Parallel Programming

Recitation Session 2

Thomas Weibel <weibelt@ethz.ch>

Laboratory for Software Technology,
Swiss Federal Institute of Technology Zürich

March 11, 2010

Executive Summary

- Solution to the last assignment
- Threads in Java
  - Create and start
  - Synchronization
  - Deadlocks
- Producer/Consumer
- Hints for the next assignment

Outline

1 **Last Assignment**

2 Threads

3 Producer/Consumer

4 New Assignment

Solution

```java
class Incrementer {
  public static void process(String arg)
      throws TerminationException {
    int tmp = Integer.parseInt(arg);
    if (tmp < 0)
      throw(new TerminationException("< 0"));
    System.out.println(tmp+1);
  }
  public static void main(String[] args) {
    try {
      for (int i = 0; i < args.length; i++)
        process(args[i]);
    }
    catch (TerminationException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Formatting Source Code

- 80% of the lifetime cost of a software product goes to maintenance
- Hardly any software is maintained for its whole life by the original author(s)
- Using good style improves the maintainability of software code
- Eclipse: "Ctrl+Shift+F" or "Source → Format"

I SAW THE CODE FOR YOUR COMPUTER PROGRAM YESTERDAY.
IT LOOKED EASY. IT'S JUST A BUNCH OF TYPING. AND HALF OF THE WORDS WERE SPELLED WRONG.
AND DON'T GET ME STARTED ABOUT YOUR OVER-USE OF COLONS.
THEY REMIND ME OF YOU, SIR.
E-Mail: SCOTTADAMS@AOL.COM
© 1994 United Feature Syndicate, Inc.
S. Adams

Java Naming Conventions

- Non-static variables and methods use camel case:
  - `thisIsAVariable`
  - not `this_is_a_variable`
- Class and interface names should start with capital:
  - `LinkedList`
  - not `LINKED_LIST`
- Non-static variable and all function names should start with lower case: `readFromFile()` or `firstName`
- All static variables upper-case: `MAXIMUM_USERS`
- Package names should be all lowercase, with no spaces between words: `ch.ethz.inf`

Pre and Post Increment

Pre Increment:

```
int i = 41;
System.out.println(++i);
System.out.println(i);
```

Post Increment:

```
int j = 23;
System.out.println(j++);
System.out.println(j);
```

Output?

Conditional Operator (Ternary Operator)

```
if (a > b) {
  max = a;
}
else {
  max = b;
}
```

can be written with the conditional operator ?: as

```
max = (a > b) ? a : b;
```

Use it wisely!

Outline

1 Last Assignment

2 Threads

3 Producer/Consumer

4 New Assignment

Creating Threads

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this:

- Provide a Runnable object
- Subclass Thread

Runnable Object

- Runnable interface defines a single method: run
- Meant to contain the code executed in the thread
- The Runnable object is passed to the Thread constructor

```
public class HelloRunnable
   implements Runnable {
  public void run() {
    System.out.println("Hello from a thread!");
  }

  public static void main(String args[]) {
    (new Thread(new HelloRunnable())).start();
  }
}
```

Subclass Thread

- The Thread class itself implements Runnable
- Its run method does nothing
- Application can subclass Thread, providing its own implementation of run

```
public class HelloThread extends Thread {
  public void run() {
    System.out.println("Hello from a thread!");
  }

  public static void main(String args[]) {
    (new HelloThread()).start();
  }
}
```

Threads

- Runnable object: more general, can subclass a class other than Thread
- Subclass Thread: easier to use in simple applications, but limited by the fact that task class must be a descendant of Thread
- Invoke `threadInstance.start()` to start the new thread
- Note: `threadInstance.run()` does not create a new thread

Sleep

```
try {
  // Doze a random time (0 to 0.5 secs)
  // to simulate workload
  Thread.sleep((int)(Math.random()*500));
}
catch (InterruptedException e) {
  // ...
}
```

- `Thread.sleep(long)` puts the current thread to sleep for the specified time in milliseconds
- An `InterruptedException` is thrown when a thread is waiting, sleeping, or otherwise paused for a long time and another thread interrupts it using the `interrupt` method in class `Thread`

Synchronized

- Every class and every object has an intrinsic lock
- `synchronized` marks code blocks where a thread must acquire the lock before proceeding
- `synchronized` can be added to methods
- The `this` pointer is used as the lock for instance methods

