No Clue Crew

Chat Software Manual

Version: 1.0

Authors:

Zachary Agness, Benedict Casasola, Jared Cruz, Jaeven Laron, Liyuan Zhao, Robert La

University of California, Irvine

Table of Contents

Glossary

User Application

Client: a device or computer that requests data or information from a host.

GUI: Graphical User Interface

Hash Table: data structure used to cycle through account info

Host: a device or computer that provides data or information requested by a client.

IP Address: Internet Protocol Address

Peer to Peer (P2P): a term used to refer to the communication/interaction of one user with another.

Server: a device or computer that acts as a hub for information or resources in a network.

TCP Address: Transmission Control Protocol Address

Txt file: A file format that contains text information.

Developer Application

Socket: An endpoint of communication between two or more users/entities communicating over a network.

Collision: problem occuring in a hashtable when two elements that share the same hash key try to take the same index in an array.

Chaining: Method of handling collision by allowing multiple elements to share the same key. Elements sharing the same key are stored in a linked list at the array index.

Hashkey: value used to find a certain element in a hashtable

Network Protocol: a set of rules, formats, and procedures that govern the communication of two or more computers/systems in a network. TCP/IP are examples of network protocols

Client Architecture Overview

1.1 Main Data Types and Structures

1.1.1 Hashtables

This project will use hash tables that will store user info. An array of struct userlist (containing two pointers, which points to the first and last entry in the index) will be used to store users. Hashkeys will be generated by taking the ascii values of each letter or number in a users name, summing them up, and modding that sum by the size of the array that stores all the user names. The hashkey will be used to choose which index in the hashtable to store the user. The current approach of dealing with collision when registering multiple users is the process of chaining.

1.1.2 Structs

Structs are going to be used to create a user data type. When a user registers, they are required to enter a username and a password. This information will be put into a struct "user" which would contain the username, password, hashkey and pointers to next and previous entries. The next and previous entries are used to incorporate chaining in a hashtable.

1.1.3 Linked Lists

The linked list would be utilized within the userlist array to handle the process of chaining. When there are multiple users in an index, the list will be cycled through until the right user is found.

1.2 Major Software Components

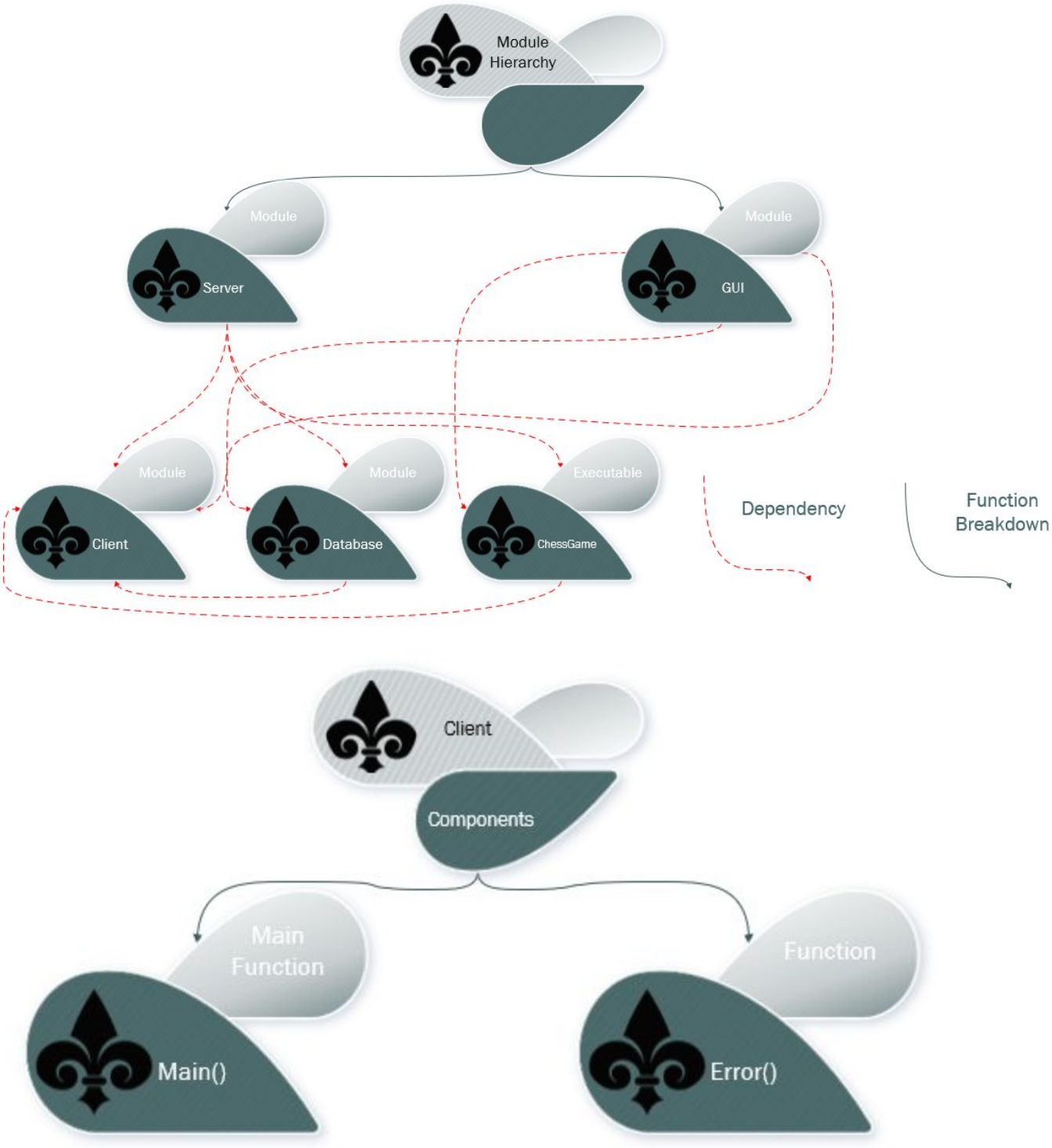

Fig 1: Module Hierarchy

1.3 Module Interfaces - API of Major Module Functions

1.3.1 Client.c

The client.c module will contain the address and the network protocol of the socket. In addition, the client will be able to send requests to the server by way of a function that sends all the necessary stuff to the server.

