HW1 – A Jump-start into Inter-Process Communications and Resource Sharing

Estimated time: 10-16 hours

Objectives

- Start to become familiar with request-reply style inter-process communication using datagram sockets in the selected programming environment
- Start to become familiar with the basic design issues for distributed system, e.g., message handling, message encoding/decoding, intra-process concurrency, and inter-process concurrency.
- Learn to setup and use virtual machines in a cloud, e.g. AWS
- Become familiar with logging tools, like log4net or log4j.
- Start to become familiar with basic unit testing techniques.

Overview

This assignment involves implementing the client part of a distributed word-guessing game that uses a simple client/server architecture. The instructor will provide an executable of the server program and you will build a client program. You will run the finished version on your client on a EC2 instance (a virtual machine) in the AWS cloud.

Using datagram (UDP) socket communications, your client program will interact with a server program to

1. start a game and retrieve a word definition with an initial hint (series of blanks and letters),
2. make guesses about the correct word,
3. ask for additional hints, and
4. exit the game

Each of the above represents a type of interaction or conversation, governed by a communication protocols. Figures 1-4 and Table 1 define these protocols and their messages in more detail.

A user (i.e., a player) runs your client program and your client program will communicate with the server in response to user interactions. Specifically, when the player starts a new game, your client will need send a *new-game* request to the server and wait for a *word-definition* response from the server. This exchange constitutes a *start-game conversation*. Your program will then need to display that word's definition and initial hint so the player can start guessing the word. When the player enters a guess, your program will send that guess to the server and the server will response with a message indicating whether the guess was right or wrong. If it was right, the response message will also include a score. If

it was wrong, the message will include a number that represents how many letters were correct (the right letter and in the right place.) Also, if the guess was wrong, the player should be able to enter another guess or ask for additional hint. The user should be able to repeat this process until the correct word is discovered. 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 user requested.

You can choose to write your client using any development tool stack that you wish, but an executable version must run on an EC2 instance (virtual machine). As part this assignment, you will obtain an AWS account, if you don't already have one, security educational credits from Amazon through the AWS Educate portal, and setup a virtual machine of your choice. Your program must ultimately run on that virtual machine.

Instructions

You can download the executable for the WordGuess server, a server monitor, and sample player from the classes shared Git repository, by using the following URL:

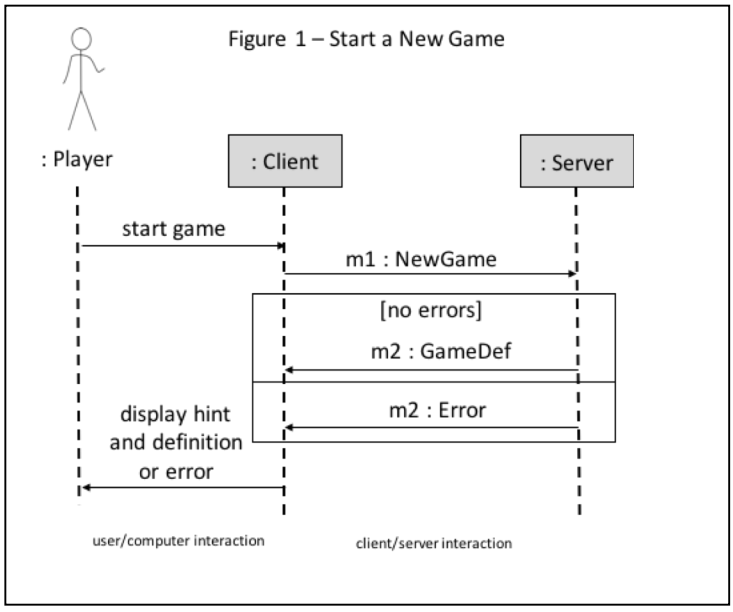
https://bitbucket.org/stephenclyde/cs5200f17-shared.git

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

Functional Requirements

1. The client program must allow the player to specify or change the address and port of the server. Ideally, it should remember what was used in the last session and provide that address and port as the defaults.

