Wearable Gait Analysis Sensor for Fall Detection

Final Project Report

ECE4220 Shining Sun

# Abstract

Falling is a common and potentially life threatening problem that faces many elderly people. Consequences of falling can include a broken hip, head or spine trauma, and even death. Many elderly people do not remember if they have fallen or are simply unwilling to admit they have due to fear of losing their independence. This project implements a way of detecting poor walking habits in elderly and works toward preventing future falls from occurring. By placing multiple sensors around the waist, gait, balance, and velocity can be measured and analyzed in order to determine whether or not the user is at risk of falling. This device records waking habits and provides a history for doctors and health professionals in order to prevent unknown falls from occurring.

# Introduction

The objective of this project is to creating a system that can tell people or their families when they have fall. The system will have one server and one client. One the Client side, it will collect data from the accelerometer and sent to the client through socket. The server will collect the data from the server and store into the faircom database. Once the fall occurs, the client will broadcast the news “!!!Fall Occurs!!!” and it will trigger the software interrupt and then the LED red light is on and the speaker will be on.

# Background

1. Client side

**Socket**

Socket is used to communicate with two processes in different computer. I used it to send data from the client side to the user side. The sending pattern is followed as the client server model we gave in class.

**Multithread**

I created a muti-threaded server system. One main thread is to setup socket and do broadcasting. Another thread called readFile is to read data from the accelerometer.

**Software Interrupt**

In the child readFile, I calculate the data from the accelerometer. Once the data is greater than the threshold. It will alert the user and broadcast the message. Then it will trigger the software interrupt and the LED light will on and the speaker will sound.

1. Server Side

**Faircom**

I used faircom to store the value of the accelerometer, to store the data and keep track the health condition of the user.

**Socket**

Socket is created to receive the data passed from the server.

1. Kernel side

**Real time task**

I set the task to be real time. It is counted by the nano second and I set period to the task in order to trigger the speaker.

**Software interrupt**

Once the software interrupt is triggered from the server side, the LED red light will be on and the speaker will sound.

**Fifo**

Fifo is used to communicate with the server side and the kernel side. In the kernel space, it will read the value from the fifo and change the current status.

1. Hardware

I used accelerometer to keep track of the acceleration and I used Arduino board to send the data to the computer.

# Implementation

## General System structure

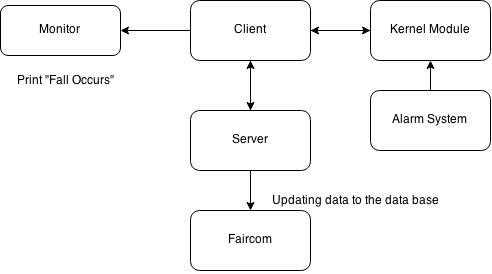


Figure : General System Structure

The client will collect data from the accelerometer and it will collect data from the accelerometer and sent to the client through socket. The server will collect the data from the server and store into the faircom database. Once the fall occurs, the client will broadcast the news “!!!Fall Occurs!!!” as the monitor shows and it will trigger the software interrupt and then the LED red light is on and the speaker will be on, which is the alarm system.

## Client Side

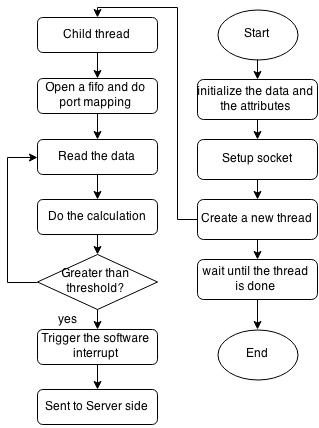


Figure : Client flow diagram

To begin with, I initialized the data structure and set different variables.

I set the structure: to store the data received.

typedef struct{

float xvalue;

float yvalue;

float zvalue;

} thread\_data\_t;

Then I create and setup the socket by using socket(). I changed the permission of the socket to allow broadcast by the function setsockopt().

The third step is creating the a child thread named “readfile”socket using to read data from the arduino board. (pthread\_create()) . After creating the thread, the main thread will wait until the child thread finished its task (using pthread\_join()).

For the child thread part,

1. it port map the port in order to trigger the software interrupt. By using open(“/dev/mem”,…) and mmap() function.
2. It reads in the data
3. Do the calculation, if the total acceleration is greater than the threshold then it means the fall occurs.

Acc

Equation : Total acceleration

1. Judging whether the total acceleration is greater than the threshold. If yes, it will set the flag of fall and trigger the kernel space and broadcast the message “Fall Occurs!!!” for several times.
2. All the data will be sent to the buffer and send to the server side by the socket. (by using function Sendto())

## Server Side

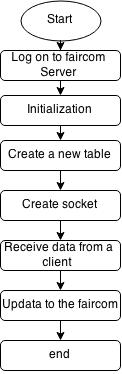


Figure : Server Side

The server part is under the circumstance of the Faircom system.

