Assignment_1_Report_CS425

Xia Shang, xshang3@illinois.edu
Xiaocong Yu, xy21@illinois.edu
03/03/2018

1. Overview

This MP is implemented using Python 3. All of the three steps in the MP requirement have been completely implemented. We have also tested the correctness of each functions in two local machines with MacOS and Windows 10, respectively. The source codes for each step are attached in the zip file.

This assignment is divided into three parts:

First, implement unicast function to deliver message from end to end with socket programming.
Then, add delay() function to simulate the real network delays.
Lastly, to implement multicast() with Total_Ordering and Causal_Ordering algorithms.

The following three python files are included along with this report:

1. Unicast.py: this file contains the requirement for step 1 and step 2. It allows multiple users to send message to the target destination. A delay function was implemented, as indicated by step 2 requirement, to set a random seconds of delay when sending messages.
2. Causal_Order.py: this file implements multi-cast functions among multiple users (4 in this case) with causal-ordering mechanism.
3. Total_Order.py: this file implements multi-cast functions among multiple users (4 in this case) with total-ordering mechanism.

In addition, a config file is included that contains the minimum and maximum delay times, and the process ID, IP and port information for each of the four processes. All of the python files read information from the config files.

2. Implementation of each steps

2.1 Unicast

One thread was used in unicast program to keep listening incoming messages and deliver (print out) received messages.
The main program perform function to send message to the target process with ID number.

2.2 Unicast with delay

For the second part, we call Delay() in Unicast() by creating a thread. In Unicast, when we have decided the destination process, we generate a thread for sendto() to be carried out. In Delay(), we use random.lib to generate an integer in range we defined in configuration file, and the int is divided by 1000 to be the delay time whose unit is second. After the thread is set to sleep for delay time, it will start to send the message to the destination process.

2.3 Multicast by causal-ordering

For implementation of causal_order(), I add a tuple to store the information of current localmarker, and attach the updated localmarker as the first severl characters of the message everytime I do a causal-order multicast. For implementation of listen(). The first rule we should follow is that we should buffer any messagae whose timestamp-marker is not in the right causal-order for current processor, which means that some message happened causally-before this message has not been received and the localmarker for that port has not been updated. The second rule we should follow is that if the message is causally right, which means localmarker for source port is only less than timestamp-marker for source port by 1.

2.4 Multicast by total-ordering

In total ordering, the process ID 0 is set as the leader by default. The leader's job is to label all of the message from every processes with a number. The labeled message is then broadcasted to all the four processes. In each process, a buffer (hashtable where key is the label and message is the corresponding value) is created to store labelled messages. The label is extracted and converted to a numeric key for the buffer. A numerical marker which will increment every time a message is delivered is used to make sure message in each process will be delivered in the same order. For example, if the message with label of 5 is received before the message with label of 4, it will be stored in the buffer. And the value of the numerical marker will increased from 4 to 5 once the process receive and deliver the message with label of 4.

In the python file, the leader and the other processes have different listen function. In the leader, a thread is used to receive messages and add label to raw messages. Then a thread is used to broadcast the labeled message to every process. In every process, a thread is used to deliver messages that are in the buffer, which is a global variable. Everytime a message is delivered, that message will be deleted from the buffer.

3. Execution of the files

The following execution methods have been tested in Terminal windows in Macbook and command window in a window laptop.

3.1 Unicast with delay

Open four terminal windows, type "Python3 MP1_Unicast.py" to start the program.
Next, enter the process number you want to use: 0-3

To perform unicast, type “send destination message” to send message to the target process with its ID number.
If you want to remove the delay, simply go to the program and set the delay_time to 0.

3.2 Multicast with causal ordering

To execute the MP1_CausalOrder.py, we should type in command “python MP1_CausalOrder.py ” under the file’s directory. And we type in the process number we want to use such as 0, 1, 2 or 3. And we can send the message now by only type in “msend message ”.

As for the testing for MP1_CausalOrder.py, we should set the delay range large, i.e 1000 - 10000, and suppose we msend message A from process 1 and we receive a multicast message A from process 2 first, then we can instantly msend a message B on process 2’s prompt, which happens causally later than message A. After messages are received by all processes, we can check the receive time with two messages, if the received time of message A is later than received time of message B and it delivers message in order of “A B”, then it is causal ordering multicast. Since the marker of message B is (0, 1, 1, 0) and the marker of message A is (0, 1, 0, 0), the process will never delivers messages in order of ”B A”.

3.3 Multicast with total ordering

Open four terminal windows, type “Python3 MP1_TotalOrder.py” to start the program.
Next, enter the process number you want to use: 0-3
To perform multicast with total ordering, type “msend message” to send multicast message to all other processes.