- Socket()
- Bzero()
- Recv()
- Fork()
- Error()
- Create_shared_memory()
- get_buffer()

1.3.2 Database.c

The database.c module will take in usernames and passwords and put them into a struct called User. The ascii values of each character in the username will be added together and modded by a value of 10, the number of entries in the array to hold the link lists of users. The use of linked lists is to apply chaining to the hashtable in order to prevent collision. In addition, the list of users will be written to a text file so that it can be read later by both the server and the client.

- bool show_ht_item(user_ht *user_ht, const char *key);
- bool new_pw_ht_item(user_ht *user_ht, const char* key, char* new_pw);
- bool database_txt_exist(const char *filename);
- user_ht *new_database(const char *filename);
- user_ht *create_database(const char *filename);
- bool save_database(user_ht *user_ht, const char *filename);
- bool database_main(const char *filename);
- bool databasetest();
- user_ht *get_db(FILE *filename, user_ht *database);
- user_ht *get_user_total(FILE *filename, user_ht *database);
- user_ht *get_user_info(FILE *filename, user_ht *database);
- user_ht *get_friends(FILE *filename, user_ht *datbase, int *key_list);
- void addFriend(user_ht *database, user_ht_item *user, user_ht_item *uFriend);
- void deleteFriend(user_ht *database, user_ht_item *user, user_ht_item *uFriend);

1.3.3 Has

The hash.c module deals with putting all the users into a hashtable which will then be converted to a text file database. Many of the functions in this module involve create a new hashtable, creating an new entry in the hashtable, and checking if a user is within the hashtable.

- user_ht *create_ht(void);

- user_ht_item *create_ht_item(const char *key, const char *value, const char *ip, const char *nickname);
- user_info *create_user_data(const char *ip, const char *nickname);
- void del_ht(user_ht *user_ht);
- void del_ht_item(user_ht_item *item);
- char *user_ht_search(user_ht *user_ht, const char *key);
- bool user_ht_search_password(user_ht *user_ht, const char *key, const char *value);
- void user_ht_insert(user_ht *user_ht, const char *key, const char *value, const char *ip, const char *nickname);
- char *user_ht_delete(user_ht *user_ht, const char *key);
- int hashCodeGenerator(const char *string);
- void displayHashtable(user_ht *hashtable);
- User_ht_item socket_find_user(user_ht *database, int socket);

1.3.3 GUI.c

This module displays the login window, create account window, friends list window, server window, and chat/chess window. The signals and functions will be connected to specific button presses. The functions used will be:

- StartWindow()
- Delete()
- GetLogin()
- Login()
- CreateAccount()
- GetRegister()
- Register()
- ShowWindow()
- ChatWindow()
- Message()
- Chess()
- Chat()
- Add()
- Remove()
- ServerWindow()

1.4 Overall Program Control Flow

The control flow for this application begins and ends with the client making a decision that is received by the main server. When the client application is initially launched and a successful

connection is made to the specified server, the user is prompted to either sign in with his/her user information if he/she already has an account, or create a new account if so desired. Once a user enters his/her sign in information, the server takes this input data and searches the user database, in this case a hash table, and fetches the matching user info from the table. If the user info is found in the hash table, then that means the login was successful and the user is valid. In the case that no matching user info was found in the database, the program returns to the login menu and informs the client that the login information was not valid. Upon a succesful log-in, the user is able to add or delete friends from a friend's list that shows which users are currently online (i.e. are also currently connected to the server and signed in). Users can then send requests to play a game of Chess with other online users. Upon an accepted request to play a game on the server, the game board is printed to both clients by the server, and users can now send and receive chat messages as well as make chess moves to play the game. All of this client input data is sent directly to the server, and the server handles all the chat and game logic processing, updating the clients' game states and chat logs after each relative input. Users have the choice to quit the game, or play until a winner is determined, both of which will disconnect the clients from the server and close the program. A visualization of the control flow for this application can be seen in Figure 1 below.

