TYPERACER – MULTIPLAYER GAME

Evaluation Component – 3 Report

JUNAIDUL ISLAM BHAT	**2014A7PS007P**
BHUVNESH JAIN	**2014A7PS028P**
HARSHID WASEKAR	**2014A7PS078P**
SHUBHAM NAGARIA	**2014A7PS143P**

PROJECT GROUP: 19

Prepared in partial fulfillment of the course:
CS F303 (COMPUTER NETWORKS)

Submitted to Prof. Rahul Banerjee (Dept. of Computer Science BITS Pilani)
On 25th April, 2017

GITHUB LINK : https://github.com/likecs/Typeracer-CSF303-MiniProject

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI, PILANI CAMPUS

Table of Contents

INTRODUCTION

1.1 Problem Statement

The project involves implementation of a 3-level multiplayer networking game, TypeRacer. The game has a common whiteboard for all the users and leaderboard for displaying the game-screen and the scores. The game should be able to concurrently handle all the requests for its users and update the scores (based on number of characters typed in the current game) accordingly. The levels of the game are defined in terms of the dictionary being chosen for the playing the game. 6 different types of dictionaries, English, French, Italian etc. are provided with the game.

1.2 Scope of work defined (Partner-wise scope of work)

TypeRacer is developed in partial fulfilment of the Mini Project component of CS F303 Computer Networks course. The primary scope of this project is to display the networking concepts learnt during this course and apply them in the implementation of the game. The project also includes testing your typing skills in comparison to others for datasets in different languages.

The whole project was divided into a total of 8 modules, 2 of which were handled by each member of the team. Overall, the code was written by the team as a whole, but the leader of the module took the responsibility of checking that everything is according to the norms and all cases are covered. The following table below shows the different modules and their respective leaders.

S.no	Work Done	Team Leader
1	Menu Model (Terminal based menu for game)	Harshid Wasekar
2	List of Dictionaries (Database of words for the game, each used as different level of the game)	Junaidul Islam Bhat
3	Network – Server logic (Server functionalities for the game)	Bhuvnesh Jain
4	Network – Client logic (Client functionalities for the game)	Shubham Nagaria
5	Graphics (Usage in menu, whiteboard and leaderboard of the game)	Junaidul Islam Bhat
6	Game Logic (Efficient Algorithms for string insertion, deletion and querying in set)	Harshid Wasekar
7	Integration (Combining functionalities of Server, Client and Game logic into complete project module)	Shubham Nagaria
8	Failure & Recovery Handling (Considering cases of networking failures, clients/server being disconnected etc.)	Bhuvnesh Jain

OVERALL DESIGN DETAILS

2.1 Working of game

TypeRacer is a multilevel multiplayer networking game involving a common whiteboard for all the clients which is updated by the central server for any change encountered. The game involves testing the typing speed of its clients in a creative way where the user needs to quickly and correctly type the words appearing on the screen. Points are awarded to the first user typing the word correctly, after which the word disappears from the screen. The word appears on the whiteboard only for a specific time interval. There are 6 levels in the game, depending on the type of dataset chosen for playing the game. The winner is decided on the basis of characters per second written by him.

The user interface just consists of the terminal in the operating system. The game has minimal GUI so as to support fast exchange of the data over the network between the server and its client. The whiteboard where the words appear moving from left to right, is common to all the clients and is handled by the server. Additionally, “Leaderboard”, showing the current detailed score of the user, including the user’s “characters per second” and accuracy level. Also, the leaderboard shows the scores of all the users connected in the game as well.

2.2 Design Challenges encountered

1. Finding an efficient scoring logic for the game so that it is fair for all and also doesn’t depend much on network parameters.
2. Usage of “curses” library forced us to give some character input to the screen at all moments of time to show the screen contents continuously.
3. Selection among select/processes/pthreads for implementation of concurrency between server and client.
4. Debugging of code in network environment.

2.3 Limitations of work done

1. An efficient method to ensure that network lag doesn’t affect the performance of the players troubled us for quite some time. Finding an efficient scoring for making the game fairer could be only implemented, while issues related to networking could not be implemented effectively as desired.
2. A more robust GUI for the game and the levels could have made the experience better for the users.

