|  |  |
| --- | --- |
| \\cetus.ece.missouri.edu\users\fischerjd\Desktop\Current Semester\common\logo-clear.png | University of Missouri – Columbia  Department of Electrical and Computer Engineering  *“A Proud Tradition, Engineering the Future!”* |

Two Input Security Mechanism

ECE 4220 Project Paper   
May 15, 2015

— By —

Nicholas Ortiz

# Abstract

This project addresses the problem of security and will add an external layer of security to a server. It will require two physical passwords in different locations to unlock the lock on the server. This will add a layer of security to the server so that not only one person can unlock the lock but requires two people each having one part of the code. The two people not only have to have the correct code but also enter it within a certain time of each other otherwise it will not be valid. If one of the two people’s codes is incorrect the process will have to be repeated.

# Introduction

The project uses two clients and one server to exchange parts of the code across a network. The two clients will have peripheral devices like a 4x3 keypad, the auxiliary board, or the keyboard. The clients will be able to choose one of these devices to input their part of the code. The clients can also choose which client goes first and does the first half of the code by setting it as a master in the menu screen. When both clients have selected their devices they wish to use then they will click start on the menu screen. The server will wait for both clients to be ready and then wait for the master client to go first. The second client will go next and the server will put both codes together and check to see if it was correct and return a message to the clients saying correct or not. The goals of the project were to have three devices the clients could use, time each code input and time between the first client sending its code to the second client sending its code, and be able to put the code together over a network and check if its correct. The motivation of the project was to implement a security mechanism that makes it so not only one person knows the whole code. This will increase the security of system and my process can be implemented with many clients so the code could be split up multiple times for even more security.

# Background

The background on my project is using the Two-man rule to unlock a lock. It is a high security method that requires actions from at least two authorized people. It is used in many applications that require high security where it’s too great of a risk for one person to be able to unlock by their self. It is also used in movies where the two people with keys turn the keys at the same time is a similar concept to one I tried to implement. My application can be used when a lock requires more security then just one person knowing the code and requires multiple code inputs from other people. The application can work across a network so the people could be in different locations and all of them can manually input their codes to unlock the lock.

# Implementation

The system uses two clients with peripheral devices attached to them for code input. The types of devices that were used are the keyboard, 4x3 keypad, and the TS-7250 buttons. The keypad uses the DIO\_0-3, 5-7 pins on the TS-7250 board just like the buttons but the keypad requires a polling method which will constantly send a high signal to the columns of the keypad and check the rows of the keypad for a high signal if a button was pressed. Then it will use the row and column number to determine which number was pressed. The TS-7250 buttons use interrupts to determine which button was pressed. I used this method because it gave a fast and accurate result which is needed in my system. The keyboard using a polling method that will check for an input from the standard input so the task could be canceled if the user does not enter the code in the correct time instead of using just a scanf which is a blocking function.

The clients have a menu screen that allows them to pick the device they want to use, set which client is the master, and start the process. When both clients have picked their device and set a master then the clients will start the process. When a client starts it will send a signal to the server saying it is ready and the server will wait for the other client to be ready before the process begins. The master client will input the code first, and then send the code to the server if it was inputted in the correct amount of time. Then the server will signal the second client to begin and this process will also be timed. If the second client does not send the other half of the code to the server in time the server will send an error signal to the clients signaling to reset the process. If the code is entered in the correct time the server will put both the codes together and check it to see if it is the correct code. It will send a signal saying it’s correct or not correct to the clients. The clients can then change devices or master and repeat the process.

The project uses real time tasks like a timer to set a flag after the certain amount time has been passed to stop another task. When a client needs to input a code it will create two pthreads, one for the real time task timer and one to read the code input. The main thread waits for the two pthreads to be done and continues on with the network communication. I did this to be able to stop the code input task if the user does not enter the code and lets it stay idle or if they do not enter the right amount of digits. The server uses a real time task with a pthread to time the process of when the code is sent from the master client to when the second client sends the code.

The system overview diagram in figure 1 shows the overall all flow of the system. The clients will receive input from the peripheral devices and the clients and server will then communicate with the appropriate signals.

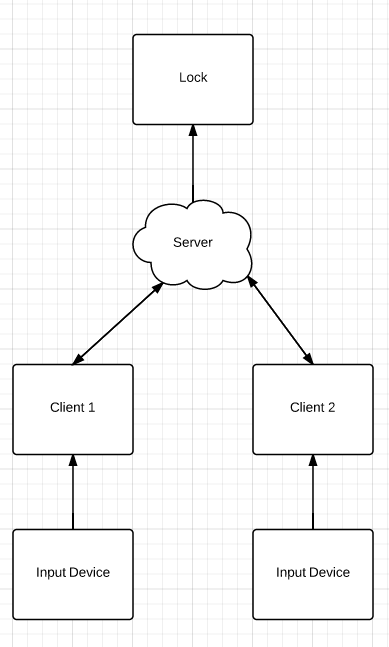


Figure 1. System Overview Diagram

Figure 2 shows how the keypad is setup. The rows and columns are connected to the DIO 0-3, 5-7 pins on the TS-7250 board.

