18-842: Distributed Systems **Spring 2014**
Lab 2: Multicast Communication

Objective: Students will demonstrate reliable, causal-ordered multicast messaging in a distributed system. Recall from lecture that a reliable multicast system is one which:

> Sends messages, m, to other processes, P, in a group, G, with an atomic property: If m is delivered to any P in G, then it will be delivered to all P in G.

Details: You will use the infrastructure you have already developed to complete this lab. Use MessagePasser to send messages between various nodes. Use ClockService to order the messages.

You will use the configuration file to specify the membership of each group. The groups are static — they don't change members. But, you can have several groups and nodes can be members of several groups. The configuration file will have another section to specify these groups, like so:

```
groups :            # A single, top-level section in the YAML file
  - name : Group1   # Group names must be unique
    members :       # Any number of members in the group
      - alice
      - bob
      - charlie
  - name : Group2   # As many groups as you'd like
    members :
      - daphnie
      - bob
      - charlie
```

You will develop another infrastructure-like service to handle the multicasting. When an application asks your service to multicast a message to a group, it will take care of sending the message to **every** node in the group.

Your multicast service won't use any multi-or-broadcast service built into the operating system. Rather, it will simply serially send( ) messages (using MessagePasser) to each peer node. You will want to be able to demonstrate what would happen if it "crashed" in the middle of this list (i.e. sent messages to some peers and not to others). You cannot show crashes with sufficient proficiency via a ctrl-C or some such, so you will have to create some such facility (hmm... or perhaps fake it with some other infrastructure system you already have created).

When your multicast service receives a message from MessagePasser, you'll want to store those until they are ready to be delivered to your application layer. Before you deliver it, you'll want to ensure two properties are met: reliability and order. How will you ensure reliability? The order that the messages are delivered should be causal-

ordering, which should be fairly straightforward, given that you have a ClockService and can causally order messages.

Note that, once again, you have a fair amount of freedom in the design space. There are at least three really good architectures for building your system. If you find yourself having trouble, section 15.4 of the textbook has a pretty algorithmic description that can assist. You are expected to implement the algorithms in a similar manner to the textbook descriptions — if you end up hacking together your own algorithm, you'll probably lose many points.

Demonstrate: The application you build does not need to fulfill any other requirement other than to help you demonstrate the correctness of the multicast service. Use the drop/delay capabilities of MessagePasser to demonstrate that your multicast service is operating properly and reliably.

Design your application with your demonstration in mind. It should be fully interactive (i.e. no compilation will be allowed during the demonstration). It should also display any information that you will need to prove your multicast system is working.

During your demonstration, you will once again need to discuss the architecture and design decisions you made. Prepare a simple drawing (not handwritten!) showing the interaction among the objects and classes in your system. Be able to show the interactions of the objects for startup as well as for send and receipt of a message.

Also discuss what happens in your system in the presence of errors. What are you doing when messages get dropped / delayed / duplicated to ensure reliable service?

Demonstration: You must demonstrate an interactive program that shows reliable, causal-ordered multicast messaging in a distributed system. TAs will be looking for:

- Basic operation: a multicast message actually gets to everyone in the group and does not get to anyone not in the group.
- Demo system is interactive (i.e. doesn't require recompilation) and easy to understand what is happening.
- Multicast messages are causally ordered properly. As a basic test, multicast messages from Alice and Bob and ensure that everyone agrees on the ordering. For most interactive sessions, it will be difficult to do more than have trivial ordering examples, as most of the sending of one message will happen before any of the sending of another. However, by using the "Delay" configuration option, it should be possible to mix up the sending of the messages in order to ensure the individual multicast operations get overlapped.
- Test reliability during the sending of a multicast message. Use "Drop" configuration options to have Alice multicast a message that actually gets delivered to Bob, but is dropped before sending to Charlie. Does Charlie eventually get the message?

- Test reliability at the targets of a multicast message. If Alice sends a message to Bob and Charlie, but Bob drops his ACK, will he eventually have the message delivered? Exactly how this is tested will depend on the multicast scheme chosen by the designers.
- A good architectural overview and ability to discuss the various components of their system and the design decisions behind them.

Teamwork and Collaboration: You are expected to work with your assigned lab partner. This means that you must communicate well (including answering email), discuss your design and share responsibilities throughout the performance of the lab. Exhibiting good teamwork skills is inherently included in the grade you receive. Therefore, the following sorts of behaviors will cost you points:

- Not answering email from a partner who wishes to meet and work on the lab
- Daydreaming while the other teammate cranks out all the code
- Letting your teammate do all the work
- Doing all the work without involving your teammate
- Not being able to show understanding of your entire system during the demonstration

All work for this lab must be your own and your teammate's. You may ask other students for general assistance, but you may not copy their work. You may use any code from Lab0 and Lab1 developed by either teammate (or in the case of Lab0 code, potentially by either teammates' Lab1 teammate).

Administrative details: The lab is due at 9:30 AM on 10 February. At that time, an archive of your source code needs to be deposited on ALE. Each team will give a quick demo, using the identical code, to a TA by 9:30 AM on 12 February. You should contact your TA to schedule the demo before the deadline. Not showing up for your demo slot is inexcusably rude behavior that will cost you points, perhaps all of them.