2.4 Architectural & Implementation Details

2.4.1. Game Logic implementation (Trie)

For implementation of the game logic, we used the data structure **Trie**. Trie also known as **prefix tree** is used to store the words in our dictionary efficiently. The data structure was chosen for efficient insertion and query operation related with it. Searching for a word to see if it exists in our dictionary would take time of the order of sum of strings lengths in our dictionary (in the worst case) for each query. This is quite inefficient as the typing speed of the user will be around 60 words per seconds (on average). But with the use of Trie, the words in the dictionary will be inserted in it beforehand with time complexity of the order of sum of length of strings in dictionary. Now, checking whether the word belongs to the set or not can be done in length of input string, independent of the number of words in the dictionary which is the best we can do with any data structure as linear time is required to just read the word itself.

Optimizing the query operation can also be done using hash table. It was not chosen of the following reasons:

1. Using closed addressing with collision resolution can still lead to large linked list aggregating at some points which may still take large time for search in some cases if the hash function is not chosen properly.
2. Also, it the hash function is not chosen properly, (for example polynomial or rolling hashing generally used for string based problems) can give the same hash value for multiple strings which is not good in our case as words need to be checked for exact matching in the dictionary.

One more useful property of Tries is the amount of memory required to store all the words in the dictionary. It is linear in the terms of the number of elements in the dictionary.

Below is the pseudo code used in the data structure **Trie.**

```
insert (root, s):
  for i from 1 to s.length():
          if (root->s[i] = NULL):
                  create_node()
          root = root->s[i]
  root->leaf = true

search (root, s):
  for i from 1 to s.length():
          if (root->s[i] = NULL):
                  return false
          root = root->s[i]
  return (root != NULL && root->leaf)
```

2.4.2 Menu Model & Graphics

All the graphics, including the menu options, whiteboard & leaderboard were made using the "curses" library in C. Terminal-based graphics were chosen for the game. The menu options for the game had the following options:

1. Entering the name of the users playing the game.
2. Selecting the option of acting as the server/client for the game.
3. If the user acts as the server, then he has the option of selecting the level of the game i.e the dictionary used for playing the game like English, French, Italian etc.
4. If the user acts as the client, then he has the option of selecting the server by writing its IP address in the option provided.

The whiteboard of the game consists of 23 rows and 80 columns (1 cell size is same as defined internally by curses library). The first 22 lines of the whiteboard contains the various words currently on the screen, moving from left to right directions. These words get added on any random row from 1 to 22 which is found empty i.e. doesn't contain any word on it. The words disappear from the screen after it has either hit the rightmost column or the word is correctly typed by one of the players. When the word is found to be correctly typed by one of the players, then the rows is emptied and new words can start appearing on the row at later moment of time. The last row of the whiteboard consists of the user's typing area, the different parameters of the users typing speed, like accuracy, number of characters typed per second etc. and the timer of the game are displayed. The timer of the game is displayed in seconds. The usual timer of the game is 120 seconds and can be set by the server before the commencement of the game.

The leaderboard of the game is shown once the game finished. It includes the complete details of the users like his typing speed in terms of characters typed per second, number of correct characters typed per second, accuracy of user typing the characters etc. It also displays the overall standing of the game between the users on the basis of the total number of characters typed by the user. This parameter, instead of the number of correctly typed words was chosen, so that users may play the game according to their own strategy i.e. either go for typing more short length words or typing less words but of larger length. It will, on average, also help in reducing the congestion in the network as well.

2.4.3. Server and Client Implementation details & working

2.4.3.1. Brief Implementation details

The overall implementation of the server and clients were done using the networking system calls, process calls, pthreads library etc., available in Linux. Also link list data structure was used for maintaining the list of clients at any moment of time at the server side. The detailed explanation & working is given below in the following sections.

2.4.3.2. Connection Establishment

Initially, the server creates a socket and listens for incoming connections from multiple clients. In this project, we need to handle concurrent requests from all the clients so that no client gets priority over other clients. We have used **pthreads** to achieve this goal. We could have also used select/processes instead of pthreads. But creating a new process can be expensive. It takes time, resources & memory. (A call into the operating system is needed, and if the process creation triggers process rescheduling activity, the operating system's context-switching mechanism will become involved. It can also lead to entire process being replicated). Also, the code using "select" function in Linux was discarded as it used blocking methods which may lead to performance degradation of the game.

