Programming Assignment 1: Reliable Total Order Multicast Protocol and Global Snapshot

Due Tuesday Oct. 9, 10 pm

1 Lab Task

In this lab, you will implement the *ISIS total order multicast algorithm* and the *Chandi-Lamport distributed snapshot algorithm*, both described in class and available in the class slides. The state to be captured by the snapshot algorithm is the state of the reliable total order multicast protocol. You will first implement the reliable multicast application, then add snapshot functionality to it, the snapshot needs to be performed while the total order application is not blocked. Details below.

1.1 PART 1: Reliable Total Order Multicast (60 points)

The multicast protocol needs to satisfy the following requirements:

- Reliability: Integrity, validity, and agreement
- Total order: All processes must agree on the order of all received messages

You should implement the protocol such that it guarantees the reliable total ordered delivery when a process multicasts a message to a closed process group. Messages can be delayed, or lost. Processes will not crash. More than one process can send at the same time.

Any process can multicast a message. **DO NOT FORGET A process should also deliver its own messages**. This self-delivery must guarantee the requirements of the multicast protocol. Each process should remain operational once started, even if there are no more messages to multicast.

For this lab, you must **NOT** use IP multicast. Instead you **MUST UNICAST** each packet to every other process. Once started, a process should not terminate by itself. You can kill your processes using **kill** command.

You need to implement this algorithm in **C/C++** . Your program (named as **prj1_tm**) **must accept** the following command line arguments.

```
Usage: prj1_tm -p port -h hostfile -c count

  -p port
    The port identifies on which port the process
    will be listening on for incoming messages.
    It can take any integer from 1024 to 65535.

  -h hostfile
    The hostfile is the path to a file that contains
    the list of hostnames that the processes are
    running on. It assumes that each host is running
    only one instance of the process. It should be
    in the following format.

    turban.ccs.neu.edu
    ...
```

```
    All the processes will listen on the same port.
    The line number indicates the identifier of the process
    which starts at 1.

  -c count
    The count indicates the number of messages the process
    needs to multicast. The value of count is non-negative.
    The process should remain operational once started, even
    if the count is 0 or the process finishes multicasting all its
    messages.
```

Cases to consider: Each type of message can be lost, or delayed. You should provide implementations of these cases, for example you can modify the command line to specify what test case are you selecting. Also your code should work correctly when more than one sender are sending at the same time.

Message format. Processes exchange multiple types of messages to guarantee total order delivery. The format of each message type are presented below. You **MUST** adhere to the following message formats.

```
typedef struct {
    uint32_t type; // must be equal to 1
    uint32_t sender; // the sender's id
    uint32_t msg_id; // the identifier of the message generated by the sender
    uint32_t data; // a dummy integer
} DataMessage;

typedef struct {
    uint32_t type; // must be equal to 2
    uint32_t sender; // the sender of the DataMessage
    uint32_t msg_id; // the identifier of the DataMessage generated by the sender
    uint32_t proposed_seq; // the proposed sequence number
    uint32_t proposer;    // the process id of the proposer
} AckMessage;

typedef struct {
    uint32_t type; // must be equal to 3
    uint32_t sender; // the sender of the DataMessage
    uint32_t msg_id; // the identifier of the DataMessage generated by the sender
    uint32_t final_seq; // the final sequence number selected by the sender
    uint32_t final_seq_proposer; // the process id of the proposer who proposed the final_seq
} SeqMessage;
```

Output. After delivering every `DataMessage` (including self-delivery), a process **must print to the standard output** the delivery order as follows:

`ProcessID`: Processed message `msg_id` from sender `sender_id` with seq (`final_seq, proposer`)

PLEASE FOLLOW INSTRUCTIONS for output printing.

1.2 Part 2: Adding Distributed Snapshot Functionality (30 points)

One of the process will initiate the snapshot after sending X number of packets. This will be specified in the command line with the additional option

```
Usage: prj1_ds -p port -h hostfile -c count -s X
 -s X
    The X indicates the number of messages after which the process
    will start the snapshot. X < count.
```

For simplicity, only one process can initiate the snapshot, the snapshot is performed only once, the state consists of the state of the processing and delivering buffers (what messages have been ordered and what messages are still to be ordered, and the sequence of the last message sent). This will be printed at the end, in this order. You can use separate TCP connections for the snapshot implementation.

2 Submission Instructions

Your submission must include the following files (10 points):

1. The **SOURCE** and **HEADER** files (no object files or binary)
2. A **MAKEFILE** to compile and to clean your project
3. A **README** file containing your name, instructions to run your code and anything you would like us to know about your program (like errors, special conditions, etc.)
4. A **REPORT** describing the system architectures, state diagrams, design decisions, and implementation issues

You should have two subdirectories, part1 and part2 for the two components of the project.

Information about submission is in post @11 in piaza. Name of the project in the submission command is prj1. Please do not submit be email.

3 Additional resources

You may find the following resources helpful

- Socket programming: `http://beej.us/guide/bgnet/`
- Unix programming links: `http://www.cse.buffalo.edu/~milun/unix.programming.html`
- C/C++ programming link: `http://www.cplusplus.com/`