Computer Networks Project Part 1

SOCKET PROGRAMMING BASED ON GAME SERVER

Summary

[In this report, I try to explain the experience that I have in socket programming by using C language. I will explain the written C code about some socket operations, connecting establishment with clients, listening connections, sending and receiving messages over created connections. Also, I will explain how we can use those transferred messages in order to implement selected game.]

Nurefşan
Sertbaş
040110078
sertbasn@itu.edu.tr

Introduction and Describing the Problem

The project has 2 main part as described in the project description. In this report, I explain the first part of the project; Game program using TCP connection. We choose Hangman as a game to implement so that it enables 2 players at the same time. Also, the written C code enables playing game between multi players and multi sessions.

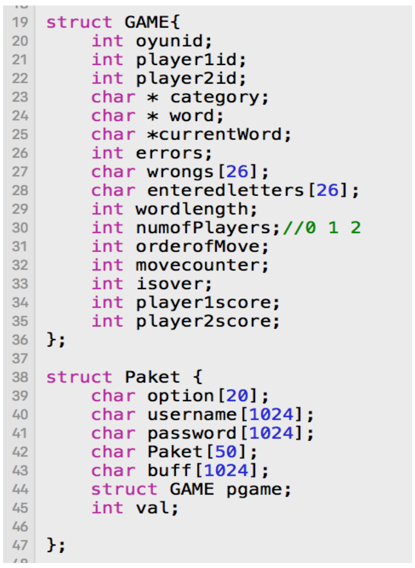
I will explain all the steps in sub sections with details as following.

Main Principle of the Program

Processing only one client at a time is not so efficient, it makes server busy during communicating with one client. Also, in our scenario the server should handle multiple connections. One of the way to realize this is using threads. Threads are special processes that share data space with the parent. This reduces the time for reloading the caches and memory operations. A thread can be assigned for each connected client so that server can handle communication with the client.

Structure and Variables of the Program

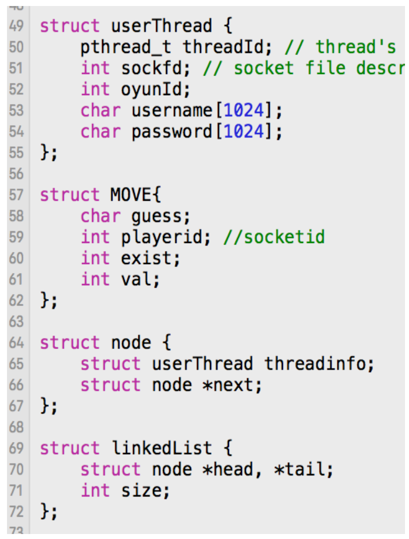
In order to realize the game, we define some structures that are given below.



```c
struct GAME{
    int oyunid;
    int player1id;
    int player2id;
    char * category;
    char * word;
    char *currentWord;
    int errors;
    char wrongs[26];
    char enteredletters[26];
    int wordlength;
    int numofPlayers;//0 1 2
    int orderofMove;
    int movecounter;
    int isover;
    int player1score;
    int player2score;
};

struct Paket {
    char option[20];
    char username[1024];
    char password[1024];
    char Paket[50];
    char buff[1024];
    struct GAME pgame;
    int val;

};
```

We hold all created games in array. Each game has a unique id. *Player1id* and *player2id* holds the socked fd's of the players. Each game belongs to one of the predefined *category*. Each game is initialized with a *word* and guessed part of the game is hold in *currentword*. *Movecounter* counts the number of moves. *Numofplayers* counts the players in the game. All entered letters by players given in *enteredletters* and entered wrong letters in *wrongs*. *Isover* is a flag to hold whether the game is finished or not. *Player1score* and *player2score* holds the scores of the players as can be clearly seen.

Paket is very important structure for the game because when we want to message to/from server we fill the related variables in *Paket* structure. Option holds the goal; I mean the purpose of usage at receiver side. We will explain those purposes later.



```
struct userThread {
    pthread_t threadId; // thread's
    int sockfd; // socket file descr
    int oyunId;
    char username[1024];
    char password[1024];
};

struct MOVE{
    char guess;
    int playerid; //socketid
    int exist;
    int val;
};

struct node {
    struct userThread threadinfo;
    struct node *next;
};

struct linkedList {
    struct node *head, *tail;
    int size;
};
```

We prefer to use treads to represent users. Also, we use linkedlist structure to hold users. Each user has its own *username*, *password* and unique *sockfd*. Also, it is assumed that one user can be in the one game at a time. User can only join new game after finishing the current one.

