Distributed Systems

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
Comp 734: Distributed Systems

Course Overview

This course will provide an implementation-oriented study of distributed systems. Some of the topics covered will include inter-process communication, group communication, synchronization, remote procedure call, peer to peer and centralized sessions, fire-walls, causal broadcast, atomic broadcast, scalability, fault tolerance, replication, and transactions/concurrency control. These are foundational concepts, which are becoming particularly relevant with the emerging areas of cloud computing and distributed games. These concepts will be introduced as layers in a general distributed infrastructure. Your projects will implement new layers and provide alternative implementations of some of the existing layers. When implementing a layer, you will act both as an application programmer, using abstractions of the layers below, and a systems programmer, defining abstractions for the layers above. The number of lines of code required by each layer will be relatively small; however the compositions of these layers will be complex.

The main difference between this course and a distributed programming/theory course is that it will address the design and implementation...
# Lectures and Assignments

## Schedule (Tentative)

<table>
<thead>
<tr>
<th>Unit (Start Date)</th>
<th>Slides</th>
<th>Chapters</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>PowerPoint 2007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Threads and Thread Coordination (Read on your own)</td>
<td>PowerPoint 2007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java Non-Blocking Socket Channel I/O</td>
<td></td>
<td></td>
<td>Distributed Non-Blocking Halloween Simulation</td>
</tr>
<tr>
<td>Java Remote Method Invocation</td>
<td></td>
<td></td>
<td>Distributed RMI-based Halloween Simulation</td>
</tr>
<tr>
<td>Sync replicated Objects</td>
<td></td>
<td></td>
<td>Replicated Sync-based Halloween Simulation</td>
</tr>
<tr>
<td>GIPC byte communication (Oct 3, 10)</td>
<td>PowerPoint 2007</td>
<td></td>
<td>Socket-based GIPC</td>
</tr>
<tr>
<td>NIC driver (Oct 24, 26, 31, Nov 2, 4)</td>
<td></td>
<td></td>
<td>Extendible Multi-Platform Serialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Synchronous Receive, Procedure and Function Call</td>
</tr>
</tbody>
</table>

- **No book**
- **PPT slides and sometimes word doc**
- **Current assignment is on the web - start working ASAP on it**
- **Outline of other assignments given**
## SOFTWARE

<table>
<thead>
<tr>
<th>Downloads</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beau Halloween Simulation (Library, keep it compressed)</td>
<td>beau_project.zip</td>
</tr>
<tr>
<td>Coupled Halloween Simulations (Eclipse project, uncompress and link to libraries)</td>
<td>CoupledTrickOrTreat.zip</td>
</tr>
<tr>
<td>ObjectEditor (Library)</td>
<td>oeall17.jar</td>
</tr>
<tr>
<td>GIPC</td>
<td></td>
</tr>
</tbody>
</table>

Software to be continuously updated
# Grade Distribution

<table>
<thead>
<tr>
<th>Category</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exams (Two midterms, no final)</td>
<td>40%</td>
</tr>
<tr>
<td>Assignments (Home work)</td>
<td>60%</td>
</tr>
<tr>
<td>Fudge Factor (Class participation, other factors)</td>
<td>10%</td>
</tr>
</tbody>
</table>
GETTING HELP

Can discuss solutions with each other at a high level

Not at the code level

Sharing of code is honor code violation

Can help each other with debugging as long as it does not lead to code sharing

Assignments may contain solution in English (read only if stuck)
Getting Help and Class Discussion

We will be using Piazza for class discussion and getting help. The system is highly catered to getting you help fast and efficiently from classmates, the TA, and myself. Rather than emailing questions to the teaching staff, I encourage you to post your questions on Piazza. If you do not get a response within a day or two on Piazza, please send mail to help401@cs.unc.edu. But try Piazza first. Do not send mail to an individual instructor, as that can overwhelm him - such mail will be ignored.

Before posing a question, please check if this question has been asked before. This will reduce post clutter and reduce our burden. Repeat questions will be ignored by the instructors.

Piazza allows anyone to respond. So if you see a question that you think you can respond to, please do so, as that will reduce our burden and help you “teach” your fellow students.

This will be a form of class participation that will be noted when I allocate my fudge points!

Hope it works well

If you have any problems or feedback for the developers, email team@piazza.com.

Find our class page at: https://piazza.com/unc/fall2013/comp734
DISTRIBUTED PROGRAM?

A program “involving” multiple computers

Specific computers must be bound at run time

→ Program can run on a single computer

Definition involves processes
Program vs. Process vs. Thread

Program

Process is an execution instance of program, associated with program and memory.

Same program can result in multiple processes.

