Keypad Security System

Khaila Pate, CE

ECE 4220: Real-Time Embedded Computing

University of Missouri – Columbia

Spring 2015

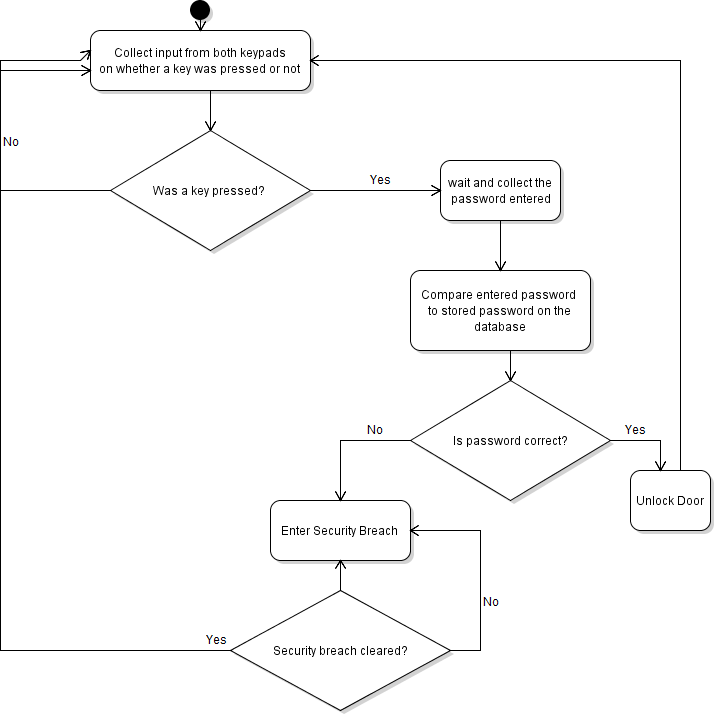
**Abstract:**

The objective of this project is to simulate a Security System that has multiple devices communicating with each other. Attached to two of these devices are keypads which will be used as input devices for pin numbers and passwords. This project will demonstrate the usefulness of Socket Communication, Real Time Tasks, P-Threads, FIFOs, and Interrupt Service Routines. The combination of these concepts will assist in the development of an effective Security System.

**Introduction:**

The first Security System was invented by Edwin Holmes in 1853. His system was just an Electromagnet Alarm System. Over the years Security Systems have improved with technology, allowing communication between devices to secure larger areas. Security is one of the most important issues in the world today. It is also one of the fastest growing and reoccurring issues, as people search for new ways to compromise security. Security Systems are a highly demanded feature in many products that are used on a daily basis. Examples of these systems are car alarms, home alarms like those supplied by ADT, and the systems that secure doors in large office buildings. In this project I have explored the necessary programming elements and logic for creating such systems. The system I have created is similar to those of secured doors inside large buildings. This system, receives a password entry from its keypad device, and sends that data to the Security System’s door server, then the door server sends that password to the client to be checked for accuracy. If the password is correct the client tells that door server to open its doors. If the password is Incorrect the client will broadcast that there is a security breach in the building, and the door servers will go into “LOCKDOWN” mode.

**Implementation:**



UML Diagram of Implementation Flow

The first section of this system I implemented was the keypad. The keypad I used in this project is a 4x4 Matrix Membrane Keypad designed by Parallax. The keypad has 8 pins. The first 4 pins are wired specifically for the 4 columns, and the last 4 pins are wired specifically for the 4 rows as shown in figure 1.

