Jacob Hoog

Username-Password Combination Security System

Final Project Report

Abstract

The core of this project uses the concepts of a client/server system and network communication to create a username and password combination security system that protects each individual users account data and allows accessibility to the system from any server program that is connected to the network.

Introduction

Online security is a topic that interests me and is an issue in our current society that has been at the forefront of many news stories. Many large businesses have had issues recently with properly protecting their customer's data. The main objective of this project is to create a password security system which utilizes sequential button presses by a user to check if they are to be allowed to alter that specific account's data. The security system is attached to a simulated banking program which serves as an example of a business where account security is crucial. In this model, messages are passed back and forth between a client program and server programs to achieve the various functionalities necessary for the system, such as updating user funds and receiving in a password. All user data is also stored locally to a file, so that it can be opened upon initialization of the system’s client, updated as the system runs, and saved when the system is shut down.

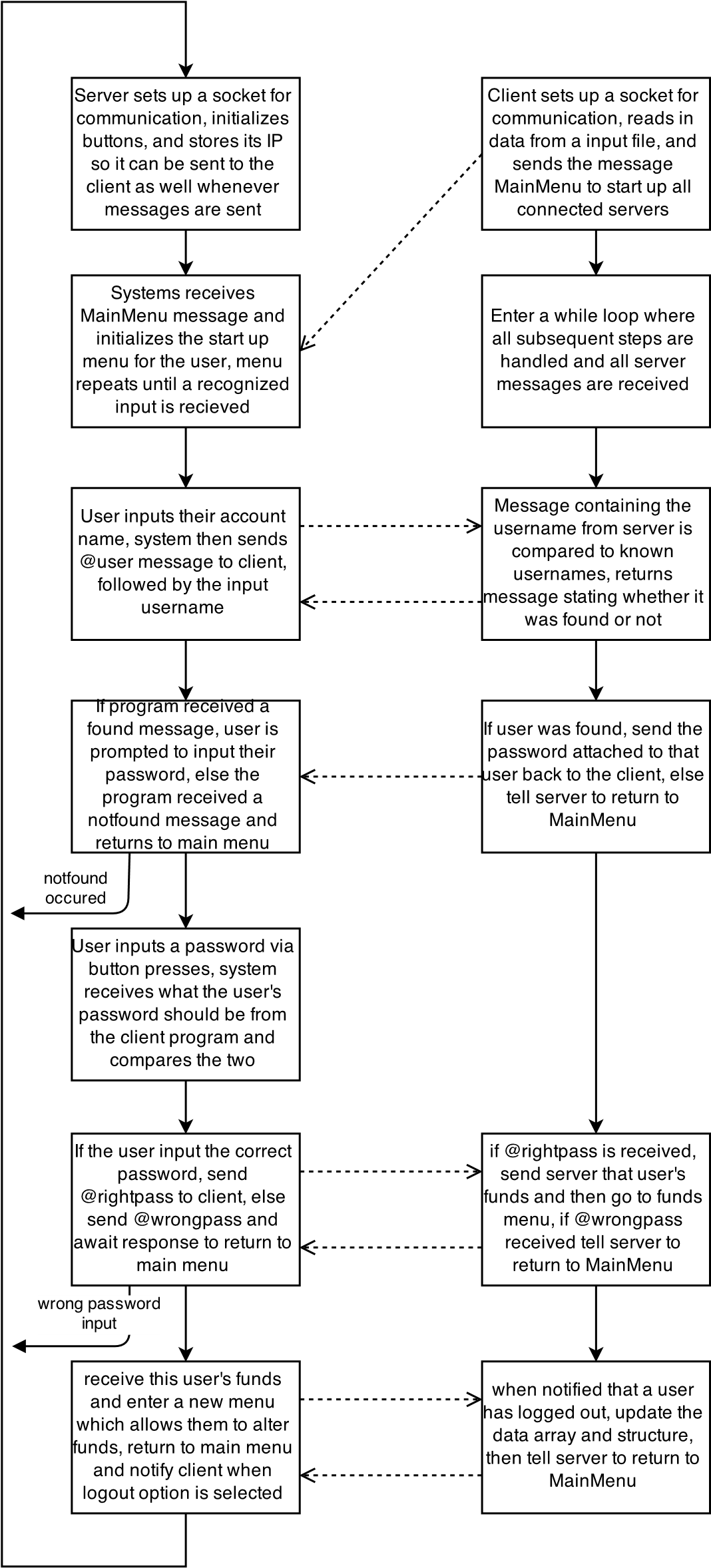
Background

The main concept of a client/server system comes from the experiments we have run within the lab of this course. In any client/server system, the client program can be used as a controller to relay information to various server programs connected to the network. This fits a security system excellently, where a user might desire to access their data from various locations. In the lab, we also came across cases where a server program will need to pass messages back to the client program. This process proved useful in creating this project and allowing for quick passing of data between the client and a server program. For example, a server program may send the client a message indicating a user's password was verified. Instead of waiting for an administrator to tell the client to send a message back, the client automatically sends a message containing that specific user's data to the server and then sends a second message that tells the server to bring up a menu for the user. A more in depth look at the application of this communication between systems is provided in the implementation section of this report below.

Implementation

