Chess with Grids-Online

Version 1.0 (Software Specification)

Producer:

Off the Grid

Developers:

Nathaniel Aponte, Andrew Lin, Sam Longo, Mason Ma, Buonkuang Priestley

Affiliation:

University of California Irvine EECS 22L

Table of Contents

1. Glossary

Account- The means by which a user is able to access the system. An account is identified by its username and requires a password for authentication. All of a user's data, such as their friend list and chat history, is stored in their account.

Chat Channels- a messaging panel between 2 or more people.

Emojis- a small digital image or icon used to express an idea, emotion. Emojis are basically special characters defined in Unicode.

Friend- A user who is on another user's friend list, allowing these two users to contact each other more easily. Users can see whether friends are online and open a chat with them via the friend list. A user becomes a friend of another user when one user sends the other a friend request and the other user accepts the friend request.

Friend List- List of users that have accepted a submitted friend request.

GIF- a format for image files that supports both animated and static images.

Login- Connecting an existing account to the program to gain access to its features. Logging in requires the user to enter the username and password for their account.

Password- A series of characters, generally known by only the user of one account, that must be entered in order for an account to access the program. Passwords are used to prevent unauthorized use of a user's account by another person. For security consideration, password is invisible while inputting and is transferred by their Hash value.

Provider- A daemon program that runs on the server platform to provide login verification, handshaking between two or more clients and contact data from its database to the clients. Additionally, the provider stores users' account details so that they can log in to the program.

Registration- Creating a new account that can be used to connect to the program. Registration requires a user to enter a username that is not used by anyone else on the system, along with a password.

Screen Name- The name for an account which is displayed to other users using the program. An account's screen name can differ from its username and can be changed in the profile menu.

User Application- The interface between the user and the provider. The user application handles the display of information to the user and sends and receives information from the provider.

Chess Panel- A popup for playing chess game between two humans.This is triggered by clicking the lower-right chess icon in the chat panel.

Username- An identifier for an account that the program uses to recognize the account. It must be entered in order for an account to access the program.

---

2. Client Software Architecture Overview

2.1 Main Data Types and Structures

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>      //defines the structure hostent