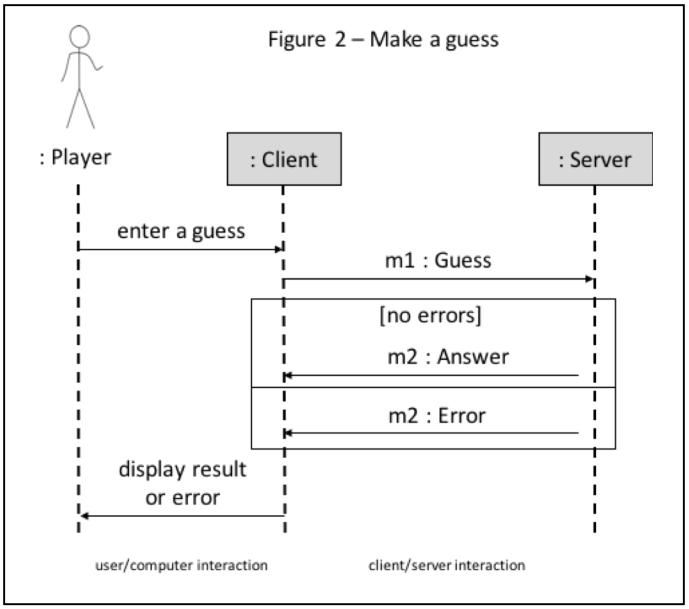
2. The client program must allow the player to start a new game at any time, even if another game is still in currently progress.
    a. To start a game, the client program must create, encode, and send a *NewGame* message to the server. See Figure 1 and Table 1.
    b. The client program must be able to receive a *GameDef* message from the server in response to the *NewGame* message.



3. The play must be able to decode the *GameDef* message to obtain word definition and hint.
    a. The word's definition will be 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). For example, "_el__" could be a hint for the word "hello".

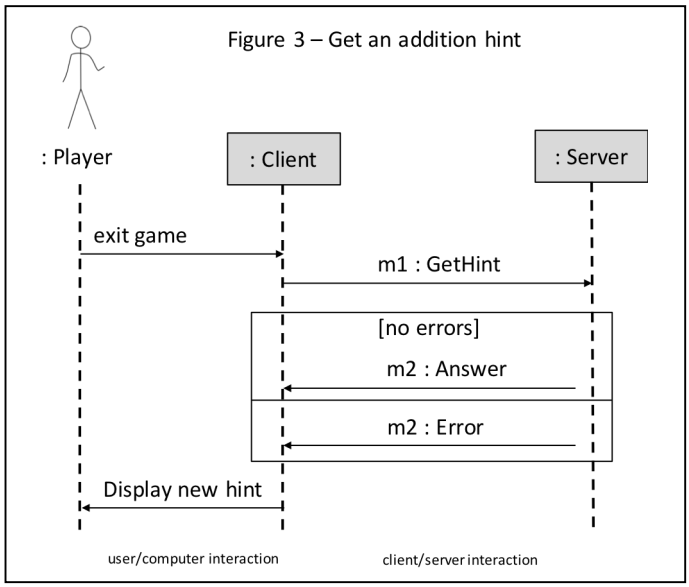
4. When a game is in progress, the client program must allow the player to see word's definition and latest hint.

5. 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 create, encode, and send a *Guess* message to the server and then be able to receive and decode an *Answer* message from the server. See Figure 2 and Table 1.

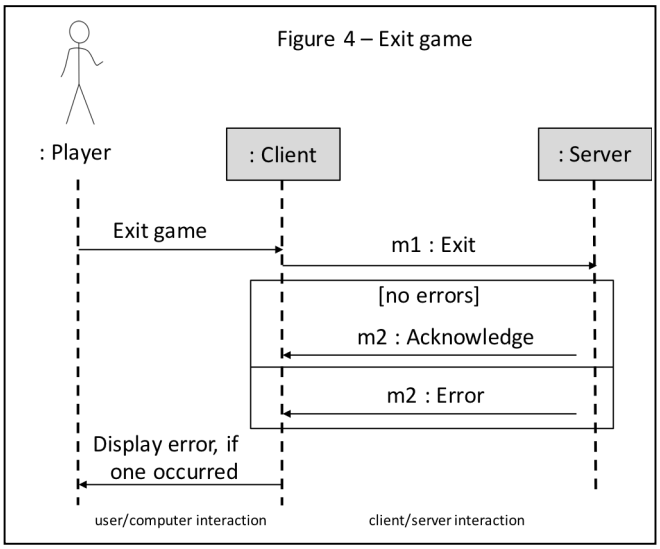


    b. If the *Answer* message indicates that the guess was correct, then the client program should let the player know and display the score that was contained in the *Answer* message. Note, if the Answer message indicates that the guess was correct, the hint will be empty since it is no longer needed.
    c. If the *Answer* message indicates that the guess was incorrect, then the client program should let the player know, display hint contained in answer message. The hint will should all the correct letters (those right place), and then let the user enter another guess.

