CS 176A: Homework #5

Part 1: Problems: (30 points total)

Complete the problems on the attached handout. *Answer each question (not sub-question) on a fresh page. Convert your document into a PDF and upload on Gradescope under HW5. When you do this identify where each question AND sub-question is (follow the Gradescope prompts on this).*

Part 2: Programming Assignment (70 points total)

Complete the Programming Assignment on the attached handout. Turn in your code using the "Homework 5 Programming Assignment" Gradescope link. PLEASE pay attention to the turn-in requirements at the end of the assignment. Also, all code must be operational on **Gradescope**. Please ensure that you pass all the tests on **Gradescope** before turning in the assignment.

Programming Assignment: Hangman Game

The aim of this project is to have you write a game-server and a player-client for a simple hangman game. You have to develop a game-server and a client (player) that establishes a connection with the game-server and plays a game of hangman. The client will be much more simple than the server. You are allowed to work with ONE partner on this assignment. You should write your code in C, and ensure it compiles with the Makefile you turn in, and passes all tests on Gradescope. If it works on your personal machine and not on Gradescope, you will not receive credit for the program. Make sure you read and follow closely the turn-in instructions at the end. If you have a partner, you must turn in a README file that indicates the name of both partners.

Server - Hangman Game Server

- The server should be able to manage 3 clients (players) at any given time.
- Each client will be playing their own separate game.
- The server should handle all administrative duties of each client's game such as choosing the word and keeping track of the score.
- The server will tell the client if the guess is correct or not and keep track of the number of guesses the client has made.
- Loop listening for connections
    - When the server starts, a connection handler starts listening for incoming game request connections from the client.
    - Server keeps listening during and after play.
- On successful connection with a client the game-server should choose a game word and wait for the packet sent from the client signaling the game should start.
- It should then respond using a game control packet (seen below), the client will add the formatting and display the following to the player:
    - For the game word "hangman"
      >>> _ _ _ _ _ _ _
      >>>Incorrect Guesses:
- After a client guesses a letter the server replies back with the updated word, filling in correct letters and supplying incorrect guesses.
    - Ex. Client has guessed "n". Server responds with:
      >>> _ _ n _ _ _ n
      >>>Incorrect Guesses:
    - Client then guesses "p". Server responds with:
      >>> _ _ n _ _ _ n
      >>>Incorrect Guesses: P
- When the game is over tell the client if they won or lost then terminate connection to that client.
    - Game ends when the client guesses 6 incorrect letters or if all correct letters for the word have been guessed.
- The server should read an input file called hangman_words.txt. This file should have between zero and 15 words, one word per line. The server should choose a word from this file at random for each game.
    - The words must range in size from a minimum length of 3 letters to a max length of 8.
    - You must choose words of at least three different lengths.
        - Ex. 5 x 4-letter words, 5 x 5-letter words, 5 x 8-letter words

    - Each word must have an equal chance of being chosen by your server for a new game. Your server does not need to remember which words it has chosen previously.
- If a fourth client (player) tries to connect to the game-server, the server should send a "*server-overloaded* " message then gracefully end the connection.
- After ending a game with a client the server should be able to accept a new connection from a clients until limit (3) is reached again and so on.

Client - Player

- Use server IP and Port to connect to a game server
- Once connected ask user if they are ready to start the game as follows:
  >>> Ready to start game? (y/n):
  - If the user says no terminate client otherwise send an empty message to the server with msg length = 0. (more on this below) signaling game start.
- The server will then respond with its first game control packet which the client should properly print out.
- The client will then ask the player for a guess exactly as follows:
  >>>Letter to guess:
- Client then sends a guess to the server. Make sure that the user guess is formatted correctly.
    - If the user guesses something invalid like a number or more than one letter please respond with an error then ask for input again as follows:

  >>>Letter to guess: 5
  >>>Error! Please guess one letter.
  >>>Letter to guess: aa
  >>>Error! Please guess one letter.
  >>>Letter to guess:
- Player keeps guessing letters while the server responds with updated game information.
- Continue until the game word is either correctly guessed or the guess limit is exceeded.
- Client should terminate when the game is over.
- Client should be able to accept lower or uppercase letters as a guess but must make sure to convert to lowercase before sending to the server.

Server Message header

When sending messages to a client the server must use the following format.

Msg flag	Word length	Num Incorrect	Data

Msg flag (1 byte) is used when the server would like to send a string message to a client. If the server is sending a message to the client the Msg flag will be set to the length of the message otherwise Msg flag will be zero. If the message flag is set then there are no Word length or Num Incorrect fields the entire packet minus the message flag is message data. When a client receives a message from the server it should directly print it out. The client should print messages out in the order they are received.

-An example of a message could be "Game Over!", "You Win!", "You Lose :("

If the message flag is zero then we have a normal game control packet.

Word length (1 byte) is the length of the word the user is trying to guess. The first n bytes in the data segment of the packet where n is the value of Word Length will be used to store the positions of the letters that have been guessed correctly so far.

Num Incorrect (1 byte) is the number of incorrect guesses a client has made. The incorrect guessed letters will be after the game word in the data segment.

Two example data packets can be seen below. In the example each box is one byte. This packet would be a game control packet with the game word being "green" and the user having guessed g, and e correctly. The user has guessed a, b, and c incorrectly.

Msg flag	Word length	Num Incorrect	Data [1]	Data [2]	Data [3]	Data [4]	Data [5]	Data [6]	Data [7]	Data [8]
0	5	3	g	_	e	e	_	a	b	c

This packet is an example of a message packet. When the client receives a message packet it should directly print out the contents.

Msg flag	Data [1]	Data [2]	Data [3]	Data [4]	Data [5]	Data [6]	Data [7]	Data [8]
8	"Y"	"o"	"u"	" "	"W"	"i"	"n"	"!"

Client Message Header

This defines the message the client will send to the server.

Msg length	Data

An example packet could be the following when a client is guessing "a".

Msg length	Data
1	"a"

Example Program flow

The following is an example game output from a client connecting to a game server at the address 0.0.0.0 and on port 8080. The server chose the game word "dog".

```
>>> ./hangman_client 0.0.0.0 8080
>>> Ready to start game? (y/n): y      ← Client asks if player is ready to start and sends empty
                                          message ("0") to server signaling game start
>>>_ _ _                               ← Server responds with first game control packet
>>>Incorrect Guesses:
>>>                                    ← Print new line after Incorrect Guesses for readability
>>>Letter to guess: a                  ← Client asks for input and user types "a"
>>>_ _ _                               ← Server responds
```

>>>Incorrect Guesses: a
>>>
>>>Letter to guess: d ← Client asks for input and user types “d”
>>>d _ _ ← Server responds
>>>Incorrect Guesses: a
>>>
>>>Letter to guess: g ← Client asks for input and user types “g”
>>>d _ g ← Server responds
>>>Incorrect Guesses: a
>>>
>>>Letter to guess: o ← Client asks for input and user types “o”
>>>The word was d o g ← Server responds
>>>You Win!
>>>Game Over!

If the player instead guesses 6 incorrect letters and loses the game, replace “You Win!” with “You Lose.”

TURN-IN REQUIREMENTS

You will hand in the client and server programs along with the Makefile to compile them. Name the client file hangman_client.c and the server file hangman_server.c. Make sure to indicate both you and your partner’s name in Gradescope when submitting your code. Only ONE of you needs to turn in the programming assignment.

Include a Makefile. The “make all” command should produce two executables named hangman_client and hangman_server.

Finally, turn in these files to the “Homework 5 Programming Assignment” Gradescope link. Be sure to test your code on Gradescope!