Developer Manual

HeroChess Version 2.0

May 26, 2021

AVENGINEERS

Paul John Lee, Irania Ruby Mazariegos,
Rachel Villamor, Keane Wong
EECS 22L, University of California Irvine, Irvine, CA 92697

Contents

Glossary of Terms Used in the Implementation

Array: collection of items of the same data types under the same name; provides efficient access to a large number of such elements
Address: location in memory
Buffer: area of memory that temporarily stores data such as inputs and outputs
char: a type of variable that can store characters and letters
Client: any hardware or software that connects, uses, and/or communicates with a server
Connection: session by which information is transmitted and received throughout a network
Error Messages: printed display that notifies the user or server of a problem occurring due to an invalid case involving expected protocols (Refer to the "Error Messages" section for more details)
Host: any hardware that permits service and communication to other hardware or networks
int: a type of variable that can store whole numbers
IP address: a unique string of characters that identifies each computer using the Internet Protocol to communicate over a network
Network: two or more computers connected together for electronic communication
Pointer: a type of variable whose value is information of a variable's location, also known as the address
Port: virtual endpoint that is associated with a host's IP address and serves as a means of communication between a server and application
Protocol: a set of rules that describe the how data is communicated and processed between devices within a network
Server: a computer or computer program which manages access to a centralized resource or service in a network.
Session: a temporary period of communication between hardware or users
Socket: A term used to refer to a single end of a communication channel used to manage data transfer
Variable: a specific name that holds value(s)
void: a data type that has no empty, and is therefore empty

1 Installation

1.1 System requirements, compatibility

To use this version of the HeroChess Multiplayer functionality, the user(s) must have access to the EECS Linux Servers, or a suitable host with a stable release of the program. Users are recommended to have the **Linux version CentOS 6.10** and perhaps even Xming if the graphical user interface is included. The amount of system memory required for the program to run is 512MB and 1GB of disk storage space. All users that use the multiplayer functionality must have access to an internet connection for the duration of their playtime.

1.2 Setup and configuration

Upon downloading the tar package, follow these steps to build and install the game:

1. On the terminal type 'tar -xvzf `P2_V2.0`.tar.gz' and press enter.
2. Then, type 'cd `P2_V2.0`' and press enter.
3. Refer to the next section, "Building, compilation, installation" for further installation instructions.

1.3 Building, compilation, installation

1. Within the `P2_V2.0` directory, type 'make' and press enter. The game's executable has now been generated and installation is now complete.
2. To run the game, type './bin/chess', press enter.
3. Type and enter './client 11900'
4. For those running the server end of this program, follow the same steps up until number 4, where instead you type and enter './server 11900'

2 Client Software Architecture Overview

2.1 Main data types and structures

The Client program is a simple modular interface that primarily communicates with the server through a socket connection established via a port number, IP address, and a host name. A listening socket will be present on the server as listed below, and much of the communication functionality is mirrored in the client-side code. The primary data types include:

- Char *array (strings): The primary medium for input and communication buffers that links both the user console input, and the two sockets. This is the primary means through which protocol codes will be sent back and forth between clients and users, which are then Processed as code.
- PIECE** Array: Pointer type arrays that represent the board. No data on the board will be modified as the board data is only transmitted for display purposes, and the board buffer is cleared or overwritten regularly
- Int: Data type used to represent or handle port numbers and file descriptors on the machine, as well as other primitive data types

Additionally, some structs are used in the programming of this module. While the primary structs are limited, those included are detailed below:
Table 1: Hostent
This is the struct used to represent the host machine that is the server, which is present in the client program but not the server program. This struct is found in the netdb.h files and contains 6 data members

Table 1:

Hostent		
Type	Name	Purpose
Char *	h_name	The 'official' name of the host machine
Char **	h_aliases	Alternative names for the host, a null terminated vector of strings
int	h_addrtype	Host address type, for our purposes it will be AF_INET but may be different if the program is on a different machine of a different host type
int	h_length	Length of each address, in bytes
Char **	h_addr_list	vector of addresses for the host, terminated by a null pointer
Char *	h_addr	Synonym for h_addr_list[0], so it is always the first host address in the address list