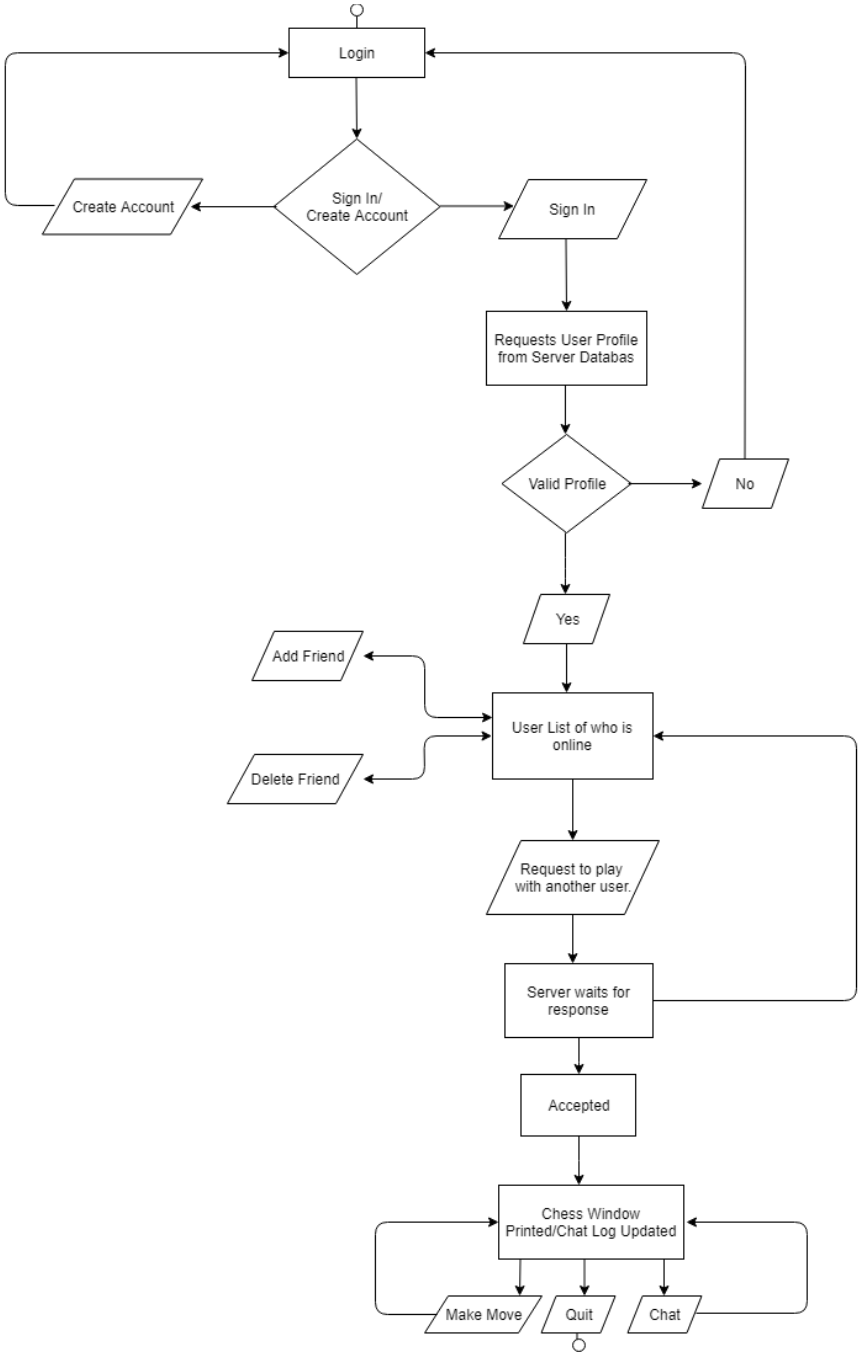

Fig 2: Client Control Flow

Server Architecture Overview

2.1 Main Data Types and Structures

2.1.1 Hash Tables

Since this program takes in user logins and passwords, the information will be stored inside of a hash table. The hashtable will start with a struct array of size 110. This would accommodate the 110 students in 22L. In the perfect scenario, all students would have their own index in the hashtable but if two or more students have the same hash code, then they will be stored in the same index through a linked list.

2.1.2 Game Logic in Server

The chess game logic will be held within the server. Since clients are only sending coordinates for the chess game, their response

2.1.3 User Structure

The user information will be held within a structure called “struct user” and this will hold the username, the password, ip address, and a key.

2.1.4 Hashtable Keys

Hash keys are generated when adding up all of the ascii values of the characters in a string. This value is then modded by the length of the array storing the names which is size 10. This key will then determine which index in the array to store the user in.