1. First we initialize the faircom table. I set the table to be 4 columns: the value of x, y, z acceleration and if the fall occurs or not. If the fall occurs, the number in the last column will be 1. If not the value will be 0.

Table : data structure of the faircom

|  |  |  |  |
| --- | --- | --- | --- |
| xacc | yacc | zacc | Fall? |
| *(Data…)* | *(Data…)* | *(Data…)* | *0/1* |

1. I do Define() and Manage() for the faircom. I change the code given by the faircom company and set corresponding figure in the table.
2. Setting up the socket() to receive data. Same with the client side, I change the permission in order to boardcast. The socket will read the data once the client side send in the data
3. Add the data to the table. Add\_Records()

## Kernel Module

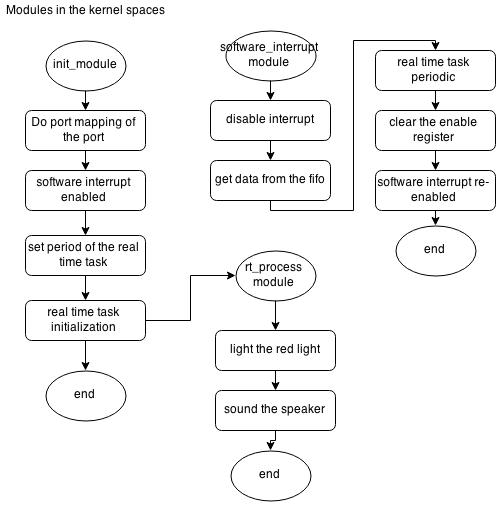


Figure : kernel part

There are four parts in my kernel modules. The first one is the init\_module. This module is basically setup for the rt\_process module and the software\_interrupt module. In this module:

1. setup value for the different registers
2. request the software interrupt by using function rt\_request\_irq()
3. enable the software interrupt by using function rt\_enable\_irq()
4. initialize the period of the real time task and setup realtime task
5. create the fifo using rtf\_create()

The second module is the software interrupt module. It will setup for the software interrupt. Following by the pattern,

1. disable the software interrupt
2. get the number in the fifo. once the number in the fifo appears to be one, it will trigger the rt\_process module
3. make the task periodic. By the function rt\_tasl\_make\_periodic()
4. Clear the register
5. Enable software interrupt again.

As for rt\_process module. It is the execution of the sound and alert. It will sound the speaker and light up the RED LED light once it is triggered. The number in the fifo will decide whether to trigger the module or not. If the fall occurs, the value sent by the fifo will trigger the alert system.

Finally the clean\_up module clears all the register and disables the software interrupt.

## Total Flow chart

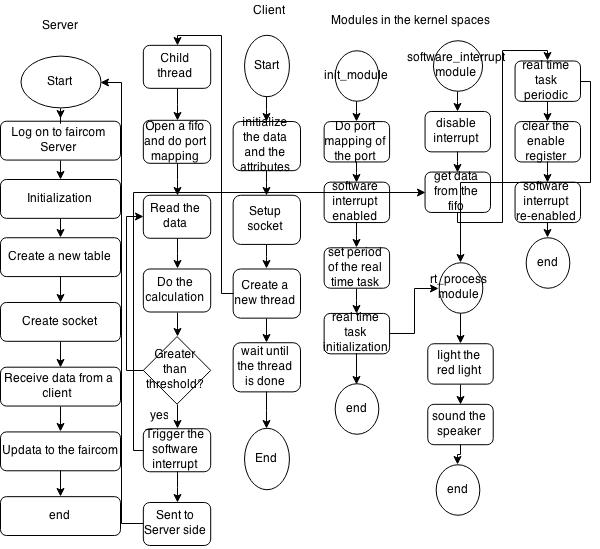


Figure : General block diagram

Just as the general block diagrams shows, the three different spaces communicate with each other. The client will collect data from the accelerometer and it will collect data from the accelerometer and sent to the client through socket. The server will collect the data from the server and store into the faircom database. Once the fall occurs, the client will broadcast the news “!!!Fall Occurs!!!” as the monitor shows and it will trigger the software interrupt and then the LED red light is on and the speaker will be on, which is the alarm system.