Execution instance

Processes are independent activities that can interleave or execute concurrently.

Thread

Thread is also an independent activity, but within a process, associated with a process and a stack.
**Distribution of Processes/Threads**

- Different processes can execute on different (distributed) computers

- A single process executes on one machine
Connected process pair: Some computation of a process can be influenced by or influence computation of the other process.

Connected process group: Each process is coupled to at least one other process in the group.

Graph created by creating pair-wise dependency links is not partitioned—every node reachable from every other node.
Logical vs. Physical Inter Process Connection Links

Physical coupling links are physical inter process communication links along which information flows in the network.

Logical links indicate computational dependencies.

Can have logical links without physical links.

Physical links usually imply logical links.

Relayer
Distributed Applications

Distributed applications?

Non distributed applications?

In today’s world, what is or should not be distributed?
**SOME DISTRIBUTED DOMAINS**

- Distributed Repositories (Files, Databases)
- Remotely Accessible Services (Printers, Desktops)
- Collaborative Applications (Games, Shared Desktops)
- Distributed Sensing (Disaster Prediction)
- Computation Distribution (e.g. Simulations)

**Full courses on some of these areas, with concepts specific to them**

- (Distributed Databases, Collaborative Applications)

**Will look at domain-independent concepts at the intersection of them**

**Will not take an application-centric view**

**Fundamental Issues?**
**DISTRIBUTION VS. CONCURRENCY**

**Process** → **Connection** → **Process**

- Distribution, no fine-grained concurrency

**Thread** → **Process** → **Thread**

- Concurrency, not distribution

**Thread** → **Process** → **Thread**

- Distribution and fine-grained concurrency (typical)
**Non-Distributed vs. Distributed Program**

**Non-Distributed**
- Creates a single process logically and physically unconnected to any other process
- Must deal with sequential and possibly concurrency issues

**Distributed**
- Creates a pair or larger group of connected processes
- Must also deal with distribution and usually concurrency issues
**SYSTEMS VIEWPOINT**

- System
- Operating System
- Database Management System
- Programming Languages
- Distributed Systems

**Computer abstractions to implement some class of programs**

- Processes, Files, Memory Management, Threads...
- Query Language, Transactions, ...
- Arrays, Loops, Classes, ...
- Data Communication, Remote Procedure Call (RPC), ...

**Byte/object communication consists of byte/object of exchange**

**RPC assumes communication consists of procedure requests and return value responses**
**Distributed Systems**

- Study of design and/or implementation of computer abstractions for developing distributed programs

- Why distributed systems?

- Why systems?

- Alternatives to understand how to program some domain of applications?

- Non distributed programs?
**Alternatives to Understanding**

Programming: Abstraction use

Programming: Use of a specific set of non-distributed abstractions (e.g., functional, MATLAB programming)

Distributed Programming: Use of a set of distributed abstractions (e.g., Socket/RPC Programming)

Systems: Abstraction design and/or implementation

Design and implementation of non-distribution abstractions (Object-Oriented vs. Functional Languages, Compilers/Interpreters)

Design and implementation of distributed system abstractions (e.g., Data Communication/RPC Design and/or Implementation)

Theory: Models and algorithms

Non-distributed model and algorithms (Turing Machines, HeapSort)

Distributed Models and Algorithms (e.g., 2-Phase commit, Group Comm. Model)
**Rationale**

Abstraction Design vs. Implémentation

Abstraction design linked to implementation: Designs are done of only efficiently implementable abstractions.

Abstractions vs. Theory (Models, Algorithms)

Abstractions are implemented operational models and have (the more) practical algorithms in them.

Abstraction Design & Implementation vs. Use

Maturity with design and implementation issues allows you to better understand the semantics of a specific abstraction.

Abstract implementations require advanced programming/software engineering techniques—“you can’t really program if you have not written a compiler”
TEACHING ABSTRACTION DESIGN & IMPLEMENTATION?

Lectures address design; assignments, implementation (e.g. Implement a PL interpreter in another PL)

Implementations can be complex and need instruction

Lectures give high-level pseudo code for complex algorithms; assignments full implementation (e.g. compilers)

Pain/gain ratio high, semester barely enough time for compiler

Lectures discuss code for a system of abstractions: assignments extend/modify this code

Code must be understandable and ideally also elegant
THE XINU APPROACH TO TEACHING OS

Layering

- Interrupt Management
- Thread Communication
- Thread Synchronization
- Thread Management

Approach not used in distributed computing
Need distributed system layers

Reuse of previous layers keeps code short (and hence presentable in class)
Can unravel a system in stages to a class
Layering good for software engineering as well as pedagogical reasons
**Layers Exist in Networking**

<table>
<thead>
<tr>
<th>UDP</th>
<th>TCP/IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>Link-Level Communication</td>
<td></td>
</tr>
<tr>
<td>Physical Communication</td>
<td></td>
</tr>
</tbody>
</table>

Physical communication in networking involves machines and used hardware machine addresses.

