A 6th semester mini project report for

Unix Systems Programming

on

Online Scrabble

Submitted by

USN 1PI11CS005, Abhilasha R
USN 1PI11CS033, Archana R
USN 1PI11CS041, B. Kruti
USN 1PI11CS050, Chitra Singh
USN 1PI11CS060, Durga R

Guide

Prof. Srishail Chari

Department of Computer Science & Engineering
PES INSTITUTE OF TECHNOLOGY
(An Autonomous Institute under VTU Belgaum)
100 Feet Road, BSK III Stage, Hosakerehalli, Bengaluru - 560 085

TABLE OF CONTENTS

INTRODUCTION

Scrabble is a word game where two to four players can play simultaneously. The objective of scrabble is to score more points than one's opponent. A player collects points by placing words on the game board. Each letter has a different point value, so the strategy becomes to play words with high scoring letter combinations.

The board is 15 X 15 grid of cells and each cell accommodates a letter forming a word. Each square contains denominations such as triple letter, triple word, double letter/word. This adds to the score depending upon the denomination.

General Rules

- Each player plays the game turnwise, placing letters from his rack (of 7 letters) on the board to form a word linking it to the already existing word.
- The word made must be a valid dictionary word defined the standard dictionary.
- Depending upon the word made, and denominations of the squares scores are updated.
- Diagonal, upside down and right to left moves are invalid.

Constraints

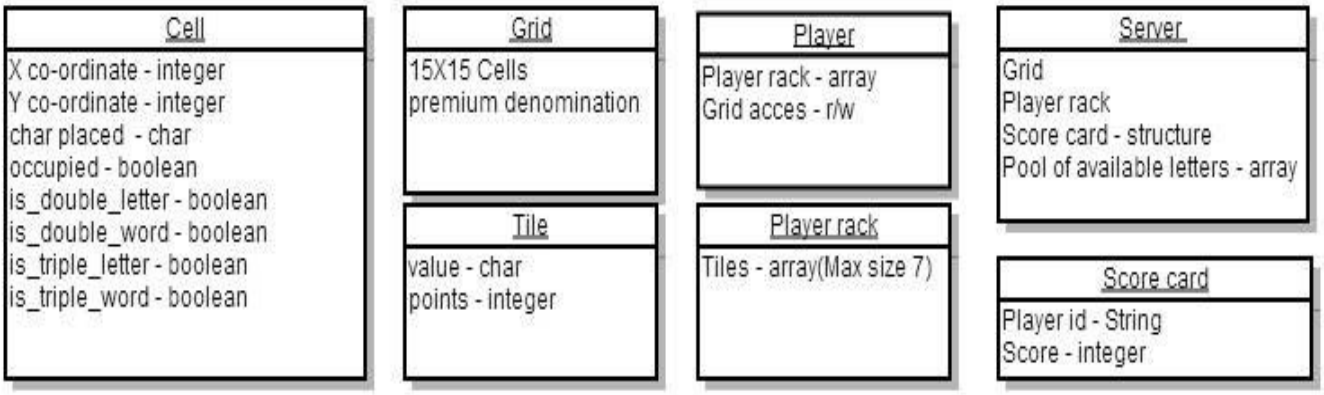
- Game can be started when at least two players join the table.
- A player has to play the game to completion, option to end a game can be given only if all players consent to the decision.
- If a player does not make a move for 5 minutes and is idle , the player is removed from the game

SCOPE

In Scope:
1. Client Server architecture is built for the game
2. Player can choose to wait or join a table.
3. Multiple tables are present on the server side
4. Error handling and validation checks, on the input message.

Out of scope:
1. Number of players is restricted to 2 to 4 players only.
2. Networking - communication across computers
3. This is not a desktop or an intranet application.

COMPONENT ARCHITECTURE AND DESCRIPTION



Player : The participants playing scrabble . The player who wants to join a table for a game , essentially establishes connection to the server and requests to join the table.