Table 2: sockaddr_in
This data struct is a struct found in netinit.h which is used to handle internet addresses, and has 5 data members. Once we have connected to a server with hostent, we use the associated name with gethostbyname to fill in the socket address's sin_addr and then use the sockaddr_in to handle the socket connection
Table 3: in_addr
A data structure used to represent an ipv4 address containing only 1 data member

Table 2:

sockaddr_in		
Return Type	Name	Purpose
short	sin_family	Family of connection, we are using AF_INET
Unsigned short	sin_port	Set to equal portnumber (Note: Always do Htons(portNo) to get the right port number)
struct in_addr	sin_addr	Set to equal Hostent->h_addr_list[0]
char	sin_zero[8]	Not used for our purposes, can be zero'd if needed

Table 3:

in_addr		
Return Type	Name	Purpose
Unsigned long	s_addr	Holds an ipv4 address

Table 4: PIECE
This is the struct used to represent game pieces detailed in project 1, but the details of which are repeated below for convenience:
Piece indicates the chess pieces placed on the board. It will contain the information about the piece type and color that is required for the game to execute player moves, check the win condition each turn and display the updated board.

Table 4:

Struct Piece		
Return Type	Name	Purpose
char	type	To identify the piece type
char	color	To identify the piece color

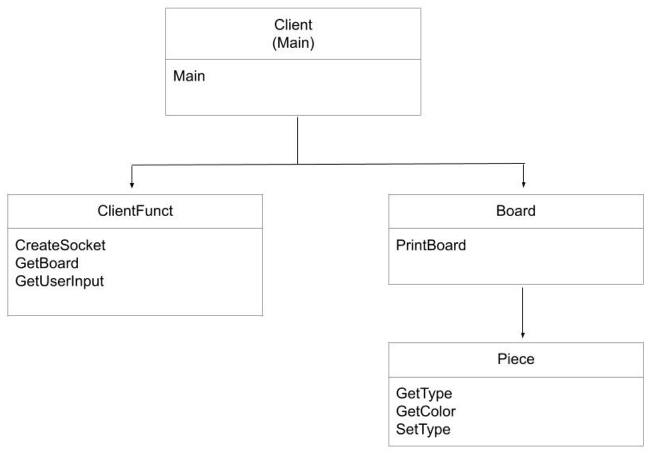

Figure 1: Server-client interaction diagram

2.2 Major software components

Diagram of Module Hierarchy

The client functionality is limited, with primarily only a driver function that determines actions to take after protocol codes are received from the server. In this sense, the client is essentially a basic input and output device, but has almost no bearing on the handling of game logic, communication with other clients, and even input error checking. Most of these are handled through the server side. However, the client does rely on certain functions in a separate file which acts like source code to simplify code. These are stored in a clientFunct file, and the client module also relies on a couple other game files to properly interface with the data. Besides these distinctions, the client side is more limited than the server.

2.3 Module interfaces

API of major module functions
ClientFunct
The client function is a module that handles some of the socket creation and handling overhead. Much of the functions in this module are used to compact and abbreviate the code in main, including input, output, and display

Table 5:

ClientFunct		
Return Type	Name	Purpose
int	CreateSocket	Makes a socket based off a portnumber
int	GetUserInput	Passes in a string buffer as parameter which writes the user console input to it, terminating with a null \0 character Returns the length of the console input
int	GetBoard	Takes in the parameter board and a socket file descriptor to accept the board data from the server into a 2D board buffer

2.4 Overall program control flow

The client program is foremost comprised of the main function, which accepts an argument of port numbers and host names. This main function drives the program, and works by creating a socket, accepting inputs, processing inputs for sending, then sending the inputs to the server. Some of these functionalities are held within the clientFunct file, which black-boxes many of these functions and makes them compact, as well as making the main function slightly more readable. Besides this, the main function does call on some data structures to represent the game board on its end but besides this, the main logic is self contained in the main function.

