SPL - 161
Assignment 3

Published on: 03/01/2016
Due date: 21/01/2016

1 General Description

In this assignment you will implement a text-based game server and client. The communication between the server and the client(s) will be performed using a simple text based protocol (TBGP), which can potentially support different games; However, your server will only support a single game - Bluffer.
The implementation of the server will be based on the **Reactor** and **Thread-Per-Client** servers taught in class. You will have to adjust them to support more complex protocols, implement the TBGP, and finally, add support for the Bluffer game in your server, and implement a client.
The rest of this document is organized as follows: section 2 presents the TBGP and its specifications, section 3 presents the Bluffer game and its implementation using TBGP, section 4 describes implementation details.

2 Text-based Game Server Protocol (TBGP) Specification

2.1 Establishing a client/server connection

Upon connection, the client must specify a nickname using the NICK command. The nickname must be unique and cannot be changed after it is set. The server will respond with a SYSMSG specifying whether the nickname was successfully acquired.

2.2 Game Rooms

Games are held in rooms, which can also be used for chat.
A user can create and join a room using the JOIN command. **A user can only be in one room at any given time**.
A game can be started using the STARTGAME command. Once started, the room will not accept new users until the game is over.

2.3 Supported Commands

The TBGP supports various commands needed in order to create chat rooms and start games in them. There are two types of commands, Server-to-Client and Client-to-Server. The commands are all formatted as follows:

<Command> <Optional: Parameters>\n

2.4 Server To Client Commands

- Command: ASKTXT
  Parameter: text - question to ask.

Used to ask the client the specified question, expecting a TXTRESP.

- Command: ASKCHOICES
  Parameter: text - question to ask.
  Used to ask the client the specified question, expecting a SELECTRESP.

- Command: SYSMSG
  Parameter: text.
  Response to every command the user sends. The format is:

  SYSMSG <Original Command> <Result> <Optional: Custom Message>\n

  Where < *Result* > is one of: $ACCEPTED$, $REJECTED$ or $UNIDENTIFIED$.

- Command GAMEMSG
  Parameter: text.
  Custom messages that are a part of the game being played.

- Command: USRMSG
  Parameter: text.
  Used to deliver a message sent by another user, in the context of a game-room.

2.5 Client To Server Commands

- Command: NICK
  Parameter: nickname.
  Requests the specified nickname.

- Command: JOIN
  Parameter: room name.
  Joins the specified room, creating it if it doesn't exist.
  This also causes the user to leave the current room, therefore is impossible while a game is in progress.

- Command: MSG
  Parameter: text to be sent.
  Sends the specified message to the users current room.

- Command: LISTGAMES
  Lists the games supported by the server.

- Command: STARTGAME
  Parameter: game name.
  Starts the specified game in the current room. Game is specified using a unique string identifier.

- Command: TXTRESP
  Parameter: text to be sent.
  Used as a response to ASKTXT command, providing a textual response.

- Command: SELECTRESP
  Parameter: integer - selected response.
  Used as a response to ASKCHOICES command, providing a numeric selection.

- Command: QUIT
  Closes the connection.

3 The Bluffer Game

The Bluffer game is a type of trivia game, with a twist - the players try to fool each other into choosing absurd answers. The game host asks a series of questions, for which the players try to provide answers that *seem* real. The players are then presented with both the real answer, and the fake answers provided by other players, and have to choose the real one.
Players are awarded 10 points for choosing the correct answer, and 5 for each player that chose one of their fake answers.

3.1 Bluffer using TBGP

The Bluffer is uniquely identified in the TBGP using the string "BLUFFER".
Once started, the server will load 3 random question from its database (3.2), and for each question (sequentially):

1. Use ASKTXT to present the question to the user.
2. Once all the users reply (using TXTRESP), their responses must be turned to lowercase and shuffled with the real answer, and then presented with ASKCHOICES.
3. Users reply with SELECTRESP. Once all the users reply, score is awarded (as described in the game description), and is announced along with the correct answer.
4. Move on to the next question.

After 3 questions, the scores summary is sent to the users using SYSMSG, and the game is over.

3.2 Questions Database

The questions for the Bluffer game are given to you in a JSON file. In it is an array of *question* objects. *question* contains two fields:

- *questionText* - String of the question.
- *realAnswer* - String of the real answer.

The file must be loaded each time questions are needed, so server restart is not required for editing the database.

4 Implementation Details

4.1 General Guidelines

- The server should be written in Java. The client should be written in C++ with BOOST. Both should be tested on Linux installed at CS computer labs.
- You must use maven as your build tool.
- The same coding standards expected in the course and previous assignments are expected here.