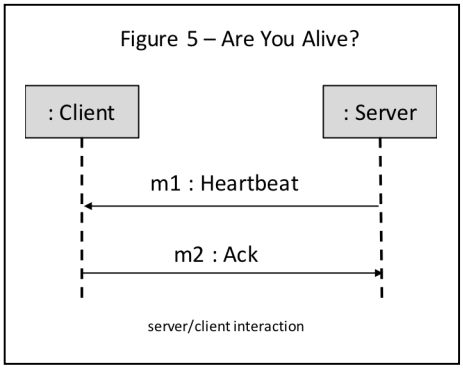
6. When a game is in progress, the client program must allow the player to request an additional hint.
    a. When the player requests a hint, the client program must create, encode, and send a *GetHint* message to the server and then be able to receive and decode a *hint* message from the server. See Figure 3 and Table 1.
    b. When the client program receives a *Hint* message, it should display it for the player.



7. A player could be able to exit at any time.
    a. When the player asks to exit, the client program must create, encode, and send an *Exit* message to the server and then be able to receive and decode an *Ack* message from the server. See Figure 4 and Table 1.
    b. When the client program receives an *Ack* message, it should stops playing the game.



8. A player must respond to a heartbeat message from the server as quickly as possible.
    a. After the player start game, the server will send a heartbeat message every 100 ms or so.
    b. A player must be able to receive that message decode it, and respond with a acknowledgement message within a few milliseconds. See Figure 5.



9. All messages in this system will be encoded into binary according to the formats outlined in Table 1.
    a. Short integers must be encoded as 2 bytes in network-standard byte order
    b. String must be encoded as a short-integer length x, follow by x bytes that are a Unicode encoding of the string. The Unicode encoding will contain 2 bytes (a short integer) per character. Each of these short integers must be in network-standard byte order.

Non-functional Requirements

1. You will create your word-guessing game client in any programming language of your choice, e.g., C# or Java
2. You must use the UDP as transport-layer protocol (i.e., datagram sockets).
3. You must test critical methods in your program using automated (executable) test cases.
4. You must setup and use a logging facility, e.g., Log4J or Log4Net, for debugging purposes
5. You must run your final version of your program on an EC2 instance. You may use the EC2 instance for development and testing, if you wish. When you run your final version, you must have it connect to a game server running on another EC2 instance provided by the instructor. The address and port number for this game will be posted to Canvas.

The instructor will provide a monitor client that will display the status of the game server. If this program has trouble connecting to the server and you are confident that you have a network connection, contact the instructor immediately.

Note that 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 about implementing reliable communications. Specifically, you can assume that your messages will be received by the server, in the order they were send, and without being duplicated by the network. As we will see later in the semester, this is not an appropriate assumption, but it is one we can make for now if we are willing to tolerate some flake behavior.

Instructions for signing up for an AWS account and creating EC2 instance will be given in a separate document.

Testing concepts will be discussed in classes, but the mechanics will depend on your chosen development environment. You may need to learn the specific of how to create and run executable test cases in your environment own your own.

Table 1 – Message Types

Type	Format (Unicode Character Sequence)
NewGame	The message's bytes will contain encodings of the following, in order: Message type=1 short integer Your A# string Your Last Name string Your First Name string Public Alias string
GameDef	The message's bytes will contain encodings of the following, in order: Message type=2 short integer Game Id short integer Hint string Definition string
Guess	The message's bytes will contain encodings of the following, in order: Message type=3 short integer Game Id short integer Guess string
Answer	The message's bytes will contain encodings of the following, in order: Message type=4 short integer Game Id short integer Result byte (0 = incorrect, 1= correct) Score short integer Hint string Note that the hint will be complete word, if the guess was correct
GetHint	The message's bytes will contain encodings of the following, in order: Message type=5 short integer Game Id short integer
Hint	The message's bytes will contain encodings of the following, in order: Message type=6 short integer Game Id short integer Hint string Note that the hint will be complete word, if the guess was correct

Exit	The message's bytes will contain encodings of the following, in order: Message type=7 short integer Game Id short integer Note that the hint will be complete word, if the guess was correct
Ack	The message's bytes will contain encodings of the following, in order: Message type=8 short integer Game Id short integer
Error	The message's bytes will contain encodings of the following, in order: Message type=9 short integer Game Id short integer Error Text string
Heartbeat	The message's bytes will contain encodings of the following, in order: Message type=10 short integer Game Id short integer

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 Canvas system.

Grading Criteria

Criteria	Max Points
Quality implementation of the New Game Protocol	15
Quality implementation of the Guess Protocol	15
Quality implementation of the Hint Protocol	15
Quality implementation of the Exit Protocol	15
Quality implementation of the Heartbeat Protocol	15
Integration of logging that is useful for debugging	10
Meaningful executable test cases for critical functionality	30
Running on AWS	5