COMP 533 - DISTRIBUTED SYSTEMS: COURSE INFO

Instructor: Prasun Dewan (FB 150, dewan@unc.edu)
**Distributed Program?**

<table>
<thead>
<tr>
<th>A program “involving” multiple computers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specific computers must be bound at run time</td>
</tr>
<tr>
<td>→ Program can run on a single computer</td>
</tr>
<tr>
<td>Definition involves processes</td>
</tr>
</tbody>
</table>
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
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.
**Distribution vs. Concurrency**

- Process
- Connection
- Process

- Thread
- Process
- Thread

- Thread
- Process
- Thread

**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 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
EXAMPLES OF PRACTICAL DISTRIBUTED APPLICATIONS

Distributed applications?

Non distributed applications?

In today’s world, what is or should not be distributed?
EXAMPLES

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

Will look at domain-independent, “foundational” concepts at the intersection of them
Non-Distributed vs. Distributed Program

Non-Distributed

- Creates a single process 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

Foundational concepts for non-distributed CS areas still relevant

How should these areas be adapted?
CLASSIFYING FOUNDATIONAL COURSES

CS-1 (Basic Programming Concepts)

CS-2 (Program Design)

Data Structures: Implementation of Big Data Structures, small algorithms

Algorithms: Proof of small data Structures, Big Algorithms

Machine organization and assembly language

(Implementation of) Data Structures and Algorithms in OS, Compilers, Networks

Abstraction design

Abstraction use

Abstraction Implementation

Implementation of Algorithms and Associated Data Structures

Proof of Algorithms and Associated Data Structures
### Abstraction and Software Layering

#### Programming Languages
- High-Level Language
- Assembly Language
- Machine Language

#### Abstraction implementation involves mapping to lower-level abstraction(s)

#### OS
- Interrupt Management
- Thread Communication
- Thread Synchronization
- Thread Management

#### Networking
- UDP
- TCP/IP
- IP
- Link-Level Communication
- Physical Communication
Adding the Distributed Adjective

We will do an implementation-based coverage of selected concepts in design, use, and implementation of domain-independent distributed abstractions and algorithms (no textbook).

- Levels of abstraction and implementation?
  - Abstraction design
  - Abstraction use
  - Abstraction Implementation
  - Implementation of Algorithms and Associated Data Structures
  - Proof of Algorithms and Associated Data Structures

Distributed operating systems covered in graduate courses and text books

Covered in Graduate course and several text books cover
# Networking vs. Distribution

<table>
<thead>
<tr>
<th>Programming Languages</th>
<th>Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-Level Language</td>
<td>Communication in networking involves machines and uses hardware machine addresses</td>
</tr>
<tr>
<td>Assembly Language</td>
<td>Communication in distributed systems is between processes and indicates routing of information among (OS) processes</td>
</tr>
<tr>
<td>Machine Language</td>
<td>TCP/IP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS</th>
<th>Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Management</td>
<td>UDP</td>
</tr>
<tr>
<td>Thread Communication</td>
<td>IP</td>
</tr>
<tr>
<td>Thread Synchronization</td>
<td>Link-Level Communication</td>
</tr>
<tr>
<td>Thread Management</td>
<td>Physical Communication</td>
</tr>
</tbody>
</table>
Base Layers for Distributed Systems

Layering in Distributed Systems?

Distributed Systems

<table>
<thead>
<tr>
<th>OS</th>
<th>Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Management</td>
<td>UDP</td>
</tr>
<tr>
<td>Thread Communication</td>
<td>IP</td>
</tr>
<tr>
<td>Thread Synchronization</td>
<td></td>
</tr>
<tr>
<td>Thread Management</td>
<td></td>
</tr>
</tbody>
</table>
**Some Important Distributed Abstractions**

- **Blocking stream object communication (Object Stream)**
- **Blocking byte communication (Sockets)**
- **Non blocking byte communication (NIO)**
- **Remote procedure call (RMI)**
## Distributed System Layers

<table>
<thead>
<tr>
<th>Network Byte Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non blocking byte communication (NIO)</td>
</tr>
<tr>
<td>Blocking byte communication (Sockets)</td>
</tr>
<tr>
<td>Blocking stream object communication (Object Stream)</td>
</tr>
<tr>
<td>Remote procedure call (RMI)</td>
</tr>
</tbody>
</table>
## Adding the Distributed Adjective: Scope of Course