We define *MOVE* structure to represent each move in the game so that we hold guessed letter in *guess* variable, *exist* to check the move is correct or not. *Playerid* is used to hold which player make this move and *val* is used for some additional purposes.

Node and *linkedlist* structures are used to implement linkedlist data structure to hold users.

Functions of the Program

Server.c

`char randomNumber(int max_number):` It generates random number up to its parameter.

`char *getWord(char * category):` It reads the file that specified with given category name and uses randomNumber function to select and return with the random word.

`int game(struct GAME *mygame,struct MOVE * mymove):` It checks the move that is given in mymove parameter and sets related fields in the game that is given by pointer named mygame. It returns with 0 means the game continue, 1 or 2 means the game is ended due to reaching max error limit or finding the keyword.

`void *cleanThreadFunc(void *param)`: It terminates the server and clears the related variables.

`void *clientThreadFunc(void *fd):`It is called when a new user is creating. It contains default recv function to receive packages and according to the option field of the package it does some operations defined in if-else cases.

`void printallgames():`It prints some information of all the games created by users.

Client.c

`void menu()`: It prints all commands to console.

`void show_hang(int i,char wrongs [], int count, char entereds []):`

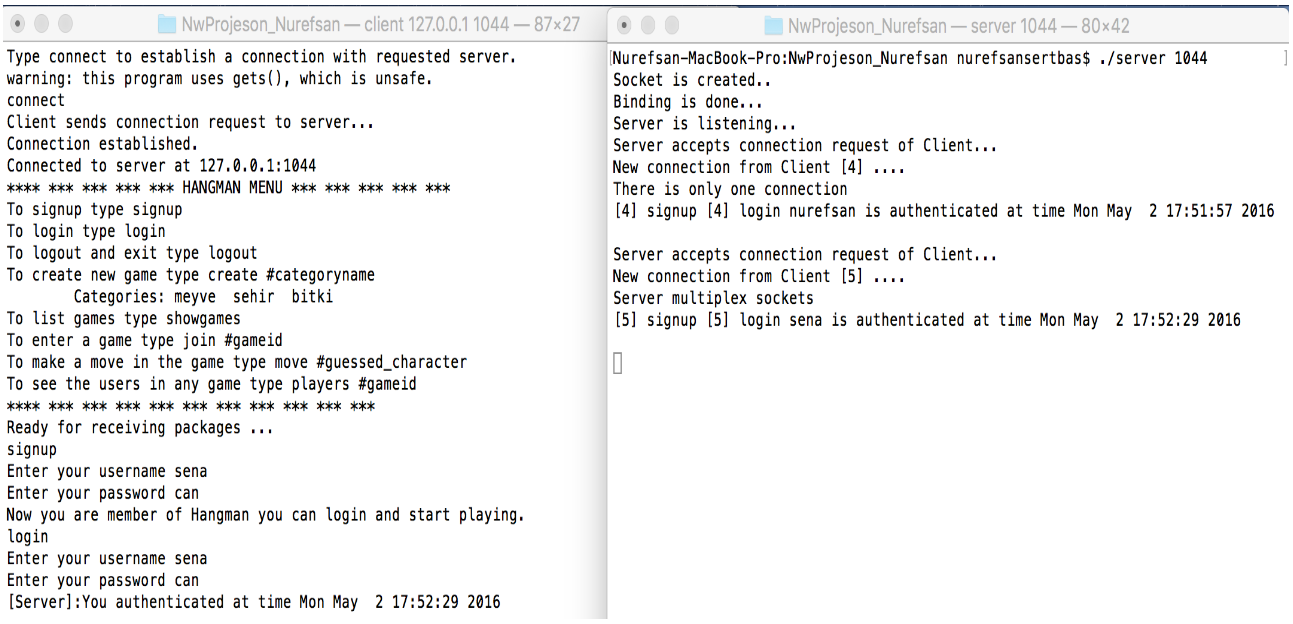
It prints the status of the hangman, entered letters and wrongs to the screen.

`void *receiver(void *param):` It contains default recv function to receive packages and according to the option field of the package it does some operations defined in if-else cases.

Establishment of TCP Connection and Commands of the Program

→Connect, signup and login commands

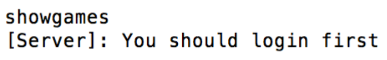
Firstly, player should type connect to connect defined server then information of the connection is given at the client console and menu is printed as can be seen below.