Struct sockaddr_in
        ServerAddress;  /* server address we connect with */
```

Data Types and data storage strategy that will be used:

- Lists
- Dynamic memory allocation
- Structs
- Int, char
- Nested loops, while loops,
- Switch statements
- Sockets

2.2 Major Software Components

- **Registration();**
  - Where it stores the user's profile information into a dynamic list
- **Socket();**
  - Creates a new socket. 3 arguments (address domain, type of socket, protocol)
- **connect();**
  - Establishes a connection with the server. 3 arguments (socket file descriptor, address of the host to which it wants to connect, and size of this address)
- **bzero();**

    - Initializes server address to zeros. Used for initializing a buffer. I.e before reading from the socket.
- **error();**
    - Handles error messages
- **Talk2Server();**
    - Talks to server directly as a client. If the server has recieved a message it will print that in client.

2.3 Module Interfaces

- Void error(const char *msg)
    - Does perror and exits
- Void talk2server(const char *message)
    - After receiving a message this will get the message from a user and send it to the server through a buffer.
- Int main(int argc, char &argv[])
    - Gets the arguments from the command line upon running the program and starts initializing the client to connect to the server.
    - Asks for user registration information and asks who to send a message to.
    - Client strncat() the user's input with the appropriate protocol so that the server can read an understand what to do next. This way the server acts passively.
    - Returns 0 when shutdown is called.

2.4 Overall Program Flow

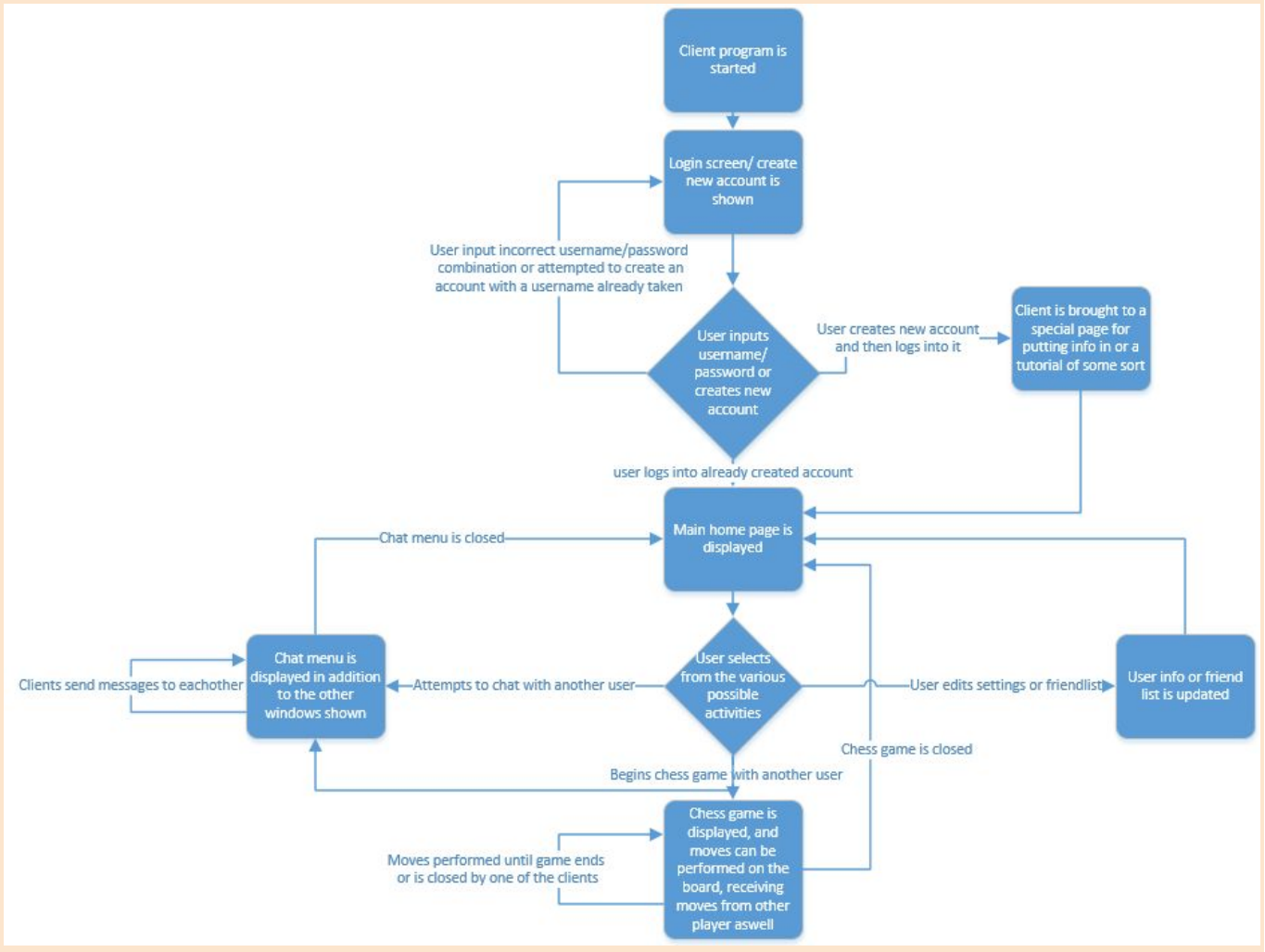

Figure 1: Flow Chart of the overall chat program. From the view of the client. The blue arrows show how it interacts within itself. I.e creating an account and the legal actions it can make.

---

3. Server Software Architecture Overview

3.1 Main Data Types and Structures