4.2 Server

You will have to implement a single protocol, supporting both the Thread-Per-Client and Reactor servers presented in class.
The *ServerProtocol* interface, as presented in class, is limited to protocols where each user message is responded by a single message, sent to the same user. In order to implement TBGP, and other more complex protocols, the server needs to provide the protocol with means to send messages to any user, at any point in time.
In order to achieve this, the *ServerProtocol* and *AsyncServerProtocol* presented in class have been refactored to send a *ProtocolCallback* along with each message. This interface, given in 6.1.1 must be implemented by each server, and a unique instance must be created for each TCP connection.
Left to you, are the following 3 tasks:

1. Refactor the **Thread-Per-Client** server to support the new *ServerProtocol*. This will require changing the server to use a *Tokenizer*, and implementing the new *ProtocolCallback*.
2. Refactor the **Reactor** server to support the new *AsyncServerProtocol*.
3. Implement the new *AsyncServerProtocol*, and support the Bluffer game. Keep in mind, while you're only required to implement one game, your implementation should support easily adding more games.

4.3 Client

The client is multithreaded with one thread for handling the socket and another thread to handle stdin.
An echo client is provided, but its a single threaded client. While it is blocking on stdin it does not get messages from the socket. The client should receive the server's IP and PORT as arguments.
You may assume a network disconnection does not happen (like disconnecting the network cable).

5 Submission Instructions

- Submission is done only in pairs. If you do not have a pair, find one. You need explicit authorization from the course staff to submit without a pair. You cannot submit in a group larger than two.
- You must submit one .tar.gz file with all your code. The file should be named "assignment3.tar.gz". Note: We require you to use a .tar.gz file. Files such as .rar, .zip, .bz, or anything else which is not a .tar.gz file will not be accepted and your grade will suffer.
- The submitted .tar.gz should contain:
  - server folder, in it: src folder and the pom.xml file.
  - client folder, in it src, bin, include folders, and a makefile.
- Extension requests are to be sent to majeek. Your request email must include the following information:
  - Your name and your partners name.
  - Your id and your partners id.
  - Explanation regarding the reason of the extension request.
  - Official certification for your illness or army drafting.

  Requests without a compelling reason will not be accepted.

6 Appendix

6.1 Server Protocols

6.1.1 ServerProtocol

```
/**
* A protocol that describes the behaviour of the server.
*
* @param <T> type of message that the protocol handles.
*/
public interface ServerProtocol<T> {

        /**
        * Processes a message
        *
        * @param msg the message to process
        * @param callback an instance of ProtocolCallback unique to the
           connection from which msg originated.
        */
        void processMessage(T msg, ProtocolCallback<T> callback);

        /**
        * Determine whether the given message is the termination message.
        *
        * @param msg the message to examine
        * @return true if the message is the termination message, false
           otherwise
        */
        boolean isEnd(T msg);

}
```

6.1.2 AsyncServerProtocol

```
/**
* A protocol that describes the behavior of the server.
*/
public interface AsyncServerProtocol<T> extends ServerProtocol<T> {

    /**
    * Processes a message.
    *
    * @param msg      the message to process
    * @param callback an instance of ProtocolCallback unique to the
       connection from which msg originated.
    */
        void processMessage(T msg, ProtocolCallback<T> callback);

        /**
        * Detetmine whether the given message is the termination message.
        *
        * @param msg the message to examine
        * @return true if the message is the termination message, false
           otherwise
        */
        boolean isEnd(T msg);

        /**
        * Is the protocol in a closing state?.
        * When a protocol is in a closing state, it's handler should write
           out all pending data,
        * and close the connection.
        *
        * @return true if the protocol is in closing state.
        */
        boolean shouldClose();

        /**
        * Indicate to the protocol that the client disconnected.
        */
        void connectionTerminated();

}
```

6.1.3 ProtocolCallback

```
/**
* An interface that bridges between the protocol and the server.
* The server should implement this interface, and pass an instance of it
    along with any message.
* The instance passed should be unique to each TCP connection.
* It therefore allows the protocol to identify the sending user between
    messages, and reply at any point in time.
*
* @param <T> type of message that the protocol handles.
*/
public interface ProtocolCallback<T> {
        /**
        * @param msg message to be sent
        * @throws IOException if the message could not be sent, or if the
            connection to this client has been closed.
        */
        void sendMessage(T msg) throws java.io.IOException;
}
```

6.2 Example

Given here is an example of a communication session between one client and a server. Note this server only asks one question, unlike the requirements in the assignment.

→ NICK Lacey

← SYSMSG NICK ACCEPTED

→ JOIN Buckkeep

← SYSMSG JOIN ACCEPTED

→ STARTGAME BLUFFER

← SYSMSG STARTGAME ACCEPTED

← ASKTXT Insanity: doing the same thing over and over again and expecting ___ results. Albert Einstein.

→ TXTRESP spectacular

← SYSMSG TXTRESP ACCEPTED

← ASKCHOICES 0. different 1. spectacular

→ SELECTRESP 0

← SYSMSG SELECTRESP ACCEPTED

← GAMEMSG Correct! +10pts

← GAMEMSG Summary: Lacey: 10pts

→ QUIT

← SYSMSG QUIT ACCEPTED

Good Luck and Have Fun!