Server : In this particular client-server architecture the server controls the board , which is a shared memory .The server accepts requests from players and initiates the game . The server communicates with the players through messages .

Board : Board is a 15 X 15 grid containing cells , also called squares that holds single letters . The cell may have denomination or premiums .

Rack : The player is given a total of 7 random letters from the pool of alphabets . The player constructs words from the letters in the rack . The rack will be filled back by the server after the letters are used to make a word.

Pool of Alphabets : The initial set of alphabets that are fixed for the game. The rack of the player is updated from this pool .

Move : A move is made by a player during his turn . The player sends the word made by the him/her to server , letter by letter along with the grid coordinate .
For eg : Move=set({char : (from the rack) , X: , Y : }).

Turn: Set of moves made by the player is a turn .

SYSTEM OVERVIEW

The game follows client-server architecture , where the server manages multiple tables . Intrinsically the server and client are different process in the same system co-ordinating amongst each other. Shared memory segment is used as a IPC medium for the this game . Appropriate permissions and synchronization using semaphores provide an efficient way of regulating the function .

SYSTEM ARCHITECTURE

The basic high level architecture of the game is as follows
High level Architecture

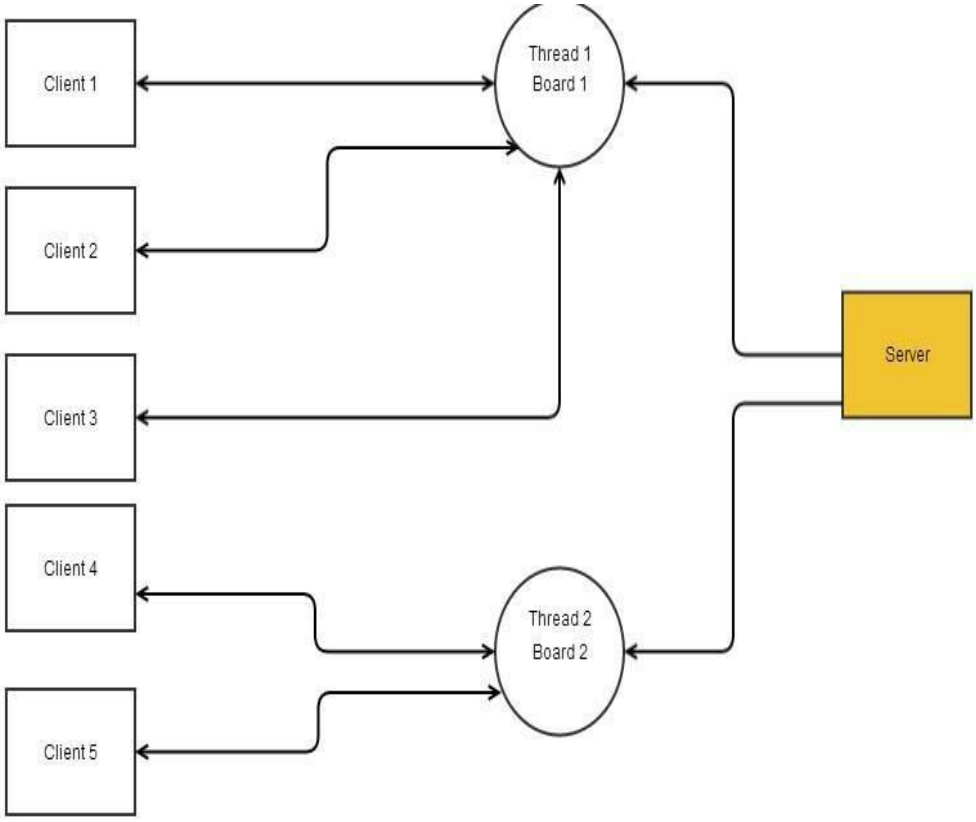
In this architecture both client and server are essentially in the same system. Inter process communication is used for server- client correspondence.

