4981 Linux Chat Pseudocode

Juliana French A00998091

Alex Xia A00991905

Table of Contents

Main

Main function

```
// entry point of program
Forever loop:
        Prompt user for input:
        If input is C // Client mode
                Create new client object and let it run
                // client mode code goes in its constructor
        If input is S // Server mode
                Create new server object and let it run
                // server mode code goes in its constructor
        If input is E // Exit
                Break (and exit program)
```

Client

Class for client mode
Runs very similar functions to Server
Instance of client, meant to live on stack
Leaving class means leaving client mode & returning to main's loop

Constructor

```
Prompt user for ip address to look for server
Prompt user for port to connect to on server
Create socket to listen for server packets
If socket is null
        Exit program with error msg
Call API function to get host at the entered Ip address
If host is null
        Exit program with error msg

Initialize server socket & port for connection
Call API function to connect to socket

If connect failed
        Exit program with error msg
```

Set socket to non-blocking now

Prompt & confirm if user wants a printed chatlog
If yes, get and open a filename
    // no need to store instance boolean on whether printed chatlog was
    // selected. Check if file is ready for writing before writing to it

// connected & set up, client can begin sending now
Print instructions for user, they can start using the application now

// background thread handles receiving from server via select
Start SelectThread in background
// main thread handles reading & sending user messages
Forever loop:
    Call function to Handle new line of user input
    If returned false
        Break
Stop SelectThread
Close socket to server
Return to idle

Client SelectLoop

Runs on background thread, so main thread can still get user input
Handles receiving from server & printing to console

Takes socket descriptor to server as argument

Initialize file descriptor set to 0
Add server socket desc to set

Initialize disposable file descriptor set
// set control variable to false from main thread to stop loop
While control variable is true
    Set disposable file descriptor set to original descriptor set

    Call API function select with following arguments:
    - serverSocket as max file descriptor
    - disposable file descriptor set to check for read ready
    - Timeout of 2 seconds
    Once select returns
    If serverSocket is in the read ready set

Call ReadServerMessage to pull data from socket

// check this return value, if false, break from main client loop
HandleNewLine
// code for main loop running on main client thread
// gets user input & handles it depending on if it's a msg or a cmd
Get a line of text from user
If char[0] is '/'
Get first word from line
If first word is /help
Print out a help message
If first word is /disconnect
Return false
Else
Print out msg to alert user of invalid cmd
Otherwise, not a command
Add timestamp string to packet
// client->server packets have no IP info in the data
Call function to send packet to socket
Return true

ReadServerMessage

Should be called on background thread
Select should already check for data, this assumes there's data on socket

Takes the server's socket desc as argument
Alloc defined buffer-length buffer, initialize it to 0
Read bytes from socket
Set buffer tail pointer
While bytes read > 0
Point tail pointer to end of buffer
Increment total bytes read by bytes read
Decrement bytes remaining
Read bytes remaining from socket
If total bytes read > 0
Print buffer contents
If chatlog file is open
Append buffer contents to file
Else
// total bytes read is 0
// only time select returns true & server socket has 0 bytes is if server

```
        // disconnected, stop all sending/receiving
        Free up buffer memory
        Exit program with error msg
Free up buffer memory
```

Server

Class for server mode.
Server very similar to client, but doesnt need IP from user and does not send messages of its own.

Constructor

```
Prompt user for port to listen for connections on
Create socket from that port
If socket is null
        Exit program with message

Initialize server socket & port for connection
Set socket to allow connection from any IP
Bind socket file descriptor to IP

Prompt & confirm if user wants a printed chatlog
If yes, get and open a filename
        // no need to store instance boolean on whether printed chatlog was
        // selected. Check if file is ready for writing before writing to it

// server is running, user can begin to admin it now
Print instructions to user on running server, similar to using a client

// background thread handles receiving from server via select
Start SelectThread in background
// main thread handles reading & executing user cmds
Forever loop:
        Call function to Handle new user cmd
        If returned false
                Break
Stop SelectThread
Close socket to server
Return to idle
```

Struct ClientInfo

Only needed in Server SelectLoop, store a socket desc together with its IP

Members:

- File descriptor : integer
- Ip Address : string

Server SelectLoop

Runs on background thread, so main thread can still get user input.
Handles receiving from clients & printing to console/file.

Takes socket descriptor to server's listening socket as argument
Initialize max file desc to listening socket's file desc

Initialize new array of ClientInfo structs, as many as system'd allow
    Set all file descriptor to -1, invalid value

Initialize file descriptor set to represent all available file desc
    Set to 0
Add server socket descriptor to set

Create empty file descriptor set for reading
// set control variable to false from main thread to stop loop
While control variable is true
    Set read file desc set to all file desc set

    Call API function select with following arguments:
    - server's listen socket as max file desc
    - read file desc set to check for read ready
    - Timeout of 10 micro secs
    Once select returns
    If listen socket is in the read ready set
        Call AddNewClient to add it to ClientInfo array
        Add new client's socket desc to all file desc set
        If new client's socket desc > max file desc
            //use new client's socket desc in all future select calls
            Set max file desc to new client's socket desc
    Initialize empty list of messages

For loop through ClientInfo array
For each untaken file desc // file desc value not -1
If its read ready // found in set returned by select
Call ReadClientMessage to recv from it
If 0 bytes recv
Close the socket & remove it from set
Set its index in ClientInfo array back to -1
Call BroadcastMessages to send all existing messages to clients
//endwhile

AddNewClient

Should call select on listen socket first, assumes there is new client waiting.

