CSE 486/586 Distributed Systems Project 1

Totally and Causally Ordered Group Messenger with a Local Persistent Key-Value Table

Update 4

The deadline has been extended to 3/26 (Monday) @ 2:59PM.

Update 3

Please do not include the permission string ("edu.buffalo.cse.cse486_586.provider.permission.ALL_PERMISSION") in your AndroidManifest.xml file. It'll cause some issues. This means that every app should be able to access your content provider without any permission.

Update 2

1. I've added some more description for Test Case 2 in Step 3. Nothing has changed, but I wanted to clarify it.
2. For each user-provided message, your app should multicast it to all others. It is not correct if your app sends each message to a leader and the leader distributes the message.

Update 1

I've changed some dashes with underscores in order to avoid Android errors. Please check to make sure that you're using the correct ones. Affected strings are:

1. ~~Permission string: "edu.buffalo.cse.cse486_586.provider.permission.ALL_PERMISSION"~~
2. Content provider URI: "content://edu.buffalo.cse.cse486_586.provider"
3. Column names: "provider_key" and "provider_value"

Introduction

In this assignment, you will design a group messenger that preserves total ordering as well as causal ordering of all messages. In addition, you will implement a key-value table that each device uses to individually store all messages on its local storage. Please note that this is not a distributed hash table (yet ;-). This assignment builds on the previous simple messenger assignment and points to the next assignment.

A “group” messenger simply means that your app should support group messaging, not just between two devices, but among ***multiple*** devices; when a user enters a message, your messenger app should multicast the message to all other devices. In addition, your app should provide total ordering of all messages, i.e., all devices should display all messages in the same order; the ordering should also preserve causal “happened-before” relations of the messages. This means that your app should provide ***both total and causal ordering at the same time***. Finally, all messages should be stored individually by every device. For this purpose, your app should implement a ***persistent key-value table*** that stores all messages on each and every device’s local storage.

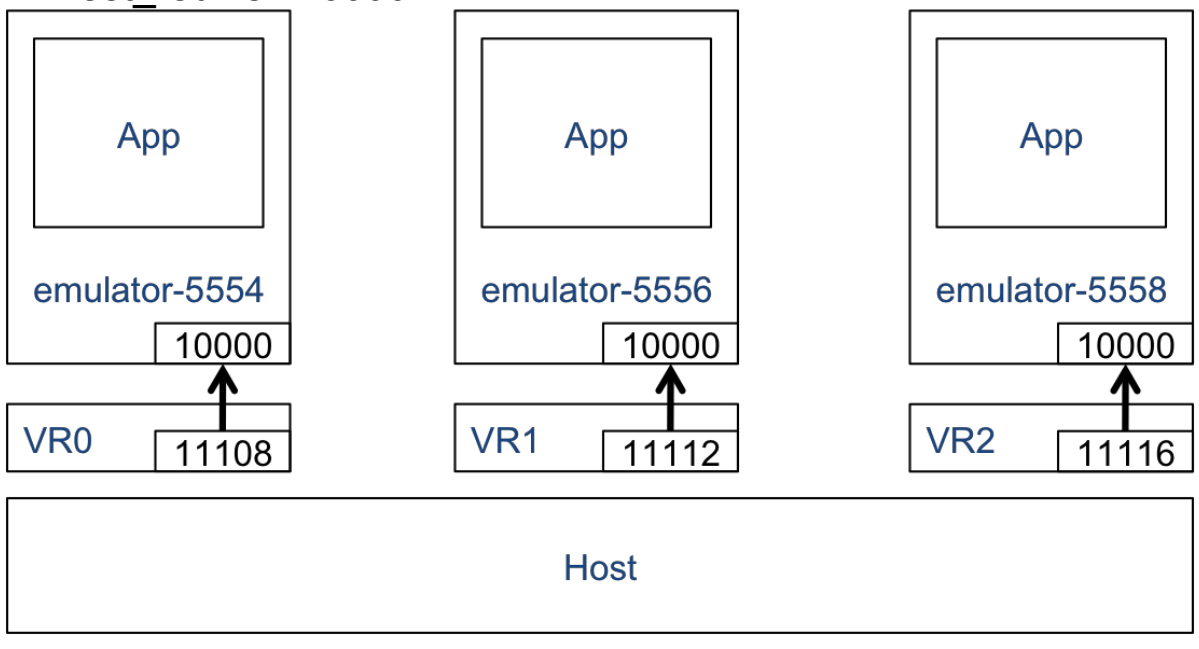
Step 0: Amazon EC2 Setup

For this assignment, you need to be able to use a virtual machine instance on Amazon EC2 created with the “CSE 486/586 Android Development Image” AMI. Since you will need to run multiple Android emulator instances to test your app, it is necessary to have a powerful enough physical or virtual machine to handle it. Amazon EC2 provides “High-CPU Extra Large” type that can comfortably handle 4-5 emulator instances. Please follow the steps below to start using Amazon EC2.

