CSE 486/586: Distributed Systems

Programming Assignment 2, Part A[1]
Group Messenger 1

Introduction

This assignment builds on the previous assignment and sets the stage for the next assignment. You will design a group messenger that can send messages to multiple AVDs and store received messages in a persistent key-value storage. Like the first assignment, *you must follow these instructions exactly.* Failure to do so may result in no credit for this assignment.

Please read this document in its entirety before you begin. It is long, but that is only because it is complicated and precise. Revisit the instructions regularly as you implement to be sure you're implementing what is required.

This assignment will require you to:

- Implement an Android `ContentProvider`.
- Use file I/O to create a persistent key-value store.
- Manage multiple sockets and multiple client connections.

1 Getting Started

Successful completion of this assignment will require both the development environment you set up in the first project and the things you learned in implementing the first project. If your development environment for the first project was not 100% operational, please go back and address that now!

As in the first project, we are providing a project template for you to get started. This template configures a number of things (such as declaring the appropriate capabilities in the Android Manifest, laying out UI components for automated testing, *etc.*) on your behalf. It is important that you do not change settings that you do not have to change to complete your project.

2 The Group Messenger App

The graded portion of this project is implementing the group messenger app. You will need to download the project from GitHub Classroom and open it in Android Studio. You should have received an email containing the GitHub Classroom invitation for this project. If you have not, please contact the teaching staff on Piazza.

[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.

Instructions for cloning the project and opening it in Android Studio are the same as for the first project, except that the base repository name is `GroupMessenger1`. You can find YouTube tutorials detailing the process here.

Take some time to understand the template code after you have read the rest of this document. In particular, note the TODO items in the three Java classes `GroupMessengerActivity`, `GroupMessengerProvider`, and `OnPTestClickListener`. You may wish to reference your SimpleMessenger code, as well as the handout code from that project (which, *e.g.*, does filesystem I/O).

The requirements for this project are in two parts: implementing an Android content provider and providing group messenger communication by implementing application-layer multicast with storage on that provider. They will be detailed here, with their specific requirements and grading criteria.

2.1 Writing a Content Provider

The first part of this assignment is implementing a `ContentProvider` that implements a persistent key-value storage. It takes a string key and a string value, and stores them for later retrieval (even across reboots!). Before you start, please read the following Android documentation on content providers:

`http://developer.android.com/guide/topics/providers/content-providers.html`

A typical Android content provider supports basic SQL statements. However, we do not require that you do so for this course. Your content provider will present a simple key-value table abstraction.

You may not fully understand all of the details in this list until you have gone back and read the relevant Android documentation. That is OK; read through the requirements first so you know what to look for in the docs!

2.1.1 Content Provider Requirements

1. *You should not require any permission to access your provider.* This is very important as it will prevent our testing program from testing your app (and therefore your app from receiving any credit!). The given code takes care of this; as long as you do not change it, you will be fine.

2. Your content provider's URI should be:

   `content://edu.buffalo.cse.cse486586.groupmessenger1.provider`,

   Any app should be able to access your provider using that URI. To simplify your implementation, your provider does not need to match or support any other URI pattern. This URI is already declared in the given code's `AndroidManifest.xml`; again, if you do not change it, you will be fine.

3. Your provider should have two columns.

   - The first column should be named "key" (all lowercase, no quotation marks). This column is used to store keys.

- The second column should be named "value" (all lowercase, no quotation marks). This column is used to store the values associated with keys.
- All keys and values stored by your provider should be Java strings.

4. Your provider should implement only insert() and query(). No other operations are necessary.

5. Given that the column names are "key" and "value", *any app* should be able to insert a key-value pair as in the following example:

```
ContentValues keyValueToInsert = new ContentValues();

// inserting <"key-to-insert", "value-to-insert">
keyValueToInsert.put("key", "key-to-insert");
keyValueToInsert.put("value", "value-to-insert");

// Assume we have already created providerUri with our provider's URI
Uri newUri = getContentResolver().insert(providerUri,
                                         keyValueToInsert);
```

6. If a new value is inserted for a key that already exists, keep *only the most recent value.* You should not preserve the history of values under the same key.

