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

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 based on 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 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 as the correct complete word,
3. ask for additional hints, and
4. exit the game

Each of the above represent a type of interaction or conversation, governed by a communication protocols. Figures 1-4 and table defines these protocols and messages that they use more detail.

As user (i.e., the player) interacts with your client program, your client program interacts will need to with the server and display certain information. For example, when the player starts a new game, it will need send a new-game request to the server and wait for a word definition and initial hint. It 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 client 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.

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.

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 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. 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). 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 character string that represents one messages types listed in Table 1
parameters is a coma separated list of parameters.

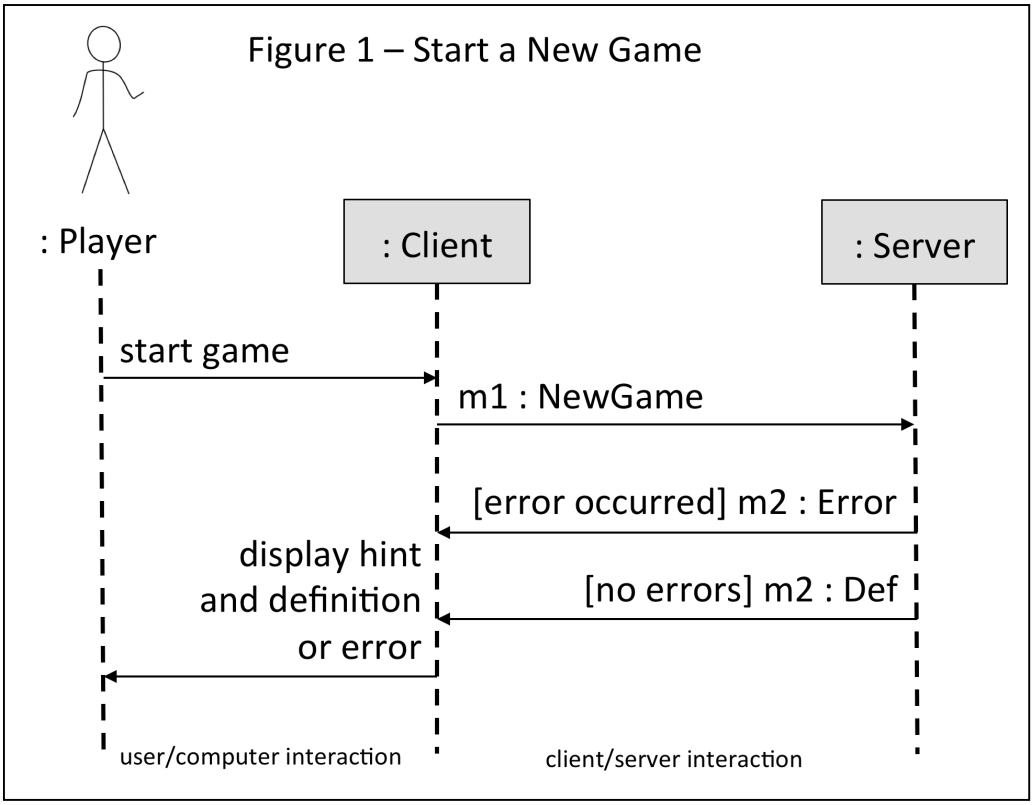
7. All messages in this system will be strings of printable ASCII characters.

Non-functional Requirements

1. You will create your word-guessing game client in programming language of your choice, e.g., C# or Java, using the UDP transport-layer protocol (datagram sockets).

2. You may development and test your client with a server running at following address and port: 129.123.41.13, 12001. If you believe that this server has gone down, please email the instructor and he will it check its status, and restart it if necessary.

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 about implementing reliable communications at this point. For example, assume that your messages will be received by the server, in the order they were send, and without being duplicated by the network.



