Course announcement for Spring ‘13
Com S 430
Advanced Programming Tools

Unofficial subtitle: Concurrency, asynchronous events, and frameworks for application integration (including .NET)

Time: MWF 12:10 – 1:00

Instructor: Steve Kautz, www.cs.iastate.edu/~smkautz

Employment opportunity: I may be looking for a TA for this course. If you don’t need to take the course for credit, but you have a strong interest and have some familiarity with concurrent programming, let’s talk.

Prerequisites: Com S 309 or 319, Com S 362 or 363, English 250, and Speech Communication 212, or permission of instructor. (Note that the catalog still lists Com S 311, not 309/319, but the latter are more relevant.) Contact me if you have questions.

Catalog Description

Topics in advanced programming techniques and tools widely used by industry (e.g., event-driven programming and graphical user interfaces, standard libraries, client/server architectures and techniques for distributed applications). Emphasis on programming projects in a modern integrated development environment. Oral and written reports. Nonmajor graduate credit.

What it’s really about

The key areas we want to investigate are:

- Concurrency fundamentals – what are threads, using low-level primitives (such as Java synchronization locks), waiting and notification, the problems of memory visibility
- Utilities and patterns for programming concurrent systems - thread-safe queues, confinement, futures, asynchronous proxies, task management
- Application integration – the role of “middleware,” remote procedure calls, remote objects and ORBs, messaging, web services, application servers
- Patterns for messaging and event-driven systems
- Modern tools for integration – what can we learn from J2EE and .NET?

Rationale: Big modern applications tend to be made up of a bunch of heterogeneous, loosely coupled, distributed components that have to communicate somehow. Often these components are legacy systems that already exist but have to be integrated with other systems or new
services, and in many cases it’s the integration that is the interesting (hard) part. Sometimes the communication can be *synchronous*, like a method or procedure call, where the caller is essentially inactive until the result comes back. However, in most cases the communication is *asynchronous* and takes place through *messages* or *events*: a caller sends a request and goes on with life, and at some future time (provided nothing has failed in the meantime) a reply comes back. The components themselves, in order to handle the asynchronous arrival of messages or events, have to be written with multiple threads of control, introducing the problem of *concurrency*.

The problem of programming concurrent applications is extremely interesting in itself and presents a number of challenging issues for programmers that are hard to get right. The basic problem is that multiple executing tasks (threads) are sharing the same memory heap, so you have to coordinate their activities so that memory isn’t corrupted and so that each thread sees a valid “view” of the objects in memory. Concurrency bugs are memory corruption problems that are hard to reproduce and find because they occur only sporadically, usually due to the effects of the scheduling of tasks or the unpredictable timing of memory writes in the presence of caching.

Most of us learned about semaphores and locks in an OS course, but for applications programming in an OO language, the issues are somewhat different and we want to work at a higher level of abstraction. An understanding of concurrency is increasingly important these days as there is more and more incentive to develop applications that can take advantage of multicore processors.

Some of our time will be dedicated to an investigation of the communication and data management facilities provided by .NET, which will necessitate learning something about C#. I would also like to investigate the new .NET libraries for parallelism and compare them with the more mature java.util.concurrent APIs.

**Workload**

Note that this is a “Group B” course in the CS curriculum, meaning that it satisfies the requirement for oral and written communication in a 400-level course. So you can expect your documentation and presentations to be scrutinized :) Planned deliverables include:

- A semester-long group project, planned in consultation with the instructor, including documentation, working code, and a final presentation.
- An individual tutorial presentation on a relevant topic
- Some programming homework involving .NET, concurrency utilities, or both
- A midterm exam
Probably Required Textbooks

Brian Goetz et al, *Java Concurrency in Practice*, Addison-Wesley 2006. Regardless of what language you’re using, this is the best introduction to concurrent programming from the high-level programmer’s point of view (as opposed to someone writing an OS kernel).
http://www.jcip.net/

Andrew Troelsen, *Pro C# 5.0 and .NET 4.5 Framework, 6th Edition*, Apress 2012. I have not had a chance to review this edition yet, but I used a previous edition and found it very helpful for getting .NET stuff to work.

There will be additional readings, e.g. on web services and messaging patterns, but I think I can just put them on reserve in the library or post online. E.g. this book is a valuable resource: Alonso et al, *Web Services: Concepts, Architectures, and Applications*, Springer 2003.
http://www.amazon.com/Web-Services-Gustavo-Alonso/dp/3540440089

A few articles to check out for more information if you are curious

“Starbucks does not use two-phase commit”, an entertaining essay about asynchronous communication by Gregor Hohpe.
http://www.eaipatterns.com/ramblings/18_starbucks.html

A good overview of the problems of application integration is the introduction to Gregor Hohpe’s big book on messaging patterns.
http://www.eaipatterns.com/Introduction.html

The JSR 133 FAQ is a good overview of the issues of thread synchronization and memory visibility. The authors talk specifically about Java, but the issues will essentially be the same using C# or other imperative languages.

If you are not at all familiar with the problem of thread synchronization, the Java tutorials provide a concise explanation.
http://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html