CSE 486/586: Distributed Systems

Programming Assignment 1[1]
Simple Messenger

Introduction

In this assignment, you will write a simple messenger app for Android. The purpose of this assignment is to help you determine whether you have the right background for this course. *If you can finish this assignment* ***all by yourself, without help from others,*** *then it is probably the case that you are prepared for this course.*

The application you develop in this assignment will simply allow two Android devices to send messages to each other. This will help you understand the basics of the Android environment, refresh your memory on network programming, help ease you into the Android development tools, and introduce you to our testing environment.

There are four high-level challenges in this assignment:

- *Reading and understanding background material:* You will need to read API documentation, tutorials, and articles online in becoming familiar with Android development.
- *Infrastructure setup:* You will have to install a number of software packages, configure them, and become familiar with their usage. You will also need to get plumbed into GitHub Classroom and UB Autograder.
- *Becoming comfortable with Android development:* You will encounter roadblocks in developing for Android, particularly if this is your first time. This is true for any platform or operating environment. Fortunately there is excellent documentation available for Android Studio and the Android APIs.
- *Working around Android virtual device restrictions:* The AVD emulator has, in particular, restrictions on network communication for virtual devices. You will need to get used to this for this assignment and for this course.

1 Getting Started

Unless you are already familiar with Android development, I suggest that you start with some Android tutorials. In addition, you should consult the Android programming assignment web page for this course and ensure that you have installed a compatible development environment (*e.g.*, Android Studio 3.0.1, Android SDK 19). **Please note** that there are some known problems running Android Studio in some environments, and there are workarounds on that page!

[1]This assignment is based, with permission, on an assignment developed and used by Steve Ko in his version of CSE 486/586. Large portions of text are used verbatim from that assignment.

The instructions for creating AVD images and configuring the emulator in various tutorials will vary. Note that recent versions of Android Studio *require* a 64-bit processor with Intel VT-x or AMD-V virtualization extensions, OpenGL graphics support, and x86 emulator images. This greatly speeds up emulation and debugging, but makes running Android Studio in a hypervisor or on older hardware difficult.

The first tutorials to look at are:

- The Building Your First App tutorial is advisable for all students, even those with previous Android experience, as a help in validating your Android development environment. This tutorial will guide you through installing the necessary software and building a simple application. *Note that we are using Android API version 19, and you should configure your build appropriately!*
- Next, look at Managing the Activity Lifecycle, which will help you understand some basic Android concepts that are used in the provided code and that will be required to successfully complete this assignment.

You will probably want to review some important Android documentation, as well, such as:

- Application Fundamentals
- Activities
- Processes and Threads

To be successful *in this and all future programming assignments*, it is critical that you understand the Android virtual device (AVD) emulator both for executing and debugging your application. You will be using the emulator through Android Studio, standalone, and via the grading scripts for this course. Please take the time to become familiar with it now! You should review:

- Run Apps on the Android Emulator
- Debug Your App
- Logcat Command-line Tool

2 Setting up a Testing Environment

You will need to run two AVDs in order to test your app. Unfortunately, Android does not provide a flexible networking environment for AVDs, so there are some hurdles to jump over in order to set up the right environment. The following are the instructions.

These instructions will assume that your Android SDK directory (configured when you set up Android Studio) is located in `$ANDROID_SDK_ROOT`, and that this variable is set appropriately in your environment. This is required for things to work properly!

- You need to have the Android SDK and Python 2.x (not 3.x; Python 3.x versions are not compatible with the scripts provided.) installed on your machine. If you have not installed these, please do it first and proceed to the next step.
- Set `ANDROID_HOME` to `$ANDROID_SDK_ROOT` in your environment.
- Add the following directories to your path:
  - `$ANDROID_SDK_ROOT/tools/bin`
  - `$ANDROID_SDK_ROOT/platform-tools`
  - `$ANDROID_SDK_ROOT/emulator`

