ASSIGNMENT 3: Client-Server Game

Out: 2/21/02; Due: 3/7/02
Programming Parallel and Distributed Systems
Computer Science 178, Spring 2002
Steven P. Reiss

OBJECTIVE

The purpose of this assignment is to let you develop and play with a client-server application. This includes not only developing the code but also attempting to debug the application across multiple machines.

THE PROBLEM

There are many types of multiple-process programs. We are going to concentrate, both in the class and in this assignment on client-server computations. Here one of the processes is designated as the “server” and is treated differently from the others with are designated as “clients”.

There are a variety of different types of client-server applications. In most of these the server controls some resource, often a database, that the clients want to access in a controlled way. However, most such database applications are pretty boring both to implement and to test.

In place of a database as our central resource, we are going to place a game engine. The server than is responsible for controlling access to, updating, and providing information from this game engine to a set of clients. The clients, each of which represents a player of the game, will want to have a reasonable interface to the state of the game and will want to play the game. Actually, there can be multiple sets of players that want to play games simultaneously using the same server and there can be multiple servers.

SPECIFICATIONS

Your first job is to choose an appropriate game. This can be something as simple as tic-tac-toe, or something more complex. You don’t want to make the game too complex (even if you are working in a group) since we want to concentrate on the client-server and networking aspects of the problem rather than the game logic.

Your server must be capable of handling at least 8 users. Moreover, your system should be able to support at least 8 servers running simultaneously on the network. The users should be divided into multiple games (where a game is run on a single server). The client should provide the facilities for connecting to an appropriate server, for choosing or connecting to a game in that server, and for playing the game.

The particular functionality we will be looking for includes:

- A mechanism for finding all active servers for the game.
- A means for starting new servers when appropriate.
- A mechanism for finding all active games in one or more servers including enough information about these games to present a new or continuing client with a choice.
- A mechanism whereby a client can connect to a given game. If multiple clients need to be connected before a game can start, this mechanism must provide the client with information on how many clients have yet to attach and when the game is ready to play.
- A mechanism for handling game playing, taking moves from the client either continually or on a turn-by-turn basis.
- A mechanism for handling non-turn client requests (e.g. history of the game thus far, board refresh) asynchronously.
- Support in the server for clients who disconnect or crash whether cleanly or not. This should include providing appropriate feedback to the other players of the game.
- Support in the client for handling a server abort. This could range from a simple error message and exit to the starting up a new server and restoring the game state.

You might also want to include lots of error checking both in the client and the server to ensure that the messages you are sending back and forth are appropriate. Also for debugging purposes, you will want to create command-line (or file-based) interfaces for the clients so that you can create appropriate test scripts.

You should build this game framework on top of Java RMI.

TESTING

You should create a test script that will demonstrate that your game works with multiple clients and multiple servers. The script should, if possible, demonstrate all the functionality of your game framework.

MECHANICS

You should hand in full source code and runnable binaries as well as annotated output from the test script that demonstrates that the system works.