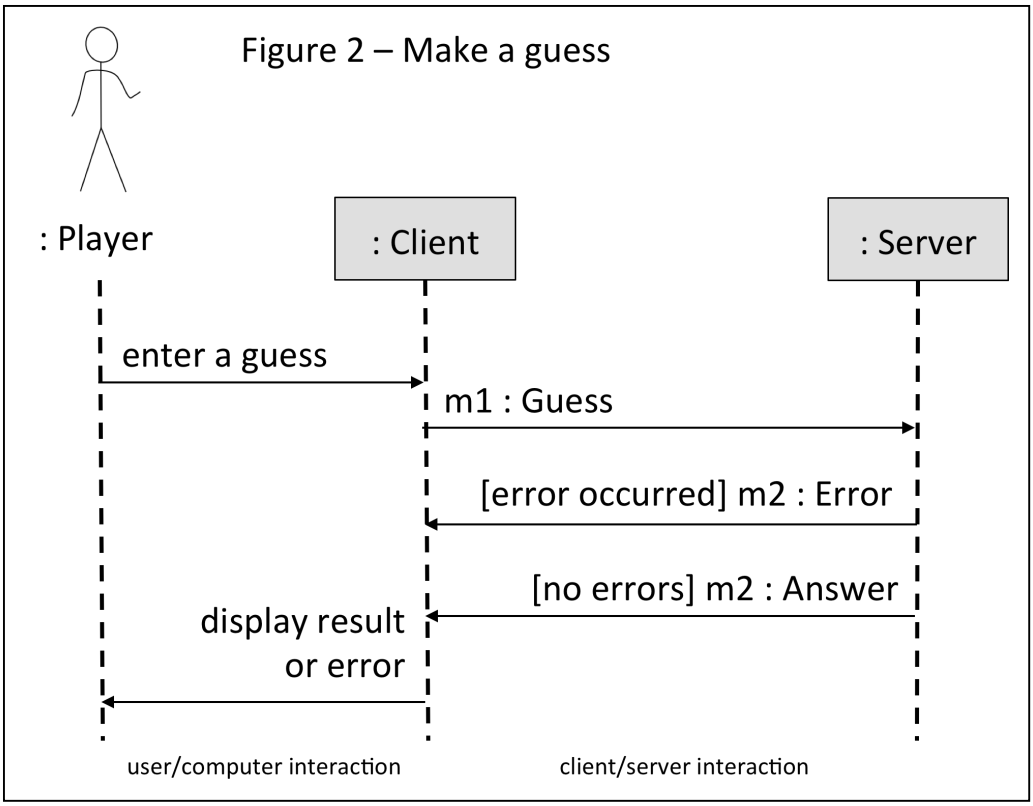


Figure 2 – Make a guess
: Player
: Client
: Server
enter a guess
m1 : Guess
[error occurred] m2 : Error
[no errors] m2 : Answer
display result
or error
user/computer interaction
client/server interaction

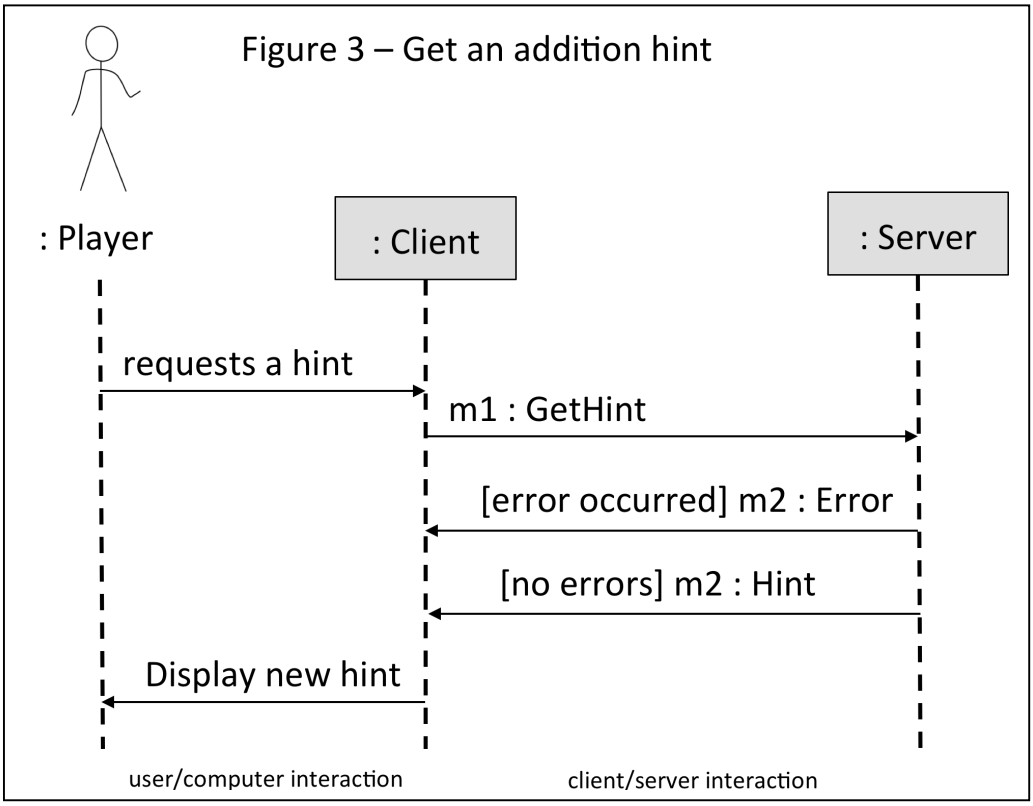


Figure 3 – Get an addition hint
: Player
: Client
: Server
requests a hint
m1 : GetHint
[error occurred] m2 : Error
[no errors] m2 : Hint
Display new hint
user/computer interaction
client/server interaction