```
public class Buffer {
  public synchronized void write(int i) {
    // ...
  }

  public synchronized int read() {
    // ...
  }
}
```

Synchronized

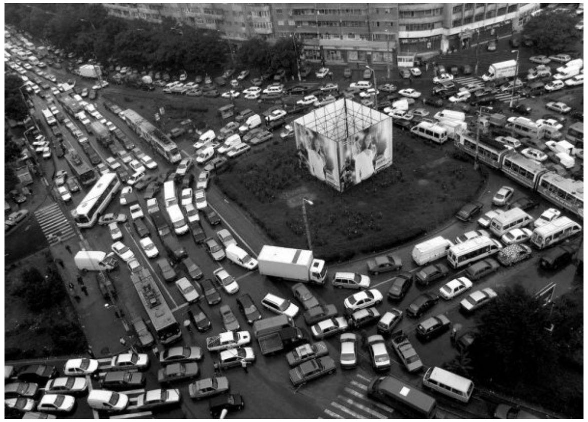
- `synchronized` can also be used to guard arbitrary blocks of code within a method, even in different classes
- It is important to use the correct object as the locks!

```
public void someMethod1() {
  //do something before
  synchronized(anObject) { /* ... */ }
  //do something after
}

public void someMethod2() {
  //do something before
  synchronized(anObject) { /* ... */ }
  //do something after
}
```

Quiz

- Can static methods be synchronized?
- What is the lock "object"?
- What is a deadlock?
- How can a deadlock occur?



Source: `http://www.vijayforvictory.com`

Deadlock

- Deadlock describes a situation where two or more threads are blocked forever, waiting for each other