1. The client establishes a connection to the server . After making the connection, the client sends a request to join a table for a game . The number of tables available is broadcasted by the server
2. If the table has seats available and if the game is yet to start, the server allots a space to the player. The player can accept the allotted place or wait till the next game.
3. The game starts when there are at least two players on the table . The pool of alphabets are made available , the player racks and other initialisation takes place. The board is empty at the beginning of the game.
4. The board is visible to all players and all players initiate the change to be made on the board during their turn.
5. Each player takes a turn to make moves on the table . No player is given access to the table while a player is making a move during his turn . This mechanism is implemented using locks .
6. The serves locks player 1 initially and after his/her turn , server unlocks the player and gives the lock to the next player in a serial manner.
7. Consider a player , player1 making a move in his turn . Once all moves are made , the server validates the word and updates the grid and score on the table . Finally server gives player2 the next turn.
8. The scores on the table is visible to all players .
9. The game ends when the pool of alphabets are exhausted or when the board has no empty cells.
10. A player may end a game with the consent of all players , else all players must play till completion .

ARCHITECTURE DESIGN

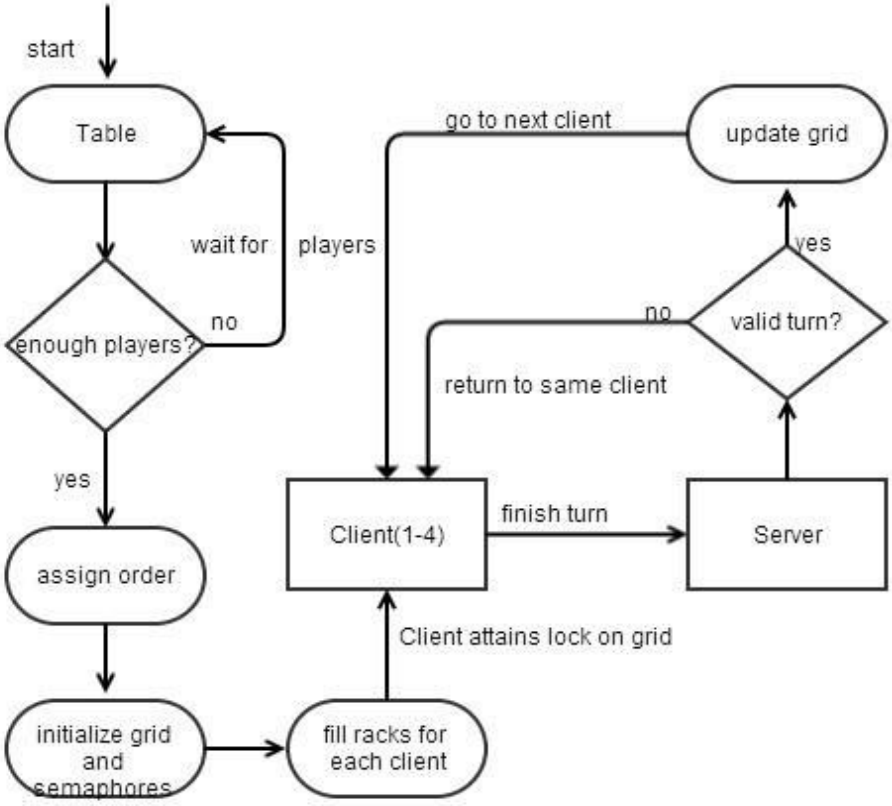
High-Level Architecture

The server process spawns new thread for each table that contains a scrabble board. Each client process is connected to the server process . A board is a shared memory segment and communication takes places using message queues . Different shared memory segment and message queues are used for different threads spawned.