# Experiments and Results

1. Client

Figure : Client Reads in data

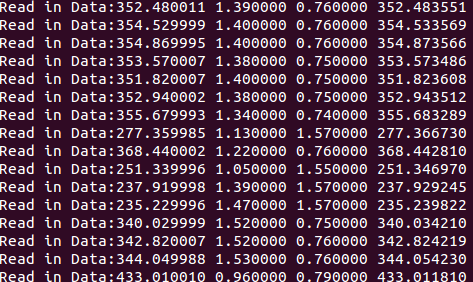
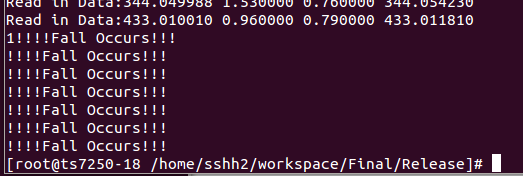


Figure : Fall Occurs



As the picture shows the client reads in data. The first three data is the x,y,z acceleration, the last data is the total acceleration once the acceleration is greater than the threshold, it means there is a fall. However once the fall occurs, it will show up ” !!!Fall Occurs!!!” in the monitor.

1. Server

The server is under the circumstance of Faircom. The following picture is how I initialize the faircom.

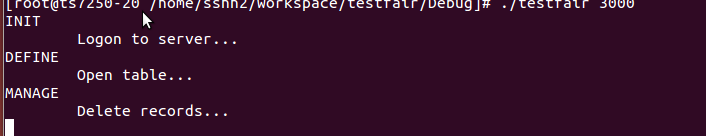


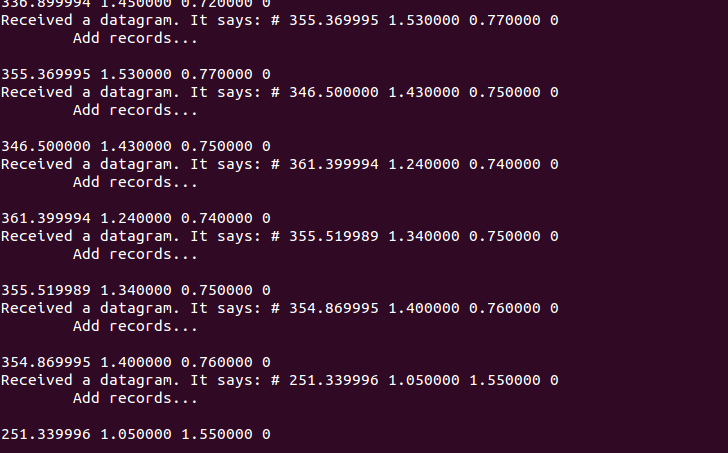
Figure : Initialize Faircom

I set up table to store the values. I set the table to be 4 columns: the value of x, y, z acceleration and if the fall occurs or not. If the fall occurs, the number in the last column will be 1. If not the value will be 0.

Table : data structure of the faircom

|  |  |  |  |
| --- | --- | --- | --- |
| xacc | yacc | zacc | Fall? |
| *(Data…)* | *(Data…)* | *(Data…)* | *0/1* |

Figure : Add record and receive data



As shown in the picture, the server receives data from the socket. The first three data is the x, y, z acceleration. And the last digit is 0 or 1, which illustrate whether the person fall or not. If it is 0, it means the normal walking. If it is 1,then it means the person fall.

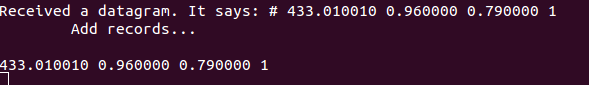


Figure : The Fall Occurs

1. General

I test my code several times and it all prints our Fall Occurs. The way I make sure my code working properly is when the fall occurs, the RED LED light will on and the speaker will sound. Besides, the last digit record in the faircom will be 1. And there will be “Fall Occurs ” present in the monitor screen.

