Project 2: Distributed Mutual Exclusion

Deadline: Monday, March 15th at midnight

Introduction

In this project you will implement an extension of Raymond's tree-based token-passing distributed mutual exclusion algorithm. The system will consist of a set of mutual exclusion servers, a location server, and a set of clients. Your location server and mutual exclusion servers should all read the environment variable `SERVER_FILE` to determine the location of a file, which has contents like the following.

```
1 onya.cs.pitt.edu 3456 1
2 bert.cs.pitt.edu 3500 1
3 ernie.cs.pitt.edu 3550 2
4 arsenic.cs.pitt.edu 3400 2
5 speedy.cs.pitt.edu 3409 3
```

The meaning of this file is as follows. Each line represents one mutual exclusion server. It contains the integer id of the server, the host and port for communication with that server, and the initial value of the "holder" variable. The latter value defines the initial structure of the tree. In the example above, server 1 is initially the root. You may assume that the file is structured correctly (i.e., there are no cycles, etc.).

The clients will first contact the location server to find a mutual exclusion server. The location server will tell them a mutual exclusion server on the same machine as the client if possible, and a random server otherwise (the idea is to minimize network traffic where possible). Once a client knows the location of a mutual exclusion server, it can repeatedly issue either a *Acquire_Exclusive* or a *Acquire_Shared* request and then a *Release* request. The mutual exclusion servers will accept requests, and will participate in the Raymond algorithm in order to acquire the token whenever necessary. Having acquired the token, the server can grant access to clients that have requested the critical section.

A mutual exclusion server can satisfy multiple requests each time it receives the token. Furthermore, if multiple clients issue *Acquire_Shared* requests concurrently, then they should be be able to enter the critical section concurrently, even if they are connected to different servers. However, *Acquire_Exclusive* requests should *not* be fulfilled concurrently. You will have to think about how to enhance Raymond's algorithm to achieve this. You should choose your own fairness criteria to serve these requests. The order that these requests arrive may not be the order that they are served. However, your criteria should be reasonable and should avoid starvation.

The advantages of this approach over just using Raymond's algorithm amongst all clients is that the set of client processes does not need to be known in advance,

and clients can come and go easily, without needing to do any maintenance on the distributed tree. Also, the ability to grant shared requests concurrently can improve throughput. Finally, clients can fail (provided they do not hold the critical section when they fail), without preventing other clients from acquiring access to their critical sections. You do not have to worry about any of the servers crashing.

You should write code for the location and mutual exclusion servers, and also for a couple of example clients. One of these clients should repeatedly choose an access type (shared or exclusive) at random, make the appropriate acquire call, and then read a file if shared and write it if exclusive, and finally call *Release*. If your mutual exclusion algorithm is working correctly, the updates will not interfere with each other, and reads will never see inconsistent values! Give some thought to how you should test this.

You are required to specify both your algorithm and fairness criteria in your README file. Before programming, it is advisable to come up with a correct and clear algorithm design.

Requirements

You should put your source code, makefile, README file, and compiled code in an `/afs` directory, and this directory should be readable and executable by both the TA (zhangyt@cs.pitt.edu) and me (moir@cs.pitt.edu). Then send email to the TA, letting him know where the files are. After the due date, the files in that directory should not be modified (otherwise, penalties will be assessed).