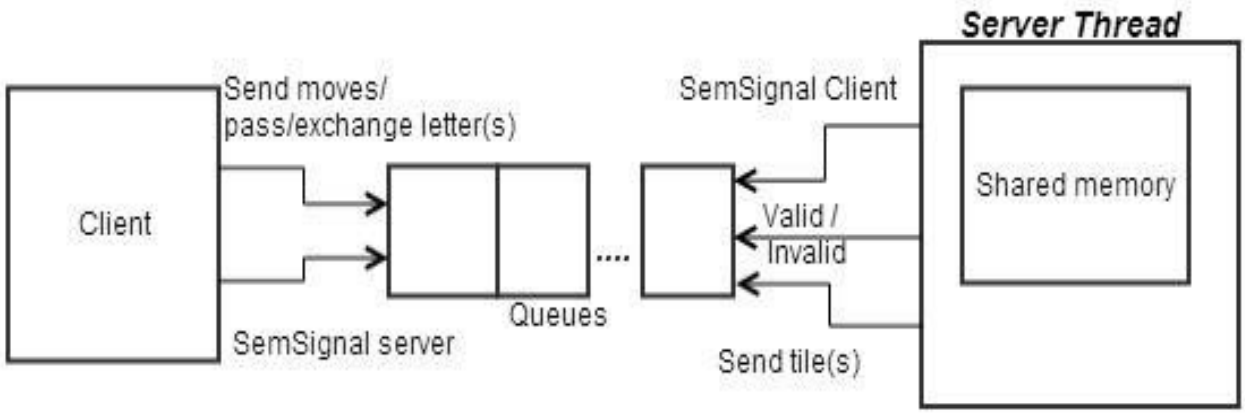
FUNCTIONAL ARCHITECTURE

Fundamental functionality



When the game commences and unfolds the server waits for the players to join the table , the game proceeds with at least two players on the table . Synchronization mechanism is implemented using semaphores .

Client-Server Design

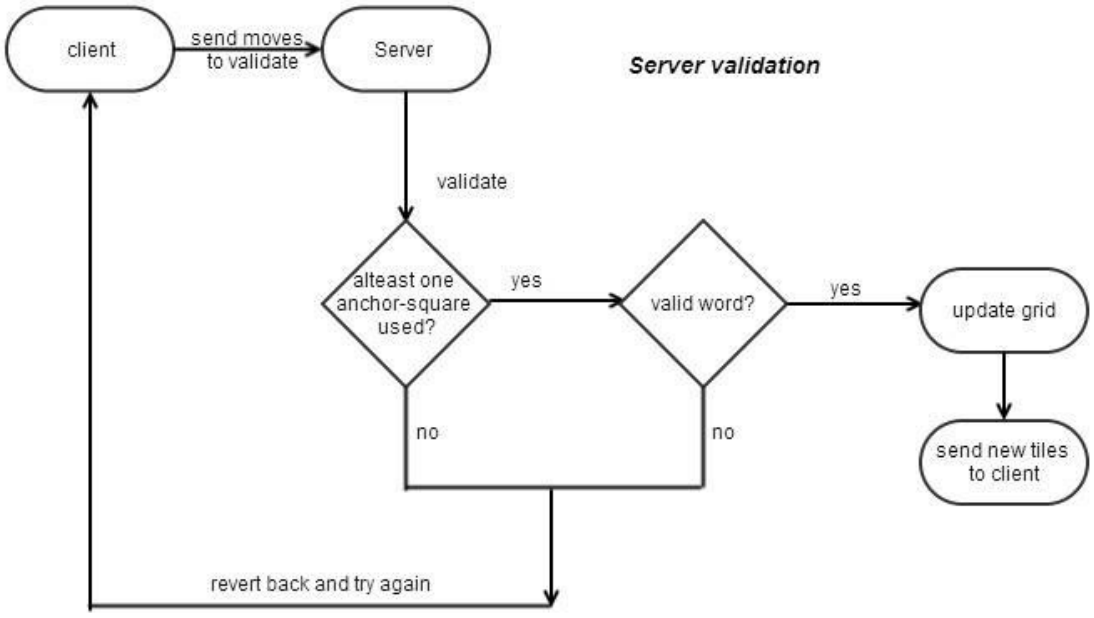


Clients SemWait on server until the server signals it

The client semWait s on the server to make its move on the board . When the server unlocks the client , the moves are encapsulated in packets and sent to the server . The server validates the move and moves on to the next player in sequence . If the move made by the client is invalid , the previous state is restored and the client has to make a new move .

