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 of systems.
# 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, 29, Nov 2, 4)</td>
<td>PowerPoint 2007</td>
<td></td>
<td>Extendable Multi-Platform Serialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Synchronous Receive, Procedure and Function Call</td>
</tr>
<tr>
<td>Current assignment is on the web - start working ASAP on it</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outline of other assignments given</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Software

### Downloads

<table>
<thead>
<tr>
<th>Download Description</th>
<th>File Name</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, uncompressed 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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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 execution instance of program, associated with program and memory

Same program can result in multiple processes

Execution instance

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

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

Windows Task Manager

Image Name | User Name | CPU | Description
---|---|---|---
WLSync.exe | dewan | 00 | Windows
wlcomm.exe | dewan | 00 | Windows
winlogon.exe | | |
**Distribution of Processes/Threads**

- Different processes can execute on different (distributed) computers.
- A single process executes on one machine.
**DISTRIBUTED PROGRAM**

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.
- Physical links usually imply logical links.
- Can have logical links without physical links.

**Diagram:**

- Logical Connection
- Physical Connection
- 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 PROGRAM**

![Diagram showing distribution vs. concurrency.]

- **Process**
  - **Thread**
  - **Process**
  - **Thread**

- **Connection**

- **Distribution, no fine-grained concurrency**

- **Concurrency, not distribution**

- **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?
ALT E R N A T I V E S  T O  U N D E R S T A N D I N G

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-distributed 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. Implementation**

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

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

Approach not used in distributed computing
Need distributed system layers
Layers Exist in Networking

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)

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.
**Language vs. OS Abstractions**

<table>
<thead>
<tr>
<th>Both operating systems and programming languages provide domain-independent abstractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating systems support processes and language-independent abstractions for accessing protected info and sharing information among processes (files, IPC)</td>
</tr>
<tr>
<td>Programming languages must provide fine-grained abstractions needed within a process</td>
</tr>
<tr>
<td>They also provide an interface to OS abstractions through libraries or language constructs</td>
</tr>
<tr>
<td>They can also extend the OS abstractions (e.g. typed files)</td>
</tr>
</tbody>
</table>
## Language vs. OS, Distributed Abstractions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
Java Abstractions

- Blocking stream object communication (Object Stream)
- Blocking byte communication (Sockets)
- Non blocking byte communication (NIO)
- Remote procedure call (RMI)
Java Layers

Remote procedure call (RMI)

Blocking stream object communication (Object Stream)

Blocking byte communication (Sockets)

Non blocking byte communication (NIO)

OS Byte Communication

Go beyond Java layers?
**Beyond Java Layers**

- Sync Replicated Objects
- Remote procedure call (RMI)
- Blocking stream object communication (Object Stream)
- Blocking byte communication (Sockets)
- Non blocking byte communication (NIO)
- OS Byte Communication

Implementation
**GIPC: Improved Abstractions and Layers with Open Source**

<table>
<thead>
<tr>
<th>Scalability</th>
<th>Fault Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Synchronization</td>
<td></td>
</tr>
<tr>
<td>Group Communication and RPC</td>
<td></td>
</tr>
<tr>
<td>Pair-wise Synchronization</td>
<td></td>
</tr>
<tr>
<td>Pair-wise Byte, and Object Communication, Pairwise RPC</td>
<td></td>
</tr>
</tbody>
</table>

GIPC layers will be replaced, augmented with assignment layers.
COURSE PLAN PRINCIPLE

Cover material for next assignment (and other relevant material)

Do next assignment

Boundary conditions?
USE NON BLOCKING I/O

Lectures

Java NIO

Use NIO Tutorials and class lectures to Implement a Distributed Program

Assignments

Distributed Non Blocking Simulation

Existing non distributed simulation

Java NIO
HALLOWEEN SIMULATION

Make Beau Anderson’s 401 Halloween implementation distributed
Use Java Tutorials and class lectures to Implement an RMI-based Distributed Program

