CSCI 4210 — Operating Systems
Homework 4 (document version 1.1)
Network Programming

- This homework is due in Submitty by 11:59PM EST on Wednesday, August 10th, 2022
- You can use at most three late days on this assignment
- This homework is to be done individually, so **do not share your code with anyone else**
- You **must** use C for this assignment, and all submitted code **must** successfully compile via `gcc` with no warning messages when the `-Wall` (i.e., warn all) compiler option is used; we will also use `-Werror`, which will treat all warnings as critical errors
- All submitted code **must** successfully compile and run on Submitty, which uses Ubuntu v20.04.4 LTS and `gcc` version 9.4.0 (`Ubuntu 9.4.0-1ubuntu1~20.04.01`)

Hints and reminders

To succeed in this course, do **not** rely on program output to show whether your code is correct. Consistently allocate exactly the number of bytes you need regardless of whether you use static or dynamic memory allocation. Further, deallocate dynamically allocated memory via `free()` at the earliest possible point in your code. of `valgrind` to check for errors with dynamic memory Also close any open file descriptors or `FILE` pointers as soon as you are done using them.

Another key to success in this course is to always read (and re-read!) the corresponding `man` pages for library functions, system calls, etc.

Homework specifications

In this fourth and final assignment, you will use C library to implement a single-threaded single-process server to run a simple multiplayer "Guess the Word" game supporting up to 5 players at once. We will focus on only writing the server, since `netcat` can be used to act as a client.

The server should support up to 5 clients, and cannot use `fork()` or threads. Since you will potentially have multiple clients, you will need to make use of the `select()` call. You should handle a client quitting at any point during the game, as well as any other error cases you think of.

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]. 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, preserving the ordering, and then immediately after the initial read, use `srand()` with the `[seed]` provided as the first argument to the program. Do **NOT** sort the dictionary. 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`

Gameplay

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.

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 Instructions

To submit your assignment (and also perform final testing of your code), please use Submitty.

Note that this assignment will be available on Submitty a minimum of three days before the due date. Please do not ask when Submitty will be available, as you should first perform adequate testing on your own Ubuntu platform.

That said, to make sure that your program does execute properly everywhere, including Submitty, use the techniques below.

First, make use of the `DEBUG_MODE` technique to make sure that Submitty does not execute any debugging code. Here is an example:

```
#ifdef DEBUG_MODE
    printf( "the value of q is %d\n", q );
    printf( "here12\n" );
    printf( "why is my program crashing here?!\n" );
    printf( "aaaaaaaaaaaaaggggggggghhhh!\n" );
#endif
```

And to compile this code in "debug" mode, use the `-D` flag as follows:

```
bash$ gcc -Wall -Werror -g -D DEBUG_MODE *.c -pthread
```

Second, output to standard output (`stdout`) is buffered. To disable buffered output for grading on Submitty, use `setvbuf()` as follows:

```
setvbuf( stdout, NULL, _IONBF, 0 );
```

You would not generally do this in practice, as this can substantially slow down your program, but to ensure good results on Submitty, this is a good technique to use.