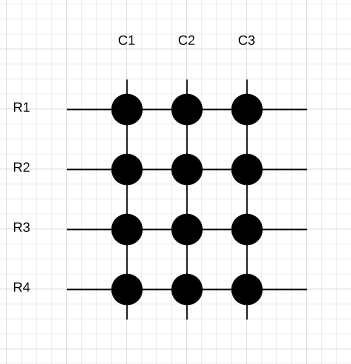


Figure 2. Keypad Schematic

Figure 3 shows the flow of all the processes of the system. The clients and server are run on different boards and communicate using sockets with UDP. The server will wait for both clients to be ready and then signal the master client to go. The slave client will go after and when the server receives both parts of the code it will check the code. I was not able to get the fairCom working so the log attempt would not happen in my implementation but was my original plan.

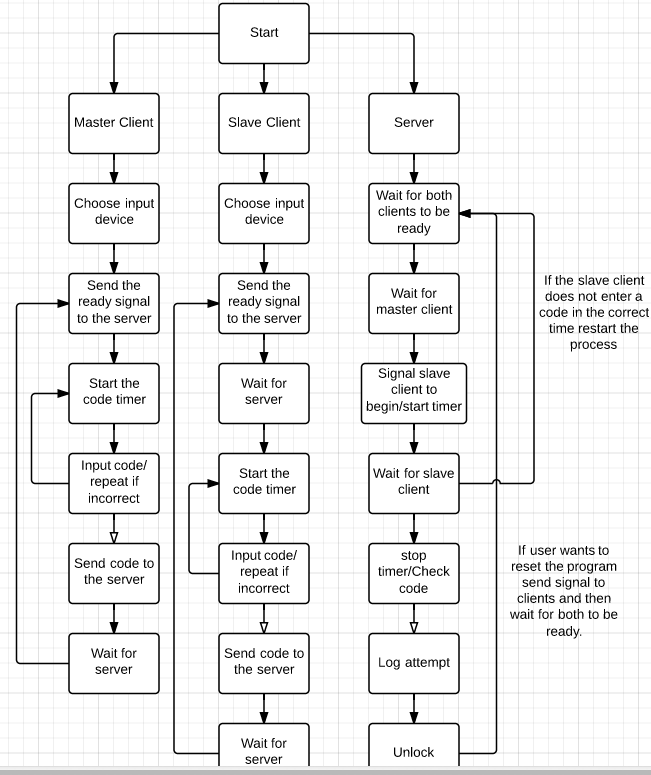


Figure 3. Software Flow Chart

# Experiments and Results

I started working on the three peripheral devices in my system to try and get them to read the input from the user. I tested this by going through each device and making sure I can produce the input code I wanted for each device. I then set up the real time task and a task to read the input with a device with pthreads and made sure the real time task would stop the reading task if it took too long. Once the hardware and client program was working correctly I started to add the network communication. When I added the network communication it would not work with the real time tasks. So I tried the network communication without the real time tasks and it worked. The system works separately I can do the real time tasks without network communication and the network communication works without the real time tasks. When they are together sometimes the pthreads stop working and one client does not receive messages from the server. The screen shots below show the parts working separately.

