HW1 – A Jump-start into Interprocess Communications and Resource Sharing

Due Date: *9/7/2012*
Estimated time: *10-16 hours*

Objectives

- Become familiar with the selected development environment
- Start to become familiar with basic inter-process communication using datagram sockets.
- Start to become familiar with of the basic design issues for distributed system, i.e. concurrency, fault tolerance, etc.

Overview

This assignment involves implementing part a distributed word-guessing game that uses a client/server architecture. The instructor will provide the server program and you will build a client program.

Using datagram (UDP) socket communications, your client program will retrieve a definition and initial hint for word from a server. It will display the definition and hint and allows the player (i.e., the user) to guess the word. When the player enters a guess, your client program will send that guess to the server and the server will responses with a message indicating that either the guess was right or wrong. If it was right, the message will also include a score. If it was wrong, the message will include number that represents how many letters were correct and in the correct place. Also, if the guess was wrong, the player should be able to enter another guess and repeat the process until the correct word is found. The server will compute the score based on the number of letters in the word, the elapse time, the number of guesses, and the number of hints that were used.

Instructions

Your client program must satisfy the following functional and non-functional requirements

Functional Requirements

1. The client program must allow the player to specify/change the address and port of the server. Ideally, it should remember what was used in the previous session and provide that address and port as the defaults. Hint: In VS Studio, look at how you create and use a Settings File in your project.

2. The client program must allow the player to start a new game at any time, even if another game is in progress.

a. To start a game, the client program must send a newgame message to the server. See Table 1.
b. The client program must be able to receive a *def* message from the server in response to the *newgame* message. Table 1.

3. When a game is in progress, the client program must allow the player to see word's definition and hint
    a. The word's definition is a string with between 1 to 200 characters
    b. The hint is a string of characters that represents the word to be guessed, where some letters are given and others are placeholders (underscores). The letters and placeholders are separated by spaces. For example, "_el__" could be a hint for the word "hello".

4. When a game is in progress, the client program must allow the player to enter a guess, by typing in the full word.
    a. When the player enters a guess, the client program must send a *guess* message to the server and then be able to receive an *answer* message from the server. See Figure 2 and Table 1.
    b. If the *answer* message indicates that the guess was correct, then the client program should let the player know and display the score that was contained in the *answer* message.
    c. If the *answer* message indicates that the guess was incorrect, then the client program should let the player know, display the number of letters that were correct and in the correct place, and then let the user enter another guess.

5. When a game is in progress, the client program must allow the player to request a hint.
    a. When the player requests a hint, the client program must send a *gethint* message to the server and then be able to receive a *hint* message from the server. See Figure 3 and Table 1.
    b. When the client program receives a *hint* message, it should display it for the player.

6. All messages in this system will be strings, consisting of printable ASCII characters, in the following format:

    <message type> : <parameters>

    where,

    message type is one of the following of types listed in Table 1
    parameters is a coma separated list of parameters.

7. All messages in this system will be strings, consisting of printable ASCII characters, in the following format:

Non-functional Requirements

1. You will create your word-guessing game client in either C# or Jave, using the communication classes provided in the .Net Framework or Java SDK.

2. You may development and test your client by running your own instance of the server. However, you must also test your client with a server running on a different machine. For you convenience, the instructor will try to keep a server listening at following address, port: 129.123.41.13, 12001.

Note, you are free to design and implement whatever kind of user interface you'd like for your client. Also, for this assignment, don't be too concern able implementing reliable communications. Assume that your messages will be received by the server, in the order they were send, and without being duplicated by the network.

Table 1 – Message Types

Type	Usage and Sample Message	Parameters
newgame	The client sends the server a *newgame* message to start a new game. In response to a *newgame* message, the server will send back a *def* message Sample: *newgame:*	There are no parameters in this message
Def	The server sends a *def* message to the client in response to a *newgame* message Sample: *def:24,_b_l__,lack of ability to make decisions*	*<game id>*, *<initial hint>*, *<definition>* *game id* – integer (numeric string) *initial hint* – string *definition* – string
Guess	After the player enters guess, the client will sends that word to the server using a *guess* messages. Note that the *game id* parameter in this message must be the same as the game id returned in the *def.* In response to a *guess* message, the server will send back an *answer* message. Sample: *guess:24,abulia*	*<game id>, <word>* *game id* – integer (numeric string) *word* – string
answer	The server sends an *answer* message to client in response to a *guess* message. The result parameter indicates whether the guess was correct ("T") or incorrect ("F"). If the guess was correct, then the score parameter contains a number that rates the user's performance. If the guess was incorrect, then the score is the number of correct letters in the right place. Sample: *answer:24,F,180*	*<game id>*, *<result>, <score>* *game id* – integer (numeric string) *result* – "T" or "F" *score* – integer (numeric string)

gethint	If the player needs a hint, the client can send the server a *gethint* message. The server will reply with a *hint* message. Sample: *answer:24*	*<game id>* *game id* – integer (numeric string)
Hint	The server sends a *hint* message to the client in response to a *gethint* message. The *word* parameter contains the hint. Sample: *answer:24,_buli_*	*<game id>, <word>* *game id* – integer (numeric string) *word* – string
Error	The server will send an *error* message back to the client in response to any mal-formed request message. Examples of malformed request messages include: any type of message that the server shouldn't response and an invalid game id. Sample: *error:Invalid GameId*	*<error>* *error* – string

Submission Instructions

Zip up your entire solution in an archive file called CS5200_hw1_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Blackboard system.

Grading Criteria

Basic Criteria (worth up to 70 points)	**Max Points**
Correct functioning of the *new game* and *guess* features	20
Quality of the client's design (good abstractions, encapsulation, low coupling, high cohesion, etc.)	30
Quality of the implementation (good programming practices, readable code, maintainable, efficient, etc.)	20
Advanced Requirements (worth up to 30 points)	**Max Points**
A feature to display errors sent by the server	2
Correct functioning of the *hint* features	10
Ability for the user to change the server's address and port at any	5

time. If the user changes the server's address and port while a game is still in progress, the game is forfeited.	
Ability for the user to create a new game, even if there is one already in progress.	3
A well-design and attractive graphically user interface	15
Detection and handling of communication failure	20