The program requires at least one server program and client program to be initialized and running within the network on the same port specified at runtime. Each program should be run through a TS-7250 circuit board, with any server programs also having access to an auxiliary board with push buttons that are able to be mapped to bits 0 through 4 of data register B. A list of known users, there associated passwords, and their account funds are read into the client program from a file and saved into an array of structures. After this, the client program is prompted to enter the message ‘MainMenu’ to startup all server programs within the network. This message is then broadcast to all connected servers, which prompts them to provide a menu to the user. It is important to mention that any time a server is sending messages to the client, the server also sends a message with its ip. The client uses this to ensure that sensitive data is only sent back to the server system which is requesting it instead of globally broadcasting to all programs. From this menu, users can select that they wish to input their account name. A server program which has had a username input sends the message ‘@user’ to the client then sends whatever username was input into it. The client program recognizes that a server is trying to verify a user from the ‘@user’ message and then compares the username the server sent to known users. If the username was not found in list of known users, the client sends the message ‘notfound’ followed by another ‘MainMenu’ message to return the server to its initial state. If instead the user was found, the client sends a ‘found’ message to the server followed by what that specific user’s password. The server program who had a recognized user asks them to input a 4 digit password with button presses on an auxiliary board and compares the input password to the one received from the client. If the password is recognized, the server sends ‘@rightpass’ to the client, otherwise it sends ‘@wrongpass’. The client sends the server the ‘MainMenu’ message is the wrong password message is received. If the client instead is told that the password was correct, it sends that user’s account funds to the server and then a second ‘MoneyMenu’ message. The server program receives both these messages and opens up a new menu allowing the user to add or remove funds, or exit back out to the main menu which sends a ‘@logout’ message to the client followed by a second message containing the user’s new fund amount. The client program takes in these messages, updates the array of structures and writes the new values back to file. The client then sends the message ‘MainMenu’ back to the server to restart the system for the next user to use that server program.

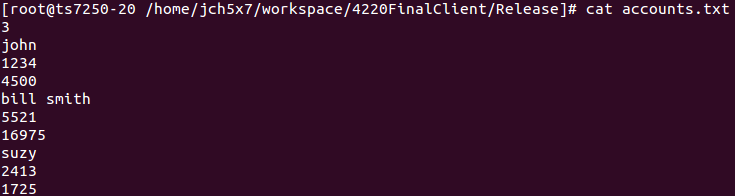
System Flowchart, messages between programs are denoted by dotted lines