7. Any app should be able to read a key-value pair from your provider using query(). Since your provider is a simple key-value table, we are not going to follow the Android convention; your provider should answer queries as in this example:

```
Cursor resultCursor = getContentResolver().query(
    providerUri,   // again, assume we already created this
    null,          // no need to support the projection parameter
    "key-to-read", // provide the key directly as the selection parameter
    null,          // no need to support the selectionArgs parameter
    null           // no need to support the sortOrder parameter

);
```

8. Your provider should store its key-value pairs using one of the possible data storage options. The details of these options are outlined here:

   https://developer.android.com/guide/topics/data/data-storage.html

   You may chose any option you like. However, the easiest option is probably to use internal storage with the key as a file name and the value as the file contents.

After implementing your provider, you can verify whether or not you are meeting the requirements by clicking the "PTest" button provided in the given code. You can look at `OnPTestClickListener.java` to see exactly what it is doing. *If your provider does not pass PTest, there will be no points for this portion of the assignment.*

2.2 Implementing Application-Layer Multicast

The final step is implementing an application-layer multicast messenger (*i.e.*, sending the same message to multiple AVDs). Unlike the first project, no code is provided to log incoming messages or display them on-screen. However, you may find that doing so is helpful for development and debugging.

For this portion of the project, you will need to run five emulators. Recall from project 1 that you can use the provided `run_avds.py` to start the emulators, and that you must use the provided `set_redir.py` to connect their virtual networks. Please refer to project 1 for more details.

In particular, we will be using the multi-port configuration as described in project 1. Your app will open one server socket that listens on port 10000, but it will connect to a different port number on the IP address 10.0.2.2 for each emulator, as follows:

emulator serial	port
emulator-5554	11108
emulator-5556	11112
emulator-5558	11116
emulator-5560	11120
emulator-5562	11124

You can use the code snippet provided in project 1 to help you with this calculation.

2.2.1 Multicast Messenger Requirements

1. Your app must multicast every user-entered message to all app instances, *including the one that is sending the message*. In the rest of this description, "multicast" always means sending a message to *all app instances*. It must be able to do this when all five emulators are running simultaneously.

2. Your app should be able to send/receive multiple messages.

3. Your app should be able to handle concurrent messages (that is, multiple messages incoming from multiple apps at the same time).

4. Your app should assign a sequence number to every message it *receives*. The sequence numbers should start from 0 and increase by 1 for each received message.

5. Each message should be stored with its sequence number as a key-value pair in your content provider. The key should be the sequence number for the message (as a Java string) and the value the message received.

6. All app instances should store every message and its corresponding sequence number individually.

For debugging purposes, you may wish to print the messages to the screen. However, this will not be graded, and the given code does not already do it (as it did in project 1).

2.3 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:

- Download a testing program for your platform. If your platform does not run it, please report it.
    1. Windows: Tested on 64-bit Windows 8.
    2. Linux: Tested on 64-bit Debian 9 and 64-bit Ubuntu 17.10 (see below for important information about 64-bit systems).
    3. Mac OS: Tested on 64-bit Mac OS 10.9 Mavericks.
- Before you run the program, please make sure that you are running all five AVDs. You can use `python run_avd.py 5` to start them.
- Remember to start the emulator network by running `set_redir.py 10000`.
- Make sure that you have installed your application on both AVDs!
- Run the testing program from the command line.
- 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!
- The testing program will give you partial and final scores.

As before, you may find that the testing program does not always give full credit even when things seem to be working, due to distributed race conditions. If this is the case, test several times.

You may find that debugging and/or the grading script perform in an unexpected way if you run your application multiple times, or if you upgrade it with `adb` or Android Studio without first uninstalling it. This is because the content provider storage is persistent, so it will be retained even if your application is restarted or upgraded. The content provider storage will be cleared when the app is uninstalled, so you can uninstall it between runs to get around this issue. It can be uninstalled with the following command:

```
adb uninstall edu.buffalo.cse.cse486586.groupmessenger1
```

Note that you may need to provide the emulator "serial number" (the string listed by `adb devices`, something like `emulator-5554`) with the `-s` argument if you are running multiple emulators.

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 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. Ensure that *all* of the following are true:

1. You *did not* create your zip file from *inside* the `GroupMessenger1` directory.
2. The top-level directory in your zip file is `GroupMessenger1` or `GroupMessenger1-<something>`, 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.*

4 Deadline

This project is due 2018-02-23 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!

5 Grading

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

- 2%: Your content provider behaves correctly.
- 3%: Your messenger app can send, receive, and correctly store messages among all AVDs.

Thus, an application that has a working content provider and can multicast messages from any or all AVDs to all other AVDs and store them at the receiver will receive a total of 5%: 2% for having a working content provider, and 3% for working multicast messaging with message storage.