Depending upon the number of valid moves made by the client , the player rack is updated again . The server replaces the rack with new letter chosen from pool of remaining alphabets .The server then signals client 2 and the same procedure follows until completion of the game .

Detailed Server Architecture



The server validates the moves made by the player . The control is transferred from player , back to server for validating process .

DATA DESIGN

Client side

1. Initial Connection to the Server .
2. Rack of letters: Rack of letters is represented as:
   char rack[7];

Server side

1. **Player Board**: Each cell on the board can accommodate a character and is represented as a 'struct'.

```
struct cell
{
        int x;                  // x coordinate or column number
        int y;                  // y coordinate or row number
        char content;           // character that fills the cell; It is initially '\0'
        int points;             // points for the cell
        bool is_triple_word;    // to triple the points obtained by the player
        bool is_double_word;    // to double the points obtained by the player
        bool is_triple_letter;  // to triple the points for the occupied character
        bool is_double_letter;  // to double the points for the occupied character
        bool is_filled;         // true if occupied by a tile; false otherwise
};
```

Before the game starts, every cell is initialised with it x-y coordinates and the content field is set to NULL. The boolean fields are set to 'true' for certain special cells on the board that never changes.

2. **Pool of letters**: The server maintains a standard pool of letter tiles with each tile representing a character. We assume the size of the pool of letters to be 100. This is an array of structure pointers. The structure is:

```
struct character_info
{
        char c;// the character between a-z
        int points; // the points associated with the character
        int num_in_pool; // the number of tiles in the pool denoting character 'c'
};
```

The array will be:
struct character_info * character_pool[26];

The players are allotted tiles of letters from this pool randomly and the count of each of the allotted letters decrement accordingly. This array is used by the server to calculate points for a player's move.

3. **Temporary linked list of letters sent by current player:** The player sends a linked list to the server where each node is a structure containing the following:

```
struct client_move
{
        char c;                 // character chosen from player's rack
        int x;                  // x coordinate of the cell where the tile is to be placed
        int y;                  // y coordinate of the cell where the tile is to be placed
};
```

The linked list is a list of nodes where each node is as follows:

```
struct node
{
        struct client_move * key;
        struct node * link;
};
```

4. **Points board**: This is a global data structure where the server updates points to each player after a turn is done. This is an array of structure pointer to a structure that is as follows:

```
struct player_info
{
        int player_id;          // player ID number
        int points;             // total points obtained by the player
};
```

The array is :
struct player_info * points_table[n]; // n is the number of players

5. **Details** of all tables in the server: Every table has a thread associated with it and this thread ID is taken as the table ID. It is an array of structure pointers of the form given as follows:

```
struct table_details
{
        pthread_t thread_id;            // acts also as the table ID
        int num_players;                // number of players in the table for that session
        int player_id1;                 // 1st player
        int player_id2;                 // 2nd player
        int player_id3;                 // 3rd player
        int player_id4;                 // 4th player
        int order_of_players[num_players];

};
```

The array will be:

```
struct table_details* all_tables[MAXTABLES];        // MAXTABLES is the maximum number of
                                                       tables in the server
```

6. **Dictionary**- This is a text file that contains valid English words for the server to check for the validity of the words formed by players.