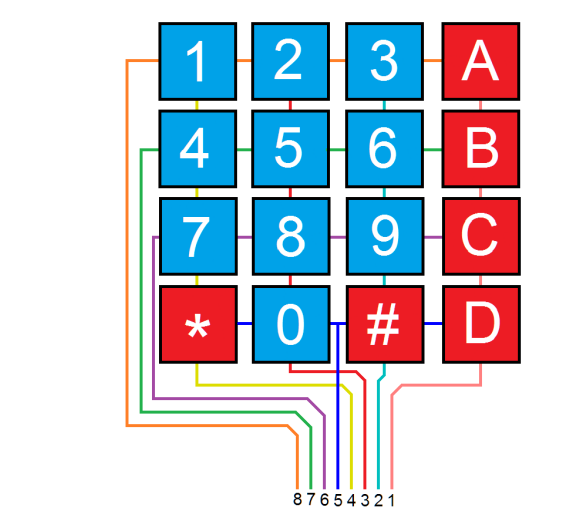


Figure 1: Matrix Keypad Wire Connection

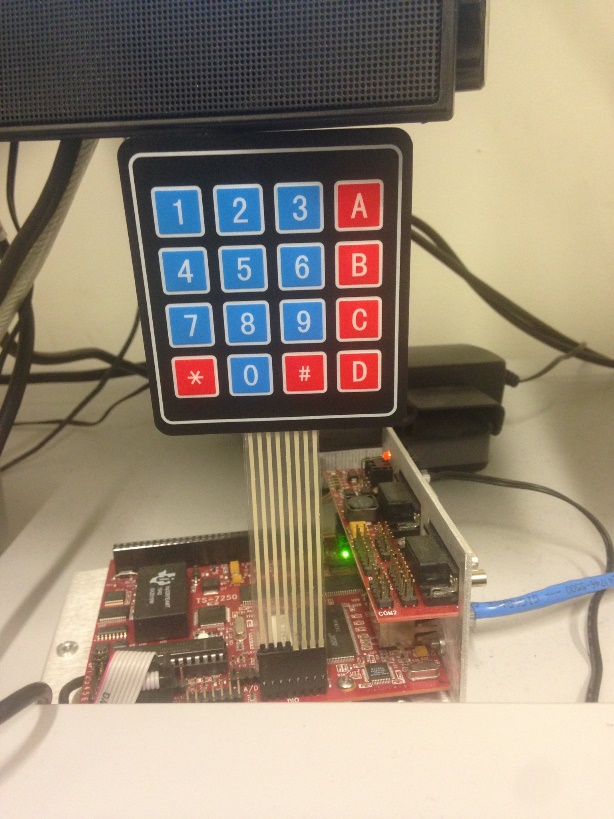


Figure 2: Actual connection of keypad to the TS-7250 Board

This keypad is a passive device, because it does not have any voltage or ground designated pins. It is a matrix of switches mapped to the 8 DIO pins on the TS-7250 board. To program this keypad I created a Kernel Module, named “doorKernel.c”. Due to this device being passive I must supply power to pins 1-4 by setting those pins high, and the pins 5-8 low, using the Port B Data Direction Register(PBDDR). I created a real-time task that would periodically write the pins 1-4 low using the Port B Data Register (PBDR), which is a way of periodically activating those columns. I configured the proper registers to set up the remaining pins for Interrupt Service Routines. This makes every row of buttons able to trigger and Interrupt. In the interrupt handler, there are a series of “IF” statements that will determine exactly which button is being pressed. In order to determine exactly which button is being pressed I must check the RawIntStsB register, to see which row the interrupt was triggered, this bit will be high. Then I must check the PBDR register to see which column was set low, or in other words, active. The “IF” Statement determines a specific button by checking if both of these conditions are met. Only if these two conditions are met can you determine which button in the matrix is pressed. If the “IF” statement is true, then the corresponding character to the button pressed will be saved and then written to a character array. This process will continue until 5 buttons have been pressed. After 5 buttons are pressed the handler will send that password to the door server program, by using a FIFO.

Door Kernel Flowchart

The Door Server, receives this password from the FIFO by using a p-thread. The Door Server communicates with the Security Client to send that password for checking. To achieve this communication within these two programs I used sockets to create a connection between the server and client. The Door Server doesn’t use the password entries on the keypad until it is given an “ALL CLEAR” message from the client, so that the door knows it is alright to proceed and accept password entries. Once the Door Server has the ok from the client the Door Server sends the password entry to the Client. The Client then accepts that password message and compares it to stored passwords on the “Passwords.txt” file. If there is a Match the Client will tell the Door Server to open its doors. If there is not a Match the Client will broadcast there is a Security Breach to all door servers. This causes the Door Servers to broadcast that the system is in “LOCKDOWN”, and lock all of its doors. If a correct password is entered during “LOCKDOWN” the Security Breach is cleared and the system will function as normal.

Door Server Flowchart

Security Client Flowchart

**Experiment and Results:**

I tested several times to make sure that the keypad was actually working. To test the keypad I would press a series of buttons and then check the DMESG of the module to see if the button was working or not. Figure 3 below shows the Printk statements located in the interrupt handler. It shows exactly which buttons are being pressed and shows them being placed inside the password character array. Once 5 buttons have been pressed the password array is reset to accept a new password entry. All of the password entries begin with a “#” to make it easier for the server and client to process that information as a password.