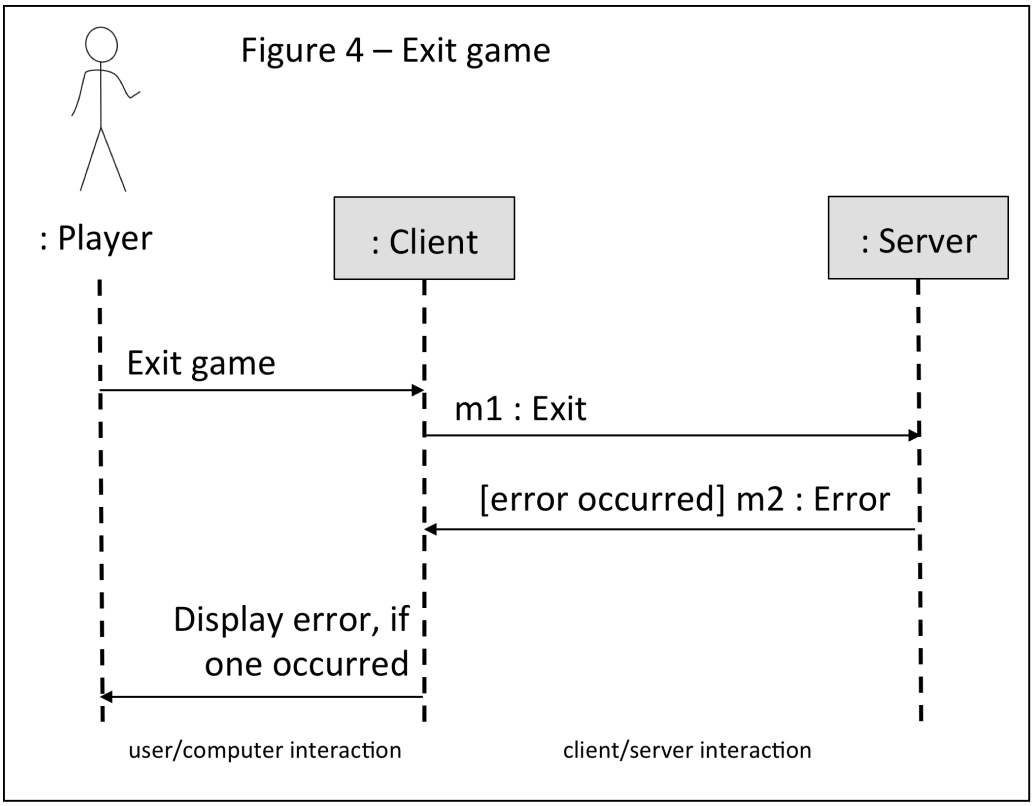


Table 1 – Message Types

Type	Usage and Sample Message	Format (Unicode Character Sequence)
NewGame	Use in Protocol #1. 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:A1234567,Jones,Joe*	newgame:<*A#*>,<last name>,<first name> where *A#* is your A# *last name* is your last last name *first name* is your last first name
GameDef	Used in Protocol #1. The server sends a *GameDef* message to the client in response to a *NewGame* message. Sample: *def:24,_b_l__,lack of ability to make decisions*	def:<*game id*>, <*hint*>, <*definition*> where *game id* is a numeric string that represents the game's identifier *hint* is a string of letters and underscores that represents the initial hint for the game's

		word. *definition* is string that represents the definition of the game's word. Note that is are commas between the game id, hint, and definition.
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 *GameDef* message. In response to a *Guess* message, the server will send back an *Answer* message. Sample: *guess:24,abulia*	guess:<*game id*>, <*word*> where *game id* is numeric string that represents the game's identifier *word* is string that contains the guess the user entered
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*	answer:<*game id*>, <*result*>, <*score*> where *game id* is numeric string *result* is a T or F *score* is an numeric string that represent that player's score if the result is "T" or the number of corrected letters if the result is "F"
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: *gethint:24*	gethint:<*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: *hint:24,_buli_*	hint:<*game id*>, <*hint*> where *game id* is numeric string *hint* is a string of letters and underscores that represents a hint for the game's word

Exit	If the player wants to exit the game before discovering the correct, the client can send an *Exit* message to the server. Sample: *exit:24*	exit:<*game id*> where *game id* is numeric 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*> *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 Canvas system.

Grading Criteria

Basic Criteria (worth up to 70 points)	**Max Points**
Correct functioning of Protocols 1, 2 and 4	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 Protocol 3	10
Ability for the user to change the server's address and port before any game. If the user changes the server's address and port while a game is still in progress, the game is forfeited.	5
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	20
Detection and handling of communication failure	20