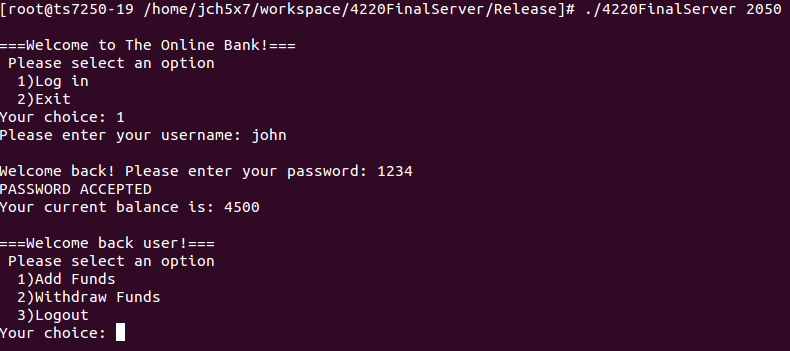
Experiments and Results

I ran this project in a single server and in multiple server scenarios to make sure the results were consistent. It was while testing multiple server situations that I found the importance of sending the client a way of identifying which server was messaging it. Without the client having this knowledge, it was sending its messages to all servers, so systems that were not awaiting a response at the time were still getting messages that they may not be ready to receive yet. Any server program connected to this system is aware of its own IP address and sends this to the client as a message so the client can use it as a return address when needed by updating the sin\_adder structure member of variable ‘anybody’ that is used in calls to the sendto function. Areas heavily dependent on user input, such as the menu for altering funds, have additional checks to ensure that whatever the user is inputting is actually a valid amount. I tested these checks with cases such as symbols and strings of characters to ensure that non-numerical inputs would be rejected and the system would not wrongly try to add them to the user’s total funds. The work done between the client and any one server is roughly split evenly between the two, with the largest amount of the effort either program performs being used in sending and receiving messages. The video demonstration shows the system running one client program and two server programs with each being on its own circuit board. The results of this test show that the client can send messages to a one circuit board at a time, which is the desired outcome in this situation. The demonstration also shows that the data the client has access to be updated every time a user closes an interaction. This means that a user can log into one terminal, make a transaction and logout, then login from a different location and have access to the funds which they had just altered from the first terminal.

Example of an acceptable input file, first value in the file is the number of users



Example of a server running and some of the inputs it accepts



Discussion and Conclusions

The current form of this project reaches its expected goals and produces the desired results. Users are able to access their data through a username-password security system that utilizes the client/server model system. An issue I came across while coding this project was ensuring that a user’s data was updated for other systems to use after any one server alters their account. I solved this issue by both updating the array of structures found on the client program and writing the new values to file after each interactions. By doing this, the new data values are now accessible to other server systems and are kept up to date for the client to read in any time it is booting up. Another issue I came across was the client broadcasting every message it tried to send to all currently connected server systems. This could lead to one user’s data being passed to a machine they were not currently using, which is not the desired outcome. I fixed this problem by having any server that is sending messages to the client send an additional message containing its IP address. The client can use this message as a destination for messages it sends back, so only the server whose address it received will hear back from the client. The largest limitation that I can see for this project would be delays as the number of known usernames increases. This would cause the amount of time taken to save to the local data file to increase and would slow down the whole system. A future implementation of this program could improve on this issue by making the writing to file portion of code its own thread, allowing it and the rest of the program to run at the same time. Overall, I would consider this project a success which has reached its original goal in making a username-password security system.

Appendices

/\*

\* 4220FinalClient.c

\*

\* Created on: Apr 18, 2015

\* Author: jch5x7

\*/

#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>

#include <arpa/inet.h>

#define MSG\_SIZE 40 // message size

typedef struct //all relevant data for a user

{

char\* name;

char\* password;

char\* totalvalue;

}account;

void error(const char \*msg)

{

perror(msg);

exit(0);

}

int main(int argc, char \*argv[])

{

int sock, n;

unsigned int length;

struct sockaddr\_in anybody, from;

char buffer[MSG\_SIZE]; // to store received messages or messages to be sent.

int boolval = 1; // for a socket option

FILE\* in;

char\* accountbuf = malloc(sizeof(char)\*40);

int numofacc, count;

int founduser = 0;

//Ensure program was initialized with right number of arguments

if (argc != 3)

{

printf("usage: %s port data\_file\n", argv[0]);

exit(1);

}

//Open input file to gather data

in = fopen("accounts.txt" , "r");

if(in == NULL)

{

printf("Input file could not be read\n");