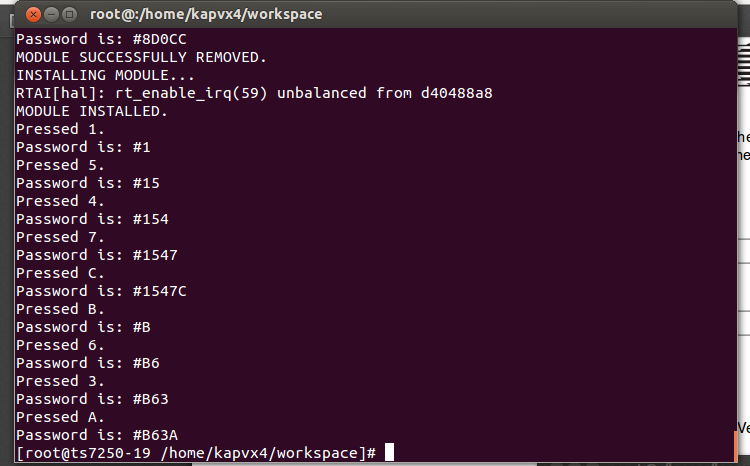


Figure 3: DMESG for doorKernel.o showing Pin Entries from Keypad

To test the Door Server and Security Client programs I ran them several times as well until the output was as desired. I made sure the 3 programs are able to work together as a complete system by testing them all together. Figure 4 shows the Door Server receiving the “ALL CLEAR” message from the client. It also shows the password entry givin from the kernel module and the keypad. In this example the password entered was 44444. This password is infact located in the password text file. The Door Server successfully reicieved the message to open it’s doors and the program continues running as normal.

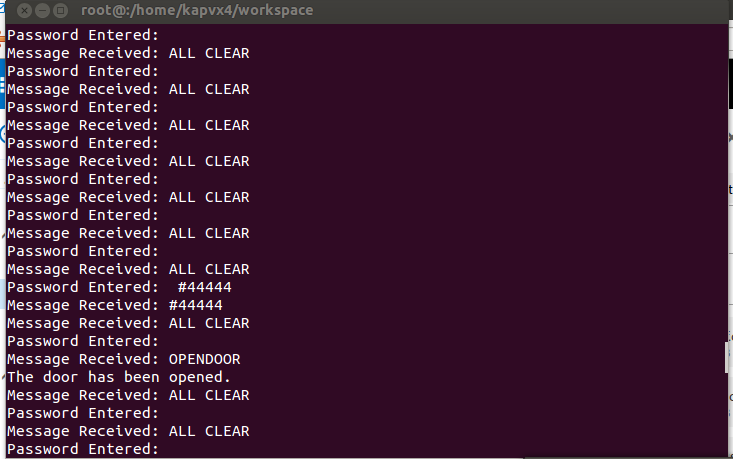


Figure 4: Door Server Receiving messages from Client and Keypad

Figure 5 shows the Client receiving it’s own message that the security status is “ALL CLEAR” . It also shows that the Client received the message from the Door Server that the password entry was 44444, and that the door was open. The security status is still “ALL CLEAR” And the Client program continues as normal also.

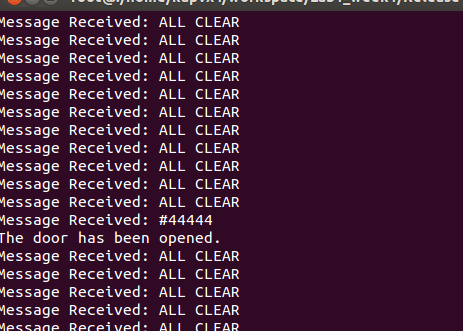


Figure 5: Client Program receiving messages from the Door Server

Figure 6 demonstrates the behavior when there is a bad password entry. The password 12333 was entered on the keypad. This is not a password entry in the text file. The figure shows the password being entered and then received as a message from the client. The client also sends to the Door Server that there is a Security Breach.

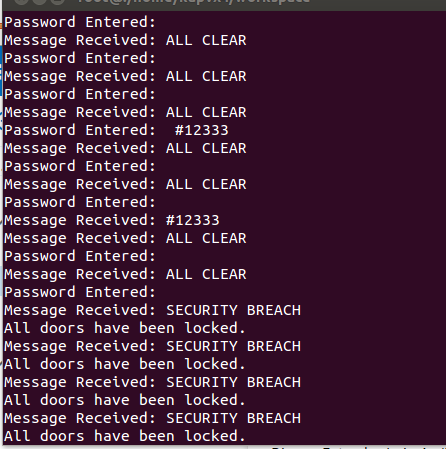


Figure 6: Door Server receiving message from client after bad password entry

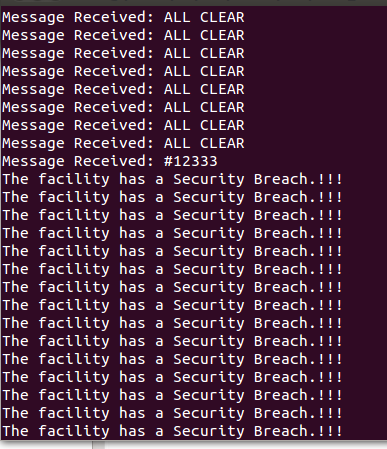


Figure 7: Security Client receiving the bad password

Figure 7 shows the Security Client receiving a bad password and telling all servers on the system there is a security breach in the building. The status of the Door Server doesn’t change until password is received from the client and the client determines that password was not an acceptable password entry.