Takes 2 arguments:
- Server's listen socket to accept & extract new client socket from
- Reference to a ClientInfo array to add new client info to

Call accept on server listen socket
If no new socket returned
Exit program with error msg
Print alert of new client joining to console
If chatlog file open
Append alert of new client joining to file
Set new client socket to non-blocking
For loop through ClientInfo array
Find first unused file desc (value = -1)
Set value at the index' file desc to client socket
Set value at the index' ip to client ip address
Break
Return client socket desc

ReadClientMessage

Should be called on background thread. Select should already check for data, this assumes there's data on socket.

Takes as argument:
- a specific client socket desc
- Reference to ClientInfo array
- Reference to List of messages (to be broadcasted)

Alloc defined buffer-length empty buffer,

```
Read bytes from socket
Set buffer tail pointer
While bytes read > 0
        Point tail pointer to end of buffer
        Increment total bytes read by bytes read
        Decrement bytes remaining
        Read bytes remaining from socket
If total bytes read > 0
        Add timestamp string to packet
        Add client ip address string to packet
        Add completed packet to end of list of messages
// total bytes read = 0
        // client has disconnected
        // let SelectLoop handle it
Free up buffer memory
Return total bytes read
```

BroadcastMessages

Pseudo-broadcasting, sends every msg to all clients except message author.

```
Take as argument:
    - Reference to ClientInfo array
    - Reference to list of messages
For loop through list of messages
        For loop through ClientInfo array
                If client file desc at index is valid // value != -1
                        Extract ip from message/packet
                        // msg author, should only happen once / msg
                        If extracted ip = current client at index's ip
                                Print msg to console as server
                                If chatlog file open
                                        Append msg to file
                        Else
                        // not author of msg, needs to get it
                                Send msg as packet to current clien at index


                //end if
        // endfor
//endfor
Clear list of messages to make way for next set
```

```
// check this return value, if false, break from main server loop
Handle command
        // code for main loop running on main server thread
        // gets user input & handles it depending on if it's a msg or a cmd
        Get a line of text from user
        // all cmds start with /
        If char[0] = /
                Get first word from line
                If first word is /help
                        Print out a help message
                If first word is /disconnect
                        Return false //dont call this func again
                Else
                        Print out msg to alert user of invalid cmd
        // not a cmd: first char is not /
        // as server, does nothing
        Return true // call this func again
```

SharedUtils

Since Client & Server uses many of the same functions, many of them can go in separate file

PromptForInteger

```
Takes 1 string msg as argument
Forever loop:
        Repeatedly prompts user with msg until the first integer parsed
        from input is an integer and is not 0
        // use PromptForString to get user string, parse result
Return parsed integer
```

PromptForString

```
Takes 1 string msg as argument
While loop true forever
        Repeatedly prompts user with msg until non-empty string read in
        // only read first string entered
Return read-in string
```

PromptForChatlog

While loop true forever
    Prompt user for a ‘y’ or ‘n’ if they want to save chatlog to a file
    If first char entered is y // yes print chatlog file
        Prompt user for a fileName or a space they want to use
        If a string containing only spaces read in
            Return string “ChatLog.txt”
        Else
            Return fileName read-in
    If first char entered is n // no dont print chatlog file
        Return empty string // calling func will check for this
    Print alert to user to make sure first char entered is only y or n

PrintWelcomeMessage

Should take 3 arguments:
- mode, either Client or Server
- IP of the room host
- chatlogOn, whether to print chat log or not as decided by user
Print available cmds and how current mode works to user
Add fancy styling, should be only function to do so

PrintHelpMessage

Print the available commands from PrintWelcomeMessage again

Die

// needless to say, only call this if errno is set
Takes 1 msg as argument
Call perror with passed-in msg
Exit program

SendPacket

Takes 2 arguments:
- socket file descriptor of where to send packet
- packet as a string

Call API function send to send as TCP packet

GetTimeString

// returns time in format <HH:MM:SS>
Create buffer to hold time string
Call API functions to get localtime of system
Store into buffer with format <HH:MM:SS>
Return buffer as string

GetIpFromPacket

Takes 1 complete packet(ip, timestamp) as arguments
// since packet format is < IP >< system time >: msg
Extract substring between first < >
Initialize startIndex var to index of first < in packet
Initialize endIndex var to index of first > in packet
Get substring of packet starting at startIndex, for length of endIndex - startIndex
Return substring

GetIpedPacket

IPs are only added by server to client-sent packets

Takes as arguments:

- IP of destination/original author
- Packet in timestamped format

Get time string
Return packet in format: “< ip >< system time >: msg”

GetTimestampedPacket

Takes 1 msg(packet body) as argument
Get time string
Return packet in format: “< system time >: msg”