  A good reference on how to change `$PATH` is here.
- Download and save these files somewhere convenient:
  - `create_avd.py`
  - `run_avd.py`
  - `set_redir.py`
- At this point, **make sure that your environment and path are set up correctly!**
- Create five AVDs by running `python create_avd.py 5 $ANDROID_HOME`.
  - You should run this command only once, and it may take quite some time to complete.
  - You may be asked to agree to a license agreement `[y/N]`; you will have to agree to continue.
  - You will be asked `Do you wish to create a custom hardware profile [no]` multiple times. *Do not enter anything when asked*, the script will handle this automatically.
- At this point you should have five AVDs, named `avd0` through `avd4`. You can view and manipulate them in Android Studio, but please do not change or delete them as they are necessary for our scripts to work.
- Start a single emulator with the command `emulator @avd0`. Wait for it to boot entirely, then click and hold the power button (⏻) until the power off menu appears on the emulated device. Click power off, then confirm, and wait for the device to shut down entirely (the emulator command will exit).
- For all of the programming assignments except this one, *you will need to run all five AVDs at the same time.* You will need to find a machine that can comfortably run five emulators simultaneously.
- In order to test this, run `python run_avd.py 5`. The first time you do this, it may take a very long time and some emulators may fail to start. If so, wait until they have all made as much progress as possible (either booted or visibly frozen, typically), shut down the emulators, and try again.

- For this assignment, you will use only two AVDs, so you will start them with `python run_avd.py 2`.
- After you successfully launch all five AVDs, run the command `python set_redir 10000` to create an emulated network connecting the five AVDs. There are some restrictions to how this works, discussed later in this document.
- After verifying your development environment, you can shut down the emulators.

3 The Simple Messenger App

The graded portion of this project is writing the simple messenger app. You will need to download the project from GitHub Classroom (you should have received an invitation to our classroom; if you have not, please contact me ASAP!) and open it in Android Studio.

1. Accept the invitation to GitHub Classroom to create your project repository.
2. Clone the GitHub your repository for SimpleMessenger. The repository URL will be `git@github.com/ub-cse586-s18/SimpleMessenger-$githubid`, where `$githubid` is your GitHub username. You can do this on the command line or in Android Studio.
3. Open the cloned project in Android Studio.
4. Take some time to understand the template code.
    - The main Activity is in `SimpleMessengerActivity.java`.
    - Please read both the code and the comments carefully!
5. Add your code to the cloned project, committing to GitHub as you make progress. There are places in the template marked "TODO"; these are the places where you will need to add your code.

3.1 Requirements

These are the project requirements. *You must follow these instructions exactly. If you do not, you will receive* ***no credit*** *for this assignment.* **When we say no credit, we mean it.** It is critical that you follow directions precisely and accurately in this course.

1. There should be only one app that you develop and need to install for grading, and it should use the manifest provided in the template code. If you use the project template code, you should satisfy this requirement.
2. There should be only one text box on-screen where the user of the device can enter a text message to be sent to the other device. If you use the project template code, you should satisfy this requirement.

3. Each device should display on-screen what it has received. The project template contains basic code for displaying messages on the screen.
4. You need to use the Java Socket API.
5. All communication should be over TCP.
6. You may assume that the size of a message will never exceed 128 bytes (ASCII characters).

3.2 Communication

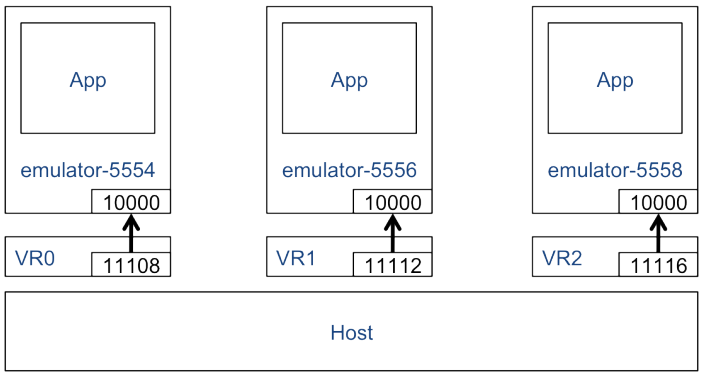
As mentioned above, the Android emulator is not very flexible in the networking facilities that it provides for communication between AVDs. Although `set_redir.py` enables networking among multiple AVDs, it is very different from a typical networking setup. When you write your socket code, you will have the following restrictions:

- In your app, you can open *only one* server socket, which must listen on port 10000 regardless of which AVD your app is running on.
- The app on `avd0` can connect to the listening server socket of the app on `avd1` by connecting to 10.0.2.2:11112 (that is, IP address 10.0.2.2, port number 11112).
- The app on `avd1` can connect to the listening server socket of the app on `avd0` by connecting to 10.0.2.2:11108.
- Your app knows which AVD it is running on via the following code snippet. If `portStr` is 5554, then it is `avd0`. If `portStr` is 5556, then it is `avd1`:

```
TelephonyManager tel =
    (TelephonyManager)this.getSystemService(Context.TELEPHONY_SERVICE);
String portStr =
    tel.getLine1Number().substring(tel.getLine1Number().length() - 4);
```

  The project template already implements the above code, but you are expected to understand the code as this is critical for the rest of the programming assignments.

