Yin Lou
CS 2026, Spring 2010

Threading and Unsafe Code

Announcement

- Assignment 2 will be graded today
  - See feedback on CMS
- Assignment 3 is released
  - due this Sunday

Review

- LINQ
- LINQ Operators

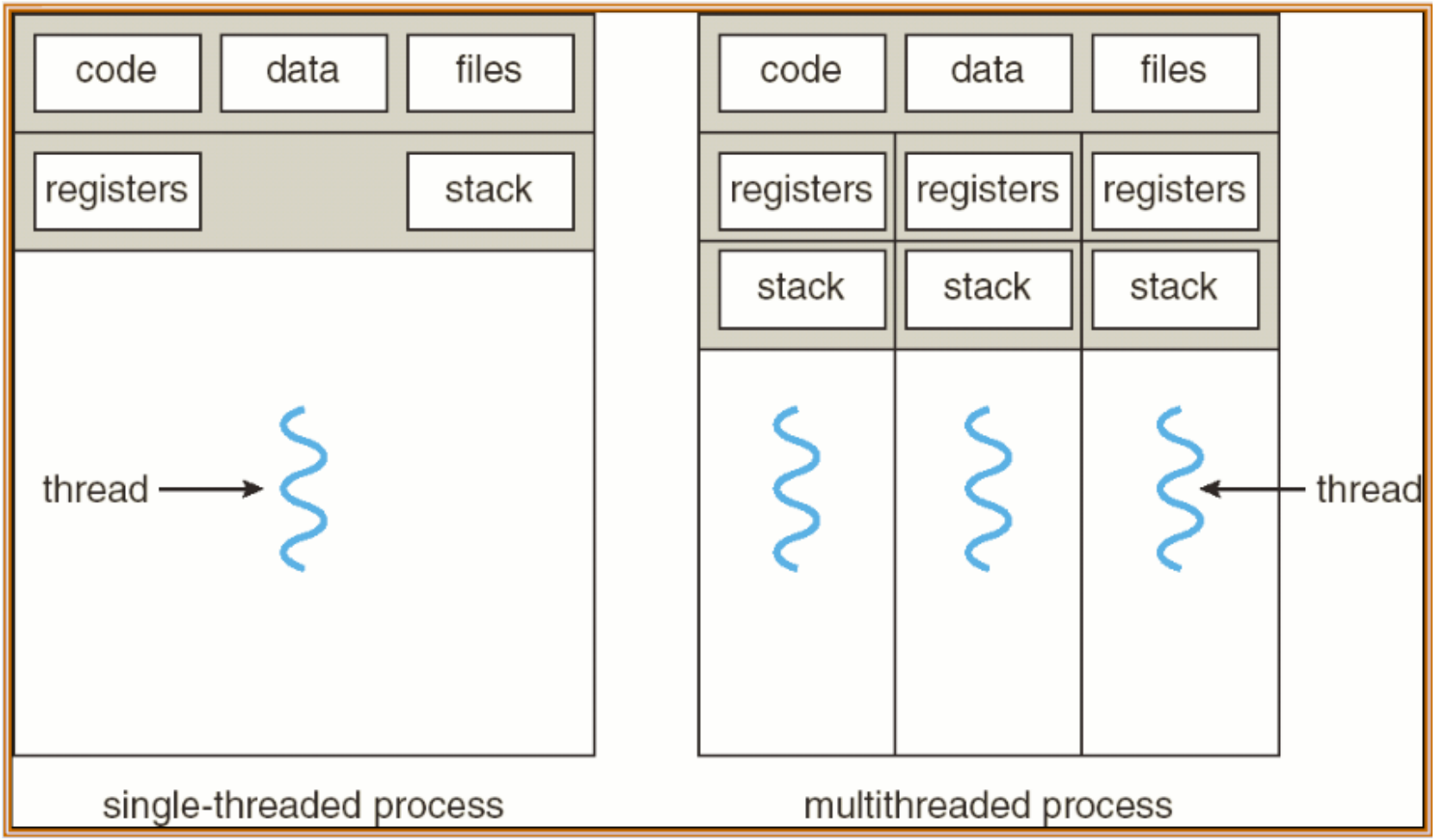
Outline

- Concept of Threading
- Synchronization
- Unsafe code

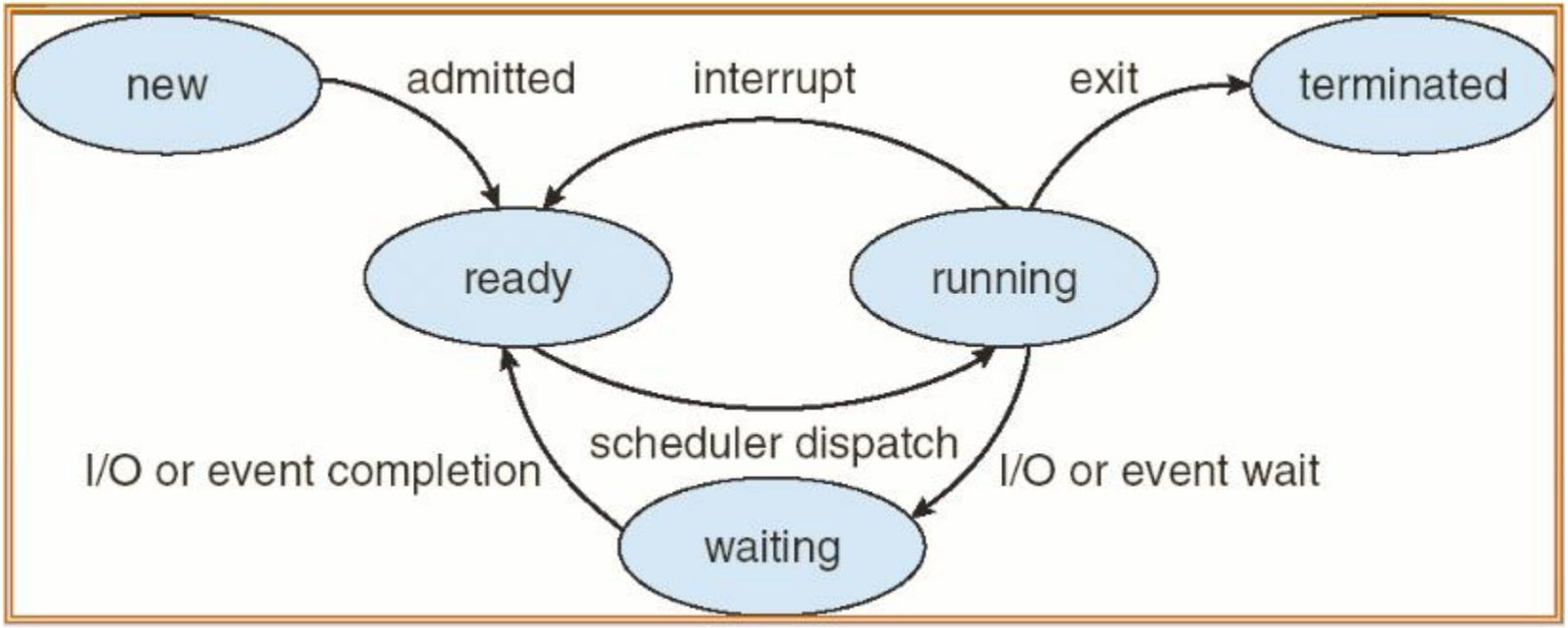
Threading Overview

- 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

Single-Threaded vs Multi-Threaded



Thread States



Create a Thread

- How to create a thread
  - 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 supressed with Thread.ResetAbort()
- Thread.Sleep method
- Thread.Join method
  - Wait for the completion of another thread
  - Better than busy-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 region at once
  - Called “critical section”
- C# provides (mostly in System.Threading)
  - lock statement
  - Monitor class
  - Interrupts
  - Several others (see Birrell's paper or MSDN)

Lock

- Basic idea: each object has a lock
- public int Increment(ref int x) {lock(this) return ++x;}
  - lock prevents more than one thread from entering
  - 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

ThreadPool Example

```
namespace ThreadPoolExample
{
        class Program
        {
                static void Main(string[] args)
                {
                        for (int i = 0; i < 20; i++)
                        {
                                ThreadPool.QueueUserWorkItem(
                                        new WaitCallback(DoWork), i);
                        }
                }

                static void DoWork(object state)
                {
                        int threadNumber = (int) state;
                        Console.WriteLine("Thread {0} reporting.", state);
                }
        }
}
```

Pointer Types

- A separate category of types
- A pointer is a variable whose value is a memory address
- Can be used like C pointers
  - Deference, get address of variables, increment ..etc
  - int x = 10; int* px = &x; *px = 5;
  - AStruct* pa = …; pa->mf; (*pa).mf
  - pa++, pa--, pa + pb, pa > pb

Unsafe Mode

- Sometimes need access to pointers
  - e.g. access to OS, memory mapped device, or implement time-critical algorithms
- Use the unsafe modifier
  - ```
    unsafe static void swap(int* x, int* y)
    {
            int tmp = *x;
            *x = *y;
            *y = tmp;
    }
    ```
  - ```
    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 from the stack
- Note int* pi, pj; // NOT int *pi, *pj;

Compiler Setting

- Turn on the /unsafe compiler flag
- Or open Visual Studio project properties