The client function, much like a computer monitor or a keyboard and mouse, is primarily a display and input device that merely reflects the state of the server as it handles the game, so the client itself actually does not synchronously track the server program as it goes through the stages of initialization and gameplay. Instead, it relies on the opcodes, which are sent in real time, to be accurate to the server state. So, in reality, the main function is simply an indefinite loop that continuously takes in these codes, prints and appropriate response, and sends user input until the exit conditions are reached.

3 Server Software Architecture Overview

3.1 Main data types and structures

The server program is a collection of functions that serve to create sockets, listen for connection, and handle games between connected clients, all the while multiplexing between multiple users. This will be the main area where the game logic, game tracking, and data management will be held. Below are detailed various data structures used in the process of this program. Some will be special to the server, but others will be similar to those found in the client.

Table 6:

Server Global variables		
Variable Type	Name	Purpose
char	strBuffer	Convert strings in printable format
Char *	Program	program name for descriptive diagnostics
int	Shutdown	Trigger to shutdown the program, keep running until Shutdown == 1

3.2 Major software components

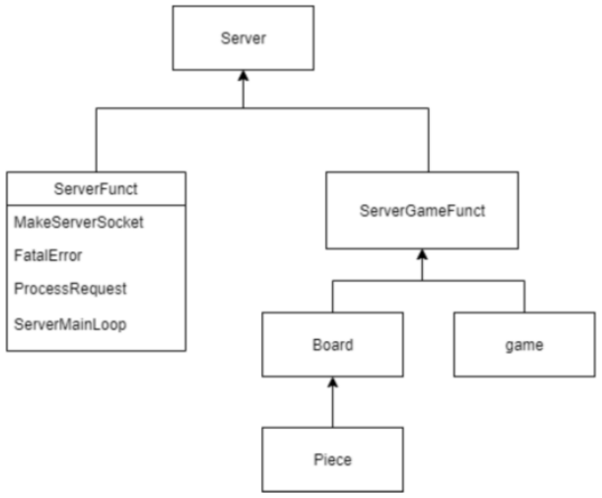

Figure 2: HeroChess V2.0 Module Hierarchy

See the next section, "Module interfaces" for further details.

3.3 Module interfaces

Functions pertaining to the Server directly.

Server		
Variable Type	Name	Purpose
int	MakeServerSocket	Creates a socket associated with the server to work with the server to work with based on the port number inputted into the program on initialization
void	FatalError	Prints a fatal error associated with the problem and shuts down
void	ProcessRequest	Takes in the client side input and performs an appropriate action to deal with it, mostly dealing with op codes
void	ServerMainLoop	The primary loop that drives the program
void	RemoveChar	Removes a char at position t

3.4 Overall program control flow

The overall program

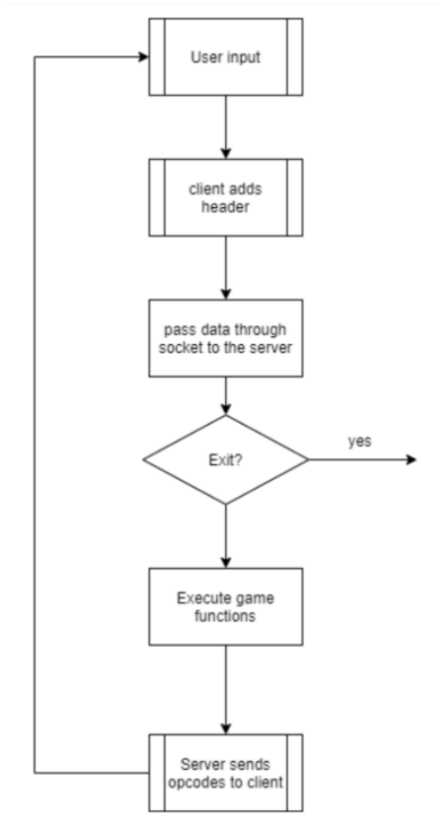
User input
client adds header
pass data through socket to the server
Exit?
yes
Execute game functions
Server sends opcodes to client

Figure 3: HeroChess V2.0 Control Flow Chart

4 Documentation of packages, modules, interfaces

4.1 Detailed description of data structures

Below are samples of the written source code, as implemented in various parts. Detailed descriptions of these functions and their specific functionality can be found in the tables in section **4.3 Detailed description of the communication protocol** starting on page 15.