}

fgets(accountbuf , 40 , in);

numofacc = accountbuf[0] - '0';

account acclist[numofacc]; //list of all known users and their data

//Read in accounts and there values to be checked against input

for(count = 0 ; count < numofacc ; count++)

{

fgets(accountbuf , 40 , in);

acclist[count].name = strdup(accountbuf);

fgets(accountbuf , 40 , in);

acclist[count].password = strdup(accountbuf);

fgets(accountbuf , 40 , in);

acclist[count].totalvalue = strdup(accountbuf);

}

sock = socket(AF\_INET, SOCK\_DGRAM, 0); // Creates socket. Connectionless.

if (sock < 0)

{

error("socket");

}

//Change socket permissions to allow broadcast

if (setsockopt(sock, SOL\_SOCKET, SO\_BROADCAST, &boolval, sizeof(boolval)) < 0)

{

printf("error setting socket options\n");

exit(-1);

}

anybody.sin\_family = AF\_INET; // symbol constant for Internet domain

anybody.sin\_port = htons(atoi(argv[1])); // port field

anybody.sin\_addr.s\_addr = inet\_addr("10.3.52.255"); // broadcast address

length = sizeof(struct sockaddr\_in); // size of structure

//Client loops until administrator either chooses to exit or go to MainMenu

while(1)

{

bzero(buffer, MSG\_SIZE); // clear buffer

printf("Please enter \'MainMenu\' to start up server programs or ! to exit: ");

fgets(buffer , sizeof(buffer) , stdin);

if((strncmp(buffer , "MainMenu" , 8) == 0) || (buffer[0] == '!'))

{

break;

}

}

//Triggers when MainMenu was input, alerts user that client is entering an automated response system

if (buffer[0] != '!')

{

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

printf("\n\nThank you for choosing to use our program!\nThe client will not enter an automated mode"

" to manage the connected servers.\nIf you wish to close the client,\nplease press the ctrl"

" button and the letter c from your terminal.\n\n");

}

while(1)

{

//Immediately break loop if client was told to quit

if (buffer[0] == '!')

{

printf("Exiting client system\n");

break;

}

//Receive message

bzero(buffer,MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

//When a server closes update the local data file

if(strcmp(buffer , "@close\0") == 0)

{

printf("\nINTERNAL: a server exited, updating data file\n");

in = fopen("accounts.txt" , "w");

fprintf(in , "%d\n" , numofacc);

for(count = 0 ; count < numofacc ; count++)

{

fprintf(in , "%s" , acclist[count].name);

fprintf(in , "%s" , acclist[count].password);

fprintf(in , "%s" , acclist[count].totalvalue);

}

fclose(in);

}

//Username was input, search for it and send feedback to server

else if(strcmp(buffer , "@user\0") == 0)

{

printf("\nINTERNAL: checking if user is recognized\n");

bzero(buffer,MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

printf("Received something: %s\n", buffer);

founduser = 0; //flag determining if user is found is reset before each search

//Attempt to locate user in array

for(count = 0 ; count < numofacc ; count++)

{

if(strcmp(acclist[count].name , buffer) == 0)

{

printf("\nINTERNAL: user found, sending password back\n");

founduser = 1;

//The following 5 lines ensure that only the server who sent the message receives the responses, occurs any time client sends sensitive data

bzero(buffer, MSG\_SIZE);

n = recvfrom(sock, buffer, 10, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

anybody.sin\_addr.s\_addr = inet\_addr(buffer);

bzero(buffer , MSG\_SIZE);

strcpy(buffer , acclist[count].password);

n = sendto(sock, "found\n", 6 , 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

//Sends the server system the password linked to this user

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

break;

}

}

//If user not found, send main menu

if(founduser == 0)

{

printf("\nINTERNAL: user not found\n");

//Updating who recipient is

bzero(buffer, MSG\_SIZE);

n = recvfrom(sock, buffer, 10, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

anybody.sin\_addr.s\_addr = inet\_addr(buffer);

n = sendto(sock, "notfound\n", 9, 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

n = sendto(sock, "MainMenu\n", 9, 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

}

}

//Password was correct, send user there funds so that they can alter them and give them a new menu

else if(strcmp(buffer , "@rightpass\0") == 0)

