TYPERACER – MULTIPLAYER GAME

Evaluation Component – 2 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)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI

(April 07, 2017)

Table of Contents

INTRODUCTION

1.1 Problem Statement

The project involves implementation of a 3-level multiplayer networking game, TypeRacer. The game should have a common whiteboard for 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.

1.2 Status of Progress made

S.no	Work Done	Team Leader
1	Game Logic (Based on Trie)	Harshid Wasekar
2	List of Dictionaries (Database of words for the game)	Junaidul Islam Bhat
3	Network – Client logic (Client functionalities for the game)	Shubham Nagaria
4	Network – Server logic (Server functionalities for the game)	Bhuvnesh Jain

Some work related to Menu model & cases for failure handling has been taken into account. Most of the work in the above subparts of the project is completed, but the full part will be completed when integration is done completely.

1.3 Perceived Challenges

1. Selection of procedure used for connection of multiple clients with server. For example – use of pthreads v/s select.
2. Efficiency of using Trie v/s hash table for our datasets (dictionary of words).
3. Finding an efficient scoring logic for the game so that it is fair for all and also doesn't depend much on network parameters.

1.4 Future Goals

1. GUI part for the menu for the game and display of the whiteboard for users.
2. Implementation of failure and recovery models for the server-client interactions.
3. Final Integration of all modules.

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 on regular basis. 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 3 levels in the game which are played for a specific interval of time. The levels are distinguished by the rate which the words appear on the screen. After each level, the player with the lowest score gets eliminated while others advance to the next level. In case of ties in score, the winner is decided on the basis of characters per second written by him. To make the game more interesting, the server will chose among dictionaries of different languages present in the database before starting each level.

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 falling from top to bottom, is common to all the clients and is handled by the server. Additionally, “Leaderboard”, showing the current score of the user and its “characters per second” count is displayed for all the users.

2.2 Game Logic implementation details (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)
```

2.3 Network – Server & Client implementation details

2.3.1. 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 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 o entire process being replicated). Also, the code using "select" function in linux was discarding 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.

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, he/she has to login with an alias 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 an option to start a game or logging out of the game.

2.3.2. Gameplay

After the player has selected the “start a game” option, he/she will be asked to select a dictionary which will be used for playing, after which the player is taken to waiting arena where the player 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 and congestion at server side.
Once all the 4 players 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 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 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, ha 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.

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.

2.3.4. Scoreboard details

The scoring is done based on the total number of characters typed by each player during the whole duration of the game. For example – If a player types the word “game”, he will be awarded 4 points. The final score of each player is the sum of the total characters typed by that player. This strategy was chosen so that total score is determined by number of character typed instead of number of words, which on average will decrease the congestion on the whole network as clients may gain more points by correctly writing larger words.

The leaderboard is displayed with decreasing order of scores of players and also separately declaring the winner of the game.

Server & Client