Lec 17

Threads
Single processor

Diagram:
- CPU
- Memory
- I/O
Multi processes

Eclipse

<table>
<thead>
<tr>
<th>code</th>
<th>data</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>registers</td>
<td>stack</td>
<td></td>
</tr>
</tbody>
</table>

thread →

PPT

<table>
<thead>
<tr>
<th>code</th>
<th>data</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>registers</td>
<td>stack</td>
<td></td>
</tr>
</tbody>
</table>

thread →

iClicker

<table>
<thead>
<tr>
<th>code</th>
<th>data</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>registers</td>
<td>stack</td>
<td></td>
</tr>
</tbody>
</table>

thread →
Multi processor

CPU

CPU

Memory

I/O

Intel Core i7

Intel Core i7
Multi-core
Logical Cores
Multi-threaded
Creating Threads

1. Write code for thread to execute
   public class Example implements Runnable {
     public void run() {
       // your code here
     }
   }

2. Create the thread and link to (1)
   Example ex = new Example();
   Thread thread = new Thread(ex);

3. Start the thread
   thread.start();
public class PrintChar implements Runnable {
    private char charToPrint;

    public PrintChar(char c) {
        charToPrint = c;
    }

    public void run() {
        for(int i = 0; i < 5; i++) {
            System.out.print(charToPrint);
        }
    }
}

public class Example {
    public static void main(String[] args) {
        Runnable a = new PrintChar('a');
        Runnable b = new PrintChar('b');

        Thread t1 = new Thread(a);
        Thread t2 = new Thread(b);
        t2.start();
        t1.start();
    }
}
main()
main()
t2.start()
main()
t2.start();
t1.start()
Creating Threads, pt II

• Create a class that extends Thread and implements run()

```java
public class Example extends Thread {
    public void run() {
        // your code here
    }
}
```

• Start the thread

```java
Thread thread = new Example();
thread.start();
```
public class Sum extends Thread {
    int start, stop, sum;
    public Sum(int start, int stop) {
        // initialize vars.
    }
    public void run() {
        for(int i = start; i <= stop; i++) {
            sum += i;
        }
    }
}

public class Example {
    private static final int SUM_NUMBERS = 5;
    public static void main(String[] args) {
        Thread t1 = new Sum(0, SUM_NUMBERS);
        t1.start();
        System.out.println(((Sum) t1).sum); // note: this is ok!
    }
}

A) Compiler error
B) 0
C) 15
D) Number between 0 and 15
E) B, C, or D

Note: when we did this in class, it was Either B or C. D is possible, but very unlikely
Thread thread = new Sum(0, SUM_NUMBERS);
thread.start();

try {
    thread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
public class Sum extends Thread {
    int start, stop, sum;
    public Sum(int start, int stop) {
        //initialize vars.
    }
    public void run() {
        for(int i = start; i <= stop; i++) {
            sum += i;
        }
    }
}

class Example {
    private static final int SUM_NUMBERS = 5000000;
    public static void main(String[] args) {
        Thread t1 = new Sum(0, SUM_NUMBERS);
        t1.start();
        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(((Sum) t1).sum); //note: this is ok!
    }
}

How would you modify this to compute the answer faster
Reasons to use threads

• HW1
• HW3?

• Avoid blocking on background actions
  – Should your GUI stop updating when waiting for, say, the hard drive to return a value?
  – Two terrible examples:
    • Your GPS not updating the screen while doing something else
    • Windows Print manager locking down the whole system
Threading

• You get up for the morning and here are your tasks before you can leave the house:
  – Brew coffee (takes 5 minutes)
  – Pour coffee in to-go mug (1 minute)
  – Get dressed (4 minutes)
  – Make cereal (2 minutes)
  – Eat cereal (5 minutes)
  – Brush teeth (2 minutes)
  – Turn on tablet to download podcast (3 minutes)

How long did it take to get ready?
Events are threads!

• Each event handler is a thread
  – helps with the User Experience (UX)
Can/should we parallelize or thread everything?

• Suppose you are playing a board game. Why make everyone take their turn in sequence? Can’t everyone go at the same time?
class WithdrawHandlerClass implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        Runnable r = new WithdrawThread();
        Thread t = new Thread(r);
        t.start();
        try { t.join();} catch...
        b.setText("$ " + balance);
        if(balance < 20) {
            btnWithdraw.setDisable(true);
        }
    }
}

class WithdrawThread implements Runnable {
    public void run() {
        int current_balance = balance;
        try {
            Thread.sleep(1000);
        }
        catch(Exception exc) {
            System.out.println(exc);
        }
        if(current_balance >= 20) {
            current_balance -= 20;
            balance = current_balance;
            System.out.println("Withdrew $20!");
        }
    }
}

What will happen when the user clicks the “Withdraw $20” button twice?

A. Nothing
B. An error
C. Java will print a message
D. The balance variable will decrease to 0
E. The balance variable will decrease to negative 20
Synchronization

How can we fix this?
1. Make sure only one thread is executing by making sure the method which operates on a shared variable is “synchronized”
   ```java
   synchronized void deposit(int amount)
   ```

2. (Advanced) Use locks explicitly to ensure 1 thread in critical region
   ```java
   Private static Lock lock = new ReentrantLock();
   lock.lock();  // take the lock, then do something
   lock.unlock(); // release the lock
   ```
“Mr Homakov worked out that making two web browsers transfer money between the same cards, at the same time, sometimes duplicated the transfer and added funds to a gift card that had not been paid for.”
Threading vs. Parallel Programming

• Threading
  – Commonly refers to the scenario where you launch a thread to take care of something which may take a while
  – Think coffee maker example (happens in the background)

• Parallel Programming
  – Commonly refers to the scenario where you have more than one computing resource and are trying to divide up your task
  – Think *summing receipts* example