{

printf("\nINTERNAL: User's password was correct, proceeding to fund access.\n");

//Updating who recipient is

bzero(buffer, MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

anybody.sin\_addr.s\_addr = inet\_addr(buffer);

//Send server system this user's total funds

bzero(buffer,MSG\_SIZE);

strcpy(buffer , acclist[count].totalvalue);

n = sendto(sock, buffer, MSG\_SIZE, 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

bzero(buffer,MSG\_SIZE);

strcpy(buffer , "MoneyMenu\n");

n = sendto(sock, buffer, MSG\_SIZE, 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

}

//Password was not found, return user to main menu to try again if they wish

else if(strcmp(buffer , "@wrongpass\0") == 0)

{

printf("\nINTERNAL: User's password was incorrect, closing login attempt.\n");

//Updating who recipient is

bzero(buffer, MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

anybody.sin\_addr.s\_addr = inet\_addr(buffer);

n = sendto(sock, "MainMenu\n", 9, 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

}

//User logs out and new funds is received, return to MainMenu

else if(strcmp(buffer , "@logout\0") == 0)

{

printf("\nINTERNAL: User logged out, taking in there updated funds to save to records.\n");

//Receive the altered funds and store them back into the array

bzero(buffer , MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

strcpy(acclist[count].totalvalue , buffer);

//File is updated when a user logs out, so if user logs in somewhere else, their updated funds will be available

in = fopen("accounts.txt" , "w");

fprintf(in , "%d\n" , numofacc);

for(count = 0 ; count < numofacc ; count++)

{

fprintf(in , "%s" , acclist[count].name);

fprintf(in , "%s" , acclist[count].password);

fprintf(in , "%s" , acclist[count].totalvalue);

}

fclose(in);

//Updating who recipient is

bzero(buffer, MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&from, &length);

if (n < 0)

{

error("recvfrom");

}

anybody.sin\_addr.s\_addr = inet\_addr(buffer);

n = sendto(sock, "MainMenu\n", 9, 0, (const struct sockaddr \*)&anybody,length);

if (n < 0)

{

error("Sendto");

}

}

}

//Write list back to file with updated values to be used upon next startup if client is closed

in = fopen("accounts.txt" , "w");

fprintf(in , "%d\n" , numofacc);

for(count = 0 ; count < numofacc ; count++)

{

fprintf(in , "%s" , acclist[count].name);

fprintf(in , "%s" , acclist[count].password);

fprintf(in , "%s" , acclist[count].totalvalue);

}

fclose(in);

close(sock);// close socket.

return 0;

}

/\*

\* 4220FinalServer.c

\*

\* Created on: Apr 18, 2015

\* Author: jch5x7

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/mman.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <fcntl.h>

#define MSG\_SIZE 40 // message size

void error(const char \*msg)

{

perror(msg);

exit(0);

}

int main(int argc, char \*argv[])

{

int sock, length, n;

int boolval = 1; // for a socket option

socklen\_t fromlen;

struct sockaddr\_in server;

struct sockaddr\_in addr;

char buffer[MSG\_SIZE]; // to store received messages or messages to be sent.

struct hostent\* thishost; //variables needed for storing this server's ip

char hostptr[64];

size\_t hostlen = 64;

char\* myip;

char mainmenu[20]; //various variable and flags used in handling user input

char moneymenu[20];

char thisuser[20];

char chkpass [5] = "0000";

int passcount = 0;

unsigned long password = 0;

long currmoney = 0;

char chkmoney[20];

char\* ismoneyammount;

long adjmoney;

//Ensures right number of arguments were received

if (argc < 2)

{

printf("usage: %s port\n", argv[0]);

exit(0);

}

sock = socket(AF\_INET, SOCK\_DGRAM, 0); // Creates socket. Connectionless.

if (sock < 0)

{

error("Opening socket");

}

//Setup for button input later

int fd = open("/dev/mem", O\_RDWR);

if (fd < 0)

{

printf("Unable to open the file.\n");

exit(1);