**Discussion and Conclusion:**

Observing these results for the most part made sense, however I did not expect the Door server to take as long as it did to receive the Security Breach message. I also didn’t expect the password to take as long as it does to reach the Door Server after entry. This is most likely due to the delays I added so that the program output wasn’t going extremely fast. A few problems I faced were that I really didn’t know how the 4x4 Matrix Membrane keypad worked, when I first tested the keypad it did absolutely nothing. I fixed this by understanding that I must supply power to the keypad from the board, and that I cannot read two high bits in the RawIntStsB register to signal an interrupt. I needed to read the values of two different registers instead. The next problem I had was that the Door Server program was not getting any data from the FIFO. I thought maybe the Interrupt handler wasn’t saving the password correctly. But actually I had not opened the FIFO, in the Door Server program. Therefore it had nothing to read from because the FIFO was closed. Once this error was fixed I was able to connect all 3 of my programs together and make sure they were able to function properly. Another issue was with the time I had to complete the project. I originally planned to test the behavior of two Door Servers and a Client, but due to time and trying to figure out the other issues, I was not able to test the second door. However my assumption is that it will function just as the first Door Server without too much of a complication. I believe on a larger scope this system will function with even more than two Doors Servers. If time were not as limited, I would be able to make the system more secure than it is. And less prone to bugs or errors.

What I’ve learned from this project is how to program a 4x4 Matrix Keypad as an Interrupt Service Routine device on a TS-7250 board. I learned how to properly send data between 3 different programs. I’ve learned the usefulness of FIFOs for sending data between kernel and user space programs. I’ve discovered how to avoid a program getting stuck during execution due to blocking functions by implementing p-threads in this project. I’ve successfully implemented a server/client model. I have obtained from this project a better understanding of how the security systems in homes, cars, and secured buildings function, and possible ways they can communicate with other machines on the system. Figure 8 shows the perspective of having more than one Door Server on the system.

