ACS7301
Assignment 1

Instructor: Alex Brodsky

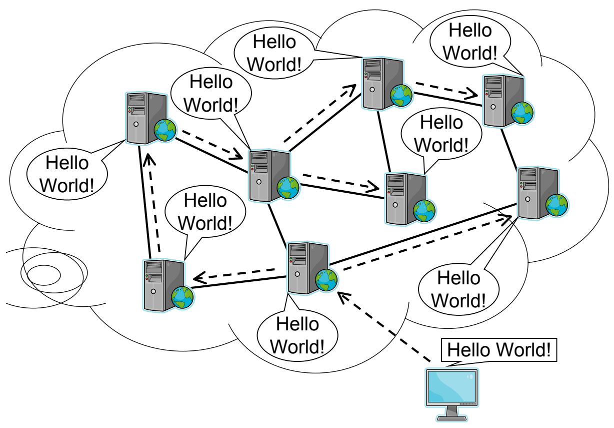
Due: 5:00pm, Monday, February 2, 2009

The purpose of this assignment is to familiarize you with writing network applications, and expose you to the complexities of implementing a simple peer-to-peer system. In this assignment you will be asked to implement a very simple peer to peer system that implements simple broadcast functionality.

Ensure that your Personal Assignment Submission System (PASS) account works!

1 Overview

A broadcast peer-to-peer system comprises a number of peers and distributes messages to all the peers in the system—the peers simply output the messages. Specifically, when a new message is injected into the system, the message must be distributed among all the peers and displayed exactly once by each peer—all peers are responsible for forwarding the broadcasts to all peers that may have not yet received it. Naturally, a peer-to-peer system is intended to be robust. Thus, the system should not crash if some peers fail, and should endeavour to deliver all messages that are injected into the system. **Your task is to design and implement such a peer-to-peer broadcast system.**



Your system will consist of two programs. A *Peer* program that participates in the peer-to-peer system and an *Inject* program that is used to inject new messages into the system. The *Inject* is a relatively simple program that contacts a peer in the system (running the *Peer* program) and sends it a message that the user has entered. The *Peer* program then initiates the distribution of the message to all other peers in the system. When a *Peer* receives a message they have not seen before, they display the message, and forward it to other peers in the system.

2 *Inject* Specification

The *Inject* program should perform the following function:

1. Ask the user to enter the IP and port number of a peer in the system.
2. Ask the user to enter the message to be injected (one line of text).
3. Connect to the specified peer.
4. Send the message to the specified peer so that it can begin distribution.
5. Disconnect from the peer.
6. Exit.

This program should take about 20 lines of code to write, and is very simple.

3 *Peer* Specification

As mentioned already, your peer-to-peer system will consist of a number of peers that you can run on various computers in the lab. To participate in the peer-to-peer system each peer will need to connect to other peers, to pass on messages, and to accept connections from other peers in order to receive messages from them as well as from *Inject*. Thus, each *Peer* will need to listen on a port.

When *Peer* is executed it should:

1. Ask the user to enter:
   (a) the port number on which it will listen for for connections from other *Peer*s and *Inject*s.
   (b) the IP and port number of an existing peer. (This line should be left blank if no other peers are presently up.)
2. Display its IP address and port number
3. Create a socket to listen on the specified port number.
4. Display the IP address and port number of the peer it will contact to join the system.
5. Join the peer-to-peer system. (How the peer joins is up to you.)
6. At this point *Peer* should enter an infinite loop in which it
   (a) Accepts connections from other *Peer*s and *Inject*s.
   (b) Receives and processes the messages
   (c) Forwards the messages if necessary by connecting to other peers.
   (d) Deals with peers that wish to join the system.
   (e) Deals with peer failures (if they occur).

   Note: This is a functional specification. The onus is on you to design and implement mechanisms that will accomplish these tasks.

4 Getting Started

Here are some suggestions for getting started on this assignment.

1. Implement the *Inject* program and a basic version of the *Peer* program that accepts messages from the *Inject* program and displays them.
2. Extend your *Peer* program so that it can join the peer-to-peer system by contacting an existing peer. Implement message forwarding among peers. Get your system to work with two peers and a single *Inject*. Note, you should be able to inject the message into either peer, and that peer should forward the message to the other peer.
3. Extend your *Peer* program (if necessary) to join and work in systems comprising more than two peers. E.g., a message injected into any of the peers should be forwarded to all the peers. This is where you need to start considering design issues. One very simple design is the following:

   > Each peer keeps a table of all other peers in the system. When a new peer joins, it receives the complete table of peers from the peer that it contacted. It then broadcasts a special message to all peers, notifying them of its presence so that they can update their tables.
   >
   > When a message is injected into the system, the peer that initially receives the message, time stamps it with the current time and its IP address and port number. The message is then forwarded to all the peers in its peer table. When a peer receives a message from another peer, it first checks if it has already receive ed this message (based on the time stamp, IP address, and port number). If so, the message is ignored. Otherwise, the message is displayed and forwarded (with the original time-stamp, IP address, and port number).

   Note, that this is a simple, but inefficient design. You can start with this one to get things working, but should also think about how to improve it.

4. Lastly, you should improve your *Peer* program to deal with peer failures. That is, if a peer fails, the remaining peers should not, and continue to receive, forward, and display broadcasts. How you do this will depend on your design. If you are using the simple design suggested above, think about what needs to be done to ensure that dead peers do not remain in the tables of the peers.

5 Running and Testing your System

You should be able to run and test your system on the computers in the graduate lab, as well as the other labs such as 3C13. As well, there is also a large unix server (with Java) on which you can run your programs. (Contact the instructor to get an account on this server.

You should be able to run multiple instances of *Peer* on the same computer, by simply assigning each one a different port number. Thus, at least for most of your initial testing your should be able to test your system on your own desktop.

6 Deliverables

The deliverables for this assignment are:

1. The *Inject* and *Peer* programs. These can be implemented in a language of your choice.
   - An electronic copy of the code, which will be tested.
   - A printout of the code, for annotation.

   The code should be well commented, indented, and structured. **Your code MUST compile. If it does not, you will receive an automatic 0**.
2. A two page high-level description (with diagrams) of your design, describing
   - The general structure of your *Peer* program.
   - The structure of the peer-to-peer system.
   - The joining mechanism.
   - The message forwarding mechanism.
   - The fault handling mechanism.
   - How you tested and validated your system.

You must submit both an electronic and a paper copy of your assignment. You can hand in the paper copy in class, to the department secretary, or to me, on or before 5pm of the due date. Both the paper and electronic submissions must be done by 5pm of the due date.

6.1 Electronic Submission

The electronic copy of the assignment should comprise all your source code files needed to compile and test your programs. Use the PASS system at `https://venus.acs.uwinnipeg.ca/` to submit your assignment. **If you do not have an account on the PASS system, please see the instructor!** Perform a test submission in advance of the deadline to ensure that you are comfortable with the process.

6.2 Paper Submission

The paper copy of the assignment should comprise print-outs of all the sources files that you submitted electronically and the written description of your system.