- `http://java.sun.com/docs/books/tutorial/essential/concurrency/deadlock.html`:
  - Alphonse and Gaston are friends, and great believers in courtesy
  - A strict rule of courtesy is that when you bow to a friend, you must remain bowed until your friend has a chance to return the bow
  - Unfortunately, this rule does not account for the possibility that two friends might bow to each other at the same time
  - What happens if both bow at the same time?
- Analyze deadlocks: "Ctrl+\\" (Unix), "Ctrl+Break" (Windows)

Outline

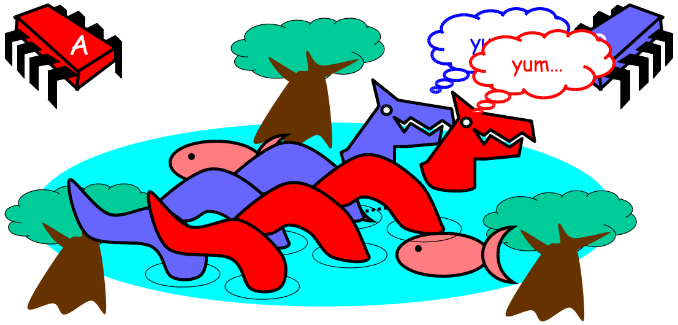
1 Last Assignment

2 Threads

3 Producer/Consumer

4 New Assignment

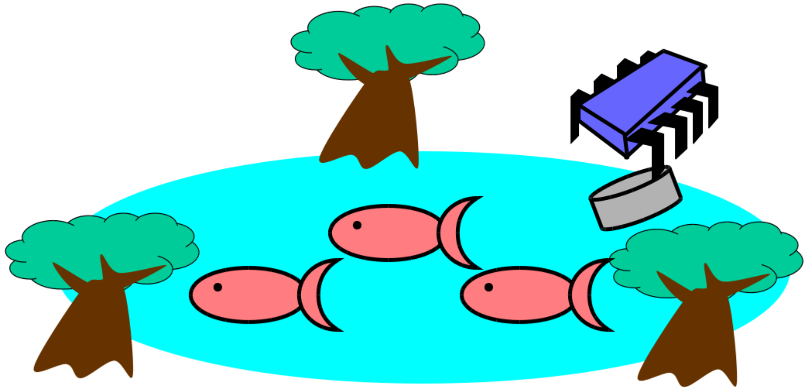
Once Upon a Time . . .



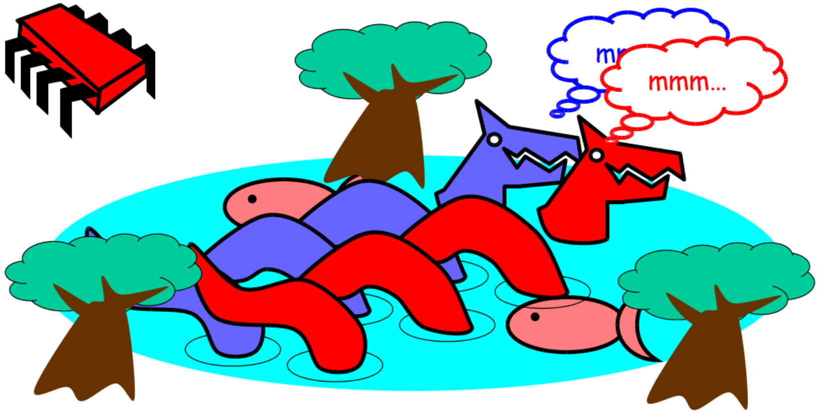
- Alice and Bob own a pet they bring to the same pond to feed
- Alice and Bob fall in love & marry
- Then they fall out of love & divorce
  - She gets the pets
  - He has to feed them