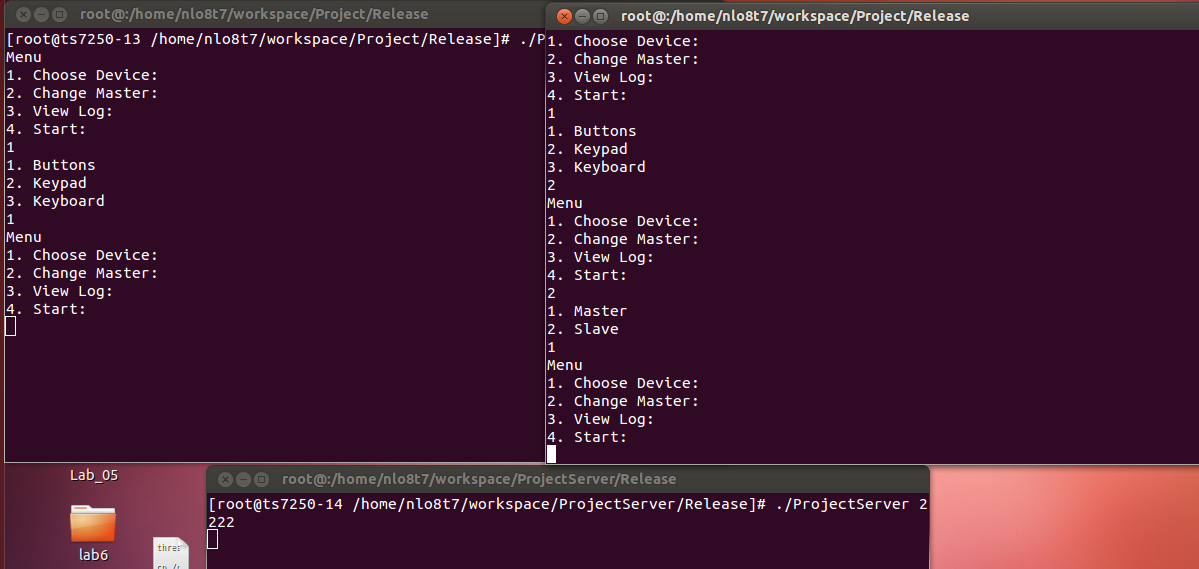


Figure 4. Beginning of the Program

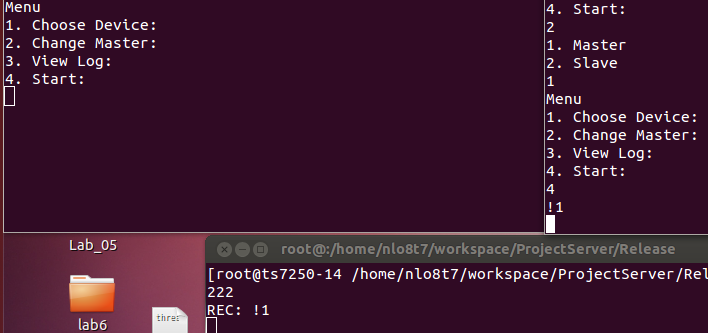


Figure 5. Waiting on the Slave Client

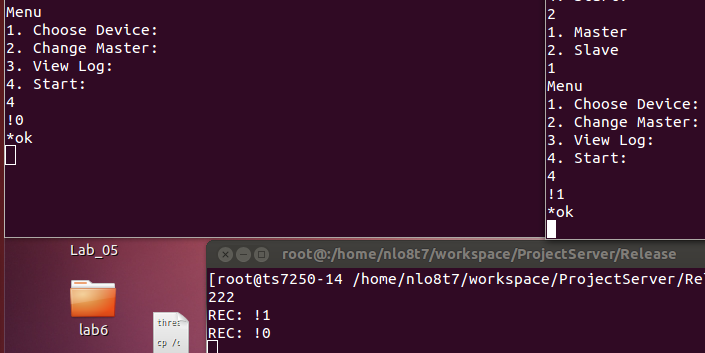


Figure 6. Both Clients are ready

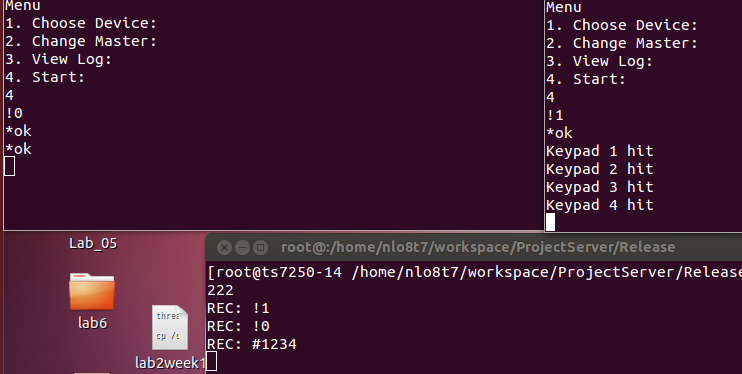


Figure 7. Master Client Enters First

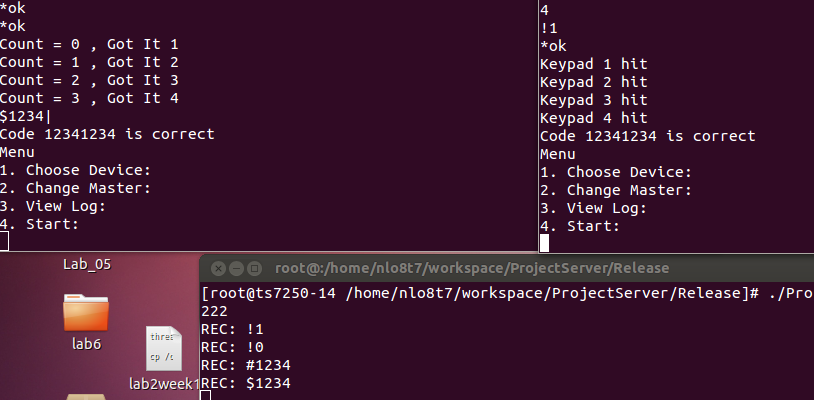


Figure 8. Both Clients Sent the Code

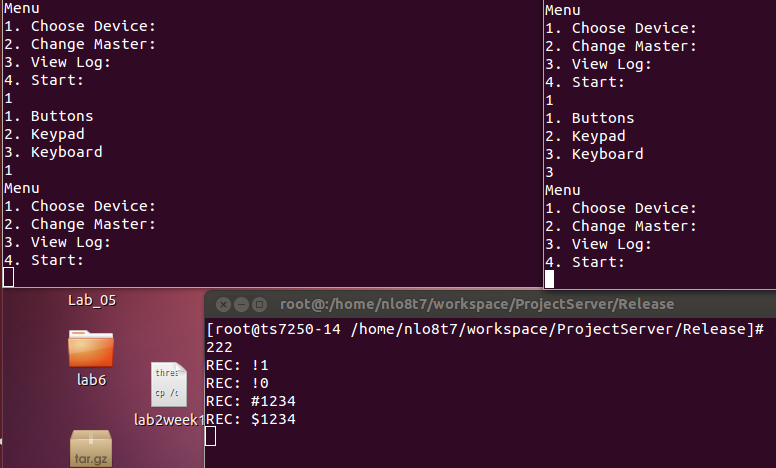


Figure 9. Master Client Changes to Keyboard

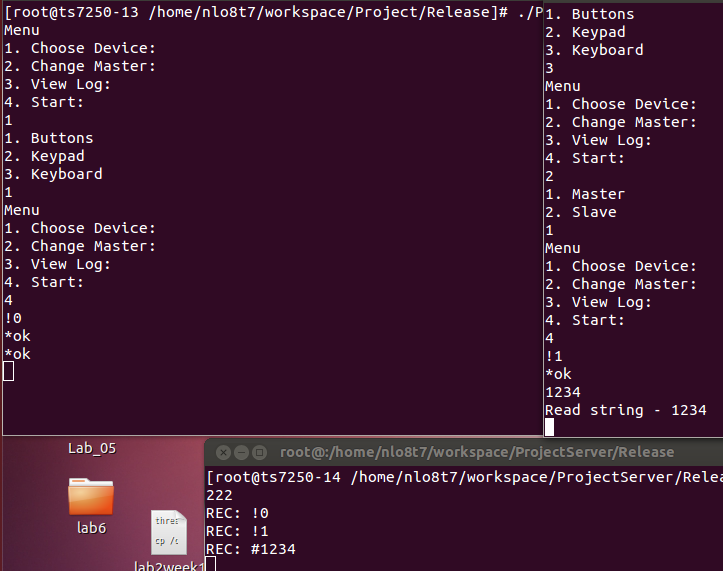


Figure 10. Sent the Code with the Keyboard

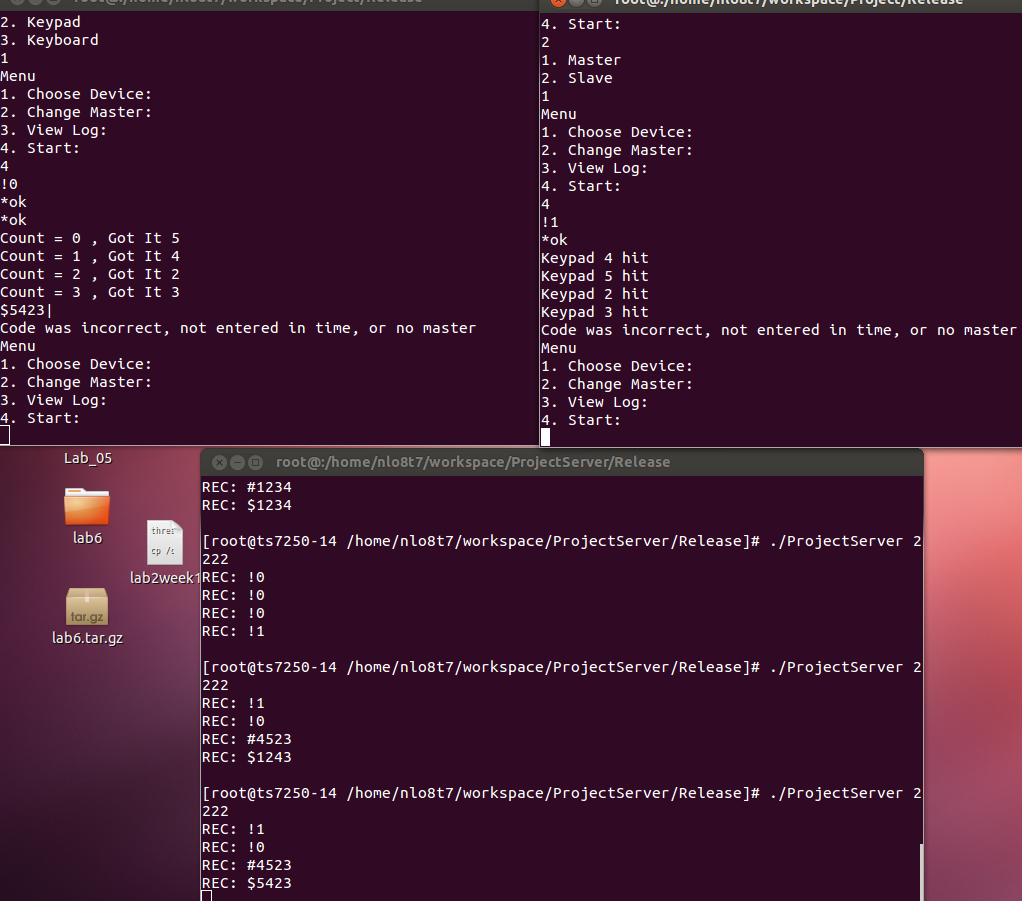


Figure 11. Incorrect Code Entered

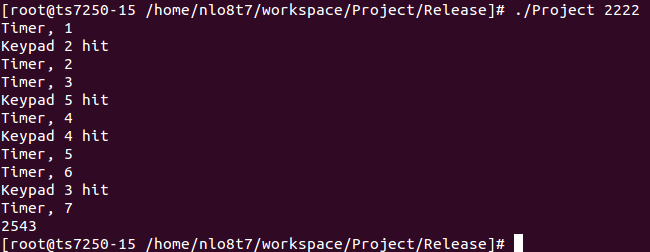


Figure 12. Real Time Task Stops After 4 Digits

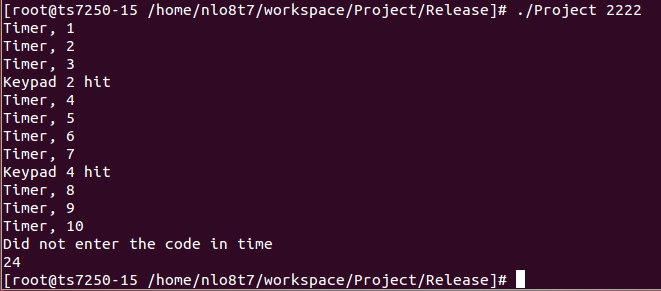


Figure 13. Real Time Task Code Not Entered In Time

# Discussion and Conclusions

The overall system did not work the way I intended it to work. I also did not implement the fairCom as I was intended to do. I did not expect to run into difficulties with the network communication and real time tasks. It wasn’t because the network communication delay varies I think it was something else. The two parts that work separately but not together is the