```
#include <stdio.h>
#include <stdlib.h>
```

#include <unistd.h>
#include <string.h>
#include <sys/types.h> //contains definitions of a number of data types used in system calls.
#include <sys/socket.h> //includes a number of definitions of structures needed for sockets.
#include <netinet/in.h> //contains constants and structures needed for internet domain address

Data Types and data storage strategy that will be used:

- Lists
- Structs
- Int, char
- Switch statements
- Sockets
- Buffers

3.2 Major Software Components

- **main(int argc, char *argv[ ]);**
  - Takes in user input to begin receiving a connection with a client.
  - Server waits until a connection with the client is established
- **encrypt();**
  - encrypting the user password and storing it in a safe location.
- **LIST *data();**
  - For storing user's information such as log history, friends list, and personal information.
- **Socket()**
  - Creates a new socket. 3 arguments (address domain, type of socket, protocol)
- **bzero()**
  - Initializes server address to zeros. Used for initializing a buffer. I.e before reading from the socket.
- **bind()**
  - Binds a socket to an address. 3 arguments (file descriptor, address to which is bound, and size of the address)

3.3 Module Interfaces

- Int main(int argc, char *argv[ ]);
    - Gets the port number and executes the main function of the program.
    - Returns 0 when the server is terminated.
- Void interaction(int sock);
    - Input is the sock descriptor from the client and handles the messaging portion of the program.
    - Returns nothing.
- Int registration(char *request);
    - Takes in the username or password.
    - Returns a value from creataccount() if 0 then failure user name is already taken if 1 then success an account has been created.
- Int login(char *request);
    - Takes in the username or password
    - Returns a value from loginattempt() if 0 then the username list or password list was empty and cannot be matched. If 1 then there was a successful matching of the login.
- Void message(char *request, char *sendbuf);
    - Takes in the username and user's input message.
    - Outputs nothing
- Void ServerMainLoop(int ServSocketFD, int Timeout)
    - Takes in the server socket file descriptor and an int for when to timeout.
    - Doesn't return anything, since the purpose of this is to just have a loop interaction with the server and the client.
- Int MakeServerSocket(uint16_t PortNo)
    - Takes in the port number provided by the command line.
    - The output of this is the server socket file descriptor.
- Void getuserlist(char *request)
    - The input is a receiving buffer.
    - Prepends the entire userlist for a specific user to receive, only excluding themselves
    - Expected to be in alphabetical order
    - Returns nothing
- Void handleoutgoingmessage(char *SendBuf, char *RecvBuf);
    - The inputs are the user's message that was sent and the receiving message stored in a buffer.

- Counts up how many of the messages destined for a specific user are in the list
- Match with the first message destined for the user in the list.
- Returns nothing.

3.4 Overall Program Flow

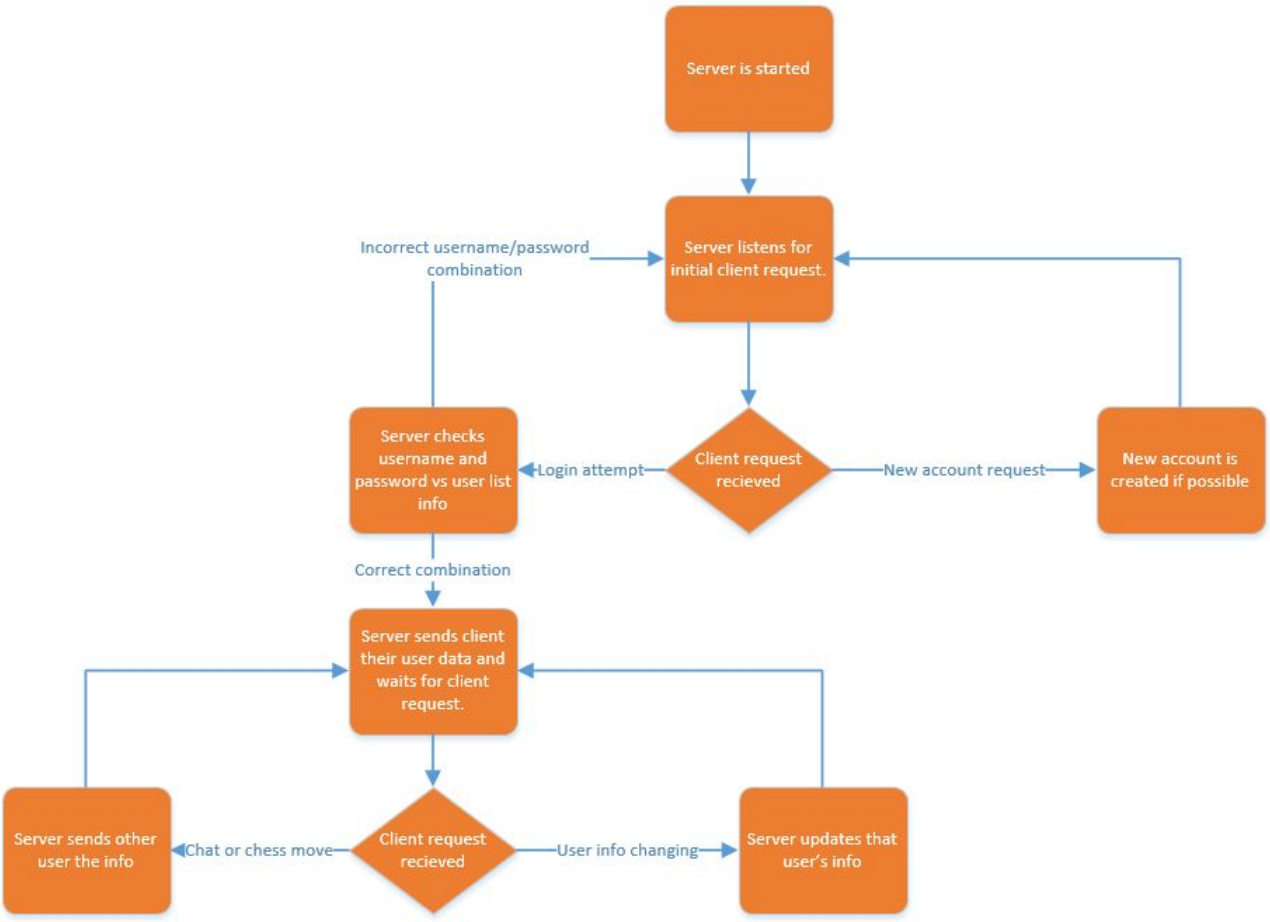

Figure 2: Flow chart of the overall chat program on the server side. Inputs are passed through from client.c after establishing a connection with the server. The stored data in the server (stored as a text file when the server shuts down and later to be read when the server turns back on) will be used to distinguish unique users.

