CSE 486/586 Distributed Systems

Programming Assignment 2, Part A

Group Messenger with a Local Persistent Key-Value Table

Introduction

The teaching staff hopes you had fun working on PA1! If you got frustrated, we feel for you and believe us, we were there too. While it is expected to be frustrating in the beginning, we promise you, it will get better and you will enjoy more and more as you do it. You might even start enjoying reading the Android documentation because it *is* actually the single best place to get great information about Android. We do hope, though, that you now understand a bit more about what it means to write networked apps on Android.

Now back to the assignment: this assignment builds on the previous simple messenger and points to the next assignment. You will design a group messenger that can send messages to multiple AVDs and store them in a permanent key-value storage. Once again, please follow everything exactly. Otherwise, it might result in getting no point for this assignment.

The rest of the description can be long. Please don't "tl;dr"! Please read to the end first and get the overall picture. Then please revisit as you go!

Step 0: Importing the project template

Unlike the previous assignment, we will have strict requirements for the UI as well as a few other components. In order to provide you more help in meeting these requirements, we have a project template you can import to Android Studio.

1. Download the project template zip file to a temporary directory.
2. Extract the template zip file and copy it to your Android Studio projects directory.
    a. Make sure that you copy the correct directory. After unzipping, the directory name should be "GroupMessenger1", and right underneath, it should contain a number of directories and files such as "app", "build", "gradle", "build.gradle", "gradlew", etc.
3. After copying, delete the downloaded template zip file and unzipped directories and files. This is to make sure that you do not submit the template you just downloaded. (There were many people who did this before.)
4. Open the project template you just copied in Android Studio.
5. Use the project template for implementing all the components for this assignment.

Step 1: Writing a Content Provider

Your first task is to write a content provider. This provider should be used to store all messages, but the abstraction it provides should be a general key-value table. Before you start, please read the following to understand the basics of a content provider:
http://developer.android.com/guide/topics/providers/content-providers.html

Typically, a content provider supports basic SQL statements. However, you do not need to do it for this course. You will use a content provider as a table storage that stores (key, value) pairs.

The following are the requirements for your provider:

1. You should not set any permission to access your provider. This is very important since if you set a permission to access your content provider, then our testing program cannot test your app. The current template takes care of this; so as long as you do not change the template, you will be fine.
2. Your provider's URI should be "content://edu.buffalo.cse.cse486586.groupmessenger1.provider", which means that any app should be able to access your provider using that URI. To simplify your implementation, your provider does not need to match/support any other URI pattern. This is already declared in the project template's AndroidManifest.xml.
3. Your provider should have two columns.
    a. The first column should be named as "key" (an all lowercase string without the quotation marks). This column is used to store all keys.
    b. The second column should be named as "value" (an all lowercase string without the quotation marks). This column is used to store all values.
    c. All keys and values that your provider stores should use the string data type.
4. Your provider should only implement insert() and query(). All other operations are not necessary.
5. Since 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");

Uri newUri = getContentResolver().insert(
    providerUri,    // assume we already created a Uri object with our provider URI
    keyValueToInsert
);
```

6. If there's a new value inserted using an existing key, you need to keep only the most

recent value. You should not preserve the history of values under the same key.

7. Similarly, any app should be able to read a <key, value> pair from your provider with query(). Since your provider is a simple <key, value> table, we are not going to follow the Android convention; your provider should be able to answer queries as in the following example:

```
Cursor resultCursor = getContentResolver().query(
    providerUri,    // assume we already created a Uri object with our provider URI
    null,           // no need to support the projection parameter
    "key-to-read",  // we provide the key directly as the selection parameter
    null,           // no need to support the selectionArgs parameter
    null            // no need to support the sortOrder parameter
);
```

   Thus, your query() implementation should read the *selection* parameter and use it as the key to retrieve from your table. In turn, the Cursor returned by your query() implementation should include only one row with two columns using your provider's column names, i.e., "key" and "value". You probably want to use android.database.MatrixCursor instead of implementing your own Cursor.
8. Your provider should store the <key, value> pairs using one of the data storage options. The details of possible data storage options are in http://developer.android.com/guide/topics/data/data-storage.html. You can choose any option; however, the easiest way to do this is probably use the internal storage with the key as the file name and the value stored in the file.
9. After implementing your provider, you can verify whether or not you are meeting the requirements by clicking "PTest" provided in the template. You can take a look at OnPTestClickListener.java to see what tests it does.
10. If your provider does not pass PTest, there will be no point for this portion of the assignment.

Step 2: Implementing Multicast

The final step is implementing multicast, i.e., sending messages to multiple AVDs. The requirements are the following.

1. Your app should multicast every user-entered message to all app instances (including the one that is sending the message). In the rest of the description, "multicast" always means sending a message to all app instances.
2. Your app should be able to send/receive multiple times.
3. Your app should be able to handle concurrent messages.
4. As with PA1, we have fixed the ports & sockets.
    a. Your app should open one server socket that listens on 10000.
    b. You need to use run_avd.py and set_redir.py to set up the testing environment.
        i. python run_avd.py 5

        ii. `python set_redir.py 10000`
    c. The grading will use 5 AVDs. The redirection ports are 11108, 11112, 11116, 11120, and 11124.
    d. You should just hard-code the above 5 ports and use them to set up connections.
    e. Please use the code snippet provided in PA1 on how to determine your local AVD.
        i. emulator-5554: “5554”
        ii. emulator-5556: “5556”
        iii. emulator-5558: “5558”
        iv. emulator-5560: “5560”
        v. emulator-5562: “5562”
5. Your app needs to assign a sequence number to each message it receives. The sequence number should start from 0 and increase by 1 for each message.
6. Each message and its sequence number should be stored as a <key, value> pair in your content provider. The key should be the sequence number for the message (as a string); the value should be the actual message (again, as a string).
7. All app instances should store every message and its sequence number individually.
8. For your debugging purposes, you can display all the messages on the screen. However, there is no grading component for this.
9. Please read the notes at the end of this document. You might run into certain problems, and the notes might give you some ideas about a couple of potential problems.

Testing

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

1. Download a testing program for your platform. If your platform does not run it, please report it on Piazza.
    a. Windows: We’ve tested it on 64-bit Windows 8.
    b. Linux: We’ve tested it on 64-bit Ubuntu 18.04.
    c. OS X: We’ve tested it on 64-bit OS X 10.15 Catalina.
2. Before you run the program, please make sure that you are running five AVDs. `python run_avd.py 5` will do it.
3. Please also make sure that you have installed your GroupMessenger1 on all the AVDs.
4. Run the testing program from the command line.
5. On your terminal, it will give you your partial and final score, and in some cases, problems that the testing program finds.
6. You might run into a debugging problem if you're reinstalling your app from Android Studio. This is because your content provider will still retain previous values even after reinstalling. This won't be a problem if you uninstall explicitly before reinstalling; uninstalling will delete your content provider storage as well. In order to do this, you can uninstall with this command:

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

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 one file described below. Once again, you must follow everything below exactly. Otherwise, you will get no point on this assignment.

- Your entire Android Studio project source code tree zipped up in .zip: The name should be GroupMessenger1.zip.
    a. Never create your zip file from inside “GroupMessenger1” directory.
    b. Instead, make sure to zip “GroupMessenger1” directory itself. This means that you need to go to the directory that contains “GroupMessenger1” directory and zip it from there.
    c. Please do not use any other compression tool other than zip, i.e., no 7-Zip, no RAR, etc.

Deadline: **2/21/2020 (Friday) 11:59:59 am**

This is one hour before class. The deadline is firm; if your timestamp is 12pm, it is a late submission.

Grading

This assignment is 5% of your final grade. The breakdown for this assignment is:

- 2% if your content provider behaves correctly.
- (additional) 3% if your messenger app can send, receive, and store messages among all AVDs.

Notes

- There is a cap on the number of AsyncTasks that can run at the same time, even when you use THREAD_POOL_EXECUTOR. The limit is "roughly" 5. Thus, if you need to create more than 5 AsyncTasks (roughly, once again), then you will have to use something else like Thread. However, I really do not think that it is necessary to create that many AsyncTasks for the PAs in this course. Thus, if your code doesn't work because you hit the AsyncTask limit, then please think hard why you're creating that many threads in the first place.

  This document gives you more details on the limit and you might (or might not, depending on your background) understand why I say it's "roughly" 5.

  http://developer.android.com/reference/java/util/concurrent/ThreadPoolExecutor.html

(Read "Core and maximum pool sizes.")

- For Windows users: In the past, it was discovered that sometimes you cannot run a grader and Android Studio at the same time. As far as I know, this happens rarely, but there is no guarantee that you will not encounter this issue. Thus, if you think that a grader is not running properly and you don't know why, first try closing Android Studio and run the grader.