NwProjeson_Nurefsan — client 127.0.0.1 1044 — 87×27
NwProjeson_Nurefsan — server 1044 — 80×42

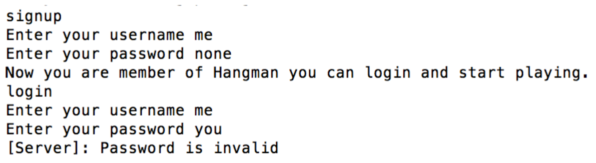
```
Type connect to establish a connection with requested server.
warning: this program uses gets(), which is unsafe.
connect
Client sends connection request to server...
Connection established.
Connected to server at 127.0.0.1:1044
**** *** *** *** *** HANGMAN MENU *** *** *** *** ***
To signup type signup
To login type login
To logout and exit type logout
To create new game type create #categoryname
        Categories: meyve  sehir  bitki
To list games type showgames
To enter a game type join #gameid
To make a move in the game type move #guessed_character
To see the users in any game type players #gameid
**** *** *** *** *** *** *** *** *** *** ***
Ready for receiving packages ...
signup
Enter your username sena
Enter your password can
Now you are member of Hangman you can login and start playing.
login
Enter your username sena
Enter your password can
[Server]:You authenticated at time Mon May  2 17:52:29 2016
```

```
Nurefsan-MacBook-Pro:NwProjeson_Nurefsan nurefsansertbas$ ./server 1044
Socket is created..
Binding is done...
Server is listening...
Server accepts connection request of Client...
New connection from Client [4] ....
There is only one connection
[4] signup [4] login nurefsan is authenticated at time Mon May  2 17:51:57 2016

Server accepts connection request of Client...
New connection from Client [5] ....
Server multiplex sockets
[5] signup [5] login sena is authenticated at time Mon May  2 17:52:29 2016
```

Player should register by typing signup and entering name and password. After that, player should login to make any operation. Otherwise it is not allowed (See below screenshot).

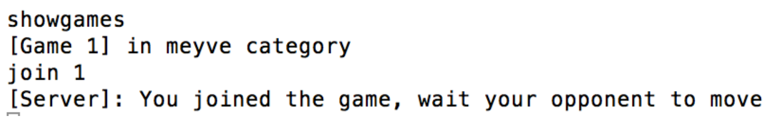


```
showgames
[Server]: You should login first
```



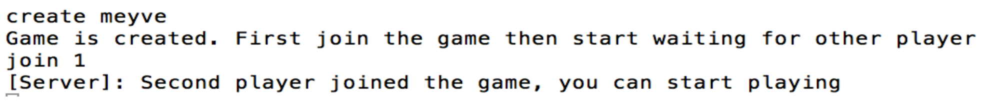
```
signup
Enter your username me
Enter your password none
Now you are member of Hangman you can login and start playing.
login
Enter your username me
Enter your password you
[Server]: Password is invalid
```

→Showgames, create, join commands



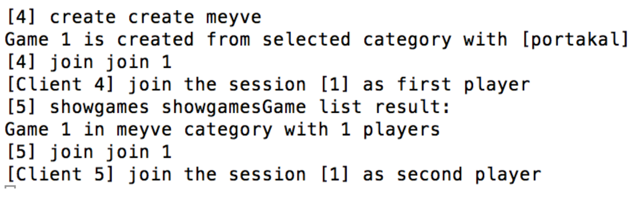
```
showgames
[Game 1] in meyve category
join 1
[Server]: You joined the game, wait your opponent to move
```

Player 1 console



```
create meyve
Game is created. First join the game then start waiting for other player
join 1
[Server]: Second player joined the game, you can start playing
```

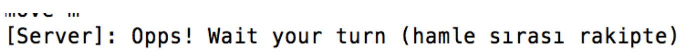
Player 2 console



```
[4] create create meyve
Game 1 is created from selected category with [portakal]
[4] join join 1
[Client 4] join the session [1] as first player
[5] showgames showgamesGame list result:
Game 1 in meyve category with 1 players
[5] join join 1
[Client 5] join the session [1] as second player
```

Server console

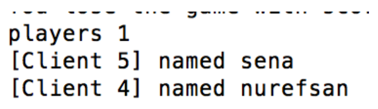
Orderofmove is used to hold the user that who will be allowed to make the next move. Its sample screenshot is given below:



```
[Server]: Opps! Wait your turn (hamle sırası rakipte)
```

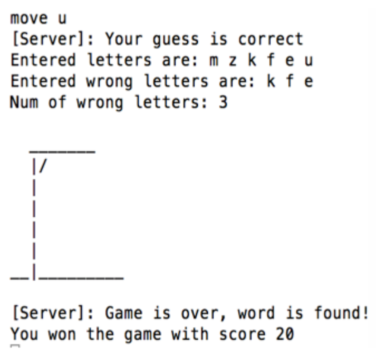
→players command

You can use players #gameid command to see the users in the selected game.