2.2 Major Software Components

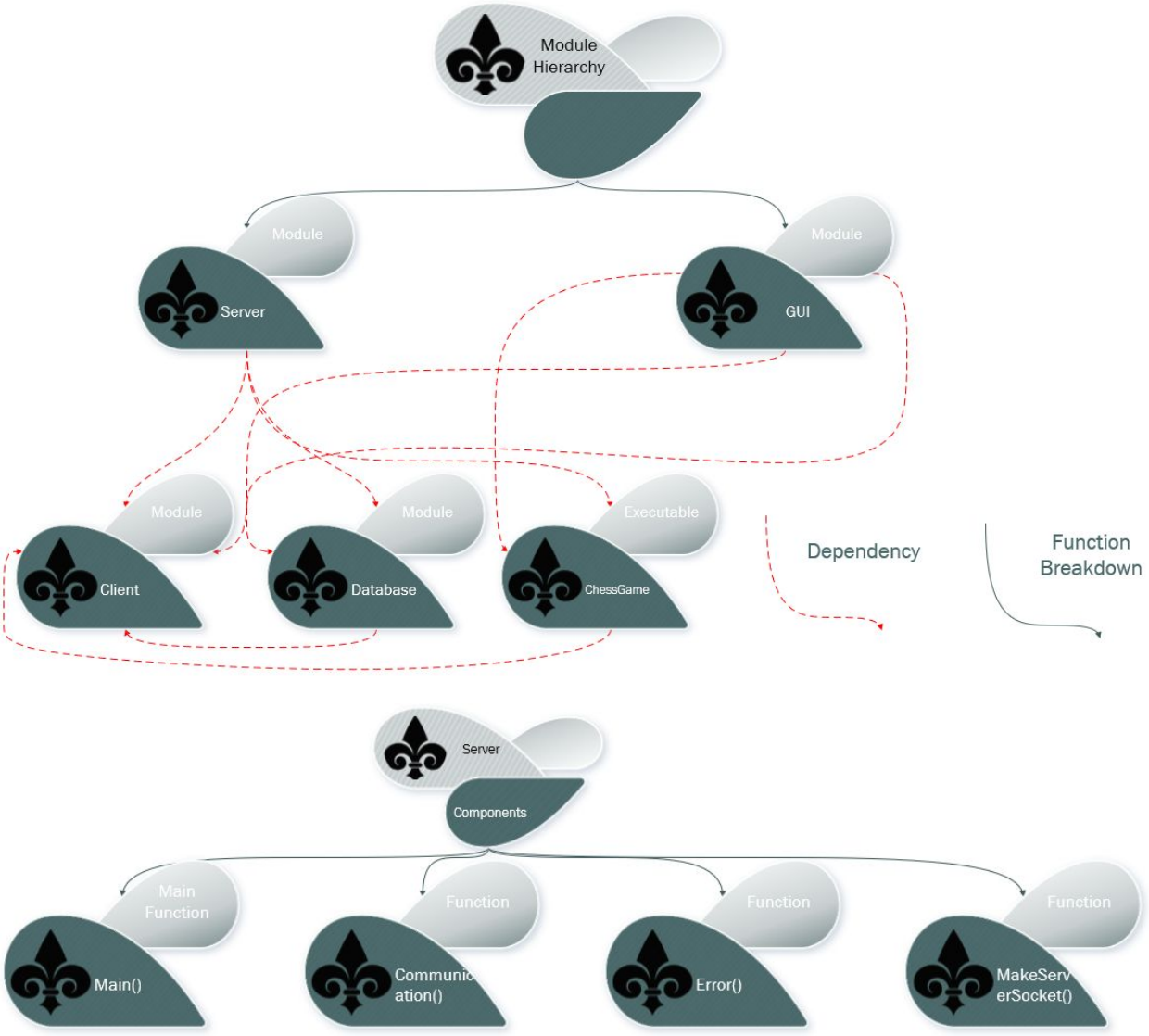

Fig 3: "Client to Server" Control Hierarchy

2.3 Module Interfaces - API of Major Module Functions

2.3.1 Server.c

The server will be initialized with a provided port, given that the port number is >2000. The server will then create a socket and listen to any clients on the same port. Afterwards, the server will create a new socket for every message sent from every user. Finally, the server will read the sent messages and write back to the clients if necessary.

- MakeServerSocket()
- Communication()
- NewClientCheck()

2.3.2 Database.c

The use of the database by the server allows it to direct communication traffic. It makes possible for user 1 to send a request to only user 2 and not to user 3. Additionally, it will be able to read the database txt file generated by the client and list all registered users.

- bool show_ht_item(user_ht *user_ht, const char *key);
- bool new_pw_ht_item(user_ht *user_ht, const char* key, char* new_pw);
- bool database_txt_exist(const char *filename);
- user_ht *new_database(const char *filename);
- user_ht *create_database(const char *filename);
- bool save_database(user_ht *user_ht, const char *filename);
- bool database_main(const char *filename);
- bool databasetest();
- user_ht *get_db(FILE *filename, user_ht *database);
- user_ht *get_user_total(FILE *filename, user_ht *database);
- user_ht *get_user_info(FILE *filename, user_ht *database);
- user_ht *get_friends(FILE *filename, user_ht *datbase, int *key_list);
- void addFriend(user_ht *database, user_ht_item *user, user_ht_item *uFriend);
- void deleteFriend(user_ht *database, user_ht_item *user, user_ht_item *uFriend);

2.3.3 GUI.c

This module displays the login window, create account window, friends list window, server window, and chat/chess window. The signals and functions will be connected to specific button presses. The functions used will be:

- StartWindow()
- Delete()
- GetLogin()
- Login()
- CreateAccount()
- GetRegister()
- Register()
- ShowWindow()
- ChatWindow()
- Message()
- Chess()
- Chat()
- Add()
- Remove()
- ServerWindow()