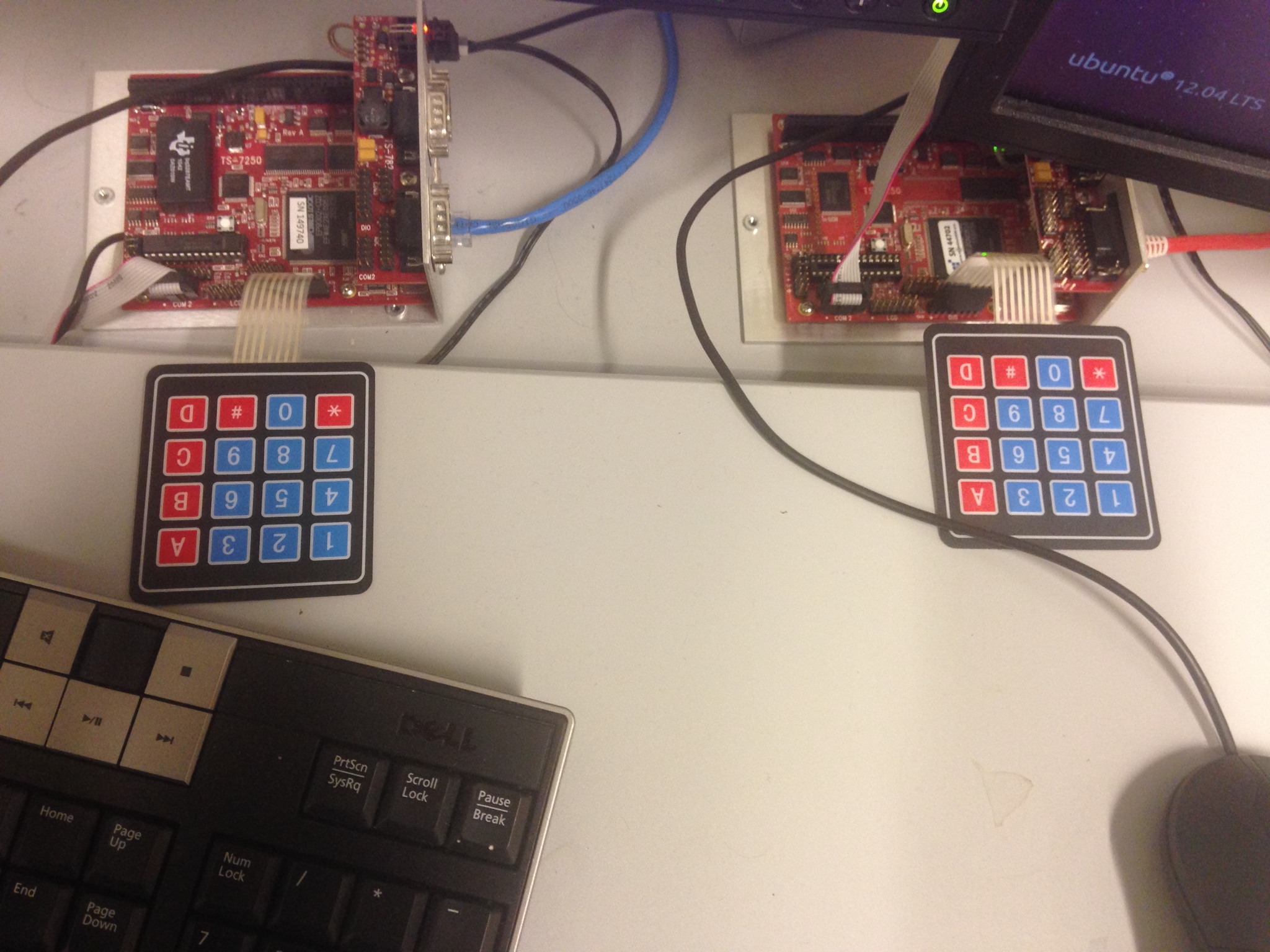


Figure 8: Actual image of the two Door Servers

**Appendices:**

**doorKernel.c**

/\* doorKernel.c

\*

\* Created on: May 01, 2015

\* Author: kapvx4

\*/

#ifndef MODULE

#define MODULE

#endif

#ifndef \_\_KERNEL\_\_

#define \_\_KERNEL\_\_

#endif

#include <linux/module.h>

#include <linux/kernel.h>

#include <asm/io.h>

#include <rtai.h>

#include <rtai\_sched.h>

#include <linux/time.h>

#include <rtai\_fifos.h>

RTIME delay;

static RT\_TASK task;

int pb = 0;

unsigned long button = 0;

unsigned long \*ptr,\*ptr2;

unsigned long \*pfdr, \*pfddr;

unsigned long \*pbdr, \*pbddr;

unsigned long \*GPIOBIntEN, \*GPIOBIntType1, \*GPIOBIntType2, \*GPIOBEOI,\*GPIOBDB, \*RawIntStsB;

unsigned long \*VIC2SoftIntEnable, \*VIC2SoftIntClear;

int fifo\_in = 1;

int fifo\_out = 0;

char pin;

int i = 1;

char password[6];

MODULE\_LICENSE("GPL");

static void supply\_power ()

{

while(1)

{

\*pbdr =\*pbdr & 0x07; //Activate first Column

\*pbdr = \*pbdr | 0x07;

rt\_task\_wait\_period();

\*pbdr = \*pbdr & 0x0B; //Activate 2nd Column

\*pbdr = \*pbdr | 0x0B;

rt\_task\_wait\_period();

\*pbdr = \*pbdr & 0x0D; //Activate 3rd Column

\*pbdr = \*pbdr | 0x0D;

rt\_task\_wait\_period();

\*pbdr = \*pbdr & 0x0E; //Activate 4th Column

\*pbdr = \*pbdr | 0x0E;

rt\_task\_wait\_period();

}

}

static void handler (unsigned irq\_num, void \*cookie)

{

//check button

rt\_disable\_irq(59); //Disable the request line

//Handle stuff

if((\*RawIntStsB & 0x80) == 0x80 && (\*pbdr & 0x07)== 0x07) //button 1 pressed

{

pin ='1';

printk("Pressed 1.\n");

}

if((\*RawIntStsB & 0x80) == 0x80 && (\*pbdr & 0x0B)== 0x0B)//button 2 pressed

{

pin = '2';

printk("Pressed 2.\n");

}

if((\*RawIntStsB & 0x80) == 0x80 && (\*pbdr & 0x0D)== 0x0D) //button 3 pressed

{

pin = '3';

printk("Pressed 3.\n");