network communication and the real time tasks. The network communication works fine when I do not use the real time task timers and the real time task work when the network communication is not there. When I put them together the pthreads stop working and one of the clients do not receive messages from the server. I think this is because I am using UDP and when the real time task is run I miss the messages from the server. Using TCP communication would be a better approach for this system sense it requires a hand shake to make sure the message is sent and received. I think instead of using a flag with the real time task to signal the other task to stop, software interrupts might be a better solution. It signals the task to stop right away instead of allowing it to chance by the OS to run through and not get preempted. I used a lot of the concepts we have covered like real time tasks, pthreads, FIFO, interrupts, network communication. I have learned more about network communication and real time tasks from this project.

# Appendix Code

# Project Client

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <unistd.h>

#include <rtai.h>

#include <rtai\_lxrt.h>

#include <pthread.h>

#include <string.h>

#include <sys/time.h>

#include <sys/select.h>

#include <sys/types.h>

#include <semaphore.h>

#include <poll.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <time.h>

#define MSG\_SIZE 40

unsigned long \*DDRB;

unsigned long \*PORTB;

RTIME BaseP;

struct timeval tv;

void \*readFIFO(void \*arg);

void \*timer(void \*arg);

void \*keyBoard(void \*arg);

void \*keyPad(void\*arg);

char read\_key(int key);

int count, deadLine, num, fifo, flag = 0, codeFlag = 0;

int lastState = 5;

char code[5], buffer, test, control, control2, control3, device = 1;

void error(const char \*msg)

{

perror(msg);

exit(0);

}

struct sockaddr\_in anybody;

struct sockaddr\_in from;

struct hostent \*hp;

int sock, length, n, boolval = 1, S\_M\_flag = 0, r = 4, c = 3;

socklen\_t clientlen;

char buf[MSG\_SIZE], key;

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

if (argc != 2){

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

exit(1);

}

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

int fp = open("/dev/mem",O\_RDWR|O\_SYNC);

unsigned long \*ptr = (unsigned long \*)mmap(NULL, getpagesize(),PROT\_READ|PROT\_WRITE, MAP\_SHARED, fp, 0x80840000);//get the mmap address

//assign addresses to the pointers for each port.

