CSE 486/586 Distributed Systems Project 0

Simple Messenger on Android

Introduction

In this assignment, you will design a simple messenger app on Android. The goal of this app is simple: enabling two Android devices to send messages to each other. The purpose of this assignment is to help you understand some of the basic mechanisms necessary to write networked apps on Android. It is intended to be simple, just to get you familiarized with Android.

Still, please start this assignment as soon as possible. There will be some learning curve in terms of socket programming as well as Android programming. While the teaching staff do not think that these are difficult subjects to learn by yourself, if you have not done these before, we cannot tell you how much time it will take for you to acquire the basics. So again, please start this assignment as soon as possible.

Step 1: Installation (Eclipse & Android SDK)

The first thing you need to do is to install Eclipse and the Android SDK on your machine. Follow the procedure described at http://developer.android.com/sdk and you should be able to install it.

Step 2: Reading the Android Dev Guide

The Android Developer's Guide at http://developer.android.com/guide/index.html is a great source of information for starters as well as experts.

Please read at least the articles listed below. However, please keep in mind that, depending on your background, not everything might make sense immediately. This is probably natural for people who didn't take a networking course or an OS course before (especially, we didn't cover processes and threads in class, so the article about them might be difficult to read). If this happens to you, don't panic and move on; later, post a question on Piazza or ask the teaching staff during our office hours.

- Android Basics
  - What is Android? http://developer.android.com/guide/basics/what-is-android.html
  - Application Fundamentals: http://developer.android.com/guide/topics/fundamentals.html
- Framework Topics
  - Activities: http://developer.android.com/guide/topics/fundamentals/activities.html
  - Services: http://developer.android.com/guide/topics/fundamentals/services.html
  - Intents: http://developer.android.com/guide/topics/intents/intents-filters.html
  - Processes and Threads: http://developer.android.com/guide/topics/fundamentals/processes-and-threads.html

A critical part of any development is testing and debugging. Especially for this assignment, you need to

test your app with two emulator instances. The following two articles describe this well:

- Using the Android Emulator: http://developer.android.com/guide/developing/devices/emulator.html
- Debugging: http://developer.android.com/guide/developing/debugging/index.html

Step 3: Following Tutorials

The Android Developer's Guide also has great tutorials you can follow. These tutorials will teach you the basics. Please do the following two tutorials at least:

- Hello World: http://developer.android.com/resources/tutorials/hello-world.html
- Hello Views: http://developer.android.com/resources/tutorials/views/index.html

Step 4: Writing the Messenger App

The final step is writing the messenger app. The following are the requirements:

- A user of one device should be able to write a message to the other device.
- The other device should be able to display what was received.
- And vice versa.
- You need to use the Java Socket API.
- All communication should be over TCP.
- You can assume that the size of a message will never exceed 128 bytes (characters).
- You do not need to show the history of messages (though this would be real nice if you choose to implement it).

There are a few things to note:

- You are free to implement it in any way; for example, you could implement it entirely in an Activity with a few threads, or you could implement it with an Activity and a few Services communicating through Intents. Other designs are also possible.
- The UI design is completely up to you. As long as both devices allow typing new messages and displaying received messages, you are free to design any UI. For example, you could display a new message with a notification (or a "toast", i.e., android.widget.Toast), or you could display a new message within an Activity.
- The Java Socket API is different from the UNIX socket API. You will need to use java.net.Socket for the client side code and java.net.ServerSocket for the server side code. java.net.Socket will give you a TCP socket.
- Your application will probably act both as a server and as a client at the same time. What this means that your app will probably do both of the following: 1) opening a server socket and listening on it, and 2) opening a client socket and connecting to the other device. This depends on how you choose to implement the app, but this is very likely. However, if your implementation has one device acting as a server and the other device acting as a client, it is also perfectly fine.
- You need to define a proper protocol. Again, a protocol is an agreement between two parties in communication on what the communication should be like. For example, in a messenger app, the remote side should be able to tell when a message starts and ends. You can define a protocol in several ways to address this. One way could simply be "the sender opens a new connection for each message, which means that every piece of data sent is a user message; the sender terminates each connection after completely sending one message." Or if you wanted to make it a bit more realistic, it could be "the first 4 bytes of data sent is the size of the message, and the rest is the actual message ended with "\n"." There are obviously other ways to define your protocol.
- It's probably easier if you write your app in steps. For example, you can first write code for getting the text input and displaying it locally without writing any network-related code. After that, you can

write code for just one-way message sending/receiving. Once that's done, you can write the rest for two-way messaging.