2.4 Overall Program Control Flow

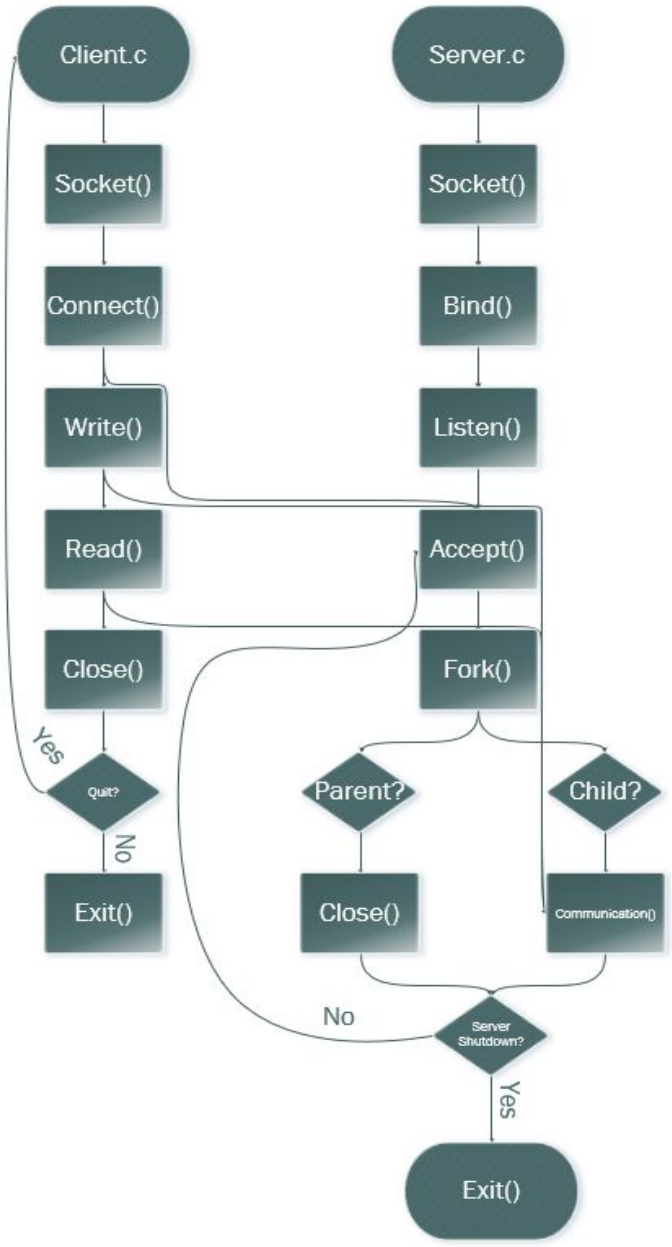

Fig 4: Client Control Flow

Installation

3.1 System Requirements

OS: Windows, Mac, or Linux operating system
Environment: Red Hat Linux Environment
Accounts: UCI EECS account
Hard Disk Space: 250 MB
RAM: 1 MB
CPU: Intel Pentium
Graphics Card: Intel Integrated Graphics

3.2 Setup and Configuration

For Windows OS:

1. Download XMING http://www.straightrunning.com/XmingNotes/
2. Follow the instructions on the installation wizard
3. Enable X11 Forwarding in your Linux environment

For Mac OS:

1. Download XQUARTZ https://www.xquartz.org/
2. Follow the instructions on the installation wizard
3. Enable X11 Forwarding in your Linux Environment

3.3 Uninstalling

1. Run the terminal emulator program
2. Enter the EECS server and the account in which the program was installed
3. Enter the directory where the program exists
4. Type the following command: % rm Chat

3.4 Building, Compilation, and Installation

For Windows OS:

1. Run a terminal emulator (PuTTY, MobaXterm, etc.)
2. Enter the EECS Linux server address (zuma.eecs.uci.edu, bondi.eecs.uci.edu, etc.)
3. Login using your UCI login information
4. In the desired directory, enter these commands:
    a. % gtar xvzf Chat_V1.0.tar.gz
    b. % evince Chat_V1.0/doc/Chat_UserManual.pdf
    c. % cd Chat_V1.0
    d. make all
    e. cd bin

f. % ./Chat <linuxserver>.eecs.uci.edu <port number> (Example: ./Chat laguna.eecs.uci.edu 9002)

5. To access the server, enter these commands:
    a. % gtar xvzf Chat_V1.0.tar.gz
    b. % evince Chat_V1.0/doc/Chat_UserManual.pdf
    c. % cd Chat_V1.0
    d. make all
    e. cd bin
    f. % ./server <linuxserver>.eecs.uci.edu <port number> (Example: ./server laguna.eecs.uci.edu 9002)

For Mac OS:

1. Run the program Xquarts (X11 in older versions)
2. Type the command:
    a. % ssh -X -Y Username@ServerName (UCI_UserName@zuma.eecs.uci.edu)
    b. Enter the password for your UCI account
3. In the desired directory, enter these commands:
    a. % gtar xvzf Chat_V1.0.tar.gz
    b. % evince Chat_V1.0/doc/Chat_UserManual.pdf
    c. % cd Chat_V1.0
    d. make all
    e. cd bin
    f. % ./Chat <linuxserver>.eecs.uci.edu <port number> (Example: ./Chat laguna.eecs.uci.edu 9002)
4. To access the server, enter these commands:
    a. % gtar xvzf Chat_V1.0.tar.gz
    b. % evince Chat_V1.0/doc/Chat_UserManual.pdf
    c. % cd Chat_V1.0
    d. make all
    e. cd bin
    f. % ./server <linuxserver>.eecs.uci.edu <port number> (Example: ./server laguna.eecs.uci.edu 9002)

Packages, Modules, and Interfaces

4.1 Detailed Description of Data Structures

4.1.1 Hashtables

The hash table will consist of a linked list array. The item at each index will be a struct user which will hold the name, the password, and the hash key of the element. Though the hash key will primarily be used to choose which index in the array to store the element in, it will also be used to search for certain items. The use of linked lists allows for chaining and helps prevent the problem of collision within the hashtable.