DDRB = (unsigned long \*)((char\*)(ptr+0x05));

PORTB = (unsigned long \*)((char\*)(ptr+0x01));

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

int i,j;

test = 'z';

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

pthread\_t thread[4];

pthread\_attr\_t thread\_attr;

pthread\_attr\_init(&thread\_attr);

pthread\_attr\_setdetachstate(&thread\_attr, PTHREAD\_CREATE\_JOINABLE);

/\*

pthread\_create(&thread[0], &thread\_attr, keyPad, NULL);

pthread\_create(&thread[1], &thread\_attr, timer, NULL);

pthread\_join(thread[1],NULL);

pthread\_join(thread[0],NULL);

\*/

while(control != 'q'){

printf("Menu\n1. Choose Device:\n2. Change Master:\n3. View Log:\n4. Start:\n");//menu

scanf("%1s",&control);

fflush(stdout);

if(control == '1'){//choose device

printf("1. Buttons\n2. Keypad\n3. Keyboard\n");

scanf("%1s",&control2);

fflush(stdout);

if(control2 == '1'){

device = '1';

}

else if (control2 == '2'){

device = '2';

}

else if (control2 == '3'){

device = '3';

}

}

else if(control == '2'){

printf("1. Master\n2. Slave\n");

scanf("%1s",&control3);

fflush(stdout);

if(control3 == '1'){

S\_M\_flag = 1;

}

else if(control3 == '2'){

S\_M\_flag = 0;

}

}

else if(control == '3'){//for fairCom but not used

printf("LOG\n");

}

else if(control == '4'){//start the unlocking process

//bzero(code,5);

bzero(buf,MSG\_SIZE);

flag = 0;

codeFlag = 0;

count = 0;

do{//wait for each client to be ready before starting

if(buf[0] != '!'){

sprintf(buf,"!%d",S\_M\_flag);

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

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

if (n < 0){

error("sendto");

}

}

bzero(buf,MSG\_SIZE);

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

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

if(!(strncmp(buf, "@no", 3))){

flag = 1;

break;

}

}

while ((strncmp(buf,"\*ok", 3)));

if(S\_M\_flag == 1){//allow the master client to go first

}

else if(S\_M\_flag == 0){

do{

bzero(buf,MSG\_SIZE);

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

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

}

while ((strncmp(buf,"\*ok", 3)));

}

if(device == '1'){//run what ever deivece was selected from the menu

//system("insmod ProjectKernel.o");

pthread\_create(&thread[0], &thread\_attr, readFIFO, NULL);

//pthread\_create(&thread[1], &thread\_attr, timer, NULL);

pthread\_join(thread[0],NULL);

if(count == 4){

bzero(buf,MSG\_SIZE);

if(S\_M\_flag == 1){

strcat(buf,"#");

}

else{

strcat(buf,"$");

}

strcat(buf,code);

printf("%s|\n",buf);

// send message to anyone there...

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

if (n < 0)

error("Sendto");

}

else{

flag = 1;

}

//read(fifo,&buffer,sizeof(buffer));

//system("rmmod ProjectKernel");

}

else if(device == '2'){

//pthread\_create(&thread[1], &thread\_attr, timer, NULL);

pthread\_create(&thread[2], &thread\_attr, keyPad, NULL);

pthread\_join(thread[2],NULL);

if(count == 4){

bzero(buf,MSG\_SIZE);

if(S\_M\_flag == 1){

strcat(buf,"#");

}

else{

strcat(buf,"$");

}

strcat(buf,code);

// send message to anyone there...

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

if (n < 0)

error("Sendto");

}

else{

flag = 1;

}

}

else if(device == '3'){

//pthread\_create(&thread[1], &thread\_attr, timer, NULL);

pthread\_create(&thread[1], &thread\_attr, keyBoard, NULL);

pthread\_join(thread[1],NULL);

//pthread\_join(thread[1],NULL);

if(count == 4){

bzero(buf,MSG\_SIZE);

if(S\_M\_flag == 1){

strcat(buf,"#");

}

else{

strcat(buf,"$");

}

strcat(buf,code);

// send message to anyone there...

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

if (n < 0)

error("Sendto");

}

else{

flag = 1;

//bzero(buf,MSG\_SIZE);

// sprintf(buf,"%qq");

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

}

}

if(S\_M\_flag == 1 && flag == 0){//wait for the server to send the result back

do{

bzero(buf,MSG\_SIZE);

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

if(!(strncmp(buf, "@error", 6))){

flag = 1;

break;

}

}

while ((strncmp(buf,"@ok", 3)));

}

else if(S\_M\_flag == 0 && flag == 0){

do{

bzero(buf,MSG\_SIZE);

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

if(!(strncmp(buf, "@error", 6))){

flag = 1;

break;

}

}

while ((strncmp(buf,"@ok", 3)));

}

if(flag == 1){//print the result

printf("Code was incorrect, not entered in time, or no master\n");

