CS 4323 Design and Implementation of Operating Systems I

Mini Group Project: Full Marks 100
(Due Date: 02/27/2022, 11:59 PM CT)

This is the mini project assignment that must be done in the allocated group and using C programming language. If you do not know the group members, then please refer the module **MiniGroupProject** under **Home** page in Canvas.

Students are free to make assumptions and have their own creativity on the project if they do not conflict with the requirements specified in the assignment. Please make sure to watch the video explanation of the assignment to understand the task better.

This project will familiarize you with Unix socket programming using TCP/IP for a simple client-server interaction and inter-process communication (IPC) using POSIX message passing.

Problem description:

You are required to develop two-players word game in a client-server environment using TCP connection.

An N number of players (aka clients) can log on to the server and can start playing the game in either single player mode or a multi-player mode (a game is played between two players). So, as soon as any client logs onto the server, he/she should get the main menu options (as shown below) from the server:

1. Single Player Mode
2. Multi-Players Mode
3. Exit

Note: N is the number of players the server can support. Your program needs to support only 3 players.

A. Single Player Mode:

If a player selects option 1 in the main menu options, then the player can start the game (to be described later) in a single player mode. In a single player mode, the game is played against a server. Only if the player is declared as a winner, his/her score may get recorded in the scoreboard (stored in a file, singlePlayer.txt), depending on his/her position in the scoreboard. The scoreboard only lists a maximum of five players with the highest score. So, if the player's score is among the top five, then his/her score needs to be included in the scoreboard and only in this case, the player is asked to enter his/her *first name*, *last name* and *country*.

The format of the scoreboard (read from the file singlePlayer.txt) looks something like this:

```
 First name    Last name     Country   Score  Number of words found  Number of words added in the dictionary
---------------------------------------------------------------------------------------------------------
   Zhao           Bao         China     34              7                            2
  Nakano        Tsutomu       Japan     32              8                            1
  Felicjan        Wos         Poland    31              9                            1
   Kaspar     Zimmermann     Germany    30             11                            0
   Karina       Maynard        USA      30             10                            0
```

Note: The scores in the scoreboard is arranged in the descending order. Pay special attention to the formatting.

The meaning of each column is as follows:

- *First name*: first name of the player
- *Last name*: surname of the player
- *Country*: country of the player
- *Score*: final score of the player
- *Number of words found*: number of words the player found in the game
- *Number of words added in the dictionary*: number of new words player added to the input file of the game (stored as input_xx.txt, described later under the game description).

Note: xx in input_xx.txt represents two-digit numbers.

Irrespective of whether the player is in the scoreboard or not, the single player scoreboard (i.e. from the file singlePlayer.txt) needs to be displayed at the end of the game.

Once the scoreboard is displayed and after pressing "Enter" key (from the keyboard), the player will return to the main menu options and the process can repeat.

B. Multi-Player Mode:

If a player selects option 2 in the main menu options, then the player enters in a multi-player mode and the game (to be described later) is played between two players. If more than two players want to play in a multi-player mode, then the server should randomly select players in a pair and the game will be played between them. The player who did not get a partner should wait for the partner for 2 minutes. Even after 2 minutes, if no partner is available, then the server needs to give an option to this waiting player to play in the single-player mode or continue to wait for a partner.

Once the players have been chosen for the game, each player is asked to enter his/her *first name*, *last name* and *country*. Each player is given a maximum of 2 minutes of time to enter each piece of this information. If there is a 3rd player waiting to play the game in a multiplayer mode and if one of these selected two players (selected to play the game in multiplayer mode) does not enter his/her relevant information within this specified time, then he/she will be removed from the game and a chance will be given to the waiting player. Only once both the players have entered all the relevant information, the game will start.

If the score of the winning player is higher than the current score in the scoreboard, then his/her score need to be included in the multi-players scoreboard (i.e. the file multiPlayer.txt). Also, if the score of the losing player is higher than the current score in the scoreboard, then he is also eligible to be include in the scoreboard. But to identify the winner vs loser, you would need to include an additional column in the scoreboard i.e. win/lose to distinguish who won the match. Like the scoreboard in the single-player mode, the scoreboard in the multi-player mode can only list a maximum of five players with the highest score.

First name	Last name	Country	Score	Win/lose	Number of words found	Number of words added in the dictionary
Zhao	Bao	China	34	Win	7	2
Nakano	Tsutomu	Japan	32	Win	8	1
Felicjan	Wos	Poland	31	Lose	9	1
Kaspar	Zimmermann	Germany	30	Win	11	0
Karina	Maynard	USA	30	Win	10	0

Whether or not the scoreboard is updated, both the players are shown the final scoreboard (i.e. the file multiPlayer.txt).

After the scoreboard is displayed and after pressing "Enter" key (from the keyboard), both players will return to the main menu options and the process can repeat.

C. Exit:

If the player selects option 3 in the main menu options, then the player will disconnect from the server. The socket should be properly closed while exiting.