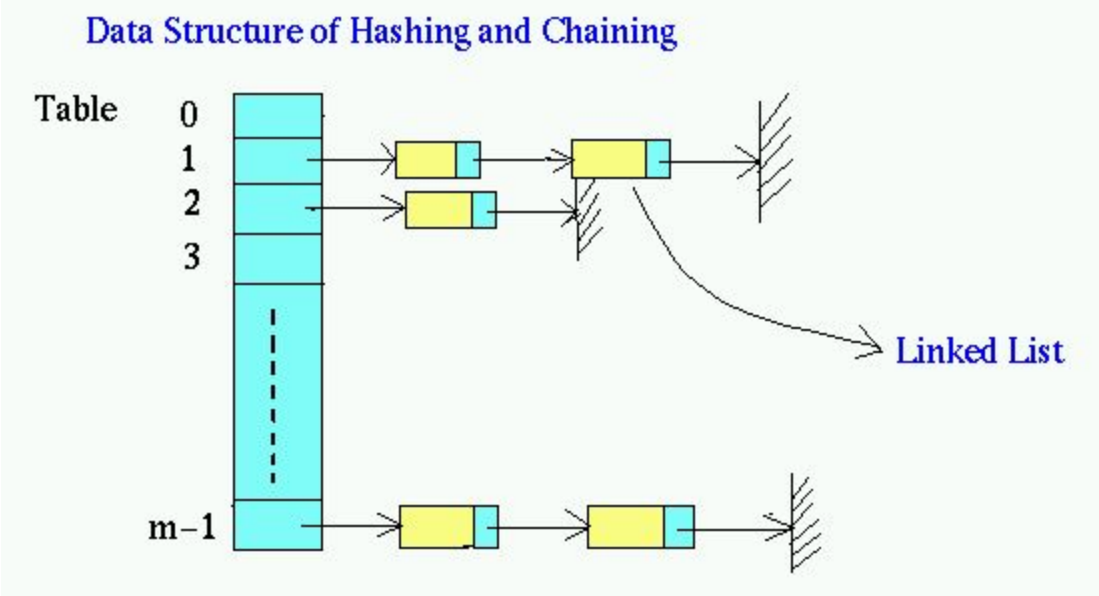

Fig 5: Diagram of chaining which shows an array of linked lists

The hashcode function takes the username and adds all of the ascii values of each of its characters. The sum is then modded by the length of the table which is 10. This produces a hash code which will be used as the index in the table in which the element will be stored.

4.1.2 Server Communication

Everytime a client sends a message, the server needs to read that information and store it into a character array and prints the output. Nothing will be printed if the client sends in nothing.

```
/*Handles read/write requests to server*/
void communication (int sock)
{
   int n;
   char buffer[256];

   bzero(buffer,256);
   n = read(sock,buffer,255);
   if (n < 0) error("ERROR reading from socket");
   //printf("Here is the message: %s\n",buffer);
   if (strcmp(buffer, "")) printf("%s\n",buffer);
   n = write(sock,"I got your message",18);
   if (n < 0) error("ERROR writing to socket");
}
```

Fig 6: Server Code of reading client messages and writing back

4.2 Detailed Description of Functions and Parameters

4.2.1 Hash.c

int hashcode(char username[256]);

The hashcode function takes the username entered while registering and produces a hash code. This determines the index in the array where the username will be stored.

struct User{
Char username[256];
Char password[256];
Int hashcode;
Int Friends_list
};

The user struct holds a users name, password, hashcode that corresponds to the main database's account repository and a friends list for that user. The friends list will be another hash table within that struct.

int Userlist[];

The userlist is an array that contains linked lists of users with the same hashcode. If two users have the hashcode of 2, then they will be placed in a linked list at Userlist[2].

bool *user_ht_search (user_ht *database, const char *key);

The search function takes a username /key and searches for a certain within the database. It takes the hashcode of the key, traverses the database hashtable and if it is able to find an entry with the given username, it returns true. Otherwise, it returns false.

void delete_ht_item(int hashcode, char username[256]);

The delete function takes the hashcode and searches the Userlist for the username that has been given as input. After it has found its target, the entry is deleted from the user list and the linked list is linked back together.

struct User *insert(struct User, struct Userlist);

The insert function is used to insert a new person into the user list. It takes the user, traverses within the inputted user struct and appends it into the user list. This will primarily be used when adding friends or when registering users.

bool user_ht_search_password(user_ht *user_ht, const char *key, const char *value);

Searches for a user in the hashtable database and checks if the value parameter matches the password in the user's profile.

4.2.2 Database.c

bool show_ht_item(user_ht *user_ht, const char *key);

This function shows all the information of an item given its key.

bool new_pw_ht_item(user_ht *user_ht, const char* key, char* new_pw);

This allows the server to change the password of a user.

bool database_txt_exist(const char *filename);

This checks if the text file for the database exists.

user_ht *new_database(const char *filename);

This creates a blank database to store in a .txt file in the parameter.

user_ht *create_database(const char *filename);

This reads the .txt file in the parameter and creates the database.

bool save_database(user_ht *user_ht, const char *filename);

This saves the database to the .txt file in the parameter.

user_ht *get_db(FILE *filename, user_ht *database);
user_ht *get_user_total(FILE *filename, user_ht *database);
user_ht *get_user_info(FILE *filename, user_ht *database);
user_ht *get_friends(FILE *filename, user_ht *datbase, int *key_list);

The above functions are used to read from the .txt file given by the first parameter and import all of the data (user total, usernames, passwords) into a hashtable that the server can use as a database.

user_ht *doesFileExist(char *filename);

This function checks if the current file exists in the same directory as the server.

void UpdateTextFile(char *filename, user_ht *database);

If there are any changes, such as a new user registering, this function will update the .txt file in the first parameter of the newly updated hashtable in the second parameter.

void addFriend(user_ht *database, user_ht_item *user, user_ht_item *uFriend);
void deleteFriend(user_ht *database, user_ht_item *user, user_ht_item *uFriend);

The above functions are used to add/delete a friend into a user's contacts. This is done by checking if both users exist and if they do, the uFriend is added into the user's friend's list. Alternatively, the uFriend is deleted if the deleteFriend function is used.