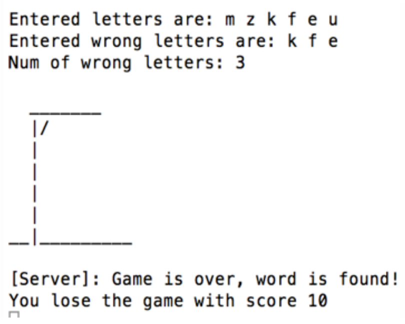


```
players 1
[Client 5] named sena
[Client 4] named nurefsan
```

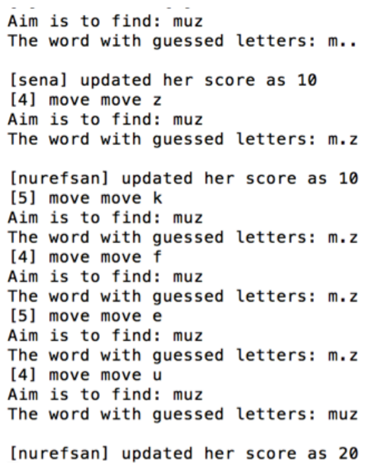
→move command



```
move u
[Server]: Your guess is correct
Entered letters are: m z k f e u
Entered wrong letters are: k f e
Num of wrong letters: 3

   ______
   |/
   |
   |
   |
   |
 __|________

[Server]: Game is over, word is found!
You won the game with score 20
```



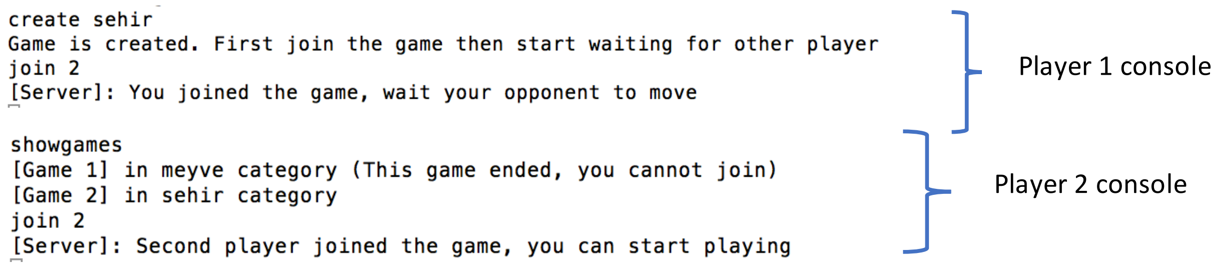
```
Entered letters are: m z k f e u
Entered wrong letters are: k f e
Num of wrong letters: 3

   ______
   |/
   |
   |
   |
   |
 __|________

[Server]: Game is over, word is found!
You lose the game with score 10
```



```
Aim is to find: muz
The word with guessed letters: m..

[sena] updated her score as 10
[4] move move z
Aim is to find: muz
The word with guessed letters: m.z

[nurefsan] updated her score as 10
[5] move move k
Aim is to find: muz
The word with guessed letters: m.z
[4] move move f
Aim is to find: muz
The word with guessed letters: m.z
[5] move move e
Aim is to find: muz
The word with guessed letters: m.z
[4] move move u
Aim is to find: muz
The word with guessed letters: muz

[nurefsan] updated her score as 20
```

Other Details of the Program

After finishing the game, user can create/join another game as can be seen following:



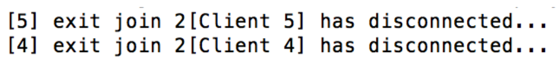
```
create sehir
Game is created. First join the game then start waiting for other player
join 2
[Server]: You joined the game, wait your opponent to move
```

Player 1 console

```
showgames
[Game 1] in meyve category (This game ended, you cannot join)
[Game 2] in sehir category
join 2
[Server]: Second player joined the game, you can start playing
```

Player 2 console

When user type logout it ends the connection of the client as can be seen from the server side (See below screenshot).



```
[5] exit join 2[Client 5] has disconnected...
[4] exit join 2[Client 4] has disconnected...
```

Note: Also, 4 users can play 2 games simultaneously because server listen the same port but when any client connect to server new socket number assigned to client. The code works properly in such a scenario.

Note: You can use menu command to print the menu at the client console.

Note: The program uses the threads so that it should be compiled with 'gcc –pthread server.c –o server'