}

unsigned long\* ptr = (unsigned long\*)mmap(NULL , getpagesize(),PROT\_READ|PROT\_WRITE , MAP\_SHARED , fd , 0x80840000);

unsigned long\* BDR = ptr + 1;

unsigned long\* BDDR = ptr + 5;

unsigned long\* BDBounce = ptr + 49;

\*BDDR = \*BDDR & 0xffffffe0; //set buttons to inputs and activate DBounce for them

\*BDBounce = \*BDBounce & 0x0000001f;

length = sizeof(server); // length of structure

bzero(&server,length); // clear structure

server.sin\_family = AF\_INET; // symbol constant for Internet domain

server.sin\_addr.s\_addr = INADDR\_ANY; // IP address of this machine

server.sin\_port = htons(atoi(argv[1])); // port number

//Binds the socket to the address of the host and the port number

if (bind(sock, (struct sockaddr \*)&server, length) < 0)

{

error("binding");

}

//Change socket permissions to allow broadcast

if (setsockopt(sock, SOL\_SOCKET, SO\_BROADCAST, &boolval, sizeof(boolval)) < 0)

{

printf("error setting socket options\n");

exit(-1);

}

fromlen = sizeof(struct sockaddr\_in); // size of structure

//Get this IP address of this server to be sent to the client

gethostname(hostptr , hostlen); //fill the hostent struct

thishost = gethostbyname(hostptr);

myip = strdup(inet\_ntoa(\*((struct in\_addr\*)thishost->h\_addr\_list[0]))); //this is used as the return address the client is told to send responses to

while (1)

{

bzero(buffer,MSG\_SIZE); // empty buffer

//Receive messages from client program

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&addr, &fromlen);

if (n < 0)

{

error("recvfrom");

}

//Run the main menu system

if(strcmp(buffer , "MainMenu\n") == 0)

{

while(1)

{

bzero(mainmenu , 20);

printf("\n===Welcome to The Online Bank!===\n Please select an option\n 1)Log in\n 2)Exit\nYour choice: ");

fgets(mainmenu , sizeof(mainmenu) , stdin);

//User inputs their username, client is alerted that a username is coming along with where the response should be sent back to

if(strcmp(mainmenu , "1\n") == 0)

{

bzero(thisuser , 20);

printf("Please enter your username: ");

fgets(thisuser , sizeof(thisuser) , stdin);

n = sendto(sock , "@user" , MSG\_SIZE , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

n = sendto(sock , thisuser , MSG\_SIZE , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

n = sendto(sock , myip , 10 , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

break;

}

//Program is closed, notify client that server is shutting down

else if(strcmp(mainmenu , "2\n") == 0)

{

printf("Thank you for using The Online Bank!\nInitiating shutdown.\n");

n = sendto(sock , "@close\0" , 7 , 0 , (struct sockaddr \*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

return 0;

}

}

}

//Username was found, program now prompts the user to input their password

else if(strcmp(buffer , "found\n") == 0)

{

bzero(buffer , MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&addr, &fromlen);

if (n < 0)

{

error("recvfrom");

}

printf("\nWelcome back! Please enter your password: ");

fflush(stdout);

//Getting password from button presses, also shows numerically what users inputs on screen

while(passcount < 4)

{

msync(ptr , getpagesize() , MS\_INVALIDATE);

if((\*BDR | 0xfffffffe) == 0xfffffffe)

{

printf("1");

fflush(stdout);

usleep(1000000);

password = password \* 10 + 1;

passcount++;

}

msync(ptr , getpagesize() , MS\_INVALIDATE);

if((\*BDR | 0xfffffffd) == 0xfffffffd)

{

printf("2");

fflush(stdout);

usleep(1000000);

password = password \* 10 + 2;

passcount++;

}

msync(ptr , getpagesize() , MS\_INVALIDATE);

if((\*BDR | 0xfffffffb) == 0xfffffffb)

{

printf("3");

fflush(stdout);

usleep(1000000);

password = password \* 10 + 3;

passcount++;

}

msync(ptr , getpagesize() , MS\_INVALIDATE);

if((\*BDR | 0xfffffff7) == 0xfffffff7)