Lectures

Java RMI

Assignments

Distributed RMI-based Simulation

Existing non distributed simulation

RMI
Use Sync Replicated Objects

Lectures
Java Sync
Use class lectures and Sync to Implement an RMI-based Distributed Program

Assignments
Replicated Simulation
Existing non distributed simulation
Sync
**Blocking vs. Non Blocking Sockets**

**Lectures**
- High-level buffer communication
- Java (Non Blocking) Channels

**Assignments**
- Understand the differences between blocking and non-blocking communication
- High-level buffer communication
- Java (Blocking) Sockets
Recursion and Serialization

Lectures

- High-level object communication
- High-level Buffer comm.
- Java Object Streams

Assignments

- High-level object communication
- High-level Buffer comm.
- Custom Object Serialization

Understand serialization and really understand recursion
SYNCHRONIZATION AND RPC

Lectures

Remote Procedure Call

High-level Object comm.

Java Thread Synchronization

Assignments

Remote Procedure Call

High-level Object Communication with Synchronization

High-level Object comm.

Java Thread Synchronization

Use and implement pairwise synchronization
GROUP COMMUNICATION AND FAULT TOLERANCE

Lectures

Fault Tolerance
Group Communication
High-level Object comm. RPC

Assignments

More Efficient Fault Tolerance
More Functional Group Communication
High-level Object comm. RPC

Use and implement group synchronization and fault tolerance and group communication
LAST PHASE

Lectures

- More Efficient Fault Tolerance
- More Functional Group Communication
- High-level Object comm.
- RPC

Assignments

- Transactions?, Distributed Hashtables?, Multiprocessor systems?, ....
OBJECTIVES

At the end of the course you will .....
DISTRIBUTED COMPUTING

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

Internet/Cloud computing increasing relevance of the fundamental concepts
**PRACTICAL RELEVANCE**

- For distributed applications, likely to use the code you implemented than existing abstractions.
- Can send objects over NIO socket channels.
- Existing Java RPC does not work on Android devices, but the one you implement will.
- Use Sync, which apparently is the basis of some new Mobile platforms.
- Will implement many abstractions not part of standard Java.
SOFTWARE ENGINEERING PRINCIPLES

**Classes**
Existing classes will be used, inherited but not modified directly

**Interfaces**
Alternative implementations will create new classes implementing existing interfaces

**Factories and Abstract factories**
These will allow easy switching between different implementations

**Generics**
Implementation rather than use of generics to unite buffer and object communication

Will be both a distributed computing and software engineering course
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**

- 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
EXTRA SLIDES
## Alternative Java Layers

<table>
<thead>
<tr>
<th>Blocking stream object communication (Object Stream)</th>
<th>Remote procedure call (RMI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking byte communication (Sockets)</td>
<td>Remote procedure call</td>
</tr>
<tr>
<td></td>
<td>Non blocking object</td>
</tr>
<tr>
<td></td>
<td>communication</td>
</tr>
<tr>
<td></td>
<td>Non blocking byte</td>
</tr>
<tr>
<td></td>
<td>communication (NIO)</td>
</tr>
<tr>
<td>OS Byte Communication</td>
<td></td>
</tr>
</tbody>
</table>

Could have more efficient RPC and non blocking object communication

Two RPC’s?
# Improved Alternative Java Layers

<table>
<thead>
<tr>
<th>OS Byte Communication</th>
<th>Blocking byte communication (Sockets)</th>
<th>Blocking stream object communication (Object Stream)</th>
<th>Non blocking object communication</th>
<th>Non blocking byte communication (NIO)</th>
<th>Remote procedure call</th>
</tr>
</thead>
</table>

Go beyond Java abstractions?  
Could do late binding between RPC and lower-level communication  
Cannot unite NIO and socket at byte or object level  
Socket communication is low level  
NIO is even lower level  
Programmers rely on usage patterns
**Pattern vs. Abstraction (1-Computer Programming)**