Game Description:

The game starts with the display of a set of random alphabets, obtained from the input .txt file. The task is that the player needs to find the valid English word using only those letters specified in the given set. There are some rules:

- Each player takes turn to come up with the words.
- Player need to come up with a single word at a time that begins with the letter or letters that the previous word ended with
    - For example: so<u>up</u> – up<u>set</u> – <u>set</u> – etceter<u>a</u> – apterous
- Only words available in dictionary.txt are considered valid English words.
- The first player to start the game is randomly selected by the server. Also, the server will randomly select the starting alphabet.
- For every valid word that have appeared for the first time in the game will get points (refer to *scoring* section below)
- If a player cannot come up with a word or there is no valid word meeting the requirements, then the player can pass his turn. If the opposite player also cannot come up with a word, then he/she can also pass. When both players have passed their turn without forming a word, then the player in the turn can select any alphabet from the given set to form a word.
- The game ends when both the players pass their turns without forming a words two times consequently.

Scoring:

- For words that are already in the input_xx.txt, the scoring is as follows:

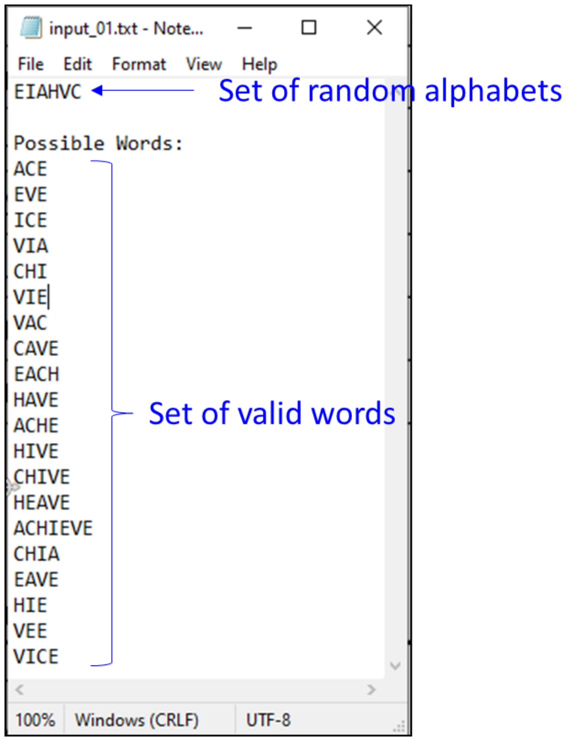
Word length	Points
3 – 4	1
5	2
6	3
7	5
8+	11

- For any new valid word (that is not already in the input_xx.txt), the players will get a bonus of 5 points.
- For any invalid word, the player is penalized with 1 point.
- Words that has already appeared in the game cannot be used again. The player who sends such words will be penalized with 2 points.

How is the game played?

There are several input .txt files: input_01.txt, input_02.txt, and so on given along the assignment.

One such input .txt file is shown below:



```
EIAHVC

Possible Words:
ACE
EVE
ICE
VIA
CHI
VIE
VAC
CAVE
EACH
HAVE
ACHE
HIVE
CHIVE
HEAVE
ACHIEVE
CHIA
EAVE
HIE
VEE
VICE
```

Description of the input .txt file is as follows:

- The first line of all the input .txt files always contains a set of random alphabets.
- Also, there are some valid words included in the input .txt files.

When the game starts, the server needs to randomly select:

- one of the input .txt file,
- the player to start the game, and

Once the input .txt file and the 1st player to start the game are selected, the server displays the 1st line of the .txt file as the set of random alphabets only to the first player. The server then randomly selects a character from this set of random alphabets and ask the first player to come up with the word starting with that random character. When the first player comes up with the word, he/she sends the word to the server. Once the server accepts the word is valid, the score is updated and both players' score is sent to the first player (second player score is zero, at this point). Only when the first layer turn is complete, the second player is shown the set of random alphabets, and then the game continues as per the rules discussed above.

When it is the player's turn, the server should send that player following piece of information:

- the set of random alphabets
- the player's current score,
- the opponent player's current score,
- words that have been used in the game so far, and
- the starting character of the word that the player needs to form the word with or the last word another player has entered.

When any player sends the word to the server, the server does the following checking:

- checks the received word with the words in the dictionary.txt to confirm that it is a valid word,
- checks that the word has not already been used in the game, and
- checks the word with the words in the selected input .txt file to assign proper score.

If the server finds the word is not valid, then the corresponding player is informed that it is an invalid word. The server needs to ask the corresponding user to enter the valid word again and subsequently penalized for invalid word(s).

If the server finds the word has already been used in the game, then the corresponding player is informed that it has been used already. The server needs to ask the corresponding user to enter another valid word and subsequently penalized for sending the previously used word(s).

Also, each player is assigned a total of 4 minutes to enter any word (either valid, invalid or previously used word). The timer is reset as soon as the player send the word to the server and the countdown begin again. If the player does not send the word within these 4 minutes, then the server will consider the player has passed his/her turn. In a single turn, if a player resets the timer for 3 times, then also the server will consider the player has passed his/her turn.

If you look at the input .txt files, then you can see there are already some valid words listed under *possible answer*. When any player finds a new valid word for the game, this valid word needs to be added in the respective input .txt file.