4. Installation

4.1 System Requirements and Compatibility

- Gtk+ Version 2 or greater
- OS: Linux with Kernel version 3.10+
- Processor: 2.0 GHz Xeon E5 2660 v4 or Higher
- Memory: 512MB RAM
- Graphics: 256 MB VRAM with Shader Model 3.0 support and needs a directX 10 capable video card.
- Storage: 200 MB available space

4.2 Setup and Configuration

To ensure that the game runs properly Xming (an X11 display server for Microsoft Windows operating systems) needs to be turned on. After that the user needs to extract the tar file and navigate inside where they will compile and run the code.

4.3 Building, Compilation, Installation

To Untar the tar file

- **Execute: “gtar -xvzf Chat_V1.0_src.tar.gz”**
- **Execute: “make”**

To run the game

Stay in the directory Chat_V1.0_src

- **Type: “make test-all” this starts the server and client**

---

5. Documentation of Packages, Modules, Interfaces

5.1. Detailed description of data structures

Major "global" data

- *Definitions*
  - *Program = NULL; //program name for descriptive diagnostics
  - Shutdown = 0; //keep running until Shutdown ==1
  - BUFFSIZE 256
  - DELIMITER "-"
  - LAST "0" //indicates this message being received is the last one.
  - MORE "1" //indicates more messages are stored in the server buffer and need to be //retrieved
- *Structures*
  - Struct sockaddr_in
    ServerAddress;
  - ULIST *userlist;
  - Char forwardingmessage[BUFFSIZE];
  - Char destiantionuser[MAXNAMELENGTH];
  - Struct message{
    Unsigned int length;
    char opcode[2]; //FAIL/SUCCESS/LOGIN etc
    char user[MAXNAMELENGTH]; //from the perspective of the main.c this is the person who wishes to send the message, or who it was sent from
    char destuser[MAXNAMELENGTH]; //from the perspective of the main.c this is the person who the message is destined for, if it is an incoming message struct, this is the current client for example, but for a sending message it's the target
    char remainingmessages[2]; //LAST if the currently received message is the last one, MORE if there are more messages waiting in the server buffer %warning not guaranteed to correspond with the exact remaining number%
    char content[BUFFSIZE]; //main content of the message sent or received, either password or some message, or a chess move etc
    MOVE *chessmove;
    } Message_struct;

- Protocols
    - FAIL: ‘a’
        - this prefix means that the server response for a request failed
    - SUCCESS: ‘b’
        - server response for request that is successful
    - LOGIN: ‘c’
        - Server will read this protocol as a login attempt and checks to make sure that login was successful.
    - REGISTER: ‘d’
        - Server will read this protocol as a registration attempt and will proceed to save that into a list.
    - MESSAGE: ‘e’
        - Server will read this as an action from a user to send a message to another determined user.
    - PING: ‘f’
        - User pings server for the next item in the waiting queue for that user.
    - GETUSERLIST: ‘g’
        - Prepends all the usernames to the message queue with the requesting user as the target
    - GETFRIENDLIST: ‘h’
        - Prepends all the usernames to the message queue with the requesting user as the target
    - REQUESTCHESS: ‘i’
        - When this is called the chess game starts up
    - CHESSMOVE: ‘j’
        - Client calls for the user’s chess move.
    - SENDFRIENDREQUEST: ‘k’