Sample section of the protocol code handling

```
if(strcmp("FIRST MENU", RecvBuf) == 0)
{
        printf("Please select 1 or 2: \n");
        printf("\t 1. New user");
        printf("\t 2. Returning user\n");
        fgets(SendBuf, sizeof(SendBuf), stdin);
                l = strlen(SendBuf);
                if (SendBuf[l-1] == '\n')
                { SendBuf[–l] = 0;
                }
                printf("%s: Sending message '%s'...\n", Program, SendBuf);
        n = write(SocketFD, SendBuf, l);
        if (n < 0)
        { FatalError("writing to socket failed");
        }
}
```

Source code example of server side protocol codes

```
if (RecvBuf[0] == '+')
{
        RemoveChar(RecvBuf, RecvBuf[0]);
        strncpy(SendBuf, "MOVE FUNCTION EXECUTED", sizeof(SendBuf)-1);
        SendBuf[sizeof(SendBuf)-1] = 0;
        strncat(SendBuf, ClockBuffer, sizeof(SendBuf)-1-strlen(SendBuf));
}

    else if (RecvBuf[0] == '/')
{
        RemoveChar(RecvBuf, RecvBuf[0]);
        strncpy(SendBuf, "FIRST LOGIN FUNCTION EXECUTED", sizeof(SendBuf)-
1);
        SendBuf[sizeof(SendBuf)-1] = 0;
        strncat(SendBuf, ClockBuffer, sizeof(SendBuf)-1-strlen(SendBuf));
}
```

Source code of socket creation

```
int l, n;
int SocketFD,         /* socket file descriptor */
        PortNo;          /* port number */
struct sockaddr_in
        ServerAddress;         /* server address we connect with */
struct hostent
        *Server;         /* server host */
ServerAddress.sin_family = AF_INET;
ServerAddress.sin_port = htons(PortNo);
ServerAddress.sin_addr = *(struct in_addr*)Server->h_addr_list[0];

    SocketFD = socket(AF_INET, SOCK_STREAM, 0);
        if (SocketFD < 0)
        { FatalError("socket creation failed");
        }
        printf("%s: Connecting to the server at port %d...\n",
               Program, PortNo);
        if (connect(SocketFD, (struct sockaddr*)&ServerAddress,
               sizeof(ServerAddress)) < 0)
        { FatalError("connecting to server failed");
        }
```

4.2 Detailed description of functions and parameters

The client's username and password will be stored in a text file database which will only be accessible by the server. The database will allow the server to append new clients to its record as well as ensure that existing clients are entering the right information.

Table 7: Database

Variable Type	Name	Purpose
void	appendUser(char username[100], char password[100])	Creates user record text file if needed. Appends username and password into the text file when a new user registers for an account.
int	checkUser(char user[100])	Searches text file for the user input. It will be used twice in the login process; once to check that the user exists and second to check that the password is correct. Returns 1 if true and 0 if false.
int	changePass(char username[100], newPass[100]);	Changes the password of an account if the user does not remember their current password. Searches the record for the username and then replaces the old password with a new password. Returns 1 if successful and 0 if unsuccessful.

4.3 Detailed description of the communication protocol

General order of execution:

1. Client accesses the listening server
2. Server accepts
3. Client sees server has accepted
4. Server Requests login
5. Client Sends Username
6. Server confirms appropriate username
7. Server requests password
8. Client sends password
9. Server confirms password
10. Server signals it is asking for the opening menu option
11. Client displays menu and sends the user's input
12. Server confirms input
13. Server sends board data
14. Client receives and displays board data
15. Client takes user move input
16. Client sends user input to server
17. Server executes or rejects move
18. Repeat 13-17 until a checkmate is achieved
19. Server signals a win and ends the game
20. Client displays the win
21. Repeat 1-20 until server is shut down

