CSE 486/586 Distributed Systems

Programming Assignment 2

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

Components used:

In order to implement Totally and Causally Ordered Group Messenger with a Local Persistent Key-Value Table, the components, Ptest(Persistent Key-Value test), Test1 and Test2 (Totally and causally algorithm), needs to be written. The Ptest needs two functions of the Content Provider needs to be implemented which can be done by creating a class which extends Content Provider. Test1 and Test2 uses sequence vector as in sequencer algorithm, and so sequencer algorithm needs to be implemented.

Content Provider:

The Content Provider is basically used in order to store messages. The OnPtestClickListener makes call to insert() and query() methods of Content Provider, where insert() function is used to insert values into the file stored, and the key-value pair is used in order to insert value into the file, where key is the file name and value is the "value" to be written into the file. To make a call to the openFileOutput() and openFileInput(), an object of Context class needs to be created. Insert is basically done using the input/output (i.e. "java.io.*") file streams, the FileOutputStream is used to insert value into the file, the ContentValues and the Uri passed as an argument to the insert(), has the current file and the value that needs to be written to it. The Uri is usually used to map the ContentValues to appropriate provider.

The query() is performed using FileInputStream, arguments passed in the query include the Uri and a selection (file name), which is retrieved and the value is then retrieved using BufferedReader. Here a MatrixCursor (which is a mutable cursor) is used in order to store the current file name and value, and add a row, using the addRow() function. After adding row, the MatrixCursor is returned which is then checked for its correctness of value returned by it. If the records inserted by the insert() function and query() is performed successfully, then the Ptest is considered to be successfully implemented.

Idea or initial setup for passing Test1 and Test2:

Total and causal ordering is done on the click of Test1 button. Now, in order to implement this, we need to design a sequencer algorithm, where the sequence in which the messages to be displayed can be handled easily. Since this test is to performed using 3 AVDs, we need to create three AVD's and run them using the python scripts, create_avd.py and run_avd.py. Now we need to set connections between these, which can be done using, set_redir.py, which will connect all the three AVDs to a common listening port. Now, one of the AVDs need to become a Master or Sequencer which can decide in what order the messages need to be dispatched, so that the total ordering can be maintained and that all the AVDs display the messages in the same order as others.

<u>Total and causally ordering:</u>

In order to do so, we assign AVD0 as the Master or the Sequencer, and now the message, needed to be broadcast is sent through this AVD to the sequencer function. In order to verify the correctness of this message, the sequence numbers of the current message and the present state is compared, if the state of current message is valid then the message is broadcast, else it is stored in the buffer, and is compared with the sequence numbers of the incoming messages, if the condition meets, it is removed from the buffer and is broadcast after the message with which it was compared has reached the AVDs and been displayed. Here the sequencer is thus used to maintain total ordering of messages and thus all the active AVDs running will follow the total ordering, and thus the order will be preserved.

In here, a “currentVal” array has been used to store cuurent sequence number of the AVD sending the message and in the sequencer function, this number is compared to the present state number “newVal”, (for ex: 1,0,0 and 2,0,0) if the difference between the two is mod|1|, then the message can be displayed, otherwise it needs to be stored(for ex: 1,0,0 and 3,0,0).

Now, whenever the difference is mod|1|, the counter ”count” is increased, this is done so as to ensure that at a time only a single change in the sequence number of the AVDs is possible. Thus the sequencer function would return “OK” string, if the sequence is correct. On receiving “OK”, master can send out the messages to all active AVDs in order. In order to pass Test1, on button click, the number of messages to multicast is numbered 5, and the 5 messages are displayed on each of the AVDs display in same order, also multicasting of every message is followed by 3 seconds of sleep of thread, which is done by calling Thread.sleep(3000).

Here, the sequencer is responsible for the ordering of messages and so the algorithm would provide both ordering guarantees at the same time.

<u>Handle first message in Test2:</u>

On button click of Test2, client thread is called and here the initial AVD multicasting the message is handled, and a socket is created which will perform wirteBytes() for that message. Using this method of handling the multicast message, helped in discriminating between the multicast message and other messages which would be displayed, on receiving of this first message. After this, the message is appended with a string “Test2”, which would discriminate and help the thread to identify that the button click was for Test2, so now, when Test2 is clicked, it would check whether the message string being passed contains “Test2”, if yes, then client thread with number of messages as 2 is called and thus on receiving of a message being multicasted by AVD, all the active app instances, multicasts 2 more messages, thus each AVD will have 7 messages, displayed, on Test2 button click.

<u>Lessons learned:</u>

This assignment helped me to understand Content Provider and its functions, and implementation of ordering algorithms.