        - Client sends a signal that it wants to send a friend request to another determined user
    - ACCEPTFRIENDREQUEST: ‘l’
        - Client sends a signal of either accepting the friend request or declining the friend request.
    - USER: ‘user’
        - This is what determines what the username will be when being read from a text file.
    - PASSWORD: ‘pass’
        - As for reading a password this opcode will let the server know the following line will contain the password. Therefore the UserLog can be read and saved into the server when restarting the server.

5.2. Detailed description of functions and parameters

Main file

- The initialization of all the GTK+2 components on the client side
- For the server its the primary code for listening on the ports and interacting with the clients

---

Structures.h file

- This is a structure header file where it contains all the structs that will be used throughout the whole coding process.
- The major structure from the perspective of the server-side is a double linked list of user accounts, storing the relevant data for each user logged in such as username, password, and a inner linked list of the friendlist for that user

---

Server.c file

- This will handle communication from one client to another. Acting as a buffer before sending the message to the other user.
- Clients will connect to this server and read and write through the same port number.
- Multiple clients can be in the same server
- When the user wants to initiate a chess game with another user the moves will be sent as a text file to the server. Where it will be deciphered and pass through to the latter user.

Client.c file

- The user will start the client up and connecting to a server port.
- There will be different functions regarding the personalization of the user's information
- After establishing a connection with the server they are allowed to send and receive messages.
- For the chess portion, the chess game will either be in a separate window or the same window as the chat messenger.

Userlist.c

- Here is where an account is created and where it stores the user's information.
- Double-linked list will be allocated and sorted according to the username. Their information will be stored as an entry.
- Login verification will be here along with an encrypt function to verify if the login was a success.
- This will also include the functions that will save the entry list into a text file. As well as reading the text file when server turns on again.

Messagestructs.c

- Holds the structures that are needed to store a pending message into a list.
- Creates and mallocs a message entry then appends to a list
- Deleting and deallocating all the entries in the list and the list itself is also included in this module.
- Defines the protocols that are used by the server and any other defines that is needed across all other modules.

5.3 Detailed description of input and output formats

For chat portion:

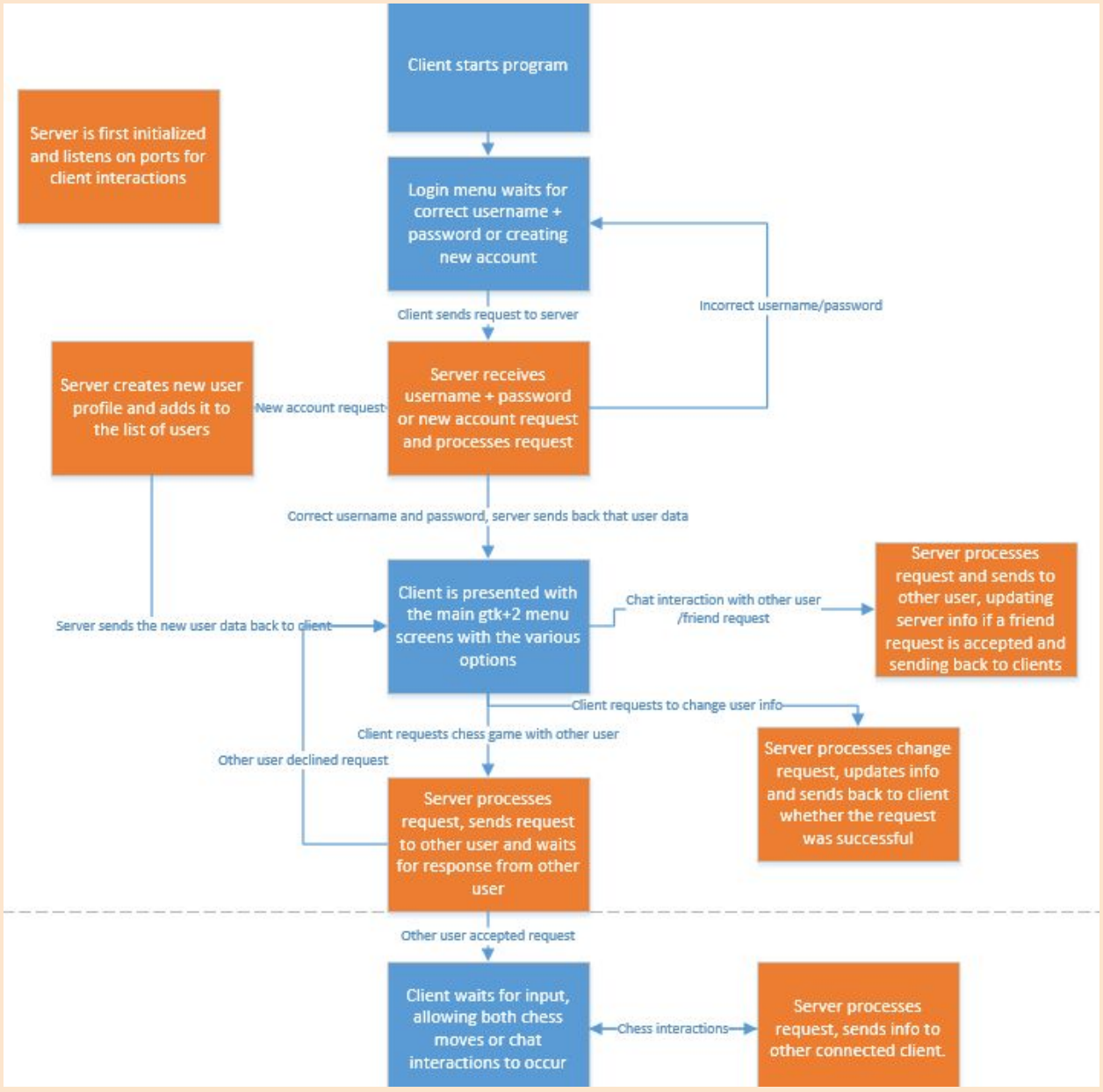

Figure 3: Flow Chart of the overall chat program. Server processes in orange, client processes in blue. The different inputs are directed with blue arrows and depending on what's given, the output will be different.

Chess portion:
The program will wait for user input through clicks, if the program is currently waiting in the menu stage, the click will be compared to the location of the menu options. Selecting the appropriate click if the position of the cursor falls within the button in question.

