**CS 425/ECE 428 Distributed Systems (Spring 2018)**
**<u>Sections T4 and V4</u>**

**Programming Assignment 1 (MP1)**
**Due: 7 p.m. March 1, 2018**

**This assignment may be performed by a group of two students, or individually.**
You may use any of the following languages: **C, C++, Java, Python**

This assignment involves **three** steps. The first step will be to create socket connections among multiple processes. The second step will be simulating network delays. Step three will be implementing two different multicast message ordering schemes: causal ordering, and total ordering.

## Step 1:
## Socket Programming

In step 1, create a socket connection between each pair of processes in your system. A **configuration file** should contain the identifiers, IP addresses and ports to use for the processes. **Each process will have a unique integer identifier (such as 1, 2, 3…)**.

Implement basic unicast functionality: **unicast_send(destination,message):** Sends message to the destination process.
**unicast_receive(source,message)**: delivers the message received from the source process

## Step 2:
## Simulating network delays

For this step, you will need to simulate communication channel with delay between each pair of processes. Implement the system in layers as follows: (i) Implement a channel with delay on top of sockets. (ii) The processes will use these channels with delay to communicate with each other. (iii) When a process 1 sends a message to a process 2, the send operation should complete once the message is handed over to the layer that implements the channel with delay. (For instance, do not put the process itself to sleep to simulate the delay). To simulate a network with variable delay, and potentially non-FIFO delivery, each unicast message should be delayed for a bounded random duration of time.  The **configuration file** should specify the minimum and maximum delay for any channel. Choose the delay for each message uniformly and randomly between the minimum and maximum delay.

Modify the implementation of **unicast_send** and/or **unicast_receive** function developed in step 1 to add the delay mechanism.

You should launch each of the processes from the command line. For instance, if four processes run on the same computer, then you should open four separate terminals, and launch one process each from each terminal. For testing purposes, we should be able to type a command in any of the above terminals, and then have the corresponding process handle that command appropriately.

For step 2, you should support a command using the syntax below, where “Message” is a plain-text string.

*send destination Message*

The message sender should print the time when it sends the message. The message receiver should print out the message and the time at which it receives

that message. For example, suppose you have two processes: 1 and 2 running on two terminals and the maximum delay from process 1 to process 2 is 3 seconds.
In process 1's command line, type in:

*send 2 Hello*

Then process 1 should print:
*Sent "Hello" to process 2, system time is -------------*
Process 2 should prints:
*Received "Hello" from process 1, system time is -------------*

During the project demo, you will need to demonstrate the message delay feature above.
You may find it convenient to implement each process using multi-threading (e.g., one thread for application functionality, one thread for the delayed-message delivery functionality etc. )

**Step 3:**
**Ordered Multicast Protocol**

Your goal will be to create a command line chat room. Each process will take input messages from a user and then multicast them to other processes. It will also receive messages via multicast from other users and display them to the user. Each message will consist of a single line of text. **Your implementation should allow at least 4 users/processes to participate in the chat.**
You will be writing an ordered multicast protocol. As such, you have to implement the multicast interface; in particular, the multicast protocol must support the **multicast(message)** call to send a message to a multicast group and **deliver(source,message)** to deliver a message (labeled with its source) to the

process. Your multicast protocol should be implemented as a layer above the delayed transmission mechanism implemented in step 2.

For this assignment, you will need to implement **two** different multicast message ordering schemes: **causal ordering** and **total ordering.**

Include a README file in your submission with a brief description of your algorithms for implementing each of the two multicast orderings.

**Testing the multicast:**

While launching your program, you may specify which multicast protocol to use as a parameter. Alternatively, you may write separate programs for the two protocols. Each process should accept “msend” command for multicast, e.g.

*msend Hello*

The recipient processes should show the **sender ID**, **system time** and **the messages** delivered according to the multicast protocol.

**Configuration file:**

The configuration file **should** have the following format

```
min_delay(ms) max_delay(ms)
ID1 IP1 port1
ID2 IP2 port2
….  ….  …….
```

The first line contains the minimum delay(in milliseconds) and the maximum delay for any communication channel. These two fields should be separated by a single space. Every line after the first line contains the ID, IP address and port number of a process, all separated by a single space. So if you have 4 processes, then the configuration file will contain 5 lines. The configuration file should end with 2 blank lines.

**Instructions for scheduling a Demo will be announced later.**

**Deliverables: (Submit Via Compass2g)**

1. Source Code
2. README (pdf or plain text) containing brief description of the implementation of steps 2 and 3. Also include a description of how to execute your program.

**The above two materials should be zipped into one file. Please use the format "netid_MP1" or "netid1_netid2_MP1" for your filename.**

**Grading Rubric:**

Implementation and correctness: 75%

a. Sockets, and Simulation of Network Delay: 25%
b. Causal Ordering (correctness and implementation): 25%
c. Total Ordering (correctness and implementation): 25%

Miscellaneous (comments, coding style): 5%

Report (README.pdf): 10%

Demo 10%