Physical communication in distributed systems is between processes and indicates routing of information among processes.
DISTRIBUTED VS. NETWORK LAYERS

Networking addresses physical connections and byte communication among processes.

No separate logical connections, object communication, synchronization, fault tolerance.

Low-level (hidden from programmers) abstractions.
## Distributed System vs. Networking Abstractions

<table>
<thead>
<tr>
<th>Programming Language Abstractions</th>
<th>Just as programming language abstractions are built on top of assembly language abstractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Language Abstractions</td>
<td>Distributed system abstractions are built on top of networked abstractions</td>
</tr>
<tr>
<td>Distributed Abstractions</td>
<td>Knowledge of assembly/networked abstractions important to implement PL/distributed abstractions</td>
</tr>
<tr>
<td>Networked Abstractions</td>
<td>Byte communication APIs, close to networked abstractions, is provided by operating systems (e.g. sockets), which hide networking abstractions</td>
</tr>
<tr>
<td>OS Byte Communication API</td>
<td></td>
</tr>
</tbody>
</table>
**Domain Independent?**

- Distributed Repositories (Files, Databases)
- Remotely Accessible Services (Printers, Desktops)
- Collaborative Applications (Games, Shared Desktops)
- Distributed Sensing (Disaster Prediction)
- Computation Distribution (e.g. Simulations)

**Distributed Abstractions**

**OS Byte Communication API**

Even though OS abstractions developed to build distributed OS (file systems), they are by definition domain-independent.

Will look at domain-independent concepts at the intersection of them.
Both operating systems and programming languages provide domain-independent abstractions

Operating systems support processes and language-independent abstractions for accessing protected info and sharing information among processes (files, IPC)

Programming languages must provide fine-grained abstractions needed within a process

They also provide an interface to OS abstractions through libraries or language constructs

They can also extend the OS abstractions (e.g. typed files)
## Language vs. OS, Distributed Abstractions

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte communication is all that operating systems provide</td>
<td></td>
</tr>
<tr>
<td>Non distributed programming languages such as C provide only OS abstractions</td>
<td></td>
</tr>
<tr>
<td>Distributed programming languages such as Java provide a richer variety of abstractions</td>
<td></td>
</tr>
<tr>
<td>Java provides threads and reflection, making it easy to implement our own replacements and extensions of Java abstractions</td>
<td></td>
</tr>
<tr>
<td>Will use Java as implementation language</td>
<td></td>
</tr>
<tr>
<td>To extend and replace Java abstractions/layers, knowledge of them useful</td>
<td></td>
</tr>
</tbody>
</table>
OBJECTIVES

At the end of the course you will .....
Internet/Cloud computing increasing relevance of the fundamental concepts
<table>
<thead>
<tr>
<th>Practical Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>For distributed applications, likely to use the code you implemented than existing abstractions</td>
</tr>
<tr>
<td>Can send objects over NIO socket channels</td>
</tr>
<tr>
<td>Existing Java RPC does not work on Android devices, but the one you implement will</td>
</tr>
<tr>
<td>Use Sync, which apparently is the basis of some new Mobile platforms</td>
</tr>
<tr>
<td>Will implement many abstractions not part of standard Java</td>
</tr>
</tbody>
</table>
SOFTWARE ENGINEERING PRINCIPLES

<table>
<thead>
<tr>
<th>Classes</th>
<th>Existing classes will be used, inherited but not modified directly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>Alternative implementations will create new classes implementing existing interfaces</td>
</tr>
<tr>
<td>Factories and Abstract factories</td>
<td>These will allow easy switching between different implementations</td>
</tr>
<tr>
<td>Generics</td>
<td>Implementation rather than use of generics to unite buffer and object communication</td>
</tr>
</tbody>
</table>

Will be both a distributed computing and software engineering course
Relevancy to OS

Inter-process communication key to design of new OS’s, even non distributed OS

Extensive use of bounded buffers

Will study and use thread synchronization in depth

Will study how distributed OS are implemented

Will gain understanding of fundamental OS concepts except memory management
Introduction to Systems

Systems: Abstraction design and implementation

Design and implementation of non-distribution abstractions (Object-Oriented vs. Functional Languages, Compilers/Interpreters)

Design and implementation of distributed system abstractions (e.g. Data Communication /RPC Design and/or Implementation)

Distributed systems covers concepts from many fields