4.2.3 Client.c

Socket(int domain, int type, int protocol)

The socket function creates a socket where the client connects. It takes in the domain which determines what type of network rules it will be using, the type which specifies what type of socket it is, and protocol which determines the protocols it will use. For default protocol, int protocol is set to zero.

Bzero(void *s, size_t n)

Bzero function sets memory of certain elements to null. It takes size n and makes the string s to null. Useful for clearing data from long strings.

Recv(int sockfd, void *buf, size_t len, int flags)

The Recv() function takes in responses from the server. This function will primarily be used when two users are communicating with each other and takes in receives responses from the server.

4.2.4 Server.c

MakeServerSocket(uint16_t portno)

This function creates a socket for the server. It uses the input of port number and uses that information to get addresses and socket information. It binds to the socket and prepares itself to listen for incoming responses.

communication(int sock)

The communication socket takes the inputted socket and sets all memory to zero using the bzero function. From there it handles read/write functions from client to server and when it is accessing the database for logins and user registration.

4.3 Detailed Description of Communication Protocol

Communication protocol for the design of this program relies heavily on the main server, with little actually happening on the client side other than sending inputs to the server. The server has a database using a hash table that stores user information, and the server is also able to access the chess game logic through a dependency on the ChessGame module, which contains all of the game logic needed to play a game of chess. Clients send optional text messages and chess moves to the server, and the server processes this information. Messages received by the server are sent straight to the corresponding destination user to be displayed as a chat message. Chess moves are taken by the server, and passed to the ChessGame module. The ChessGame module processes the move, updates the current state of the game, and sends this information back to the server. The server can then pass the game state to the other respective client so that he/she can respond with their own game move and optional message. When a client is connected to the server and is in a session with a partner client, a graphical user interface is displayed to the client with separate text entries for a chess move and a chat message. When the client inputs his/her move and optional message, the server responds with a confirmation that the inputs were received, and the server relays this data to the second client's GUI in a likewise fashion. Visualization of communication protocol can be seen in Figure 2 below.

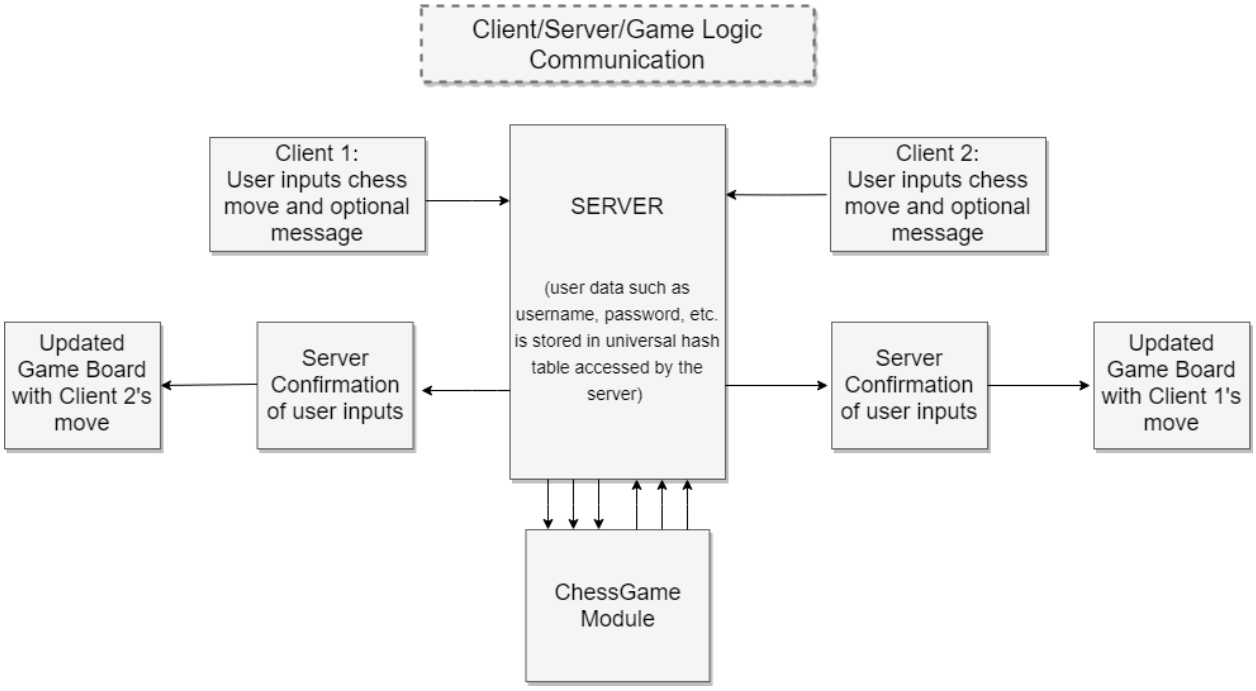

Fig 8: Communication Protocol Visualization

4.3.1 Communication Protocols

Implemented Protocol

--LOGIN <username> <password>

Client: --LOGIN <username> password

Server: Login as <username>. Access Granted.

Currently, the --LOGIN command can be inputted from the client. When the client sends this token to the server, the server will respond with a message saying "Please enter username and password." At this point, the next message that the client sends will be taken as a username. The server will receive this message from the client and store that input in a pointer. The message will be tokenized and taken by the server to be compared to all entries in the database, which will be a text file that the server reads.

Client: --LOGIN <username> password1

Server: Login as <username>. Access Denied.