<table>
<thead>
<tr>
<th>Design and use of (Java) NIO</th>
<th>Abstraction design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design and use of (Java) RPCs (RMI and GIPC)</td>
<td>Abstraction use</td>
</tr>
<tr>
<td>Implementation of (Java) RPC in GIPC</td>
<td>Abstraction Implementation</td>
</tr>
<tr>
<td>Design Space of Interprocess Communication</td>
<td>Implementation of Algorithms and Associated Data Structures</td>
</tr>
<tr>
<td>Implementation of Algorithms for Fault-Tolerant Distributed Consensus in Replicated Objects</td>
<td>Proof of Algorithms and Associated Data Structures</td>
</tr>
</tbody>
</table>
**Non-Distributed vs. Distributed**

- **Programming:** Abstraction use
  - Use of a specific set of non-distributed abstractions (e.g., functional, MATLAB programming)
  - Use of a set of distributed abstractions (NIO, RMI, Replicated Objects with Consensus)

- **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 (Serialization and RPC design and implementation)

- **Theory:** Models and algorithms
  - Non-distributed model and algorithms (Turing Machines, HeapSort,)
  - Distributed Models and Algorithms (Fault-Tolerant Consensus)
Goal is to teach concepts that allow implementation of a set of course assignments
Make Beau Anderson’s 401 Halloween implementation distributed in multiple ways
Base Layers for Distributed Systems (Review)

Layering in Distributed Systems?

Distributed Systems

<table>
<thead>
<tr>
<th>OS</th>
<th>Networking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Management</td>
<td>UDP</td>
</tr>
<tr>
<td>Thread Communication</td>
<td>TCP/IP</td>
</tr>
<tr>
<td>Thread Synchronization</td>
<td>IP</td>
</tr>
<tr>
<td>Thread Management</td>
<td>Link-Level Communication</td>
</tr>
<tr>
<td></td>
<td>Physical Communication</td>
</tr>
</tbody>
</table>
IMPLEMENTATION/ASSIGNMENT-BASED COURSE

Goal is to teach concepts that allow implementation of a set of course assignments.
Adding the Distributed Adjective

We will do an implementation-based coverage of selected concepts in design, use, and implementation of domain-independent distributed abstractions and algorithms (no textbook).

Levels of abstraction and implementation?

Distributed operating systems covered in graduate courses and text books

Abstraction design

Abstraction use

Abstraction Implementation

Implementation of Algorithms and Associated Data Structures

Covered in Graduate course and several text books cover

Proof of Algorithms and Associated Data Structures
ASSIGNMENTS NATURE: HALLOWEEN SIMULATION

Make Beau Anderson’s 401 Halloween implementation distributed in multiple ways
 ASSIGNMENT 1: USE NON BLOCKING I/O

<table>
<thead>
<tr>
<th>Distributed Non-Blocking Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing non distributed simulation</td>
</tr>
</tbody>
</table>
ASSIGNMENT 2: USE SYNCHRONOUS RMI

<table>
<thead>
<tr>
<th>Distributed RMI-based Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing non distributed simulation</td>
</tr>
<tr>
<td>Java Object Serialization</td>
</tr>
<tr>
<td>Sockets</td>
</tr>
</tbody>
</table>
**Assignment 3-Part 1: Use Asynchronous RMI**

<table>
<thead>
<tr>
<th>Distributed GIPC RPC-based Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing non distributed simulation</td>
</tr>
<tr>
<td>Java Object Serialization</td>
</tr>
<tr>
<td>Java NIO</td>
</tr>
</tbody>
</table>
ASSIGNMENT 3-PART 2: CENTRALIZED, NON-FAULT TOLERANT CONSENSUS ALGORITHM

Adaptive communication with consensus on current communication mechanism

- Distributed Non-Blocking Simulation
- Distributed RMI-based Simulation
- Distributed GIPC-based Simulation
ASSIGNMENT 3-PART 3: DISTRIBUTED CLOUD-BASED EXPERIMENTS

Adaptive communication with consensus on current communication mechanism

- Distributed Non-Blocking Simulation
- Distributed RMI-based Simulation
- Distributed GIPC-based Simulation

Atmosphere Cloud-Based Infrastructure (Cyber Infrastructure)
ASSIGNMENT 3-PART 1: USE ASYNCHRONOUS RMI