For a single-player mode, to allow the player to compete and win against the server, we will only allow the server to refer to the words present in the input .txt file to come up with the words. Server will not refer to the dictionary.txt to come up with a word.

Programming requirements:

Your program should illustrate the usage of following concepts:

1) The server-side program should use fork to serve the clients.
2) A separate forked process in the server should do all the checking of the word in the dictionary.txt, writing of the new word in the respective input .txt file, and the scoreboard files.
3) Any form of communication between the processes at the server side need to use POSIX message passing.
4) The client-side program does not store any information. All the information needs to be passed from the server. So, the client program is just the program to interact between the server and the player, by accepting data from the server (or player) and passing to the player (or server).
5) The scoreboard (singlePlayer.txt, multiPlayer.txt) should only have a list of top players.

Grading Rubrics:

The grading will be as follows:

• Correct display of the main menu options	[2 Points]
• Proper display of the set of random alphabets, players' score, words used so far in the game and the starting character that the player need to use to come up with the word	[3 Points]
• Correct synchronization of the game i.e. game is played in turns	[4 Points]
• Correct computation of scores in the game along with penalties	[4 Points]
• Correct working of the scoreboard (with proper formatting) for single player mode	[3 Points]
• Correct working of the scoreboard (with proper formatting) for multi-player mode	[3 Points]
• Correct termination of the game	[3 Points]
• Usage of TCP protocol for client-server communication	[10 points]
• Correct working of single player mode ○ Correct use of fork to handle multiple clients in a single player mode	[10 Points]
• Correct working of multi-player mode ○ Correct use of fork to handle multiple clients in a multi-player mode	[10 Points]
• Proper working of 4 minutes maximum time to each player on each turn	[3 Points]
• Implementation of a separate forked process to do the word checking in the dictionary.txt, update of respective input .txt file, and the scoreboard files.	[8 Points]
• Proper use of POSIX message passing to communicate among the processes at the server side	[35 Points]
• Proper implementation of exit option, with no zombie process left in the server	[2 Points]

Note:

- Failure to follow standard programming practices will lead to points deduction, even if the program is running correctly. Some of the common places where you could lose points are:
    - Program not compiling successfully: -20 points
        - The TA will run the program in CSX machine.
    - No comments on code (excluding function): -5 points
    - No comments on function: -5 points
    - Not writing meaningful identifiers: -5 points
    - Failure to submit files as specified: -10 points
    - Not enough or appropriate message displayed for user interaction: -5 points

Grading Criteria:

This is a group project. So, grading will focus on students' ability to work in the group and successfully completing the work. Each member in the group should coordinate as a team rather than individual and that is the key to scoring maximum points.

Points to remember:

- Failure to complete the project as a group will deduct 20 points, even though every individual has completed their part. As this is a group project and students are expected to learn to work in the group and meet the team's objective, and not just individual objective(s).

- It is not necessary that all group members get equal points. It depends on what responsibility each student has taken and whether the student has delivered the task.

- Work need to be distributed uniformly among all group member and should be clearly specified in the report.
  - Be sure to submit a progress report as specified in the progress report file.

- It is the group's responsibility to alert the instructor with any issue that is happening in the group.
  - Students are supposed to use the group created in the Canvas for all correspondence. That will serve as the proof, in case there is any dispute among the group members.
  - Time is very important. Group members are expected to respond quickly.
  - You need to have sufficient days to combine individual work. You are going to make one submission as a group and not individually.

- If any of the group member have plagiarized the code, then the complete project will be considered plagiarized, and all members will be awarded zero. So, each group member needs to be very alert of plagiarism issue, not only of their code but fellow member's code. Any plagiarism issue should be brought to the instructor's attention as soon as possible. <u>University and department have strict policy on plagiarism and will be strictly followed.</u>

Submission Guidelines:

- Progress report needs to be submitted by the progress due date/end date. The due date and the end date are same for the progress report.
  - Please refer to the progress report file for further instruction

- Final project submission instruction:
  - Attach a final UML diagram in pdf format. If nothing has changed, then it could be same as that you have submitted in the progress report. Otherwise, you need to make a new UML diagram showing the changes.
  - Include a single readMe.txt file that shows how to run your program.

    - Each student’s work needs to be in a separate .c file.
        - Each student needs to write the code in their own file. That will let the TA see each individual group member’s work.
        - All students code should be compiled in the respective server and client programs. TA will not run each individual student’s work separately.
    - Each .c file should include the header information, which should include:
        - Group Number
        - Group member name
        - Email
        - Date
    - There should be sufficient comments in the program.
        - Each function should be clearly described what it does along with the input arguments and return values.

    - All the codes should be submitted in both .pdf format (copy paste in textual form, no screenshot) as well as in .c files (including any .h file).

- Final project will be evaluated together with the progress report. So, they should have a correlation. Commitments on the progress report should be reflected on the final submissions.

- Remember, since the TA and the instructor will have no way of knowing the project activities, we will consider what is presented in the submitted file as the final contribution. Based on that submission, we will be grading. So, if you fail to provide sufficient information, it will be difficult for us to grade fairly. It is the group responsibility to provide as much information as possible in the final submission.