Interconnecting Two Emulator Instances

Although the Android Dev Guide describes how to interconnect two instances (http://developer.android.com/guide/developing/devices/emulator.html#connecting), it only describes it for a one-way client-server scenario. If your implementation only has one device acting as a server and the other device acting as a client, the link is probably good for you. However, if your app acts both as a server and as a client on a single device, you need to set up both ways, which is not covered in the above link. Please follow the rest if you want to set up both ways.

Again, what we mean by "server" below is really a piece of code (either a complete program or a block of code inside of a program) that opens a server socket and listens on it. Likewise, what we mean by "client" below is really a piece of code that opens a client socket and connects to some other server sockets.

To allow one emulator instance to communicate with another, you must set up the necessary network redirections as illustrated below.

Assume that your environment is

- A is your development machine
- B is your first emulator instance, running on A
- C is your second emulator instance, running on A too
- And you want to run a server on B, to which C will connect.
- You also want to run a server on C, to which B will connect.

Here is how you could set it up:

1. Set up the server on B, listening to 10.0.2.15:<serverPort1>
2. On B's console, set up a redirection from A:localhost:<localPort1> to B:10.0.2.15:<serverPort1>
3. On C, have the client connect to 10.0.2.2:<localPort1>

For example, if you wanted to run an HTTP server, you can select <serverPort1> as 80 and <localPort1> as 8080:

- B listens on 10.0.2.15:80
- On B's console, issue redir add tcp:8080:80
- C connects to 10.0.2.2:8080

In addition,

1. Set up the server on C, listening to 10.0.2.15:<serverPort1> (this doesn't have to be different from B's <serverPort1>)
2. On C's console, set up a redirection from A:localhost:<localPort2> to C:10.0.2.15:<serverPort1> (notice localPort2, which should be different from localPort1 that B uses.)
3. On B, have the client connect to 10.0.2.2:<localPort2>

For example, if you wanted to run an HTTP server on C, you can select <serverPort1> as 80 and <localPort2> as 8081:

- C listens on 10.0.2.15:80
- On C's console, issue redir add tcp:8081:80
- B connects to 10.0.2.2:8081

Useful Hints

- Add uses-permission as Internet and Access_network_state in Manifest file to avoid permission errors.
- Create a separate AVD for each emulator
- Use Log class for debugging and logging (Check the logs in Eclipse/Windows/Other/Android/LogCat)
- To check log for different emulators, add a DDMS perspective in Eclipse (You can add by clicking perspective icon present at top right corner of Eclipse window)
- For showing history, you can use StringBuffer to append text or if you are using TextView, TextView.append().

Project Group

This assignment should be done individually. This includes both the design document and code. There should be no project group for this assignment.

Design Document

You need to write a design document of up to 2 pages (12 pt font in .pdf). This should include:

1. What components you designed and what they do
2. The protocol between two devices
3. Difficulties you faced (what things took time to figure out, …)
4. (Rough) # of hours you put
5. How to test your app
    a. What port numbers to use
    b. How to set up redirections in order to test your app properly
    c. Any other special instructions for testing

Please include figures if necessary.

Submission

We use the CSE submit script. You need to use either “submit_cse486” or “submit_cse586”, depending on your registration status. If you haven’t used it, the instructions on how to use it is here: https://wiki.cse.buffalo.edu/services/content/submit-script

You need to submit the following three files:

- Your app’s .apk
- Your design document in .pdf
- Your entire Eclipse project source code tree zipped up in .zip. To do this, please go to your Eclipse workspace directory, find your app, and zip it.

Deadline: 2/6/12 (Monday) 2:59pm

This is right before class @ 3pm. The deadline is firm; if your timestamp is 3pm, it is a late submission.

Grading

This assignment is 5% of your final grade. The break-down for this assignment is:

- 3%: your messenger app
- 2%: your design document

Late Submission Policy

- Late submissions will result in a 20% penalty per day, e.g., (your regular score * 80 / 100) for the first day, (your regular score * 60 / 100) for the second day, etc. A day is defined as 24 hours after the day/time the assignment is due (excluding weekends or school holidays).
- No help will be available from the TAs or from the instructor for a project after its scheduled due date.
- After five (5) days, the assignment will no longer be accepted.

Academic Integrity

- Under any circumstance, you should not copy others' code.
- You need to get permission first when you copy from other sources, e.g., the Web, books, etc. If you get the permission, then you also need to clearly comment in your code which part you copied. This is for your protection; otherwise, the course staff might not know whether you have gotten the permission or not.
- The exception is Android Developers. It contains many useful coding examples, and you are free to copy any code from there. However, **you still need to clearly comment in your code that you copied from the developer website.** Again, this is for your protection.