- Distributed GIPC RPC-based Simulation
  - Existing non distributed simulation
  - GIPC RPC
- Java Object Serialization
- Java NIO
**Assignment 4: Custom RPC Component**

- Distributed GIPC RPC-based Simulation
- Existing non-distributed simulation
- Sync RPC
- GIPC RPC
- Java Object Serialization
- Java NIO
Assignment 3-Part 2: Centralized, Non-Fault Tolerant Consensus Algorithm

Adaptive communication with consensus on current communication mechanism

Distributed Non-Blocking Simulation
Distributed RMI-based Simulation
Distributed GIPC-based Simulation
Assignment 5-Part 1: Understand and Trace Consensus Algorithms under Different Configurations/Algorithms

- Example Replication
- Replicated Objects with Configurable Fault-Tolerant Consensus
- GIPC RPC
- Java Object Serialization
- Java NIO

Change configuration, introduce faults and understand algorithms behavior
## Assignment 5-Part 2: Use Configurable Consensus Mechanism for Sharing Commands

<table>
<thead>
<tr>
<th>Distributed Consensus-based Simulation</th>
<th>Replicated Objects with Configurable Fault-Tolerant Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing non distributed simulation</td>
<td>GIPC RPC</td>
</tr>
<tr>
<td></td>
<td>Java Object Serialization</td>
</tr>
<tr>
<td></td>
<td>Java NIO</td>
</tr>
</tbody>
</table>
## Assignment 5-Part 3: Reach Consensus-Configuration Consensus

<table>
<thead>
<tr>
<th>Distributed Consensus-based Simulation with Consensus on Consensus Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing non distributed simulation</td>
</tr>
<tr>
<td>Replicated Objects with Configurable Fault-Tolerant Consensus</td>
</tr>
<tr>
<td>GIPC RPC</td>
</tr>
<tr>
<td>Java Object Serialization</td>
</tr>
<tr>
<td>Java NIO</td>
</tr>
</tbody>
</table>
## Assignment 6: Custom Serializer (Extra Credit)

<table>
<thead>
<tr>
<th>Distributed Consensus-based Simulation with Consensus on Consensus Mechanism</th>
<th>Replicated Objects with Configurable Fault-Tolerant Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing non distributed simulation</td>
<td>GIPC RPC</td>
</tr>
</tbody>
</table>

- Custom Serializer
- Java NIO
Adding the Distributed Adjective

We will do an implementation-based coverage of selected concepts in design, use, and implementation of domain-independent distributed abstractions and algorithms (no textbook).

- Levels of abstraction and implementation?
- Distributed operating systems covered in graduate courses and text books
- Covered in Graduate course and several text books cover
- Abstraction design
- Abstraction use
- Abstraction Implementation
- Implementation of Algorithms and Associated Data Structures
- Proof of Algorithms and Associated Data Structures
Earlier Version: Fall 13, 15

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 distributed systems.
First assignment was a problem, assignment 3 needs to be cleaned up, auto grading needed, relatively little and simple code needs to be written for regular assignments, compared to Beau’s project at least (distributed debugging is the challenge)
LEARNING RESOURCES

No textbook!

Alternatives?
PDF of Slides

ISSUES IN MESSAGE PASSING

Process/Thread

message³

Message Queue

message²

message¹

Port (Mailbox)

Reliable message delivery?
In-order message delivery?
Port access (and message routing)?
Operations?
Synchronous vs asynchronous?
Blocking vs non blocking?
Buffering of messages at queue?
Location of communicating threads?

RUNELIABLE VS. UN-RELIABLE

Process/Thread
POWERPOINT OF SLIDES

ISSUES IN MESSAGE PASSING

- Process/Thread
  - Message Queue
  - message 1
  - message 2
  - message 3
- Port (Mailbox)
- Reliable message delivery?
- In-order message delivery?
- Port access (and message routing)?
- Operations?
- Synchronous vs asynchronous?
- Blocking vs non-blocking?
- Buffering of messages at queue?
- Location of communicating threads?
Can escape out into unsynchronized or no audio mode (WPS Office on Android will play synchronized audio)
PowerPoint Slides With Unsynchronized Recordings and Media Control