**Pattern**

```java
public final static int RED = 0;
public final static int BLUE = 1;
public final static int GREEN = 2;
int color = RED;
```

```java
public final static int LIKE = 0;
public final static int DISLIKE = 1;
public final static int NEUTRAL = 2;
int response = NEUTRAL;
```

**Abstraction**

```java
public enum Color {RED, BLUE, GREEN};
Color color = Color.RED;
```

```java
public enum Response {LIKE, DISLIKE, NEUTRAL};
Response response = Response.NEUTRAL;
```
public void run() {
    while (true) {
        try {
            // Process any pending changes
            synchronized(this.changeRequests) {
                Iterator changes = this.changeRequests.iterator;
                while (changes.hasNext()) {
                    ChangeRequest change = (ChangeRequest) changes.next();
                    switch(change.type) {
                        case ChangeRequest.CHANGEOPS:
                            SelectionKey key = change.socket.keyFor(this.selector);
                            key.interestOps(change.ops);
                            break;
                    }
                }
                this.changeRequests.clear();
            }
        } catch (Exception e) {
        } finally {
            .....}
}

Vast majority of tutorial readers will copy and edit this pattern

Much better to identify a corresponding abstraction and implement it to understand channels

NioServer.java
EchoWorker.java
ServerDataEvent.java
ChangeRequest.java
NioClient.java
RspHandler.java
### Problem with Java Abstraction Level

<table>
<thead>
<tr>
<th>Blocking stream object communication (Object Stream)</th>
<th>Remote procedure call</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Non blocking object communication</td>
</tr>
<tr>
<td>Blocking byte communication (Sockets)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Non blocking byte communication (NIO)</td>
</tr>
<tr>
<td>OS Byte Communication</td>
<td></td>
</tr>
</tbody>
</table>

- Cannot unite NIO and socket at byte or object level
- Socket communication is low level
- NIO is even lower level
- Programmers rely on usage patterns

New picture?
## Improved Alternative Java Abstractions and Layers

<table>
<thead>
<tr>
<th>Remote procedure call</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Object Communication</td>
</tr>
<tr>
<td>New Byte Communication</td>
</tr>
<tr>
<td>Blocking byte communication (Sockets)</td>
</tr>
<tr>
<td>OS Byte Communication</td>
</tr>
</tbody>
</table>

- Alternative high level layer to socket and NIO based byte communication
- Can be bound to either lower level layer
- Design and implementation challenge
- Picture complete? More abstractions and layers?
- Top down vs. bottom up view point
**Improved Alternative Java Abstractions and Layers**

<table>
<thead>
<tr>
<th>Remote procedure communication</th>
<th>New Object Communication</th>
<th>New Byte Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non blocking byte communication (NIO)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS Byte Communication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Link setup and communication:** How to create a group of physically/logically connected processes and communicate informing along these links?

**Scalability:** How to allow group size to increase without degrading performance?

**Distributed fault tolerance:** How to recover when one end of the link goes down but the other does not?

**Process synchronization:** How to block a (thread in a) process until the information it needs to proceed is received from a (thread in a) remote process

**No direct support for group setup and communication, scalability, fault tolerance, and any process synchronization**
New Abstractions: Design Challenge

Fault Tolerance

Pair-wise Synchronization

Group Synchronization

Group Communication and RPC

Scalability

Pair-wise Byte, and Object Communication, Pair-wise RPC

Layering?
Distribution Issues

- Link setup and communication: How to create a group of physically/logically connected processes and communicate informing along these links?
- Distributed fault tolerance: How to recover when one end of the link goes down but the other does not?
- Process synchronization: How to block a (thread in a) process until the information it needs to proceed is received from a (thread in a) remote process?
- Thread synchronization: How to block a thread until a condition for proceeding is enabled by a local thread?
- Scalability: How to allow group size to increase without degrading performance?