CIS–4020 Lab
Distributed Mutual Exclusion

Last Revised: December 2, 2008

Introduction

In this lab you will write a program that illustrates Lamport's distributed mutual exclusion algorithm. Your program will participate in a network with those of your peers to control access to a single hypothetical resource. You should use C++ for your implementation language. C++ supports standard library component (STL vectors) that can be extended in size dynamically with a minimum of fuss.

1 Outline

Your program should dynamically allocate the array needed in Lamport's algorithm. As new nodes send you messages, the array should be expanded to incorporate them. You can assume that the nodes in the network are number in order from 0 to N-1. You will be assigned a node number in lab; your program must know what it is (use a command line argument).

Your program should use on thread to listen to incoming UDP packets on port 9999[1]. When a packet arrives this thread needs to access the array, the clock value, and perhaps send a reply message. Also, for each message received this thread needs to notice if a previous resource request can now be acted on. If so, it should start up a "resource-use" thread that "uses" the resource in some way. I suggest printing a message on the console.

[1] On the Windows machines in the lab it may be necessary to configure the firewall to allow external access to this port.

Messages must use an agreed upon format if everyone's program is to interoperate properly. The content of each message should be as follows:

```
type:timestamp:source_node
```

Where `type` is the message type (one of REQUEST, REPLY, RELEASE), `timestamp` is the message timestamp as an integer, and `source_node` is the node ID of the sending node as an integer. No additional characters should be in the messages; in particular they should not be terminated with a carriage return, line feed, or null character.

Meanwhile the program should use another thread to interact with the user, allowing the user to manually request and release the resource. When the user requests the resource, this thread should access the array and the clock value, and broadcast a request message. When the user releases the resource, this thread should terminate the resource-use thread, access the array and the clock value, and broadcast a release message.

Be sure you use mutual exclusion as necessary to protect the shared resources.

2 Report

Turn in a commented listing of your software. There is no other report.