18-842: Distributed Systems **Spring 2015**
Lab 3: Mutual Exclusion

Objective: Students will demonstrate mutual exclusion among processes in a distributed system.

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. Use your multicast service to handle any multicast requirements.

Your goal is to implement Maekawa's algorithm and explore some limits of that algorithm. The algorithm was discussed in lecture 5 and is shown on page 639-641 of the text. You also read Maekawa's paper, "A SQRT(N) Algorithm for Mutual Exclusion."

Maekawa's algorithm requires that request sets or groups need to be defined ahead of time. A general method of defining those groups is too advanced to attempt here. Instead, groups will be defined by the user (i.e. you) and specified in the configuration YAML file, just like you did for Lab 2. You may add a "memberOf" list to the configuration section to make looking up each node's membership affiliations easier, if you wish. Like so:

```
configuration :
  - name : alice
    ip   : 192.168.1.52
    port : 12344
    memberOf :
      - Group1
  - name : bob
    ip   : 192.168.1.112
    port : 14255
    memberOf :
      - Group1
      - Group2
  - name : charlie
    ip   : 128.2.130.19
    port : 12998
    memberOf :
      - Group1
      - Group2
  - name : daphnie
    ip   : 192.168.1.87
    port : 1987
    memberOf :
      - Group2
```

```
groups :
  - name : Group1
    members :
      - alice
      - bob
      - charlie
  - name : Group2
    members :
      - daphnie
      - bob
      - charlie
```

Requirements: The application you build does not need to fulfill any other requirement other than to help you demonstrate the correctness of the mutual exclusion service. Be able to communicate among 7 processes (hosted on at least 4 nodes), to request and release a single critical section, and to display to the user which process has the critical section (i.e. each node should be able to show if it has the critical section. If it doesn't have the critical section, it doesn't necessarily need to know which process does). Also, show the number of messages sent and received by each node.

Use the delay, duplicate, drop capabilities of MessagePasser to demonstrate that your mutual exclusion service is operating properly. Your implementation can assume that network communication is reliable. You may still need to use the drop capability in order to examine what happens if a node crashes.

The basic algorithm as discussed in class is deadlock-prone. In the paper, Maekawa uses INQUIRE and RELEASE messages to prevent deadlock. You do not need to implement the deadlock prevention mechanisms.

Once again, your application needs to be fully interactive (i.e. no compilation allowed during the demo) and needs to display enough information to be able to prove to a TA that it is operating correctly. You should be able to initiate requests for critical sections from any process at any time. The user interface needs to let the user know whenever a process believes it has the critical section. The user should then be able to release the critical section at will.

Explore the repercussions of different group constructions. Recall from lecture the four properties for how the groups are put together. Demonstrate what happens when each of those properties is not met. Some of these demonstrations might result in disastrous consequences. Others might just show inefficiencies in the number of messages sent.

You also should provide a hardcopy illustration of your system architecture to illustrate the interaction among the various components of your system.

Demonstration: You must demonstrate an interactive application that shows mutual exclusion in a distributed system. The TAs will be looking for:

- Basic operation: 7 clients (on at least 4 different machines) can request access to a "resource" and release the resource. Blocked clients know they are blocked. Granted clients know they got the resource. Each client displays number of messages sent and received.
- Mutual Exclusion: If the resource is held by one client, a second client who requests access will be blocked (or not get access in some other way. I would imagine most groups have blocking access, but that isn't really a requirement).
- Resource Release: When the client who had the resource releases it, the next requester (in logical clock order) is given access.
- Test that the system works beyond two participants. Among 4 clients of a 7 client system (Alice, Bob, Charlie, David), have Alice get access to the resource, Bob and Charlie request it in that order (and get blocked) and David doesn't care. When the resource is released, Bob gets it (in logical clock order, right), while Charlie is still blocked. When Bob releases, Charlie gets it.
- Test reliability: Repeat the above scenario, but delay Bob's message. It shouldn't affect the outcome.
- Discuss group construction rules. Students should be able to provide either live experiments or data from their own experiments to show what happens when each of the 4 group construction rules is violated.
- A good architectural overview and ability to discuss the various components of their system and the design decisions behind them.

Teamwork and Collaboration: Exactly like previous labs, 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 previous labs developed by either teammate or incorporated in their work from previous teammates.

Administrative details: You will work in teams of two students, with the same partner as previous labs. The lab is due at 9:30 AM on 23 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 the assigned TA by 9:30 AM on 25 February. Contact your TA well ahead of time to schedule your demo.