Table 8: Protocol Codes for Client to Server Table Part 1

Server to Client Table Pt. 1		
Protocol Code + Example Input	When it shows up	Usage
“FIRST_MENU”	The initial menu asking if the user wants to be a new or returning user	The Client program prints "Please select 1 or 2 1. New user 2. Returning user" And takes the input, (ASCII version of ’1’ or ’2’ char, not the integers 1 or 2)
“NEW_USERNAME”	Asking for the new username	The client program prints "Welcome, nice to meet you! Please enter your new username for HeroChess (Suggestions: BlackXwidow, Fe_Man, CorporalAmerica48):" Then takes in the new name from console and sends it through socket to server
“REQUESTING_USERNAME"	Server is asking the client to send a username	The client program prints "Please enter your username:" Then takes in a string from console and sends it to the server through socket
“INVALID_USERNAME”	Server signals the client did not succeed in inputting a pre-existing username	The client program prints "Invalid Username" This protocol does NOT cause the program to take in any input, it just causes the client to print the error message. Relies on the server sending another REQUESTING_USERNAME to request another username
“REQUESTING_PASSWORD"	Server is asking the client to send a string of password	The client program prints “Please enter your password” And takes in an input for the password from console, sending it to the server

Table 9: Protocol Codes for Server to Client Table Part 2

Server to Client Table Pt. 2		
Protocol Code + Example Input	When it shows up	Usage
“NEW_PASSWORD”	Server is asking for the password associated with the new username	The client prints “enter a new password for the new username” And takes in an input for the password from console, sending it to the server
“INVALID_PASSWORD”	Server signals the client did not succeed in inputting a password in wrong format	The client prints “Invalid password” This does NOT cause the program to take in any input, it just causes the client to print the error message
“REQUESTING_MOVE”	Server is asking the client to send a string of move <SOURCE><DESTINATION>	The client prints ”Please enter a move” Takes in an input for the move from console sending it to the server
“INVALID_MOVE”	Server signals the client that the input move is invalid	The client prints “Invalid move” and "Please enter another move" on the next line Does NOT take in another input, requires server to send another ’requesting move’ request to get input

Table 10: Protocol Codes for Server to Client Table Part 3

Server to Client Table Pt. 3		
Protocol Code + Example Input	When it shows up	Usage
“PRINT_BOARD”	Server signals the client to print the current board status. Signals that the server is also about to send the board data in the form of a PIECE** array buffer	Client listens for the buffer and receives it. The client prints the current board in the right format
“SUCCESSFUL_MOVE _CHECK_W"	Server signals the client to print a successful move has been made and that the white player is in check	Client prints “White player is in check”
“SUCCESSFUL_MOVE _CHECK_B"	Server signals the client to print a successful move has been made and that the black player is in check	Client prints “Black player is in check”
“WIN_ACHIEVED W"	Server signals the client that the game has been won by white	Client prints “Checkmate; Player White wins!”
“WIN_ACHIEVED B"	Server signals the client that the game has been won by black	Client prints “Checkmate; Player Black wins!”

5 Development plan and timeline

5.1 Partitioning of tasks

- Rachel / Irania : Database, GUI
- Keane / Paul : Server, Client

5.2 Team member responsibilities

- **Members and roles**
  - Manager: Paul Lee
  - Presenter: Keane Wong
  - Recorder: Rachel Villamor
  - Reflector: Irania Mazariegos

5.3 Timeline

Timeline Overview

1. Planning - Week 7
2. Development - Week 7 to 8
3. Prototyping - Week 7 to 8
4. Testing - Week 7 to 9
5. Pre-launch - Week 9 to 10
   (a) Alpha Release - Week 10
   (b) Beta Release - Week 10
6. Launch - Week 10 to Finals
   (a) Master Release - Finals
7. Competition - Week 9

6 Back matter

Copyright

References

Chess Against Computer Expert-Chess-Strategies.com - Hr.prodaja2021.com." n.d., hr.prodaja2021.com/content?c=play ches online vs cpuid=2. Accessed 27 April 2021.bib-itemTextbook [1] "Play Chess Online vs Cpu P

Index