**jpmathe2**

**yumochi2**

**Simple Multicastor Simulater**

A simple multicastor simulater that is capable of unicasting and multicasting based on causal and total ordering.

**Getting Started**

To run the program

* Download the mp1.py, config.yaml files.
* Run python mp1.py to start the program

**Prerequisites**

What things you need to install the software and how to install them

Python 3 recommended

**Operate**

The program work based off of 4 proposed processes that are meant to run on one machine.

To start the program, run

python mp1.py –i # -c #

i and c are argument parsed from the terminal using arg parser package

i – can be any integer from 0 to 3 to designate the process id.

c – use 0 or 1 to indicate total ordering or causal ordering respectively.

# - indicate the number to use for i and c respectively.

Once the program starts use the following command to navigate from the user interface.

‘help’ - to enact helping messages

‘send destination message’ - sending message to destination process

‘msend message’ - sending message to everyone

‘show clientTable’ - to show client table

‘show messageDict’ - to show message table

‘show tMessageDict’ - to show meta message information

‘show stamp’ - to show time stamp

‘exit’ – to exit program

**Implementation**

The code is written in python, based on simple socket programming. The simulated multicast works through 3 separate threads, one for user interface, another for listening, and the last for monitoring received messages.

Network Delay

Network delay is implemented using a simple non-blocking timer, ie threading.Timer. The interval for the timer is randomly drawn from the min and max values associated with each.

Causal Ordering

Causal ordering is implemented in the simulator based on vector time stamps, ie releasing messages based on the value of each respective process.

In each process, a dictionary of messages based on process id is kept. In turn, the value of these ids are lists of messages tuples in the form of (stamp, message). True to the algorithm discussed in class, each process also maintains its own time stamp. The comparison between the process’s own and each message’s determines whether the message is delivered to the process.

Total Ordering

Total ordering is established through a sequencer (P0) using a count for the messages that the sequencer receives and individual counters for messages received from each of the processes.   
Each process sends a message and a count identifying which message of the process it is. The sequencer stores both of these and uses the metadata (process, messageNum) to implement total ordering. When a process receives the metadata, it looks in its own table and delivers the message if it contains it, otherwise it waits until its table is updated (i.e it receives the message) and then delivers it.

Upon receiving each message, the sequencer sends a multicast containing

**Authors**

**Joel Mathews – jpmathe2**

**Yumo Chi – yumochi2**

**Acknowledgments**

* To all the TA’s that helped clarify any confusions