# Threads

# Motivation

- Begin looking at concurrent programming

# Threads

- “The assembly language of parallel programming”
- Very little abstraction:
  - Low overhead
  - Available nearly everywhere
  - Little safety
  - Hard to reason about concurrent operations

# Idea

- We can create threads easily using *lambda functions*
  - Lambda function = Anonymous function
- Need to do:
  using System.Threading;
- To create the thread:

```
var T = new Thread( () => {
    Console.WriteLine("im in ur thread");
});
T.start();
```

# Uses

- We can use threads to organize tasks that need to go on concurrently
- Ex: Game program
  - AI thread (maybe one per enemy/NPC)
  - Network thread (get/send packets)
  - User input thread (wait for keypresses/mouse clicks/gamepad events)
  - Graphics thread (render images)
  - Sound thread (do 3D audio computations)
  - Physics thread

# Waiting

- Sometimes, we need to be sure some other thread has finished its work
- Use join() function from the thread that should wait: T.join()
  - Blocks until thread T finishes
  - Then join() returns

# Example

- Suppose we have a several files on disk
- We want to know total of how many times the word “foo” appears in the files
- Example: Program.cs

# Output

- Output:

```
File test2.txt has 6 instances of 'foo'
File test1.txt has 3 instances of 'foo'
File test3.txt has 1 instances of 'foo'
```

- When I run it again:

```
File test1.txt has 3 instances of 'foo'
File test2.txt has 6 instances of 'foo'
File test3.txt has 1 instances of 'foo'
```

- Notice: No guarantee of order

# Question

- What if we want to just print the sum total?
- Need the threads to communicate back to main function
- How can threads communicate?
  - Need to use some sort of shared memory
- Example: Program.cs

# Important Points

- Need to send array to function
  - Ordinary integer wouldn't do it!
- Could have used static variable, but that's kind of messy
  - Program.cs
  - Not obvious that threads modify results unless you read the thread code

# Race Condition

- Suppose we decide, “Why use an array when an integer will work?”
- We code it like so:
  Program.cs
- It gave the correct result when I ran it once.
  - Package it up and ship it!
  - Right?

# Problem

- Consider this code:
  Program.cs
- What's the output?

# Result

- I got this:
  Foo: 2246931
  - That seems...Wrong...
  - Why?

# Implementation

- Suppose we have this statement:
  result++
  - This is just like
    result = result + 1
- What does CPU do?
  - Load value of result to register
  - Add 1 to register
  - Write value of register to memory

# Problem

- If two threads do this at the same time: Corruption!
  - Diagram on board...
- What's the solution?

# Solution

- Option 1: Make sure threads write to distinct memory locations
  - Like the array example we showed previously
- Option 2: Use special function for increment:
  Interlocked.Add( result, 1 );

# Pitfall

- Beware of variable capture!
- Ex: Consider this:

```
public class X{
    public static void Main(string[] args){
        string[] x = new string[]{ "foo","bar","baz","bam" };
        for(int i=0;i<x.Length;++i){
            new System.Threading.Thread( () => {
                System.Console.WriteLine(x[i]);
            }).Start();
        }
    }
}
```

- What gets printed (ignoring race conditions with the console)?

# Pitfall

- Indeterminate!
- When I run this, I get an error:

```
[ERROR] FATAL UNHANDLED EXCEPTION: System.
    IndexOutOfRangeException: Index was outside the bounds of the
     array.
  at capture.cs:6
```

- How?

# Problem

- This is an example of *variable capture*
  - (Diagram on board)

# Solution #1

- foreach loops don't exhibit capture

```
public class X{
    public static void Main(string[] args){
        string[] x = new string[]{ "foo","bar","baz","bam" };
        foreach(var s in x ){
            new System.Threading.Thread( () => {
                System.Console.WriteLine(s);
            }).Start();
        }
    }
}
```

# Solution #2

- Make a local copy of the loop counter:

```
public class X{
    public static void Main(string[] args){
        string[] x = new string[]{ "foo","bar","baz","bam" };
        for(int i=0;i<x.Length;++i){
            int i_ = i;
            new System.Threading.Thread( () => {
                System.Console.WriteLine(x[i_]);
            }).Start();
        }
    }
}
```

- Note that if ’int i_’ was declared outside the loop, this would not work

# Solution #3

- Factor out code into a separate function:

```
public class X{
    static makeThread(string[] x, int i){
        new System.Threading.Thread( () => {
            System.Console.WriteLine(x[i]);
        }).Start();
    }
    public static void Main(string[] args){
        string[] x = new string[]{ "foo","bar","baz","bam" };
        for(int i=0;i<x.Length;++i){
            makeThread(s,i);
        }
    }
}
```

# Assignment

- Write a program which takes a single *command line argument*
- This will be the name of a file with an arbitrary number of lines in it
- Each line will be a http URL
- Download the files specified to the local computer via http.
- Your code should download all the files simultaneously.
  - Name the file according to the last element of the URL path. IF there is no last element, use “index.html” as the name
  - Ex: If the file contains the line
    http://www.example.com/foo/bar
    save the result to the file “bar”.
- I suggest using System.Net.WebClient to do the work
- More details follow...

# Assignment

- Your program must be correct and avoid race conditions and the possibility of incorrect results. (Remember: Just because you don't see corruption after a single test run doesn't mean it can't or won't happen! You must reason through the logic of your code to make sure problems cannot happen.)
- Your program should exit once all the data has been downloaded. **Don't busy wait!**
- Turn in your source code (.cs files) only.
  - Don't zip up your VS project folder (it will include a bunch of project and debug stuff. I don't want 15GB worth of data to download when I go to grade the code.)
- Target .NET Framework, ***not .NET Core!***

# Sources


- Microsoft Corp. .Net documentation. https://docs.microsoft.com/en-us/dotnet/api

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org