Issues in Message Passing

Process/Thread

<table>
<thead>
<tr>
<th>Reliable message delivery?</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-order message delivery?</td>
</tr>
<tr>
<td>Port access (and message routing)?</td>
</tr>
<tr>
<td>Operations?</td>
</tr>
<tr>
<td>Synchronous vs asynchronous?</td>
</tr>
<tr>
<td>Blocking vs non blocking?</td>
</tr>
<tr>
<td>Buffering of messages at queue?</td>
</tr>
<tr>
<td>Location of communicating threads?</td>
</tr>
</tbody>
</table>

Message Queue

message¹

message²

message³
Recorded YouTube Videos

Issues in Message Passing

Play 2X, rewind, pause, fast-forward to match understanding pace

Youtube video generated from PPT Recordings, does not allow slide-based browsing

PPT modes allow slide-based browsing but requires downloading PPT
Office Mix

Issues in Message Passing

Process/Thread

Message Queue

Port (Mailbox)

Process/Thread

- Reliable message delivery?
- In-order message delivery?
- Port access (and message routing)?
- Operations?
- Synchronous vs asynchronous?
- Blocking vs non blocking?
- Buffering of messages at queue?
- Location of communicating threads?

Create your own!
SLIDE-BASED BROWSING
Long pauses, you may know the answer

Cannot hear student answer

Audio is not the fastest way to get information, specially when studying for an exam

Recordings of live lectures with q/a rather than 15 minute lessons

Can fast forward

You can get a clue from my answer
**WRITTEN DOCUMENTS**

- **Interprocess Communication**
  - Shared Memory
  - Software Interrupt
  - Message Passing
    - Reliability of Messages
    - Order
    - Access
    - Remote Assignment vs Procedure Call
    - Synchronous vs Asynchronous
    - Buffering of Messages
    - Pipes
    - Selectivity of Receipt
    - Integration with Programming Language
    - Integration with I/O
  - Xinu Low-Level Message Passing
    - Semantics
    - Implementation

- Communication across a Network

Incomplete

Concise
DETAILED ASSIGNMENTS

Consult this section only if you are stuck trying to implement some aspect of the assignment. You will, of course, learn more by figuring out the implementation structure on your own.

You essentially have to extend the “coupled simulation” program using the distributed pattern illustrated in the “meaning of life” distributed program.

The trace of the coupled simulation program illustrates how it couples the two simulation models.

The following steps occur before any interaction occurs:

1. \texttt{I**{main} (AddedPropertyChangeListener) EvtSrc(AHalloweenCommandProcessor)}
2. \texttt{I**{main} (AddedPropertyChangeListener) EvtSrc(AHalloweenCommandProcessor)}
3. \texttt{I**{main} (AddedPropertyChangeListener) EvtSrc(AHalloweenCommandProcessor) coupledsims.ASimulationCoupler@3445341}
4. \texttt{I**{main} (AddedPropertyChangeListener) EvtSrc(AHalloweenCommandProcessor) coupledsims.ASimulationCoupler@14cc1f5c}

The first two steps are \texttt{ObjectEditor} registering view observers (whose \texttt{toString()} methods return the empty strings) with each of the two coupled models described by the class,
**What do we do in class?**

<table>
<thead>
<tr>
<th></th>
<th>Without existing resources</th>
<th>With existing resources</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
<td>Live Lectures only on new material</td>
<td>Can listen to recordings in class using headphones</td>
</tr>
</tbody>
</table>
WHAT DO WE DO IN CLASS?

Homework

Deep thinking done solo?

Limited discussion with classmates?

Can do debugging
WHAT DO WE DO IN CLASS?

**Quizzes:** Test that you understood support material

- No time pressure, open book, collaborative, submittable later
- One hour testing, puts pressure
- May not have discipline to access material
What do we do in class?

- Participate in lectures/discussions
- Do quizzes (collaboratively)
- Troubleshoot or implement assignments
- Listen to recorded lectures

Choice is good

Choice can paralyze or be excuse to procrastinate
WHAT DO WE DO IN CLASS? (REVIEW)

- Participate in lectures/discussions
- Do quizzes (collaboratively)
- Troubleshoot or implement assignments
- Listen to recorded lectures

Choice is good

Choice can paralyze or be excuse to procrastinate
**Progress Diary**

- Participate in lectures/discussions
  - Explain some concept that was presented or discussed
- Do quizzes (collaboratively)
  - Explain some concept on which you took a quiz
- Troubleshoot or implement assignments
  - Explain some progress you made on an assignment or some issue that prevented progress
- Listen to recorded lectures
  - Explain some concept that was presented or discussed

Describe at least one specific piece of work you did in each class.
Q/A DIARY

Give all (correct/incorrect/partially correct) answers to questions asked together with the questions
DIARY POSTING MECHANISM

Piazza post visible to instructors (and possibly students also)
Date: 1/11

Activity: Learned (from live lecture/recorded lecture/quiz) that a selector can wait for a specific set of NIO operations to complete.