}

if((\*RawIntStsB & 0x80) == 0x80 && (\*pbdr & 0x0E)== 0x0E) //button A pressed

{

pin = 'A';

printk("Pressed A.\n");

}

if((\*RawIntStsB & 0x40) == 0x40 && (\*pbdr & 0x07)== 0x07) //button 4 pressed

{

pin = '4';

printk("Pressed 4.\n");

}

if((\*RawIntStsB & 0x40) == 0x40 && (\*pbdr & 0x0B)== 0x0B) //button 5 pressed

{

pin = '5';

printk("Pressed 5.\n");

}

if((\*RawIntStsB & 0x40) == 0x40 && (\*pbdr & 0x0D)== 0x0D) //button 6 pressed

{

pin = '6';

printk("Pressed 6.\n");

}

if((\*RawIntStsB & 0x40) == 0x40 && (\*pbdr & 0x0E)== 0x0E) //button B pressed

{

pin = 'B';

printk("Pressed B.\n");

}

if((\*RawIntStsB & 0x20) == 0x20 && (\*pbdr & 0x07)== 0x07) //button 7 pressed

{

pin = '7';

printk("Pressed 7.\n");

}

if((\*RawIntStsB & 0x20) == 0x20 && (\*pbdr & 0x0B)== 0x0B) //button 8 pressed

{

pin = '8';

printk("Pressed 8.\n");

}

if((\*RawIntStsB & 0x20) == 0x20 && (\*pbdr & 0x0D)== 0x0D) //button 9 pressed

{

pin = '9';

printk("Pressed 9.\n");

}

if((\*RawIntStsB & 0x20) == 0x20 && (\*pbdr & 0x0E)== 0x0E) //button c pressed

{

pin = 'C';

printk("Pressed C.\n");

}

if((\*RawIntStsB & 0x10) == 0x10 && (\*pbdr & 0x07)== 0x07) //button \* pressed

{

pin = '\*';

printk("Pressed \*.\n");

}

if((\*RawIntStsB & 0x10) == 0x10 && (\*pbdr & 0x0B)== 0x0B) //button 0 pressed

{

pin = '0';

printk("Pressed 0.\n");

}

if((\*RawIntStsB & 0x10) == 0x10 && (\*pbdr & 0x0D)== 0x0D) //button # pressed

{

pin = '#';

printk("Pressed #.\n");

}

if((\*RawIntStsB & 0x10) == 0x10 && (\*pbdr & 0x0E)== 0x0E) //Button D pressed

{

pin = 'D';

printk("Pressed D.\n");

}

password[0] = '#';

password[i] = pin; //save entries

printk("Password is: %s\n", password);

i++;//count how many # entries

if(i == 6) //if 5 entries put onto fifo

{

rtf\_put(fifo\_out, &password, 6\* sizeof(password[0]));

strcpy(password, "\0"); //clear string

i = 1; //reset counter

}

\*GPIOBEOI = \*GPIOBEOI | \*RawIntStsB; //clear the end of the interrupt register

rt\_enable\_irq(59); //re-enable the interrupt line

}

int init\_module(void)

{

printk("INSTALLING MODULE...\n");

ptr = (unsigned long \*)\_\_ioremap(0x80840000, 4096, 0);

//Configure the Registers

GPIOBIntEN = (unsigned long\*) ((char\*) ptr + 0xB8);

\*GPIOBIntEN = \*GPIOBIntEN | 0xF0;

GPIOBIntType1 = (unsigned long\*) ((char\*) ptr + 0xAC);

\*GPIOBIntType1 = \*GPIOBIntType1 | 0xF0;

GPIOBIntType2 = (unsigned long\*) ((char\*) ptr + 0xB0);

\*GPIOBIntType2 = \*GPIOBIntType2 & 0xFFFFFF0F;

GPIOBEOI = (unsigned long\*) ((char\*) ptr + 0xB4);

\*GPIOBEOI = \*GPIOBEOI | 0xF0;

RawIntStsB = (unsigned long\*) ((char\*) ptr + 0xC0);

//\*RawIntStsB = \*RawIntStsB | 0xF0;

GPIOBDB = (unsigned long\*) ((char\*) ptr + 0xC4);

\*GPIOBDB = \*GPIOBDB | 0xF0;

pbdr = (unsigned long\*)((char\*) ptr + 0x04);

pbddr = (unsigned long\*)((char\*) ptr + 0x14);

\*pbddr = \*pbddr & 0xFFFFFF0F; //Set button direction pointer to 00001111

\*pbddr = \*pbddr | 0x0F;

//create fifo

rtf\_create(fifo\_out,10\*sizeof(password));