}

else{

bzero(buf,MSG\_SIZE);

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

printf("Code %s is correct\n",buf);

}

}

}

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

close(fifo);

close(sock);

return 1;

}

//real time task timer

void \*timer(void \*arg){

RT\_TASK\* rttask = rt\_task\_init(nam2num("timer"),0,512,256);

rt\_task\_make\_periodic(rttask,rt\_get\_time()+0\*BaseP,1000\*BaseP);

deadLine = 0;

while(count < 4){

rt\_task\_wait\_period();

deadLine++;

if(deadLine > 10){

break;

}

printf("Timer, %d\n",deadLine);

}

if(count < 4){

write(fifo,&test,sizeof(buffer));

}

pthread\_exit(0);

}

//read the FIFO from the Kernel Module for the button interrupts.

void \*readFIFO(void \*arg){

count = 0;

while(count < 4){

if(read(fifo,&buffer,sizeof(buffer))){ //when the named pipe is ready

if(buffer == 'z'){

break;

}

printf("Count = %d , Got It %c\n",count, buffer);

code[count] = buffer;

count++;

}

if(deadLine > 14){

break;

}

}

pthread\_exit(0);

}

//read from the keyboard

void \*keyBoard(void \*arg){

struct pollfd mypoll = { STDIN\_FILENO, POLLIN|POLLPRI };

count = 0;

while(1){

if(poll(&mypoll, 1, 5000)){

scanf("%4s", &code);

printf("Read string - %s\n", code);

count = 4;

}

else if(deadLine > 14){

break;

}

else{

break;

//puts("Read nothing");

}

}

pthread\_exit(0);

}

//read from the keypad using the polling method

void \*keyPad(void\*arg){

int i,j;

\*DDRB = 0xE0;//Switches to be input / led output

\*PORTB = 0xFF;//Turn on all pins in PORTB

count = 0;

while(count < 4){

for(i = 0; i < c; i++){

\*PORTB ^= 1<<(i+5);

for(j = 0; j < r; j++){

if(!(\*PORTB & 1<<j)){

key = read\_key((i+j\*3));

printf("Keypad %c hit\n",key);

code[count] = key;

count++;

}

}

usleep(40000);

\*PORTB = 0xFF;//Turn on all pins in PORTB

}

if(deadLine > 10){

printf("Did not enter the code in time\n");

break;

}

}

pthread\_exit(0);

}

//for turning the keypad rows and cols into a number.

char read\_key(int key){

char a = ' ';

switch(key){

case 0:

a = '1';

break;

case 1:

a = '2';

break;

case 2:

a = '3';

break;

case 3:

a = '4';

break;

case 4:

a = '5';

break;

case 5:

a = '6';

break;

case 6:

a = '7';

break;

case 7:

a = '8';

break;

case 8:

a = '9';

break;

case 9:

a = '\*';

break;

case 10:

a = '0';

break;

case 11:

a = '#';

break;

default:

break;

}

return a;

}

# Project Server

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <unistd.h>

#include <rtai.h>

#include <rtai\_lxrt.h>

#include <pthread.h>

#include <string.h>

#include <sys/time.h>

#include <sys/select.h>

#include <sys/types.h>

#include <semaphore.h>

#include <poll.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <time.h>

#define MSG\_SIZE 40

void error(const char \*msg)

{

perror(msg);

exit(0);

}

RTIME BaseP;

struct timeval tv;

void \*timer(void \*arg);

int deadLine, timeFlag = 0, stopFlag;

struct sockaddr\_in server;

struct sockaddr\_in client;

struct hostent \*hp;

int sock, length, n, portNumber,mcount, count;

int boolval = 1; // for a socket option

socklen\_t clientlen;

char buf[MSG\_SIZE], code[10];

char correctCode[10] = "12341234";

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

if (argc == 2){

portNumber = atoi(argv[1]);

}

else {

portNumber = 2000;

}

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

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

bzero(&server,length); // sets all values to zero. memset() could be used

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

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

// the server is running

server.sin\_port = htons(portNumber); // 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");

// set broadcast option

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

{

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

exit(-1);

}

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

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

pthread\_t thread;

pthread\_attr\_t thread\_attr;

pthread\_attr\_init(&thread\_attr);

pthread\_attr\_setdetachstate(&thread\_attr, PTHREAD\_CREATE\_JOINABLE);

while (1){

bzero(&buf,MSG\_SIZE);//clear the buffer

n = recvfrom(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, &clientlen);//receive the message

printf("REC: %s\n",buf);

if(buf[0] == '!'){//wait for the clients to signal they are ready.

count++;

if(buf[1] == '1'){

mcount++;

}

if(count == 2 && mcount == 1){

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "\*ok");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

count = 0;

mcount = 0;

}

else if(count == 2 && mcount != 1){//both slaves or both masters causes an error

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "@no");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

count = 0;

mcount = 0;

