Winter 2006

CEG 730-01: Distributed Computing Principles

Prabhaker Mateti
Wright State University - Main Campus, prabhaker.mateti@wright.edu

CEG 730: Distributed Computing Principles

Dr Prabhaker Mateti
pmateti@cs.wright.edu (937) 775 5114

Catalog Description: [4 Credit Hours] Communicating sequential processes, clients and servers, remote procedure calls, stub generation, weak and strong semaphores, split-binary-semaphores, and distributed termination. Example languages: SR, Linda. **Prerequisite:** CEG 634.

Home Page http://www.cs.wright.edu/~pmateti/Courses/730/ Bookmark this and visit it periodically. All course related announcements will be posted on this page.

Source Materials

- Text Book: Gregory R. Andrews, *Concurrent Programming: Principles and Practice*, Benjamin/Cummings, 1991. The book is now in its fifth printing. Errata sheets are available for the first (www.cs.arizona.edu/sr/cperr1.ps) and second (www.cs.arizona.edu/sr/cperr2.ps) printings.
- A book of your choice on client/server programming in Java, and
- Other references listed below.

This course is about the foundations and principles involved in distributed systems with an introduction to recent systems and languages. The next course (CEG 830) concentrates on systems related issues, and on the design of distributed systems and applications.

It is necessary that you are comfortable in the *use* of networked machines based on Unix and X11. It is also assumed that you are fluent in C/C++ and Unix process-related system calls. It is not assumed that you are familiar with either RPC, or Java.

I lecture on only the conceptually difficult sections of Andrews' book, leaving the rest as reading assignments.

Course Content

The following is tentative. The order of topics will be roughly as indicated, except for project related discussions. The numbers in parentheses at the end of each topic is an estimate of the number of (75-minute) lectures I intend to devote.

Fundamentals (4)

Programming logic. Pre-, post conditions, and loop invariants. Starvation, deadlocks, livelocks. Interference, fairness, safety and liveness properties. Fine-grain and coarse-

grain parallelism. Review of classic problems in concurrency. Review of semaphores, and conditional critical regions. Review of monitors. Proper definition of semaphore. Weak and strong semaphores. Split-binary semaphores. Starvation-free mutual exclusion. The technique of ``passing the baton."

1. Andrews, Chapters 1, 2, 3.1, 3.5, 4, 5, 6, and Section 10.2.
2. Jan Tijmen Udding, ``Absence of Individual Starvation Using Weak Semaphores," *Information Processing Letters*, Vol. 23, 1986, 159-162.

Remote Invocation (6)

Remote Procedure Calls (RPC), and Clients and Servers. Implementation of RPC. Stub generation, marshaling arbitrary data structures. Sun XDR and RPCGEN. Study of an RPC example. Remote Method Invocation (RMI) and Object Serialization of Java.

1. Programming with ONC RPC, a tutorial by Digital (Compaq), http://www.tru64unix.compaq.com/docs/base_doc/DOCUMENTATION/HTML/AA-Q0R5B-TET1_html/TOC.html
2. Java tutorial, http://java.sun.com/docs/books/tutorial/

Distributed Algorithms (4)

CSP, communicating sequential processes. Asynchronous and synchronous message passing. Logical clocks, ordering of events. Heartbeat, probe/echo, and broadcast algorithms. Distributed mutual exclusion, Distributed implementation of semaphores, and distributed termination detection.

1. C. A. R. Hoare, ``Communicating Sequential Processes," *Communications of the ACM*, 1978, Vol. 21, No. 8, 666-677.
2. Andrews, Chapters 7 and 8.

Languages for Distributed Programming (5)

SR: Multiple primitives, *Linda*: Distributed data structures., and Java, Jini and JavaSpaces.

1. Andrews, Chapter 10.
2. Greg Andrews and Gregg Townsend, The MPD Programming Language, www.cs. arizona. edu/ mpd/
3. Sudhir Ahuja, Nicholas Carriero and David Gelernter, ``Linda and Friends," *IEEE Computer* (magazine), Vol. 19, No. 8, 26-34.
4. Java tutorial, http://java.sun.com/docs/books/tutorial/
5. JavaSpaces™ FAQ, http://java.sun.com/products/javaspaces/faqs/jsfaq.html

Grading

Class and Newsgroup Participation 5%

Concurrent algorithms and concepts are often subtle. Please participate in the class room discussions. Feel free to discuss openly, to ask what may appear to be dumb questions, and to catch any errors that I make. Be alert, study the backup material right away, and solve the exercises promptly.

No problems will be assigned specifically as ``home work." However, you are expected to have studied problems that appeared in the old exams of this course, and the exercises of our textbook.

There is a local newsgroup, called `wright.ceg.730` that you should read and participate regularly as a vehicle for discussing your solutions. I will read this group regularly, and note silently who is contributing what. I expect other students to be able to find flaws and fix them. Only when the most fundamental errors are being made will I break my silence and contribute a response.

Exams 30+35%

There will be two exams contributing 35% plus 35% to the final grade. The date for the midterm exam will be announced in class. The date for the final exam is as set by the Registrar. Both exams are closed-book, and comprehensive. Many of the old exams for this course are available at `gamma` in the directory `/usr/local/lib/Languages/730/Exams/`

Projects 15+15%

It is necessary that you are comfortable in the *use* of networked machines based on Unix and X11. It is also assumed that you are fluent in C and Unix process-related system calls. It is not assumed that you are familiar with either RPC, or Java.

There are two programming projects using

1. C/C++ and RPC, and
2. Java, Jini, JavaSpaces

that implement a multi-workstation oriented *WhiteBoard* (WB) client-server program. In each of these, you will start from a working and well-documented program written by Mateti. Working Versions of this program written in C and RPC, and also in SR are in `/usr/local/lib/Languages/730/` visible from all networked workstations of the College.

Project P0: Weight == 0%; Obtain the files either from gamma, or download them from here. This version will compile cleanly on gamma. It will need tweaking on other platforms. A Linux i386 version that compiles and runs cleanly is here. Your first task is to run the C/X11/RPC version so that the server is on one node, and there are at least four clients on four different nodes of at least two different architectures and two different whiteboards. Turn in printouts of the X11 screen.

Project P1: Weight == 15%; Extend the P0 server as follows so that it now responds to

messages from a new ``client'' that we would call `wbadmin`.

- Server must respond to a query with full information (name of the board, names of the host machines where the boards are in use, etc.) regarding all whiteboards.
- Server must respond to a request to initiate a new server on a given node. The newly given node may be the same node as that of an existing server. The newly created server begins its life with no boards to serve.
- A server must respond to a request to transfer all clients on a given whiteboard from one server to another. This transfer should be transparent to the clients.

The projects are required. Not producing a demonstrable (even if faulty) project will award you a grade no better than a C in the whole course regardlesss of your performance in the exams. The projects must be work done by *you* as a *single individual*. The projects are evaluated both by a demo and by skimming the source code and associated documentation.

Project P2: Weight == 15%; Re-design the P1 above but using Java, Jini and/orJavaSpaces. Whether we will use Jini + JavaSpaces or limit ourselves to Java RMI will depend on how our class is progressing.

 33,302