- In general, `set_redir.py` creates an emulated, port-redirected network like this (VR stands for "virtual router"):



3.3 Testing

We have testing programs to help you see how your code does with our grading criteria. If you find rough edges in the testing programs, please report it so the teaching staff can fix it. The instructions for using the testing programs are the following:

1. Download a testing program for your platform. If your platform does not run it, please report it.
   (a) Windows: Tested on 64-bit Windows 8.
   (b) Linux: Tested on 64-bit Debian 9 and 64-bit Ubuntu 17.10 (see below for important information about 64-bit systems).
   (c) Mac OS: Tested on 64-bit Mac OS 10.9 Mavericks.
2. Before you run the program, please make sure that you are running two AVDs (`avd0` and `avd1`). You can use `python run_avd.py 2` to start them.
3. Make sure that you have installed your SimpleMessenger application on both AVDs!
4. Run the testing program from the command line.
5. It may issue some warnings or errors during execution. Some of them are normal, some may indicate errors in your program. Examine them to find out!
6. At completion, it will give you one of three outputs:
   - No communication verified: The SimpleMessenger instances cannot communicate with each other. This would be worth 0 points.
   - One-way communication verified: The SimpleMessenger on `avd0` can send a message to the SimpleMessenger on `avd1`. This is worth 2 points.
   - Two-way communication verified: Both AVDs can communicate with each other. This is worth an additional 3 points, for a total of 5.

Remember that this is only a simple test against the grading criteria. It is NOT a test for development progress! You will want to create your own tests as you go along.

Notes for 64-bit Linux: The testing program is compiled 32-bit. If you get an error like the following, install the 32-bit `libz` for your system:

```
./simplemessenger-grading.linux: error while loading shared libraries:
    libz.so.1: cannot open shared object file: No such file or directory
```

On Debian-based distributions, you can accomplish this with the command `apt-get install libz1g:i386` as root (you may need to use `sudo` or `su`). If `apt-get` reports an error about the architecture or says the package is not found, you may need to enable multiarch. To do this, run `dpkg --add-architecture i386` as root, then update your APT repositories with `apt-get update` as root. Once this is done, you should be able to install the 32-bit `libz`.

For other distributions you will need to consult your distribution documentation.

3.4 Submission

We use UB CSE autograder for submission. You can find autograder at https://autograder.cse.buffalo.edu/, and log in with your UBITName and password.

Once again, *it is critical that you follow everything below exactly.* Failure to do so **will lead to no credit for this assignment**.

Zip up your entire Android Studio project source tree in a single zip file named `SimpleMessenger.zip`. Ensure that *all* of the following are true:

1. You *did not* create your zip file from *inside* the `SimpleMessenger` directory.
2. The top-level directory in your zip file is *SimpleMessenger*, and it contains `build.gradle` and all of your sources.
3. You used a zip utility and *not any other compression or archive tool:* this means no 7-Zip, no RAR, no tar, *etc.*

3.5 Deadline

This project is due 2018-02-05 11:59:00 AM. This is one hour before our class. This is a firm deadline. If the timestamp on your submission is 11:59:01, it is a late submission. You are expected to attend class on this day!

3.6 Grading

This assignment is 5% of your final grade. Credit for this assignment will be apportioned as follows:

- 2%: Your messenger app can send one-way messages from `avd0` to `avd1`.
- 3%: Your messenger app can send two-way messages between `avd0` and `avd1`.

Thus, an application that can send two-way messages will receive a total of 5%: 2% for sending from `avd0` to `avd1`, and 3% for also being able to send messages back.

4 Other Resources

In addition to Android Developers and the course Android documentation (which will be updated with FAQs and fixes, so keep an eye on it!), I have prepared a few videos to help with getting the development environment set up and preparing the handout code using GitHub Classroom. You can find them on YouTube.