During the game the player will interact through clicking on the board, which the program will then feed the coordinates to the game.c function call. If the click occured on a square with a player piece, the game.c will then generate a movelist for that piece and return it to main.c which will then feed the movelist to Render.c in order to highlight the possible moves for that particular piece. If the movelist returned is NULL, the program will continue waiting.

On a second click after a piece is selected, the game.c will check if the position of the click corresponds with a position of a legal move that has been highlighted. If it does, the move will be performed and the board updated and the other team's turn is set to go. If the click does not occur on a legal square, the game.c will return null instead and the piece unselected and the squares unhighlighted and the main.c will return to waiting for selection of a piece with none selected. The output will be done through a gtk+2 window with a graphical display of the current board setup or the current menu.

Modules

How many modules will be needed?

- **All the modules in Chess_V1.tar.gz**: includes all the necessary modules to run a chess game with gui.
- **Server.c**: where clients connect to and send information to. Server.c will act as a buffer to send and receive information between two clients.
- **Client.c:** where the user will start registering their information and inputting the commands for the server. Basically a big input module for the user to interact with the server.
- **main.c:** holds the struct for graphical interface pieces. The module that will get gtk+2 to work with the code and have a chess board interface along with the chat messenger.
- **messagestructs.h:** a header file for a list of defines that will be used across multiple modules
- **Userlist.c:** Similar to structures.c for the chess application. Userlist.c will allocate memory and create a double-linked list to sort out the usernames and the information linked with the users. There are also other lists that are created such as encrypt pass, which will encrypt the password from the user login.

- **Structures.c**: There are the structures for the chess game. This is where we would call and received the chess moves.

What are the APIs(application program interface) for each modules?

- Registration:
  - Input: two char arrays, one for the username and the other for the password
  - Output: outputs a 1 for success or a 0 for a fail.
  - Description: Registration function basically looks up the user's name and password in the database to see if its a match or if the name is already taken and then returns a value for the corresponding response. There will be a section where it prompts the user to register their information for the first time as well.
- Encrypt:
  - Input: the username and password char arrays
  - Output: returns 1 for correct password 0 for wrong password
  - Description: this encrypts the password and stores it in a text file. Another function is to verify the password so if the typed password matches the linked username and password then the output is a success otherwise it fails.
- Data_storage:
  - Input: char arrays
  - Output: void
  - Description: This function is using a double linked list to sort out the users and their information. So the list will contain a list of users sorted by name, and it's contents will contain the password and the list of friends.

Functions in userlist.h

