Distributed Systems Project, Spring 2011

Second assignment: Vector clocks and causal multicast

In this assignment, you need to write two small programs that implement the vector clock algorithm and causal multicast and demonstrate that your programs works correctly in at least small setups.

Tasks

1. Write a program in the language of your choosing that implements the vector clock algorithm. The program must be able to read a configuration file which describes how many clients to start. Each client reads an input file and acts according to the instructions in the file (see format below). At the end of execution, each client must print out its vector.
2. Write a program in the language of your choosing that implements a causal multicast. The program takes the same inputs as the vector clock program and multicasts messages to all clients. At the end, each client prints out its vector.

Specifications

Command line interface:

```
program configuration_file line input_file
```

where `configuration_file` is the name of the configuration file and `line` is the line of this client (see below), and `input_file` is the name of the input file (see below).

The configuration file has an undetermined number of lines, each in the following format:

```
host port
```

where `host` is either the hostname or the IP address of the client and `port` is the port on which it is listening at that address. Your client program takes as argument one integer, which is the number of the line in the configuration that indicates what port this client should use. The client can ignore the hostname for its own line, but needs to use the other lines to know who are the other clients in the system. There is no upper limit to the number of lines in the configuration file. If the argument given to the program is negative or larger than the number of lines, your program is allowed to crash immediately.

NOTE: It's easiest to have all the clients run on the same machine.

The input file for the vector clock program has an undetermined number of lines, each in the following format:

```
ci X cj
```

where c_i and c_j are positive integers and `X` is either `M` or `L` (capital letter). Line "c_i `M` c_j" means that client c_i sends a message to client c_j. Line "c_i `M` `n`" means that client c_i advances its local clock by `n`, but does not send any messages. Note that both kinds of lines look the same and the letter determines whether to send a message or advance the local clock.

NOTE: When a client is reading through the input file and notices that it is supposed to receive a message, it must block until it has received the message.

The final output for the vector clock program has the following format:
`v1 v2 v3 ... vn`
where `vi` is the value of the vector clock for process `i` held by the client. There should be one entry for each client in the configuration file and if the client does not know anything about that client, the value should be zero.

NOTE: Because we use partly automated tools for checking the output, your output must match exactly the format above. Do not add any other text or lines. Failure to comply will lead to a reduced grade.

The input file for the causal multicast program has an undetermined number of lines, each in the following format:
`c1 [ | c2 | c3 | ... ]`
where `ci` is the number of the client who sends the multicast message. One line can contain several concurrent multicasts which are separated by `|`. The number of concurrent multicasts is not defined in advance. You can implement a multicast as a series of unicast messages.
NOTE: As the client processes the input file, it must block to wait for each incoming multicast message. In case the line has several concurrent multicasts, make sure the programs do not end in a deadlock.

The causal multicast program should output the following:

- Every time a message is ready to be delivered, output the number of the client who sent that message
- At the end, output the vector held by the client

Guidelines

You are free to choose any programming language, but we recommend using a higher level language, e.g., Ruby or Python, even if you have to learn the language from scratch during the assignment.

Use the vector clock algorithm defined in the Tanenbaum book. In short, each message contains the vector of the sending client and the receiving client will do the standard Lamport clock update between each element of the received vector and its own vector.

The actual contents of the messages are irrelevant, so you can even send empty messages as long as the vector is included. No interoperability between groups is required so you are free to choose the format in which you send the vector.

Deliverables

Program source code with documentation and a successful demonstration on February 22nd during the normal course session.

Timeline

The assignment is due on February 15th at 10:00. No extensions will be given.

Return

Return your code by email to Liang.Wang@cs.helsinki.fi. Please indicate clearly all the group members in every source code file.