Administrative Announcement

- Assignment #3 released
  - Using LINQ and Reflection
  - Due Friday 2/13 by midnight
  - Questions?
Threading Overview

- Threading provides “concurrent” execution
  - As opposed to sequential execution
  - Single processors can give illusion of concurrent execution (time slicing)
  - Multiprocessors and multicore can give true concurrent execution

- Threads vs. Processes
  - Threads share address space
  - Less expensive to communicate within threads
Creating Threads

- How to create threads?
  - System.Threading namespace
  - New Thread instance with ThreadStart delegate
    - public delegate void ThreadStart()
  - Start() method to start the thread
Thread Operations

- Thread.Abort method
  - Terminate a thread
  - Raise ThreadAbortException
  - Can be suppressed with
    Thread.ResetAbort();

- Thread.Sleep method

- Thread.Join method
  - Wait for the completion of another thread
  - Better than polling on Thread.IsAlive

- Thread.Priority property
Thread Synchronization

- What happens when two threads access the same data?

  ```
  public int Inc(ref int x) { return ++x; }
  ```

  - What happens when called by two threads at the same time?
Thread Synchronization

- Synchronization primitives
  - Way to ensure that only one thread executes code in a “critical section” at once
- C# provides
  - Lock statement
  - Monitor class
  - Interrupts
  - Several others (see Birrell's paper or MSDN)
Lock

- Basic idea: each object has a lock
- public int Inc(ref int x) {
  lock(this) return ++x;
}
  - Lock prevents more than one thread from entering at same time
  - Forces sequential order
Lock

• What should we lock on?
  – For instance variables: `this`
  – For globals and statics:
    `typeof(container)`
  – Something that will be the same for all threads that access this shared memory
ThreadPool

• Instead of explicitly creating threads
  – Create a pool of threads
  – Enqueue jobs with QueueUserWorkItem
    • Takes a WaitCallback delegate, to be passed to worker threads

• Good for large amounts of parallel work
  – “embarrassingly parallel” problems
  – Automatically scales to number of processors

• ThreadPool.cs
Pointer Types

- A separate category of types
- A pointer is a variable whose value is a memory address
- Can be used like C pointers
  - Dereference, get address of variables, increment ..etc
Unsafe Mode

- Sometimes need access to pointers
  - e.g. Access to OS, memory mapped devices, or implement time-critical algorithms

- Use the `unsafe` modifier

```c
unsafe static void swap(int *x, int *y) {
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```

```c
int x=0, y=1;
unsafe { swap(&x, &y); }  
```
Pointer Details

- Can only refer to value types
  - Can not be refer to a reference type
  - Can not refer to as struct that contain reference types
- Can be passed as `ref` or `out` function parameters
- No pointer arithmetic allowed on `void*`
- `stackalloc` gets memory form the stack
- **Note:** `int* a, b; // NOT int *a, *b;`
Compiler Setting

- Must turn on the `/unsafe` compiler flag
  - `-unsafe` in mono compiler
- Can be done in visual studio by selecting the appropriate checkbox from the project properties box.