```
/* Structure that keeps each username and its corresponding password together */
typedef struct {
        char username[SLEN];
        char password[SLEN];
} USERENTRY;

typedef struct {
        unsigned int length;
        ACCOUNT *first;
        ACCOUNT *last;
} ULIST;
```

/* Creates a new account with the specified username and password. Returns a pointer to the account. */
ACCOUNT *createAccount(char username[SLEN], char password[SLEN]);

/* Creates a new account list and returns a pointer to the list. */
ALIST *createAccountList();

Functions in Server.c

/* Checks if the database contains an account with the specified username. If not, creates a new account with the given username and password. Returns 0 if a new account is successfully created. */
int register(char username[SLEN], char password[SLEN]);

/* Checks if the database contains an account with the specified username. If so, checks whether the given password matches that account's password. Returns 0 if both the username and password match.*/
int login(char username[SLEN], char password[SLEN]);

Functions in Client.c

/* Takes strings entered in the login fields (password should be encrypted, encryption TBD) and reads them into the server, which will handle registration or authentication based on the usernames and passwords stored in the server database. This function will also need to send an additional indicator so that the server recognizes these inputs as a username and password. How this is implemented depends on what data types can be sent via sockets and how. */
void sendAuth(int function, char username[SLEN], char password[SLEN]);

/* Takes the string entered in the text field and reads it into the server. Also takes a list of recipients to send the message to. */
void sendMsg(char msg[SLEN], ALIST *recipients);

Functions in messagestructs.c

MESSAGELIST *createmessagelist(void){
    //mallocs and returns pointer to a created empty message list

