CSCI 4220 Assignment 2

Word Guess Game

Due Date: Friday, October 18, 11:59:59 PM

This is a team-based (max. 2) assignment.

For this assignment, we will be building a server we can `netcat` into to play a simple word guessing game.

Server Details

The server should support up to 5 clients, and cannot not use `fork()` or threads. Since you will potentially have multiple clients, you will need to make use of the `select()` call.

The program will take four arguments: `./word_guess.out [seed] [port] [dictionary_file] [longest_word_length]`

It should use TCP and listen on the port given as the second arguemnt `[port]`.

When the server starts, and whenever a game finishes, it should select a word randomly from a dictionary, which will be referred to as the `secret word`. The filename of the dictionary, which just has one word per line, is passed in as the third argument to the program (`[dictionary_file]`), and the length of the longest word (number of letters) is passed in as the fourth argument (`[longest_word_length]`). Word lengths will not exceed 1024 bytes, and will not exceed `[longest_word_length]`. An example dictionary can be found here. The example dictionary does use ISO-8859-1 encoding instead of UTF-8, so don't be surprised if you see strange characters in your terminal when using this dictionary. The autograder input will only use A-Z and a-z, you do not need to write code in your solution to handle any character encoding issues. Usernames, guess words, and secret words are **not** case sensitive, "Bob", "bOB", and "bob" should all be treated as the same thing.

To make the grading deterministic, you should read in the entire dictionary once, sort the dictionary (C users may want to look at `qsort()`, C++ users may want to look at `std::sort()`), and then immediately after sorting, use `srand()` with the `[seed]` provided as the first argument to the program. Secret words should then be selected by using `rand() % dictionary_size`.

Usernames

When a client joins, the server should send a message asking the client to select a username, which will be used to uniquely identify the player among all currently connected players: `Welcome to Guess the Word, please enter your username.`

If a client disconnects, its username is no longer reserved. For example client 1 could be the first to connect and claim the username `bob`: `Let's start playing, bob`

If client 2 then connected and requested `bob`, the server would reject the username by asking for the username a second time: `Username bob is already taken, please enter a different username`

If client 1 then disconnected and client 3 connected and claimed the username `bob`, the game would continue with client 3 being `bob`. `Let's start playing, bob`

(continued on next page)

Gameplay

(Updated 10/6) Once a username is selected, the server should then notify that user about how many players are currently playing **(including the new user)**, and what the length of the secret word is: `There are 3 player(s) playing. The secret word is 5 letter(s).`

Any user can send as many guesses as they want, there is not a concept of "taking turns" in this game. However, each message should contain one word of the same length as the secret word, followed by a newline. Whenever the server receives a word from a user (called a `guess word`), it should send a message to **all** connected users in the format: `Z guessed G: X letter(s) were correct and Y letter(s) were correctly placed.`

X counts duplicate letters as a separate letters, so if the secret word is `GUESS` and `bob` sent a guess word of `SNIPE`, the message would be:
`bob guessed SNIPE: 2 letter(s) were correct and 0 letter(s) were correctly placed.`
(Once for one S and once for the E.)

Similarly, if the sceret word is `GUESS` and the guess word is `CROSS`, the response would be:
`bob guessed CROSS: 2 letter(s) were correct and 2 letter(s) were correctly placed.`
(Once for each S in the secret word.)

Finally, if the secret word is `GRUEL` and bob guesses `spill`, the response would be:
`bob guessed spill: 1 letter(s) were correct and 1 letter(s) were correctly placed.`
(Only one L counts since there is only one L in the secret word.) Note that game is **not** case-sensitive.

(Updated 10/6) If a user correctly guesses the secret word, all connected users should receive the message `Z has correctly guessed the word S`, and then all users should be disconnected from the server. The server should continue to run and should select a new word. Z is the username of the user who correctly guessed the word, and S is the secret word. If a user sends a guess word that is not the correct length, the server should send an error message to **only** the user with the invalid guess, but should not disconnect that client: `Invalid guess length. The secret word is 5 letter(s).`

Submission

You should handle a client quitting at any point during the game, as well as any other error cases you think of. You can provide any decisions or other comments for the graders to read in a **README.txt** file. Like Assignment 1, you should submit a **Makefile** that produces **word_guess.out** and use C or C++. Also like the previous assignment, we will provide libunp.a, unp.h, and config.h in the same directory as your files are built, and to ensure compilation if using these features you should have libunp.a in your compile line and should use clang instead of gcc.