FUNCTIONS

Client side

1. Request to join a table: Each player can choose which table to join and which position he/she wants and then send a request to the server . The server then accepts the client process as a player process.

```
void request_to_join(ipthread_t thread_id,nt position);
```

2. Make a move: This function is to select a character from the rack of cells and choose the x,y coordinate of the board on which the tile will be placed.

```
void make_a_move(void);
```

3. Complete turn: This function is to acquire a lock on the board, make a maximum of 7 moves, send the list of letters to the server and then unlock the board.

```
void send_moves_list(pthread_t thread_id,struct node* wordlist);
```

4. Exchange letters: This function is called when the client wants to exchange some or all of his/her letters in his/her rack.The player loses a turn on exchange .

void exchange(ipthread_t thread_id,,nt *posn_in_rack);
‘posn_in_rack’ is an integer array that holds the positions of the elements to be exchanged. Its maximum size is 7.

5. Pass: This function is called when the user cannot make a valid move and doesn’t wish to exchange his/her letters in the rack.

void pass(pthread_t thread_id,);

6. View the board: The client calls this function to read the board which is a shared memory.

void show_table(pthread_t thread_id,)

Server side

1. Enter new player: When a request to join is received from a client, the server has to make entries in the points table about the new player.

int enter_new_player(player_details, pthread_t thread_id); // returns player_id

2. Refresh board: When a new game begins, the board has to be refreshed and reset with all the cells initialised to initial default settings.

void refresh_board(pthread_t thread_id);

3. Update board: When a player’s turn is accepted as valid, the changes made by the player has to be updated.
void update(pthread_t thread_id, struct node* head_word_list);

4. Display board: This function is to present a view of the board to all players on the table. It is called every time an update happens.

void display(pthread_t thread_id);

5. Validate user move: This function in turn calls 3 functions:

a. Validate word: This function checks if the word framed by the user is a valid word from the list of words in the text file.

bool is_valid_word(char* word);

b. Validate cell: This function checks if the cell to be filled is already occupied or not.
bool is_valid_cell(struct cell* cell_chosen);

c. Validate position: This function checks the common column or row filled by the player and checks if at least 1 cell in the word was already occupied. The move is only then valid.

bool is_valid_posn(struct cell* *cells);

d. Validate_first_move: This is a special function that checks if the first move filled the centre cell of the board.

bool is_valid_first_move(void);

7. Update points table: This function is called every time a player's turn is accepted valid.

void update_points(int player_id, int new_points);

8. Calculate points: For every valid turn completed, the points for that move are calculated based on the values in the linked list that has nodes of moves made by the user. This function calls update_points to update the table with new points.

void calculate(pthread_t thread_id,struct node* word_list);

IPC Structures

1. Shared memory - Player board
2. Message Queues - Communication between Client and Server

Thread Function

1. A board game where 2-4 players can play. So if there are more than a table available, the client can choose the table where he wants to play.
2. Each thread is spawned for a table in the server process .
3. All communications are routed through the server
4. When a player requests the server to join a specific table, the server maintains a log of which player is being assigned to which table, and henceforth redirects all communication
5. When a player requests to move, the server reassigns them a table and proceeds to route all communication to the thread of that table

Synchronisation

Binary semaphores are used for this purpose. Each client has 2 semaphores- a semaphore for itself and the server. The server has 5 semaphores, one for itself and 4 for the client processes

Pseudocode:

Client side:

```
if (current_player == my_player_id )
{
        semwait(mysem);
        // do operation
        semsignal(serversem);
}
```

Server side:

```
// common function for i = 1,2,3,4
if(current_player == i)
{
        semwait(serversem);
        // do operation
        semsignal((i+1)%num_players);
}
```

CONCLUSION

The Online Scrabble application deals with multiple client processes communicating with a server process that involves the use of several IPC mechanisms. The application also requires the use of semaphores for synchronization process. Hence, this project design intends to illustrate the importance of incorporating IPC structures, synchronizing mechanisms and other data structures that is an integral part of the application.The Unix Programming concepts are put to efficient design of the project.