CSE 486/586 Distributed Systems

Programming Assignment 2, Part B

Group Messenger with Total and FIFO Ordering Guarantees

Introduction

We are now ready to implement more advanced concepts and in this assignment you will add ordering guarantees to your group messenger. The guarantees you will implement are total ordering as well as FIFO ordering. As with part A, you will store all the messages in your content provider. The different is that when you store the messages and assign sequence numbers, your mechanism needs to provide total and FIFO ordering guarantees. Once again, please follow everything exactly. Otherwise, it might result in getting no point for this assignment.

Step 0: Importing the project template

Once again, you need to download a template to start this assignment.

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 “GroupMessenger2”, 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

As with the previous assignment, you need to have a working content provider. The requirements are almost exactly the same as the previous assignment. The only exception is the URI, which is “content://edu.buffalo.cse.cse486586.groupmessenger2.provider”.

Step 2: Implementing Total and FIFO Ordering Guarantees

This is the meat of this assignment and you need to implement total and FIFO guarantees. You will need to design an algorithm that does this and implement it. An important thing to keep in mind is that there will be a failure of an app instance in the middle of the execution. The requirements are:

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 use B-multicast. It should not implement R-multicast.
3. You need to come up with an algorithm that provides a total-FIFO ordering under a failure.
4. There will be at most one failure of an app instance in the middle of execution. We will emulate a failure only by force closing an app instance. We will ***not*** emulate a failure by killing an entire emulator instance. When a failure happens, the app instance will *never* come back during a run.
    a. Each message should be used to detect a node failure.
    b. *For this purpose, you can use a timeout for a socket read;* you can pick a reasonable timeout value (e.g., 500 ms), and if a node does not respond within the timeout, you can consider it a failure.
    c. This means that you need to handle socket timeout exceptions in addition to socket creation/connection exceptions.
    d. ***Do not just rely on socket creation or connect status to determine if a node has failed.*** Due to the Android emulator networking setup, it is **not** safe to *just* rely on socket creation or connect status to judge node failures. Please also use socket read timeout exceptions as described above.
    e. You cannot assume which app instance will fail. In fact, the grader will run your group messenger multiple times and each time it will kill a different instance. Thus, you should not rely on chance (e.g., randomly picking a central sequencer) to handle failures. This is just *hoping* to avoid failures. Instead, you should implement a decentralized algorithm (e.g., something based on ISIS).
5. When handling a failure, it is important to make sure that your implementation does not stall. After you detect a failure, you need to clean up any state related to it, and move on.
6. When there is a node failure, the grader will not check how you are ordering the messages sent by the failed node. Please refer to the testing section below for details.
7. As with the previous PAs, 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.
    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"
8. Every message should be stored in your provider individually by all app instances. 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.

9. For your debugging purposes, you can display all the messages on the screen. However, there is no grading component for this.
10. 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 32- and 64-bit Windows 8.
    b. Linux: We've tested it on 32- and 64-bit Ubuntu 12.04.
    c. OS X: We've tested it on 32- and 64-bit OS X 10.9 Mavericks.
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 GroupMessenger2 on all the AVDs.
4. Run the testing program from the command line.
5. There are two phases of testing
    a. Phase 1---Testing without any failure: In this phase, all the messages should be delivered in a total-FIFO order. For each message, all the delivery sequence numbers should be the same across processes.
    b. Phase 2---Testing with a failure: In this phase, all the messages sent by live nodes should be delivered in a total-FIFO order. Due to a failure, the delivery sequence numbers might go out of sync if some nodes deliver messages from the failed node, while others do not. This is OK; the grader will only examine the total-FIFO ordering guarantees for the messages sent by live nodes. (Note: in phase 2, the message sequence numbers can go out of sync due to a failure. Thus, when the grader output says that a key is missing, the key means the message sequence number that the grader is verifying. It may not be the exact key.)
6. Once again, you should implement a decentralized algorithm to handle failures correctly. This means that you should not implement a centralized algorithm. This also means that you should not implement any variation of a centralized algorithm that randomly picks a central node. In our grading, we will run phase 2 as many as possible.
7. If your implementation uses randomness in failure handling or is centralized, the score you get through the grader is not guaranteed.
8. On your terminal, it will give you your partial and final score, and in some cases, problems that the testing program finds.
9. Unlike previous graders, the grader for this assignment requires you to directly give the path of your apk to it. The grader will take care of installing/uninstalling the apk as necessary.

10. The grader uses multiple threads to test your code and each thread will independently print out its own log messages. This means that an error message might appear in the middle of the combined log messages from all threads, rather than at the end.
11. The grader has many options you can use for your testing. It allows you to choose which phase to test and for phase 2, how many times to run. It also has an option to print out verbose output, which can be helpful for debugging. You can enter the following command to see the options:

    $ <grader executable> -h

12. 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.groupmessenger2

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 GroupMessenger2.zip.
    a. Never create your zip file from inside "GroupMessenger2" directory.
    b. Instead, make sure to zip "GroupMessenger2" directory itself. This means that you need to go to the directory that contains "GroupMessenger2" 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: 3/17/2017 (Friday) 11:59:59am

The deadline is firm; if your timestamp is 12pm, it is a late submission.

Grading

This assignment is 10% of your final grade. The breakdown for this assignment is:

- 4% if your group messenger provides total-FIFO ordering guarantees.
- (additional) 6% if your group messenger provides total-FIFO ordering guarantees under a failure.

Notes

- Please do not use a separate timer to handle failures. This will make debugging very difficult. Use socket timeouts and handle all possible exceptions that get thrown when there is a failure. They are:
    - SocketTimeoutException, StreamCorruptedException, IOException, FileException, and EOFException.
- Please use full duplex TCP for both sending and receiving. This means that there is no need to create a new connection every time you send a message. If you're sending and receiving multiple messages from a remote AVD, then you can keep using the same socket. This makes it easier.
- Please do not use Java object serialization (i.e., implementing Serializable). It will create large objects that need to be sent and received. The message size overhead is unnecessarily large if you implement Serializable.
- Please do not assume that there is a fixed number of messages (e.g., 25 messages) sent in your system. Your implementation should not hardcode the number of messages in any way.
- 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.