QA:
Question: Why allow multiple pending operations?  
My answer: To increase concurrency without requiring thread overhead.
# Why Class Participation and Diaries

<table>
<thead>
<tr>
<th>Instructor needs feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn to speak in public (I will ask questions every one can answer such as what was interesting in the material you studied)</td>
</tr>
<tr>
<td>Oral “exam” over whole semester</td>
</tr>
<tr>
<td>Retain material better, logical vs physical attendance</td>
</tr>
<tr>
<td>Recruiters regularly ask about oral and verbal skills</td>
</tr>
<tr>
<td>NSF Survey asked what was being done to improve these skills</td>
</tr>
</tbody>
</table>
### More Oral Skills

<table>
<thead>
<tr>
<th>&lt; 5 minute Videos of Demos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safeguard against lack of and errors in auto-grading</td>
</tr>
</tbody>
</table>
# Grade Distribution

<table>
<thead>
<tr>
<th>Category</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Midterm</td>
<td>25%</td>
</tr>
<tr>
<td>Final</td>
<td>15%</td>
</tr>
<tr>
<td>Regular Collaborative Open-Book Quizzes</td>
<td>10%</td>
</tr>
<tr>
<td>Answers to In-Class Q/A (based on diary)</td>
<td>5%</td>
</tr>
<tr>
<td>Class activity report (based on diary)</td>
<td>5%</td>
</tr>
<tr>
<td>5 Individual Assignments</td>
<td>40%</td>
</tr>
<tr>
<td>Extra Credit Serialization Individual Assignment</td>
<td>12%</td>
</tr>
<tr>
<td>Extra Credit Serialization Open-Book Collaborative Quizzes</td>
<td>3%</td>
</tr>
</tbody>
</table>

Finish assignments (> 90%) and median on other parts assures some form of B or better.
Comparing the Two Approaches

Traditional: Live Lectures + Exams

Mixed Mode: Live Lectures + Exams + Quizzes + Assignments + Diaries.
Live Lectures + Exam

- Much lower instructor effort
- Illusion of or actual understanding
- 1/2/3 Exams show gaps in retention and understanding
ACTIVE LEARNING

Continuous demonstration of gaps in retention and understanding

Continuous fixes to these problems

At the end of course may feel less comfortable with the experience and unsure about your understanding
FRUSTRATION VS CREATIVITY

How frustration can make us more creative

Challenges and problems can derail your creative process ... or they can make you more creative than ever. In the surprising story behind the best-selling solo piano album of all time, Tim Harford may just convince you of the advantages of having to work with a little mess.

This talk was presented at an official TED conference, and was featured by our editors on the home page.
HOW FRUSTRATION CAN MAKE US MORE CREATIVE

A pianist working with a defective piano created the best-selling solo album, after initially refusing to use it.

Studying from a harder to read font resulted in better results.

Diverse groups created better solutions but were more frustrated and less comfortable with their results.
# Lecturing vs. Learning Experience

<table>
<thead>
<tr>
<th>Activity</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Midterm</td>
<td>25%</td>
</tr>
<tr>
<td>Final</td>
<td>15%</td>
</tr>
<tr>
<td>Regular Collaborative Open-Book Quizzes</td>
<td>10%</td>
</tr>
<tr>
<td>Answers to In-Class Q/A (based on diary)</td>
<td>5%</td>
</tr>
<tr>
<td>Class activity report (based on diary)</td>
<td>5%</td>
</tr>
<tr>
<td>5 Individual Assignments</td>
<td>40%</td>
</tr>
<tr>
<td>Extra Credit Serialization Individual Assignment</td>
<td>12%</td>
</tr>
<tr>
<td>Extra Credit Serialization Open-Book Collaborative Quizzes</td>
<td>3%</td>
</tr>
</tbody>
</table>
ASSIGNMENT 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)