The advantage of using a thread group over using a process group is that context switching between threads is much faster than context switching between processes (context switching means that the system switches from running one thread or process, to running another thread or process). Also, communications between two threads is usually faster and easier to implement then communications between two processes. Also, using pthreads also gave us the benefit of using mutex locks over the common variables, like updating of score of a particular word only for a single user, maintain the linked list of clients at any particular moment of time etc. Also, all threads were given equal priority by default so that each client have equal chance of sending data to the server.

Whenever a new client tries to connect to the server, the server creates a new thread for each client connected. After the player connects to the server, the alias (name of player) used by him/her before start of game will be send to the server which will be used to uniquely identify that player during a game and for assigning score to the player in the leaderboard. After logging in, the player is presented with option of selecting the server by tying its IP address and Port number before joining the game.

2.4.3.3. Gameplay

After the server has logged in, he/she is given the chance to select a dictionary which will decide the level of the game. After this, the server is taken to waiting arena where he/she has to wait till atleast one more opponent joins the game. The maximum number of players in each game is restricted to 4 players, considering the complexity of thread handling, congestion at server side and maximum number of threads supported by the computers on which testing of the code was done.

Once the required number of clients enter the playing arena, the game and the countdown timer starts. Initially, the server selects some subset of words from the selected dictionary and sends them to each player which is then displayed in the player's whiteboard. The player then types one of the words visible to him in the common whiteboard.

As soon as, player hits enter/space after typing the word, a search is done locally to check if the word matches any of the words in the screen. If the word doesn't match any word, then it is ignored and the typing space is cleared. If the word matches some word, then the index of the word (row on which the word appears initially) is returned on search and that index is communicated to the server. The checking is done locally so as to reduce the load on the server and network. This also requires that clients when installing the game, have the dictionaries installed on his/her side. Also, as soon as client attempts connection with the server, the Trie for the corresponding dictionary is built locally which take time of the order of milliseconds.

Once the server receives the index, it does a quick check if the word still exists in the current list of words as it is possible that some other player has already typed the word and the server has deleted that word from the current list of words. This part has been implemented using pthreads and mutex locks. Also, care has been taken that no priority is given to any client and he game is fair to all. If the word at the index is still present, then the server removes the word from the list and also relays the index to all the other players and at the same time updating the scoreboard. For maintaining the score of each client, concepts & algorithms of concurrency were used to avoid race conditions and maintain fairness to the users sending the correctly typed word first.

The game stops when the timer hits zero and then all players are taken to the leaderboard where each player can see his and the opponent's score.

2.4.3.4. Extra Features

An additional feature of showing the available IP addresses (of different interfaces on the server side) was implemented so that the server doesn't need to open a new terminal window to see the IP addresses of various interfaces available at his side. (i.e. implementation of "ifconfig" command available in Linux terminals)

We have also given the provision of maintaining the server & client log details in a separate file, depending on the action taken and the time at which it was done. This not only gave us a clear picture of how the things in the project were going, but also helped us to find some corner cases / implementation flaws in our project as well.

MAJOR LEARNING / TAKEAWAY FROM PROJECT

1. Learning how to implement a networking game using the networking calls, process calls etc.
2. Maintaining list of clients using the pthreads & link list data structure on the server side.
3. Finding out ways to implement game logic in manner which can reduce the network congestion on average.
4. Implementation of graphics for the first time using the "curses" library.
5. Using theoretical reasons to argue about using different method to implement the algorithm, finding advantages and disadvantages of all of them and then selecting the most appropriate one depending the practical utility of each one of them.
6. Getting a hands-on experience on what is taught in class and what design issues one might face while implementing a networking project. This also gave us a better picture on how TCP works and information in packets are exchanged between server and clients.
7. Learning methods of doing crash handling and storage of server & clients logs.
8. Integration of smaller modules into a larger one and working together as a team with leaders doing their jobs effectively and efficiently.

CONCLUSION / FUTURE SCOPE

The project had basic networking features implemented, which strengthened the theoretical concepts as well. Extra features like more advanced error and crash handling like server shutting down, even though clients are still present, making the game more robust for environments containing clients with both fast and slow networks etc. could be implemented so that the overall experience of the game improves. Also, a GUI version of the game, instead of the terminal based one would also allow users who have no knowledge of even basic terminal commands could play and enjoy our game, Typeracer.