

# COMP 4704 – Spring 2009

## HW#1

### Instructor:

Prof. Matt Rutherford – [mjr@cs.du.edu](mailto:mjr@cs.du.edu)

### Due by:

5pm on April 9, 2009

Please choose **ONE** of following two options. The assignments should be emailed to me at [mjr@cs.du.edu](mailto:mjr@cs.du.edu) before the due date.

## Option #1: Chat-Room Program

The overall goal of this assignment is to implement a very basic client/server application that can function as a chat room.

### Server Implementation

You will program a basic chat room server and a simple client in your language / environment of choice. The server will listen on a TCP socket and accept all incoming connections. Once a connection is established, the client program will send a single newline delimited record that contains the client's unique identifier. The server program must check that the identifier is indeed unique and terminate the connection if it is not.

After this, the server program will read newline delimited records from the clients and write them to all other currently connected clients with the originating client's identifier prepended.

The following requirements are placed on the message delivery:

1. Each client should receive the messages in the order they were received by the server program; and
2. Clients should not receive messages that they originated.

Implement your server program so that it can use two different mechanisms for managing the connections and message propagation (e.g., a thread for each client or the use of a `select`, etc.). You may create two separate server programs or use some sort of parameter/argument to determine which management mechanism to use.

### Client Implementation

The client should be a simple program that is capable of initiating a TCP connection to the server program (host and port), writing its unique identifier to the server, and then writing a series of  $N$  newline-delimited messages to the server with random time separation uniformly distributed in the range of 1 to 5 seconds. The messages can be sequentially numbered so that they are unique, something like "message 10 from client1" where 10 is the message sequence number and *client1* is the client's unique identifier.

The client program will mostly likely take the following arguments:

- host and port of the server to communicate with

- the client's unique identifier (you could also just generate this from the combination of the client-side host/port)
- the total number of messages to send to the server.

Keep in mind that your client also has to read messages from the server. The simplest thing to do here might be to use a separate thread for reading the messages from the server and writing them to standard output.

## Testing

Implement a simple scenario with a single server and 5 clients each of which transmits 10 messages to the server. Verify correct operation by capturing the messages received by each client and verifying that each receives the correct messages (40 messages received, in the right order).

## Deliverable

Please create an archive containing the source code and any build / test scripts used to accomplish this. I would like to be able to execute the test scenario and ensure that it works. You will be graded on the following:

- A client that is capable of:
  1. Connecting to the specified TCP host and port
  2. Sending its unique identifier
  3. Sending the specified sequential messages to the server with a random time delay between them
- A server that is capable of:
  1. Listening on the specified port
  2. Accepting connections from clients and reading the unique identifier
  3. Verifying that the identifier is unique for each client
  4. Receiving and propagating messages between clients in two different ways
- A script or program that demonstrates the system is basically working correctly

## 1 Option #2: DNS Analysis

The goal of this homework is to examine the two original DNS RFCs (RFC 1034 <sup>1</sup> and RFC 1035 <sup>2</sup>) in the context of the 8 pitfalls of developing distributed systems (textbook, pg. 16). For each of the pitfalls, describe **ALL** design decision made in order to avoid problems associated with the pitfalls. The RFCs may not explicitly handle all of the pitfalls; if this is the case, simply note this.

You will turn in a document with 8 sections, one for each of the pitfalls. For each pitfall, describe all design decisions (high-level or low-level) that were made in order to protect against the pitfall.

---

<sup>1</sup><ftp://ftp.rfc-editor.org/in-notes/rfc1034.txt>

<sup>2</sup><ftp://ftp.rfc-editor.org/in-notes/rfc1035.txt>