1. ***Note: before following the steps below, please remember that we are providing the EC2 credit for testing with 4-5 emulator instances, not for general development & simple debugging.*** You can probably do most of the development locally on your machine without using EC2. Likewise, you can probably do a lot of debugging locally with two instances without using EC2. We are emphasizing this because we do not want you to run out of your credit before the semester ends.
2. Create an account at http://aws.amazon.com/
3. Redeem your AWS credit coupon (under “Account” --> “Payment Method” --> “Redeem an AWS Credit Coupon”): you should have received an AWS credit coupon from the teaching staff. This is intended to cover all your cost of using AWS this semester. However, since the credit is limited, ***it is very important to watch your AWS usage.*** If the credit runs out, your credit card will be charged automatically.
4. Launch a High-CPU Extra Large (c1.xlarge) instance with “CSE 486/586 Android Development Image”: using the regular process of launching a new instance, search for “CSE 486/586 Android Development Image” AMI and use it to launch a High-CPU Extra Large instance. ***Please note that once you launch an instance, the URL of your instance changes every time you stop it and start it again;*** use the correct URL to log in to the instance you start.
5. Download an NX client for your platform at http://www.nomachine.com/download.php: there is a client for Windows, Linux, and Mac. All client software is free.
6. Launch the NX client and enter the URL of your instance.
    a. ***The login username*** to enter for the NX client is “cse486-586” (without quotes).
    b. ***The TAs should have told you the password*** in the recitations. Please change the password the first time you log in.
    c. ***The desktop installed is “GNOME”***; there is no other desktop installed.
    d. For all other settings and options on the NX client, ***please do not change the default***; otherwise, you might not be able to log in to your EC2 instance.
7. Log in. If you cannot log in, please post a public message on Piazza and wait for an answer from the teaching staff. It is important to post your question as a public message since others might experience the same problem.
8. Once you log in, use the terminal application to launch Eclipse. There are also two scripts provided to help you launch multiple emulator instances and set up the redirection.
    a. “**run_emu.sh <the number of instances>**” launches multiple instances.
    b. “**set_redir.sh <app server port number>**” sets up the redirection. If you run the script, the output will tell you how the redirection is set up. However, if you are curious, set_redir.sh script sets up the redirection in the following way for each instance.

i. Take the input port number (<app server port number>) which your app will listen on, e.g., 10000.
ii. Take the emulator instance's console port, e.g., 5554.
iii. Multiply it by 2, e.g., 11108.
iv. Set up the redirection with the command "redir add tcp:<console port * 2>:<app server port number>", e.g., "redir add tcp:11108:10000"

c. For example, the diagram below shows the set up if you run:
i. run_emu.sh 3
ii. set_redir.sh 10000



Step 1: Implementing the main Activity

Your main Activity should display messages. This is really just for testing purposes, so the UI design is again completely up to you. It is also OK to reuse your code from Project 0. Here are the requirements for your main Activity:

1. ***Users should be able to enter messages.***
2. ***The messages should be displayed in the order according to your app's ordering guarantee.*** If you implement all the requirements of this assignment, this ordering is supposed to be total-causal. However, if you do not implement all the requirements for some reason, then the ordering might be different.
3. This means that your app should display all the messages exchanged while a user is using the app. However, if the user closes the app and restarts it, your app does not need to display the message exchanges in the previous run.
4. There has to be two buttons that trigger test cases. The details on the test cases are in the final step "Step 3: Implementing Total-Causal Ordering".

Step 2: Writing a Content Provider

Along with the main Activity, your app should have a content provider. This content provider should be used to store all messages, but the abstraction it provides should be a general key-value table. The following are the requirements for your content provider:

1. ***Any app (not just your app) should be able to access (read and write) your content provider*** ~~if the app has the permission of "edu.buffalo.cse.cse486_586.provider.permission.ALL_PERMISSION". For this, you need to add it with the "android:permission" attribute of the <provider> element~~.
2. ***Your content provider's URI should be*** "content://edu.buffalo.cse.cse486_586.provider", which

means that any app should be able to access your content provider using that URI. To simplify your implementation, your content provider does not need to match/support any other URI pattern.

3. ***Your content provider should have two columns.***
    a. The first column should be named as “provider_key” (an all lowercase string). This column is used to store all keys.
    b. The second column should be named as “provider_value” (an all lowercase string). This column is used to store all values.
    c. All keys and values that your provider stores should use the string data type.
4. ***At the minimum, your content provider should implement insert() and query().***
5. Since the column names are “provider_key” and “provider_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(“provider_key”, “key-to-insert”);
keyValueToInsert.put(“provider_value”, “value-to-insert”);

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

6. 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 providers column names, i.e., “provider_key” and “provider_value”. You probably want to use android.database.MatrixCursor instead of implementing your own Cursor.

7. ***Your content 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.