//Set up Hardware ISR

rt\_request\_irq(59, handler, 0, 1);

rt\_enable\_irq(59);

rt\_set\_periodic\_mode(); //set to periodic mode

delay = start\_rt\_timer(nano2count(1000000));

rt\_task\_init(&task,supply\_power, 0, 256,0,0,0);

rt\_task\_make\_periodic(&task, rt\_get\_time()+delay, 70\*delay);

printk("MODULE INSTALLED.\n");

return 0;

}

void cleanup\_module(void)

{

rt\_task\_delete(&task);

rtf\_destroy(fifo\_out);

printk("MODULE SUCCESSFULLY REMOVED.\n");

}

**doorServer.c**

/\*

\* doorServer.c

\*

\* Created on: Apr 29, 2015

\* Author: kapvx4

\*/

#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>

#include <pthread.h>

#include <fcntl.h>

#define size 40

#define ROW 4

#define COL 4

char hostName[50];

struct hostent \*myHost;

struct in\_addr \*addresses;

struct sockaddr\_in server, from;

unsigned int length;

int sock, n;

char buffer[size] = " ";

socklen\_t fromlen;

int boolval = 1;

unsigned short pnumber = 2000;

int rnum = 0;

char password[6];

char Oipaddress[10], Aipaddress[10];

int i;

int status = 0; //0 - clear 1- breach

char pin[6];

int fifo\_in = 0;

char repeat[6];

char temp[6];

char string10[2];

pthread\_t thread [1];

void error(const char \*msg);

//pthread to get the password from the board

void get\_password(void)

{

fifo\_in =(open("/dev/rtf/0",O\_RDWR));

while(1)

{

read(fifo\_in, &password, 6\*sizeof(password[0]));

//strcpy(repeat, password); //save last password

}

pthread\_exit(0);

}

int main (void)

{

//get the host name

gethostname(hostName,sizeof(hostName));

//get host by name

myHost = gethostbyname(hostName);

//Print host information

printf("Host Name: %s\n", myHost->h\_name);

printf("Host IP Addresses: \n");

addresses = (struct in\_addr\*)myHost->h\_addr\_list[0];

printf("%s \n ", inet\_ntoa(\*addresses));

bzero((char\*)&server, length);

server.sin\_family = AF\_INET; //Symbol constant for internet domain

server.sin\_addr.s\_addr = INADDR\_ANY; //IP address of the machine on which

printf("What is the port number?\n");

scanf("%hu", &pnumber); //read unsigned short port number from the user

server.sin\_port = htons(pnumber); //port number\*\*

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

from.sin\_family = AF\_INET;

from.sin\_port = htons(pnumber);

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255"); //Broadcast Address

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

if(sock <0)

error("socket");

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

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

error("Error Binding");

//set broadcast option

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

{

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

exit(-1);

}

//create pthread

pthread\_create(&thread[0], NULL,(void\*)&get\_password, NULL);

while(1)

{

sleep(1);

bzero(buffer, size);

//receive a message

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

if(n < 0)

error("recvfrom");

printf("Message Received: ");

printf("%s\n", buffer);

if (strncmp("ALL CLEAR", buffer, 9)==0) //if all clear proceed as normal

{

status = 0; //clear

//get password from the fifo

//strcpy(password,"#12345");

printf("Password Entered: %s\n", password);

if(strcmp(password, repeat) != 0 && (password[0] != '\0')) // while password is not equal to last password.

{

//Send password entry to the client

strcpy(buffer,password);//Copy message into the buffer

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255");

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr\*)&from, fromlen); //Send message to client

if(n < 0)

error("There was an error sending to the client\n"); //report an error

}

strcpy(repeat, password);//save last password

password[0] = '\0';//clear password

}

if (strncmp("SECURITY BREACH", buffer, 15)==0) //if all clear proceed as normal

{

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255");

char warning[size] = "LOCKDOWN\n";

//Broadcast warning to all systems

strcpy(buffer,warning);//Copy message into the buffer

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr\*)&from, fromlen); //Send message to client

if(n < 0)

error("There was an error sending to the client\n"); //report an error

//check password entries

if(strcmp(password, repeat) != 0 && (password[0] != '\0')) // while password is not equal to last password.

