
    elementCount++;
    mutex.signal();
    nonEmpty.signal();
}

public T delete() {
    nonEmpty.wait();
    mutex.wait();
    T elem = elementData[start];
    start = (start + 1) % elementData.length;
    elementCount--;
    mutex.signal();
    nonFull.signal();
    return elem;
}
```

Monitors

- [Brinch-Hansen, Dahl, Dijkstra, Hoare]
- Monitor combines
 - Private data
 - Set of methods
 - Synchronization policy
 - at most one process may be executing a method inside the monitor at a time
 - if a process is in the monitor, other processes will be delayed from entering it
- Implemented in Java with locks
 - `synchronized(obj) { ... }`
 - every object has a mutual exclusion lock
 - (can be implemented with semaphore)

Java Buffer

```
public class Buffer<T> {

    private T[] elementData;
    private int elementCount;
    private int start;
    private int end;
}
```

Monitor methods

```
public synchronized void insert(T t) {
    end = (end + 1) % elementData.length;
    elementData[end] = t;
    elementCount++;
}

public synchronized T delete() {
    T elem = elementData[start];
    start = (start + 1) % elementData.length;
    elementCount--;
    return elem;
}
```

Thread Coordination with Wait/Notify

```
public synchronized void insert(T t) {
    if (elementCount == elementData.length) wait();
    end = (end + 1) % elementData.length;
    elementData[end] = t;
    elementCount++;
}

public synchronized T delete() {
    T elem = elementData[start];
    start = (start + 1) % elementData.length;
    elementCount--;
    notifyAll();
    return elem;
}
```

Thread Coordination with Wait/Notify

```
public synchronized void insert(T t) {
    if (elementCount == elementData.length) wait();
    end = (end + 1) % elementData.length;
    elementData[end] = t;
    elementCount++;
    notifyAll();
}

public synchronized T delete() {
    if (elementCount == 0) wait();
    T elem = elementData[start];
    start = (start + 1) % elementData.length;
    elementCount--;
    notifyAll();
    return elem;
}
```

Thread Coordination with Wait/Notify

```
public synchronized void insert(T t) {
    while (elementCount == elementData.length) wait();
    end = (end + 1) % elementData.length;
    elementData[end] = t;
    elementCount++;
    notifyAll();
}

public synchronized T delete() {
    while (elementCount == 0) wait();
    T elem = elementData[start];
    start = (start + 1) % elementData.length;
    elementCount--;
    notifyAll();
    return elem;
}
```

Details...

```
synchronized void insert(T t) throws InterruptedException {
    while (elementCount == elementData.length) wait();
    end = (end + 1) % elementData.length;
    elementData[end] = t;
    elementCount++;
    notifyAll();
}

synchronized T delete() throws InterruptedException {
    while (elementCount == 0) wait();
    T elem = elementData[start];
    start = (start + 1) % elementData.length;
    elementCount--;
    notifyAll();
    return elem;
}
```

Consumers With Handler

```
class Consumer extends Thread {
    private final Buffer<Character> buffer;

    public Consumer(Buffer<Character> b) {
        buffer = b;
    }

    public void run() {
        try {
            while (true) {
                char c = buffer.delete();
                System.out.print(c);
            }
        } catch (InterruptedException e) {
            // thread interrupted, so stop loop
        }
    }
}
```

Interrupting Threads

```
class Example {
    public static void main(String[] args) {
        Buffer<String> buffer = new Buffer<String>(5);
        Producer prod = new Producer(buffer);
        Consumer cons = new Consumer(buffer);
        prod.start();
        cons.start();
        try {
            prod.join();
            cons.interrupt();
        } catch (InterruptedException e) {
            System.out.println("...");
        }
    }
}
```