Clock synchronization

Dario Brazzi, Francesco Pegoraro

March 04, 2013

Abstract

Implement in Java a clock synchronization primitive. Choose the strategy that you think best to synchronize the clock of a requesting machine against that of a reference (server) machine.

Contents

Part I
Problem description

1 Problem analysis

The goal of this project is to create a Java server providing a clock synchronization primitive.

The server must implement at least one standard protocol for clock synchronization, possibly more than one. A suitable choice is the Cristian's algorithm. The server will reply to each query with the current time, provided by the OS.

The client must be able to ask to a server the current time and to receive it to be displayed it to the user.

2 Assumptions

- For the Christian's algorithm, the network messages have comparable travel time, possibly identical.

Part II
Solution description

3 Cristian's algorithm

The simplest method to achieve clock synchronization is Cristian's algorithm. It requires a server who knows the correct time, a client that wants to synchronize its clock and a link between the client and the server.

The message exchange is the following:

- The client send a message to the server, requesting the current time, and starts a timer. M_1= (REQ_TIME)
- The server receives message $M_1$and immediately starts a timer. Than, process the request, building the response which is constituted of the current time and the time interval needed to build the response. So, just before sending the response stops the timer, than put the timer value and the current time in the response and send the response back to the client. M_2= (current_time, server_processing_time)
- The client receive message $M_2$from the server and stops its timer. Than, sets its current time to
  t = server_current_time + $\frac{1}{2}$(client_processing_time - server_processing_time)

4 Program Design

4.1 Server

The server must open a ServerSocket and listen to the first free port in the range [4444;4454]. If no free port in this range is available, the server will shut down displaying an error.

After the server has registered, will wait any incoming message from any client, until it's shutdown.

When a message from a client is received, the response is delegated to a new thread of the server, such that is possible to reply to a new client before finishing previous requests. The new thread will receive the connection with the client and the timer started by the parent process, timer that will measure the time took by the server to serve the response. It will reply to the client with the most recent current time and the server processing time, eventually closing the socket with the client.

The server will expose the following methods:

- *setPort()*: sets the server listening port to a specified value
- *getPort()*: returns the current listening port, or the default one if not specified
- *startServer()*: starts the server and opens a ServerSocket on the specified port. If the port is not specified, use one in the range [4444;4454].
- *stopServer()*: if the server is active, stop it and close the open socket.

4.2 Client

The client ...

4.3 Protocol

The protocol ...