{

printf("4");

fflush(stdout);

usleep(1000000);

password = password \* 10 + 4;

passcount++;

}

msync(ptr , getpagesize() , MS\_INVALIDATE);

if((\*BDR | 0xffffffef) == 0xffffffef)

{

printf("5");

fflush(stdout);

usleep(1000000);

password = password \* 10 + 5;

passcount++;

}

}

//Convert input password to string and determine next step in program

bzero(chkpass, 5);

snprintf(chkpass , sizeof(chkpass)+1, "%lu", password);

passcount = 0;

password = 0;

if(strncmp(chkpass , buffer , 4) == 0)

{

printf("\nPASSWORD ACCEPTED\n");

n = sendto(sock , "@rightpass" , MSG\_SIZE , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

n = sendto(sock , myip , 10 , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

//Receive the account funds, next menu prompt is handled by main recvfrom

bzero(buffer , MSG\_SIZE);

n = recvfrom(sock, buffer, MSG\_SIZE, 0, (struct sockaddr \*)&addr, &fromlen);

if (n < 0)

{

error("recvfrom");

}

//This will turn money to a long and notify user how much money they have access to

currmoney = strtol(buffer , (char\*\*)NULL , 10);

printf("Your current balance is: %lu\n" , currmoney);

}

else

{

printf("\n\nI'm sorry, this password is not correct. Returning to main menu.\n\n");

n = sendto(sock , "@wrongpass" , MSG\_SIZE , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

n = sendto(sock , myip , 10 , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

}

}

//Username was not connected to a known user, return to MainMenu is handled by main recvfrom

else if(strcmp(buffer , "notfound\n") == 0)

{

printf("\nSorry, we could not find you in our database.\n\n");

n = sendto(sock , myip , 10 , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

}

//Once user is successfully logged in, allow them to alter funds

else if(strcmp(buffer , "MoneyMenu\n") == 0)

{

while(1)

{

bzero(moneymenu , 20);

printf("\n===Welcome back user!===\n Please select an option\n 1)Add Funds\n 2)Withdraw Funds\n 3)Logout\nYour choice: ");

fgets(moneymenu , sizeof(moneymenu) , stdin);

//Deposit funds

if(strcmp(moneymenu , "1\n") == 0)

{

printf("Please enter amount you are depositing: ");

fgets(chkmoney , sizeof(chkmoney) , stdin);

adjmoney = strtol(chkmoney , &ismoneyammount , 10);

if (chkmoney[0] != '\n' && (\*ismoneyammount == '\n' || \*ismoneyammount == '\0'))

{

currmoney = currmoney + adjmoney;

}

else

{

printf("Sorry, you did not enter a number. Please try again.\n");

}

}

//Withdrawing funds

else if(strcmp(moneymenu , "2\n") == 0)

{

printf("Please enter the amount you wish to withdraw: ");

fgets(chkmoney , sizeof(chkmoney) , stdin);

adjmoney = strtol(chkmoney , &ismoneyammount , 10);

if (chkmoney[0] != '\n' && (\*ismoneyammount == '\n' || \*ismoneyammount == '\0'))

{

if((currmoney - adjmoney) > 0)

{

currmoney = currmoney - adjmoney;

}

else

{

printf("Sorry, you do not have the required funds for this withdrawal.\n");

}

}

else

{

printf("Sorry, your transaction could not be completed as entered. Please try again.\n");

}

}

//Send the client this user's funds and get back to main menu

else if(strcmp(moneymenu , "3\n") == 0)

{

printf("Thank you for using The Online Bank!\nReturning to mainmenu...\n\n");

n = sendto(sock , "@logout\0" , 8 , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

//Turn the funds back into a string and send them to client

bzero(buffer , MSG\_SIZE);

snprintf(buffer , MSG\_SIZE, "%lu", currmoney);

strcat(buffer , "\n");

n = sendto(sock , buffer , MSG\_SIZE , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

n = sendto(sock , myip , 10 , 0 , (struct sockaddr\*)&addr , fromlen);

if(n < 0)

{

error("sendto");

}

break;

}

}

}

}

return 0;

}