Alternatively, if a client sends an invalid username or an incorrect password, the server will send the client an error message and redirect them to the login screen.

--REGISTER <username> <password>

Client: --REGISTER <user> password

Server: User <user> has been registered.
The --REGISTER command will take the name given by user and checks the password twice to confirm that that its the password they wish to have with the account. Once they enter this command, the server will add this user into the database.

Client: --REGISTER <user> password
Server: User <user> has been already been registered!
Alternatively, if a client tries to register an already existing user, the server will not accept the registering user.

--CHAT <friend's username>
Client: --CHAT <friend's username>
Server: Chat with <friend's username>. Request sent.
The --CHAT command allows a person to send a request to chat with another user that is on their friends list.

--QUIT
Client: --QUIT
Server: Chat exited.
The --QUIT command allows the user to exit from the chat mode.

GLOBAL
Client: GLOBAL <message>
Server: Message sent to global channel.
The --GLOBAL command allows the user to send a global message while in a private chat with another user.

--ADD <user>
Client: --ADD <user>
Server: Request to add <user> as a friend sent.
The --ADD command allows a user to send a friend request to another user.

--DELETE <user>
Client: --DELETE <user>
Server: <user> had been deleted from your friend's list.
The --DELETE allows users to remove people from their friend's list.

--HELP
Client: --HELP

Server: Displaying all user options …
The --HELP protocol allows the server to print all the available options to the client who sent the help request.

--QUIT
Client: --QUIT
Server: Session Ended.
The --QUIT command allows the user to exit the client program. This quit command only exits the program when the client is not in chat mode.

4.4 Detailed Description of Database

We are using hash tables to create a database of user login and passwords. The information stored in the hash table would be stored in a text file. When the user registers on the client, their information will be stored in a struct array. Within the array, there will be a linked list that holds all the users with the same hashkey. When multiple users hold the same hash key, they would be put in the same index in the array but would also be appended to the link list at that index. This method is known as chaining and is implemented to prevent collision.
The use of a text file allows for easy writing of the client to register new users and allows for the server to direct requests to the right users.

4.5 Detailed Description of Client GUI

We are using regular GTK 2.4 syntax code in C to program our GUI with a title screen with sign in options and second GUI containing the chess game and chat interface.
The first title screen GUI will allow the user to sign in and create a new account. There are two entry boxes for the username and password and two buttons either to login or create new account.

If the create new account button is pressed, then the login window will be closed and a new create account window will be opened. This new window will be have three entry boxes for username, password, and confirm password and two buttons for create new account and return to sign in. When the create new account button is pressed, the entered username will be checked if it already exists inside of the database and if the two password strings match. If the username exists within the database already, an error message will be displayed with a dialog box saying "Username is taken. Please enter another one." If the passwords don't match, a dialog box that says "Passwords do not match. Enter the passwords again." will be shown. If the username is available, the GUI will get the entries of the username and password and set them into a structure that will be added into the database.

In the start window, when the login button is pressed, the GUI will get the entries of the username and password and check the database if the username exists, and if the password matches within the database. If both are valid, the login window will be closed and a new window for contacts will be opened. This contact window will display all the users within the server and show their online status and whether they are a friend. At the bottom of the window, there are three buttons: to send a friend request, to accept friend requests, and to remove a contact.

Development Plan and Timeline

5.1 Partitioning of Tasks

5.1.1 Week 6

The primary task on week 6 was to completed the user manual and to allocate the work with each team member. The team split into two groups, one group focused on GUI and the other focused on the network communication.

5.1.2 Week 7

The user manual is completed.

The primary task of week 7 was to complete the developer manual for the code. In addition, each team member focused on the basics of their group, such as mastering GTK for the GUI subteam and doing tutorials for socket communication for the network subteam.

5.1.3 Week 8

The developer manual is completed.

The primary goal of week 8 is to have a client that can take create user accounts and can log in users by checking if the given username and password is in the database. The server will be able to listen to users and receive requests. With user input, the server will be able to respond to requests and send responses to users if another user wants to chat.

5.1.4 Week 9

First release of program is due. The team wishes to have a working client that can register users and save their information in the database. The server should be able to take a request from one user and send it to the correct user. The primary goal for this week is to make sure the program with all the required functions complete. This includes a friends list for each user and a functional database that has all user accounts.

5.1.5 Week 10

Second release of program is due. All primary functions are completed and will be refined if needed. Team will incorporate a recommended friends list by having users input information about hobbies. Extra feature implementation and continued polishing of base program.

5.1.6 Final Week

Final program release.

5.2 Team Member Responsibilities

Name	Role
AJ	GUI/GTK
Jaeven	Network Communication
Jared	Network Communication
Liyuan	GUI/GTK
Robert	GUI/GTK
Zach	Network Communication

Fig 9: Table of team members and their role on the project

The team has split up into two subgroups for this project, GUI and network communications. The GUI group is working on creating a GUI for the chess game and will implement a GUI for the client. This involves learning the functions within GTK and learning how to integrate it into programs.
The network communications group is focusing on creating the server and client for the multiplayer chess project. This involves studying the basics of socket communication and learning which protocol would best suit the project.

Back Matter

Copyright

Index

References

Chess_SoftwareSpec, 1st ed. No Clue Crew, Irvine, CA, 2019.

Harrison, Colin. “Xming X Server for Windows.” Xming X Server for Windows - Official Website, 2015, www.straightrunning.com/XmingNotes/.

McKay, Kyle J. “XQuartz.” XQuartz, www.xquartz.org/.