Assignment 3

Start:	29 October 2010
End:	15 November 2010

Introduction & Objectives

In this exercise, you will develop an application that will enable users to take part in an n-person chat using their mobile phones while preserving the causality and temporal ordering of messages in light of the unreliability inherent to the UDP protocol. To do this, you will actually implement two concepts for determining the order of events in a distributed computer system that you have encountered in the lecture (and exercise). This is only a short summary – consult the slides[1] for details and additional hints:

- **Lamport Timestamps** represent a simple algorithm to partially order distributed events. The simple rules that this algorithm follows were determined by Leslie Lamport and defined in the article "Time, Clocks, and the Ordering of Events in a Distributed System" (1984). Distributed processes that implement Lamport Timestamps satisfy the so-called *clock consistency condition* ("If one event comes before another, then that event's logical clock comes before the other's") and thus implies that "If event a's logical clock comes before event b's, then a may have happened *before or at the same time* as b, but did not happen after b".
- **Vector Clocks** represent an extension of Lamport Timestamps in that they guarantee the *strong clock consistency condition* which dictates that, additionally to the clock consistency condition, "If one event's clock comes before another's, then that event comes comes before the other" (i.e., it is a two-way condition). This is achieved by having each process hold a vector/array of n logical clocks (where n is the number of processes) and include these values in all interprocess message.

To support this assignment, we have two servers running at `http://vswot.inf.ethz.ch`:

- Port **`3999`:** Provides a "capitalization" service; may be used to test UDP-based communication. This server replies on port 3999.
- Port **`4000`:** Provides the actual "chatting" service by taking care of distributing all messages received from registered clients to all other registered clients. The server randomly delays the messages to add additional unreliability to the communication such that the effects of the causality-preserving algorithms may be better demonstrated. While this server listens for incoming datagrams on port 4000, it distributes received chat messages to the registered clients at port 4001.

1 Getting familiar with Datagrams (not assessed)

To familiarize yourself with the sending and receiving of UDP messages, create an Android application that provides a capitalizing service to its user while relying on the server at `http://vswot.inf.ethz.ch:3999`. You do not need to submit this program.

[1] `http://vs.inf.ethz.ch/edu/vs/android/`

What to do

- Create a new application and setup the UI to enable the user to enter and submit a text message and also display the server's answer. Do also read the other tasks of this assignment – maybe you can reuse parts of your application later.
- Create UDP Sockets (`DatagramSocket(int port)`) for communication with the server.

2 Starting the Conversation

In this task, you will create a chat application that leverages the server at `http://vswot.inf.ethz.ch:4000`. The application will use Lamport Timestamps to delay the delivery of messages to the user if appropriate. The aim is to create a program that enables the user to choose a user name, register with the server and start chatting with other clients using the guidelines defined in the accompanying slides that are provided at

`http://vs.inf.ethz.ch/edu/vs/android/`

What to do

- Create a new application and configure the UI to provide a server registration/deregistration button, to display incoming messages and to let the user send chat messages. For displaying chat messages, you can, for instance, use the Android `ArrayAdapter` that provides an implementation of an `Adapter` and uses an array of arbitrary objects to manage a `ListView`.
- Provide the user with the ability to enter a username when registering with the server. The server will return the assigned username, the assigned vector index and the current time vector. *Ignore* the information on the assigned index, this will be needed for task 3.
- You will have to create a multi-threaded application as one thread will listen for incoming messages (at port 4001) in the background: Familiarize yourself with Java threads. Implement the background listener thread. For cross-thread information exchange, you will most probably want to use the `Handler` class.
- Test your listener thread (if you use the emulator for testing, remember to set port redirects). Depending on how fast you have progressed with the assignment, there will already be lots of chit-chat going on. At the least, `QuestionBot` and `AnswerBot` will be engaged in their eternal scrutiny and cantankerousness, respectively.
- Add functionality to allow the user to send messages to the server (via port 4000). The registration and deregistration buttons should also make use of this function. The server will reply to commands on the same port, so also add functionality to receive these replies (also make use of the `setSoTimeout(int timeout)` function). Remember to keep your `onCreate()` method as clean as possible and to use threads for delegating potentially long-running tasks.
- Enhance your listener thread: Use Lamport Timestamps (index 0 in time vector) to delay the displaying of incoming messages if their timestamp indicates that they should not yet be delivered to the user. To do this, create a method `isDeliverable(...)` that explicitly inspects the timestamp of every incoming message and decides whether or not to delay its delivery. As soon as this function returns `true` for a message, the message shall be delivered to the user.
- Design your application such that it can handle multiple incoming messages simultaneously.

3 Vanquishing the Desequencer

3.1 Vector Clocks

For this task, create a new application that implements the same basic functionality as the one developed during task 2 (you may, of course, copy the code...) but uses vector clocks for determining the order of messages in the distributed system. Your application will of course also be able to handle the dynamic joining and leaving of clients.

What to do

- Create a new application and setup the UI and functionality as before.
- Instead of using Lamport Timestamps, use Vector Clocks to determine the order of messages and display them to the user accordingly. Thus, you now will have to parse and utilize the index number that the server assigns to your application when registering.
- Create a method `isDeliverable(...)` that explicitly inspects the (vectorial) timestamps of incoming messages and decides whether or not to delay the delivery of a message. As soon as this function returns `true` for a message, the message shall be delivered to the user.
- Design your application such that it can handle multiple incoming messages simultaneously.

3.2 Additional Questions

To fulfill this task, include a thorough discussion of some issues and considerations related to Vector Clocks in your report. Specifically, answer these questions:

- When *exactly* are two Vector Clocks causally dependent? How does this relate to:
  - We decided in the exercise that we would not let our applications trigger a tick when receiving a message. What would be the implications of ticking on receive?
  - Does a clock tick happen before or after the sending of a message. What are the implications of changing this?
- Read and assess the referenced paper (below). Briefly describe the problems that are mentioned regarding dynamic vector clock systems. What is the difference between the approach described in the paper and ours?

`http://vs.inf.ethz.ch/edu/vs/exercises/DVC_Landes.pdf`

Deliverables

The following 2 deliverables have to be submitted by 09:00am on November 15:

1. **report.[txt,doc,pdf]** As part of this assignment, you should produce a report (max. 2 pages) that includes information on the design and implementation of tasks 2 and 3 and motivate any choices you have made during the process. This report should also contain your thoughts on task *3.2*.
2. **code.zip** You should create a zip file containing the two Eclipse projects (`VS_Assignment_3_2`, and `VS_Assignment_3_3`) created in this assignment. The projects should have been thoroughly tested on the HTC Desire and the Android emulator. Please use UTF-8 encoding for your documents and avoid special characters like umlauts if you can.

Marks

The distribution of marks is as follows (partial solutions will be marked individually):

- Tasks 1 and 2 and report: 4.0
- Tasks 1, 2, and 3.1 and report: 5.0
- Tasks 1, 2, 3.1 and 3.2 and report: up to 6.0

Submission

Report and code must be uploaded through:

`https://www.vs.inf.ethz.ch/edu/vs/submissions/`

The submission script will not allow you to submit any part of this exercise after the deadline. However, you can re-submit as many times as you like until 09:00am on November 15.