```
void appendmessage(MESSAGELIST *list, MENTRY *entry){
    //appends a entry to a list and increments length of list

MENTRY *createmessage(char messagetype[2],char sendinguser[MAXNAMELENGTH], char
targetuser[MAXNAMELENGTH], char newcontent[BUFFSIZE]){
    //creates and mallocs a message entry, returning the pointer

void createandappendmessage(MESSAGELIST *list, char messagetype[2],char
sendinguser[MAXNAMELENGTH], char targetuser[MAXNAMELENGTH], char
newcontent[BUFFSIZE]){
    //creates and mallocs a message entry then appends to the list given

void createandprependmessage(MESSAGELIST *list, char messagetype[2],char
sendinguser[MAXNAMELENGTH], char targetuser[MAXNAMELENGTH], char
newcontent[BUFFSIZE]){
    //creates and mallocs a message entry then pends to the list given

void deletemessageentry(MENTRY *entry){
    //deletes and deallocates a message entry from a list and decrements the length of the list by
1, also properly changing the surrounding linked list structure to accommodate
```

6. Development Plan and Timeline

6.1. Partitioning of Tasks

Manager - Buonkuang Priestley
Reflector - Mason Ma, Sam Longo
Presenter - Andrew Lin
Recorder - Nathaniel Aponte

The Main.c will need to be made first along with Structure.h header file.
The Server.c and **Client.c** can be done in parallel.
Userlist.c holds most of the important components so it should have priority. **Gtk** can be implemented later on.
When doing **Server.c** there will be coordination with **Client.c and Main.c** with the implementation of gtk and storing of information such as user name, user password.

6.2. Team member responsibilities

server.c assigned to **Buonkuang Priestley**

- Connection between multiple clients
- Able to read and write to server.
- Documentation
- Saves accounts when server is offline and checks if a previous user was registered

gtk.c assigned to **Mason Ma**

- Functionality of emojis
- Implementation of chat interface and adjustments to it
- accepting/rejecting friend requests.

server.c assigned to **Sam Longo**

- Registration
- login attempt.
- established a working interaction between 2 clients.
- Backend coding with protocols and messaging flow

client.c assigned to **Andrew Lin**

- Commented the code to allow more understandability.

client.c assigned to **Nathaniel Aponte**

- Added functions for appending and depending friends

Things to do:

- adding the chess game functionality with another user

---

7. Back Matter

7.1. Copyright

Copyright Information

Contact Information

Developer Emails:
Nathaniel Aponte: aponten@uci.edu
Andrew Lin: ajlin4@uci.edu
Sam Longo: slongo@uci.edu
Mason Ma: jiaaom@uci.edu
Buongkang Priestley: bpriestl@uci.edu

Disclaimer of Warranty

Unless explicitly stated to the contrary in this user manual, Off the Grid does not make any warranties, express or implied, regarding merchantability or the ability of this product to meet the specific needs of its user. Off the Grid provides this product “as is.”

7.2. References

"Chess." *Wikipedia*, Wikimedia Foundation, 1 Feb. 2019, en.wikipedia.org/wiki/Chess.
Chess icons and board where retrieved from this website.

Figure 1: created in Visio.

Rules of Chess obtained from "Let's Play Chess Summary of the moves of Chess" created by the United States Chess Federation.

Graphical User Interface being used will be GTK+ 2.0 created by The GNOME Project hosted by Red Hat

---

7.3 Index

Terms: