

## Transmission Control Protocol

- o TCP provides a connection-oriented, reliable, byte stream service (RFC793)
- o TCP is an independent, general purpose protocol that can be adapted for use with delivery systems other than IP.

## TCP Streams

- o A stream of 8-bit bytes is exchanged across a TCP connection.
- o The treatment of the byte stream by TCP is similar to the treatment of a file by the UNIX operating system.
- o Connections provided by TCP allow concurrent transfer in both directions. Such connections are called *full duplex*.
- o TCP always involves a client and a server and they can be on the same machine

## TCP Ports

- o TCP uses protocol port numbers to identify the ultimate destination within a machine.
- o How does one determine the port to communicate with?
  - o Well-known Ports (list shown in class)
  - o Randomly Assigned Ports

## TCP Classes

- o [see examples given in class]

## Applications

- o There are some interesting application level classes in C#
- o See the Programmer's Cookbook for details

## What if?

- o You want to have an application with a GUI and multiple network connections?
  - o How does your program remain responsive to both the user and the connections?
- o An example: An animation should run, but you should still be able to click buttons

## Processes

- o Your computer does this all the time:
  - o Many different programs or processes can run at one time even when there is only 1 cpu. They take turns running for short amounts of time.

## Threads

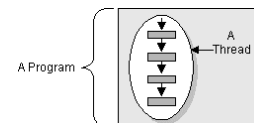
- o A thread is a flow of control in a program. It is a **lightweight** process
- o You may have multiple threads of execution running concurrently in a single program.
- o Example: the C# garbage collector runs when your program is also running

## Processes and Threads

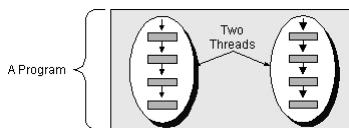
- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>o Processes<ul style="list-style-type: none"><li>o Allow different programs to multitask</li><li>o Do not share the same address space</li><li>o Context switching between processes may be expensive</li><li>o Communication between processes may be expensive</li></ul></li></ul> | <ul style="list-style-type: none"><li>o Threads<ul style="list-style-type: none"><li>o Allow multitasking within a single program</li><li>o Share the same address space</li><li>o Context switching between threads is generally inexpensive</li><li>o Communication between threads is generally cheap</li></ul></li></ul> |
|--|--|

## Single Threads

- o Good when you have a simple program
- o Easier to debug



## Multiple Threads



- o Some uses:
  - o To move a time-consuming initialization task out of the main thread, so that the GUI comes up faster. Examples of time-consuming tasks include making extensive calculations and blocking for network or disk I/O (loading images, for example).

## Multiple Thread Uses (cont'd)

- o To move a time-consuming task out of the event-dispatching thread, so that the GUI remains responsive.
- o To perform an operation repeatedly, usually with some predetermined period of time between operations. Good for animations.
- o To wait for messages from other programs.

## Exercise Caution

### o Don't overuse threads

- o "The first rule of using threads is this: avoid them when you can. Threads can be difficult to use, and they tend to make programs harder to debug."

– Javadoc Tutorial (<http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html>)

## Understanding Threads

### o Questions about Threads:

- o What code does a thread execute?
- o What states can a thread be in?
- o How does a thread change its state?
- o How does synchronization work?

## Creating and Using Threads

### o In C#:

- o A parameterless delegate called ThreadStart

### o In Java:

```
public interface Runnable {  
    public void run();  
}
```

```
public interface Executor {  
    void execute(Runnable command);  
}
```

Why not inherit from Thread?

## Inheritance?

### o Don't extend the Thread class unless you really need to in Java:

- o Extending the Thread class means that the subclass can not extend any other class
- o A class might only be interested in being runnable, and therefore inheriting the full overhead of the Thread class would be excessive.
- o This is good as the Thread class in C# is sealed!

## Delegates

- o Basically, they make methods like a variable type
- o This allows methods to be passed around and called
- o Very useful for handling events (key press, mouse move)
- o See Delegates section of Java vs. C#

## Example : ThreadStart

Please Copy and Run

```
using System;  
using System.Threading;  
public class CounterThread {  
    public static void run() {  
        for ( int i=0; i<10; i++)  
            Console.WriteLine("Count: " + i);  
    }  
  
    public static void Main(string[] args) {  
        Thread ct = new Thread(  
            new ThreadStart( run ) );  
        ct.Start();  
        Console.ReadLine();  
    }  
}
```

## Now Do the Following

---

- o Add a print statement before and after the `ct.Start()` line. What happens and why?
- o Look up the `Thread` class/members on the C# API page
- o Use `Thread.Sleep` in different parts of the code and see how it affects the running of the program. As a note: `Sleep` will always yield your current thread
- o What if you increase the count in the for loop?
- o What is you make 5 threads and run them all?

## And Then...

---

- o Look at examples for threads that the MS site has and from your book. Play with how threads work and their different abilities.

## Some Thread Info

---

- o Description of threading in C#: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconthreadsthreading.asp>
- o Start on multithreading in C#: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconUsingThreadsThreading.asp>