}

}

else if(buf[0] == '#'){//read from the master client into the first half of the code.

code[0] = buf[1];

code[1] = buf[2];

code[2] = buf[3];

code[3] = buf[4];

//pthread\_create(&thread, &thread\_attr, timer, NULL);

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "\*ok");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

}

else if(buf[0] == '$' && timeFlag == 0){//read from the slave client into the second half of the code.

code[4] = buf[1];

code[5] = buf[2];

code[6] = buf[3];

code[7] = buf[4];

stopFlag = 1;

}

else if(buf[0] == '%'){//error in the code

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "@error");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

}

if(stopFlag == 1 && !(strncmp(correctCode,buf,8))){// the code is correct signal the clients

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf,"@ok");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "%s",code);

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

stopFlag = 0;

}

else if(stopFlag == 1){//error in the code

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "@error");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

stopFlag = 0;

}

if(timeFlag == 1){//code did not finish in time

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "@error");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

timeFlag = 0;

}

}

}

void \*timer(void \*arg){

RT\_TASK\* rttask = rt\_task\_init(nam2num("timer"),0,512,256);

rt\_task\_make\_periodic(rttask,rt\_get\_time()+0\*BaseP,1000\*BaseP);

timeFlag = 0;

stopFlag = 0;

deadLine = 0;

while(deadLine > 10){

rt\_task\_wait\_period();

deadLine++;

if(stopFlag == 1){

break;

}

if(deadLine >= 10){

timeFlag = 1;

bzero(&buf,MSG\_SIZE);//clear the buffer

sprintf(buf, "stop");

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

n = sendto(sock, buf, MSG\_SIZE, 0, (struct sockaddr \*)&client, clientlen);

}

}

pthread\_exit(0);

}

# Project Kernel

#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 <rtai\_fifos.h> // for FIFOs

#include <linux/time.h>

#include <linux/delay.h>

MODULE\_LICENSE("GPL");

//register address variables

unsigned long \*PBDR;

unsigned long \*PBDDR;

unsigned long \*PFDR;

unsigned long \*PFDDR;

unsigned long \*GPIOBIntEn;

unsigned long \*GPIOBIntType1;

unsigned long \*GPIOBIntType2;

unsigned long \*GPIOBEOI;

unsigned long \*IntStsB;

unsigned long \*GPIOBDB;

static void my\_handler(unsigned irq\_num, void \* cookie){

char num;

rt\_disable\_irq(irq\_num);//disable the interrupt while this handler executes

//determine which button has been pressed

if (\*IntStsB & 1<<4) num = '5';

else if (\*IntStsB & 1<<3) num = '4';

else if (\*IntStsB & 1<<2) num = '3';

else if (\*IntStsB & 1<<1) num = '2';

else if (\*IntStsB & 1<<0) num = '1';

else num = '0';

\*GPIOBEOI |= 0x1F;//interrupt clear

//printk("Hello %c\n",num);

rtf\_put(0,&num,(sizeof(char)));//put the button that i am pressing number into the fifo

//mdelay(100);

rt\_enable\_irq(irq\_num);////re enable the interrupt

}

int init\_module(void){

rt\_request\_irq(59, my\_handler, 0, 1); //hardware interrupt

unsigned long \*ptr = (unsigned long \*)\_\_ioremap(0x80840000,4096,0);//reading register address

//assigning registers

PBDDR = (unsigned long \*)((char\*)(ptr+0x05));//assigning the address of PBDDR

\*PBDDR = 0xE0;//Switches to be input

PBDR = (unsigned long \*)((char\*)(ptr+0x01));//assigning the address of PBDR

\*PBDR = 0xFF;//Turn on all pins in PORTB

PFDDR = (unsigned long \*)((char\*)(ptr+0x0D));//assigning the address of PFDDR

\*PFDDR |= 1<<1;//make GPIOF second bit to be output

PFDR = (unsigned long \*)((char\*)(ptr+0x0C));//assigning the address of PFDR

\*PFDR |= 1<<1;

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

\*GPIOBIntEn = 0x1F;//GPIOB interrupt enable register

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

\*GPIOBEOI |= 0x1F;//GPIOB end of interrupt

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

\*GPIOBIntType1 |= 0x1F;//0-level sensitive 1-edge sensitive

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

\*GPIOBIntType2 &= ~0x1F;//1-Falling edge 0-rising edge

IntStsB = (unsigned long\*)((char\*)ptr + 0xBC);

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

\*GPIOBDB = 0x1F;

rt\_enable\_irq(59);//hardware interrupt enable

rtf\_create(0, sizeof(char)); //create fifo 0 to communicate with the other process

return 0;

}

void cleanup\_module(void){

\*PBDR = 0xFF; //Turn on all pins in PORTB

rt\_release\_irq(59); //releasing hardware interrupt

rtf\_destroy(0);

}