Step 3: Implementing Total-Causal Ordering

The final step is supporting both total and causal ordering. You will need to design an algorithm that does this and implement it. ***You will probably need to spend most of your time on this part.*** The requirements are:

1. ***Your app should multicast all user-entered messages to other devices.***

2. ***Your app should use B-multicast.*** It should not implement R-multicast.
3. ***You need to combine a total ordering algorithm with a causal ordering algorithm.*** We have discussed two total ordering algorithms and one causal ordering algorithm in class. You will need to choose one total ordering algorithm and find a way to combine it with the causal ordering algorithm.
4. ***To avoid confusion, we will fix all the port numbers.***
    a. Your app should open one server socket that listens on 10000.
    b. You need to use run_emu.sh and set_redir.sh to set up the testing environment. Because of this, if you run 5 instances, the redirection ports are 11108, 11112, 11116, 11120, & 11124. (Please refer to the diagram above to see how the redirection is set up.)
    c. You should just hard-code the above 5 ports and use them for multicast in your final submission. Again, this is to avoid confusion. This obviously doesn't mean that you always need to test/debug your app with 5 instances. For example, if you want to test/debug your app with 2 instances, you can just hard-code 11108 and 11112.
5. ***Your (final) app should enable group chatting among 5 emulator instances.*** The list of 5 instances and their port numbers should be fixed as the above requirement details.
6. ***The following code snippet will give you the "local" emulator instance's console port as a string, e.g., "5554", "5556", etc.***

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

Here the "local" instance means the emulator instance that your app is currently running on. It extracts the last 4 digits of the phone number of your local instance, which is the console port number. You can use this to determine your local emulator instance since the console port number uniquely identifies each emulator instance. You can also use this to identify the redirection port to your local instance by multiplying it by 2, since that is how set_redir.sh selects redirection ports.

7. ***Every message should be stored in your content provider.*** Each message should be stored as a <key, value> pair. The key should be the final delivery sequence number for the message (as a string); the value should be the actual message (again, as a string). The delivery sequence number should start from 0 and increase by 1 for each message.
8. ***You need to write two test cases.*** Manually testing the correctness of your app is going to be difficult since you need to verify the ordering with concurrent events. A better way is to write test cases that can automatically generate messages. This is still not perfect, but it can assist you when you debug your app. You can write as many test cases as you want. However, it is required to implement two test cases at the minimum. The next two points detail the test cases.
9. Test case 1
    a. Your main Activity should have "Test1" button that triggers the test case.
    b. When the button is touched, it should create a thread that multicasts 5 messages in sequence. Multicasting of one message should be followed by 3 seconds sleep of the thread. This is just to make sure that we can spread messages from different emulator instances.
    c. The message format should be "<console port number>:<sequence number>". <console port number> is the emulator instance's console port number, e.g., 5554, 5556, etc. <sequence number> is a number starting from 0 and increasing by 1 for each message. For example, if your first emulator instance multicasts 5 messages, then the messages should be "5554:0", "5554:1", "5554:2", "5554:3", and "5554:4".
    d. With this test case, at least the following two ordering guarantees can be verified by just looking at the sequence of messages on each instance.
        i. Total ordering: every emulator instance should display the same order of message
        ii. FIFO ordering: all messages from the same instance should preserve the local order. This should be preserved because causal ordering implies FIFO ordering.
10. Test case 2

a. Your main Activity should have “Test2” button that triggers the test case.
b. When the button is touched, your app should wait for 3 seconds and multicast one message. The message format should be the same as the above test case.
c. Your app running on the n-th (mod 5) emulator should multicast exactly ***one more message*** upon receiving ***the first message*** sent by the (n-1)-th (mod 5) emulator, where the first emulator instance is emulator-5554, the second is emulator-5556, etc. For example, emulator-5554 should multicast exactly one more message only when it receives the first message from emulator-5562. Likewise, emulator-5558 should multicast exactly one more message only when it receives the first message from emulator-5556, etc.
d. The purpose of this test case is to test causal ordering by doing something similar to circulating a message over a ring of emulator instances. For example, if you touch “Test2” button from emulator-5554, it should multicast a message, then emulator-5556 should multicast a message, etc., until everyone multicasts a message at least once. After emulator-5562 multicasts a message, emulator-5554 should multicast once again. However, emulator-5556 should not multicast anymore.
e. What this means is that your code should recognize the message of this test case and reacts accordingly when it receives this message.

11. ***This is not required, but you probably want to use AsyncTask to multicast messages and handle incoming messages.***

Design Document

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

1. What components your group designed and what they do
2. The description of the total-causal ordering algorithm your group designed. Please include an English description as well as pseudo-code.
3. Any special things to note.

Project Group

Your group should share one design document. However, coding should be done individually; please write your own code.

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 group’s design document in .pdf. Please note that everyone should submit the design document individually even though your group writes the document together. This is to make sure that we do not miss anything from you.
- 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: ~~3/9/12~~ ~~3/23/12~~ 3/26/12 (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 15% of your final grade. The breakdown for this assignment is:

- 10%: your app
  - 2% if the content provider behaves correctly
  - 3% if total ordering is preserved
  - 3% if causal ordering is preserved
  - 2% if you write test cases correctly
- 5%: your design document
  - 2% for the overall clarity of the description
  - 3% for the correct total-causal ordering algorithm description

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.