Example: "The Art of Multiprocessor Programming", Herlihy, Creative Commons Attribution-ShareAlike 2.5 License

Bob Puts Food in the Pond



Alice Releases Her Pets to Feed



Producer/Consumer

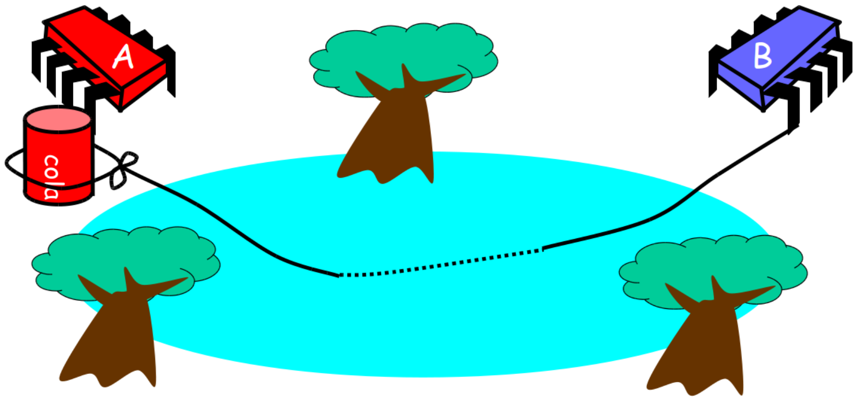
- Alice and Bob can't meet
  - Each has restraining order on other
  - So he puts food in the pond
  - And later, she releases the pets
- Avoid
  - Releasing pets when there's no food
  - Putting out food if uneaten food remains

Producer/Consumer

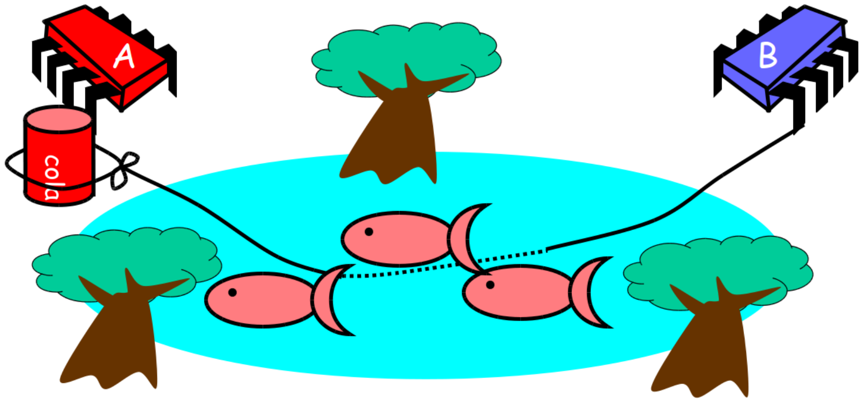
Need a mechanism so that

- Bob lets Alice know when food has been put out
- Alice lets Bob know when to put out more food

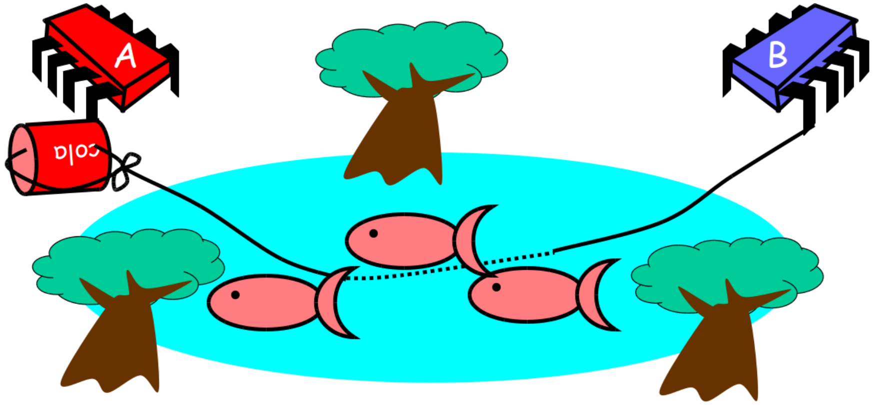
Solution



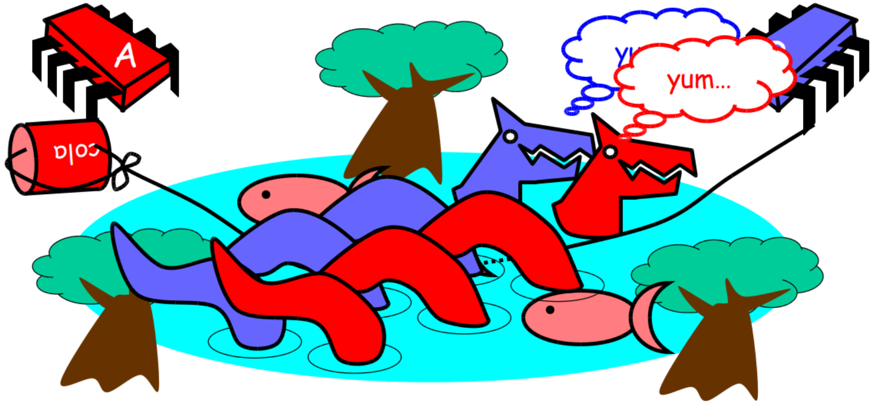
Bob puts food in Pond



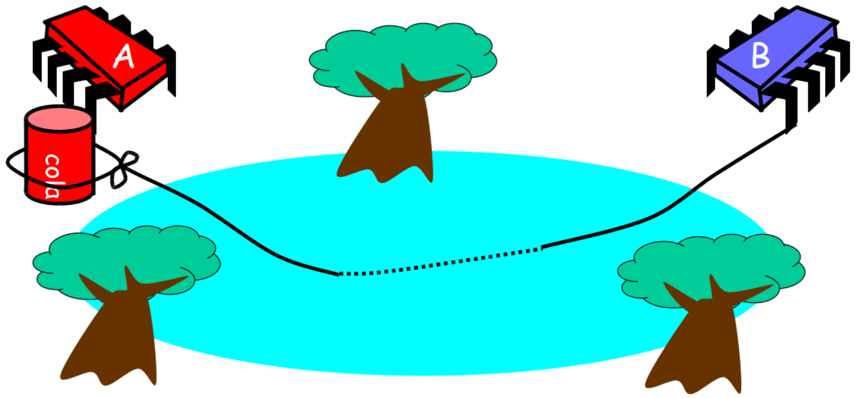
Bob knocks over Can



Alice Releases Pets



Alice Resets Can when Pets are Fed



Pseudocode

Alice:

```
while (true) {
  while (can.isUp()){};
  pet.release();
  pet.recapture();
  can.reset();
}
```

Bob:

```
while (true) {
  while (can.isDown()){};
  pond.stockWithFood();
  can.knockOver();
}
```

Correctness

- Mutual Exclusion: Pets and Bob never together in pond.
- No Starvation: if Bob always willing to feed, and pets always famished, then pets eat infinitely often.
- Producer/Consumer: The pets never enter pond unless there is food, and Bob never provides food if there is unconsumed food.

Outline

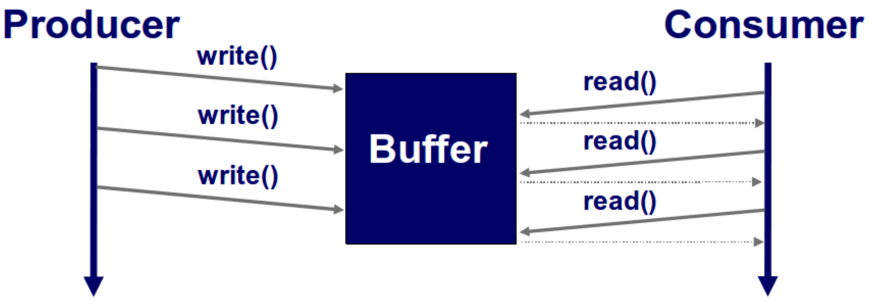
1 Last Assignment

2 Threads

3 Producer/Consumer

4 **New Assignment**

Buffer



Producer

Consumer

write()

write()

write()

Buffer

read()

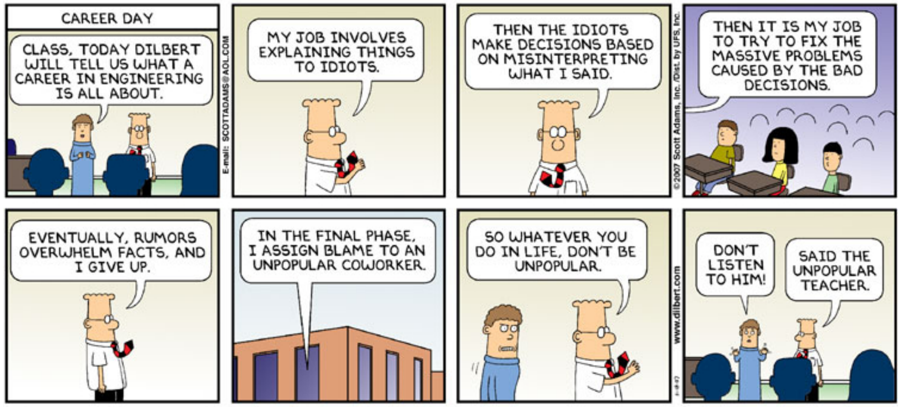
read()

read()

- A producer thread constantly produces values and writes them into a shared buffer
- A consumer thread reads a value from the shared buffer and uses it
- Premise: Every value must be consumed exactly once
- Question: How to synchronize those two?

Summary

- Create and start threads
- Thread synchronization
- Problem with synchronization: Deadlocks
- Producer/Consumer pattern


CAREER DAY
CLASS, TODAY DILBERT WILL TELL US WHAT A CAREER IN ENGINEERING IS ALL ABOUT.
E-mail: SCOTTADAMS@AOL.COM
MY JOB INVOLVES EXPLAINING THINGS TO IDIOTS.
THEN THE IDIOTS MAKE DECISIONS BASED ON MISINTERPRETING WHAT I SAID.
©2007 Scott Adams, Inc./Dist. by UFS, Inc.
THEN IT IS MY JOB TO TRY TO FIX THE MASSIVE PROBLEMS CAUSED BY THE BAD DECISIONS.
EVENTUALLY, RUMORS OVERWHELM FACTS, AND I GIVE UP.
IN THE FINAL PHASE, I ASSIGN BLAME TO AN UNPOPULAR COWORKER.
SO WHATEVER YOU DO IN LIFE, DON'T BE UNPOPULAR.
www.dilbert.com
DON'T LISTEN TO HIM!
SAID THE UNPOPULAR TEACHER.

© Scott Adams, Inc./Dist. by UFS, Inc.