{

//Send password entry to the client

strcpy(buffer,password);//Copy message into the buffer

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255");

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr\*)&from, fromlen); //Send message to client

if(n < 0)

error("There was an error sending to the client\n"); //report an error

}

strcpy(repeat, password);//save last password

password[0] = '\0';//clear password

}

if ((strncmp("LOCKDOWN", buffer, 8)==0)) //Lock down, dont allow entry of passwords

{

status = 1; //breach

printf("All doors have been locked. \n");

}

if(strncmp("OPENDOOR", buffer, 8)==0)

{

printf("The door has been opened.\n");

}

}//end while loop

}//end main function

**securityClient.c**

/\* securityClient.c

\*

\* Created on: Apr 29, 2015

\* Author: kapvx4

\*/

#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>

#include <sys/time.h>

#define size 40

#define row 5

#define col 7

char hostName[50];

struct hostent \*myHost;

struct in\_addr \*addresses;

struct sockaddr\_in server, from;

unsigned int length;

int sock, n;

char buffer[size] = " ";

socklen\_t fromlen;

int boolval = 1;

unsigned short pnumber = 2000;

int rnum = 0;

char password[6];

char Oipaddress[10], Aipaddress[10];

int i, j;

int status = 0; //0 - clear 1- breach

char string10[2];

int clear = 0;

FILE \*fp;

char database[5][7];

void error(const char \*msg);

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

{

//open file

fp = fopen("Passwords.txt", "r");

//read from password file store in database array

for(i =0; i < row; i++)

{

//for(j = 0; j < col; j++)

{

fscanf(fp, "%s", database[i]);

}

}

printf("database i = %s\n",database[1]);

//get the host name

gethostname(hostName,sizeof(hostName));

//get host by name

myHost = gethostbyname(hostName);

//Print host information

printf("Host Name: %s\n", myHost->h\_name);

printf("Host IP Addresses: \n");

addresses = (struct in\_addr\*)myHost->h\_addr\_list[0];

printf("%s \n ", inet\_ntoa(\*addresses));

bzero((char\*)&server, length);

server.sin\_family = AF\_INET; //Symbol constant for internet domain

server.sin\_addr.s\_addr = INADDR\_ANY; //IP address of the machine on which

printf("What is the Port Number?\n");

scanf("%hu", &pnumber); //read unsigned short port number from the user

server.sin\_port = htons(pnumber); //port number\*\*

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

from.sin\_family = AF\_INET;

from.sin\_port = htons(pnumber);

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255"); //Broadcast Address

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

if(sock <0)

error("socket");

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

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

error("Error Binding");

//set broadcast option

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

{

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

exit(-1);

}

while(1)

{

sleep(1);

//broadcast all clear

if(clear == 0)

{

clear = 0;

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255");

char message[size] = "ALL CLEAR";

strcpy(buffer, message);

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr \*)&from, fromlen); //Broadcast message

if(n <0)

error("Error Broadcasting message\n");

//sleep(1000);

bzero(buffer, size);

//receive password from servers

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

if(n < 0)

error("recvfrom");

printf("Message Received: ");

printf("%s\n", buffer);

//check password on database

if(buffer[0] == '#') //Password data

{

clear = 1;

for(i = 0; i < row; i++)

{

if(strncmp(buffer, database[i], 6)==0) //if password is equal open the door

{

clear =0;

char message[size] = "OPENDOOR";

strcpy(buffer, message);

//sleep(1000);

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr\*)&from, fromlen); //Send message to client

}

}

} //End of Password data

}

if(clear == 1)// security has been breached

{

from.sin\_addr.s\_addr = inet\_addr("10.3.52.255");

char message[size] = "SECURITY BREACH";

strcpy(buffer, message);

//sleep(10000);

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr \*)&from, fromlen); //Broadcast message

if(n <0)

error("Error Broadcasting message\n");

//Check password to clear breach

if(buffer[0] == '#') //Password data

{

clear = 1;

for(i = 0; i < row; i++)

{

if(strncmp(buffer, database[i], 6)==0) //if password is equal open the door

{

clear =0;

char message[size] = "OPENDOOR";

strcpy(buffer, message);

//sleep(1000);

n = sendto(sock, buffer, strlen(buffer), 0, (const struct sockaddr\*)&from, fromlen); //Send message to client

}

}

}

}

if (strncmp("ALL CLEAR", buffer, 8)==0) //if all clear proceed as normal

{

clear = 0;

}

if (strncmp("SECURITY BREACH", buffer, 15)==0) //if all clear proceed as normal

{

clear = 1;

printf("The facility has a Security Breach.!!!\n");

}

if ((strncmp("LOCKDOWN", buffer, 8)==0)) //Lock down, dont allow entry of passwords

{

clear = 1; //breach

printf("The facility is on Lockdown.\n");

printf("All doors have been locked. \n");

}

if(strncmp("OPENDOOR", buffer, 8)==0)

{

printf("The door has been opened.\n");

}

}

}