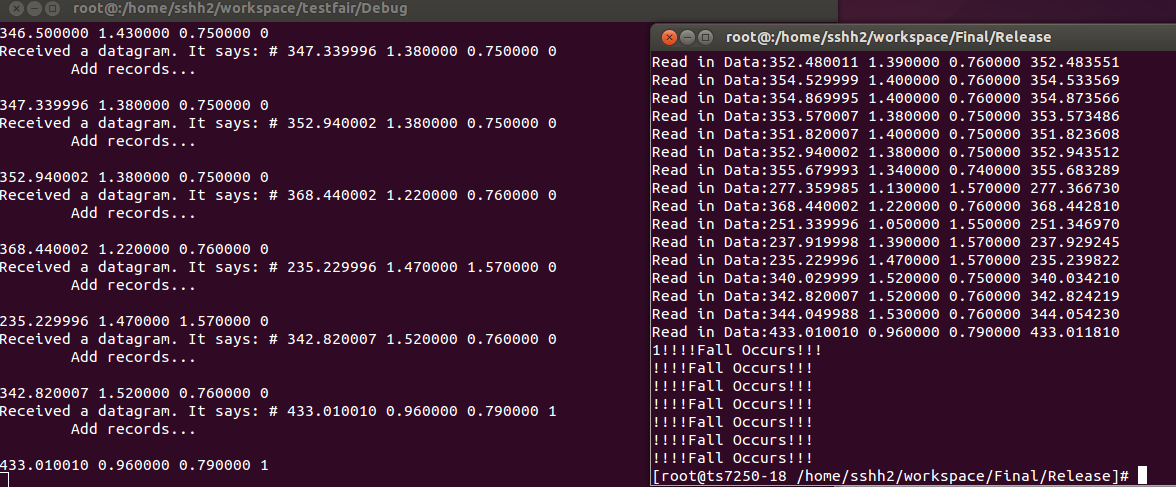


Figure : General test

I just tried the falling cases. And I run my code several times. The way I divide my work is I used client to read in data and judging the data. I used server to collect the data. And I used kernel space to do the alert system.

# Discussion and Conclusions

In my personal opinion, my code works out fine, due the test that the red light is on and the speaker is sound. The “Fall occurs” appears on the monitor and the data represent fall is 1. This is my expecting result.

I do have problem in the setting up the fifo and the setting up the faircom part. The first time, I set the fifo and the common buffer to write only one data which does not make any sense. And then I check my program and I found I did not place the code in the right place and didn’t clear the buffer. Since this is the first time I tried faircom. I faced a lot of difficulties. Through reading different files, it does help me with my server part.

I think I learned a lot from the final project. The interesting part is that I used a lot of elements we learned in class. And it is pretty amazed when you find you combined fifo, socket, real time task, user and client space, kernel module, mlti-threaded, interrupt and faircom together.

The limitation of my project is right now it still have to connect with the computer. It has to send data to the computer and then get the data. Another problem is that I don’t know how to stop the speaker. Since my program ends whenever the fall occurs but the speaker continues sounding.

And maybe the alternative way to solve the first problem is to do the Bluetooth sending data or do the wireless. But I haven’t figure out how to do it. For the second problem, I think I still have to do more research on that.

My project is to do the wireable device, which keep track of people’s walking pattern, which updates to the faircom and judging whether the fall occurs in people’s life. This project succeeded in detecting falls and walking patterns in the person wearing the device. The main goal of this project was to create a wearable device that could provide real time feedback on the user’s walking habits and alert them when they encounter situations that may result in a possible fall.

# Appendices

1. Client side

///\* Name : test2.c

// Author : Shining Sun

#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 <math.h>

#include <fcntl.h>

#include <pthread.h>

#include <semaphore.h>

#include <ifaddrs.h>

#include <time.h>

#include <sys/mman.h>

#include <sys/time.h>

#include <sys/stat.h>

#define MSG\_SIZE 40 // message size

#define Maxsize 100000 // message size

typedef struct{ //this structure is the argument for passing variables into the childThread

float xvalue;

float yvalue;

float zvalue;

} thread\_data\_t;

void error(const char \*msg)

{

perror(msg);

exit(0);

}

//a child thread to read in data

void \*readFile(void \*arg);

//setup variables

int sock, n;

struct sockaddr\_in anybody, from;

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

unsigned int length;

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

{

//port mapping

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

unsigned long \*ptr\_sft = mmap(NULL, getpagesize(), PROT\_READ|PROT\_WRITE, MAP\_SHARED,fd, 0x800C0000);

unsigned long \*VIC2SoftInt = (unsigned long\*) ((char \*)ptr\_sft + 0x18);

int boolval = 1; // for a socket option

//setup for the thread

pthread\_t thread;

pthread\_attr\_t thread\_attr;

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

//read port number from the user

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

pthread\_create(&thread, &thread\_attr, readFile, NULL); //creating the readKernel

pthread\_join(thread, NULL);

close(sock); // close socket.

return 0;

}

//multi-thread

//thread to read the file

void \*readFile(void \*arg){

//port mapping

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

unsigned long \*ptr\_sft = mmap(NULL, getpagesize(), PROT\_READ|PROT\_WRITE, MAP\_SHARED,fd, 0x800C0000);

unsigned long \*VIC2SoftInt = (unsigned long\*) ((char \*)ptr\_sft + 0x18);

//set up fifo

int fifo\_0 = open("/dev/rtf/0", O\_RDWR); //opening the named fifo

if(fifo\_0<0) printf("error opening fifo\n");

int i = 0;

thread\_data\_t data[Maxsize];

float total[Maxsize];

//set flag to illustrate fall

int fall=0;

//read file

FILE \*fp;

fp =fopen("TEST.TXT", "r");

do

{

// bzero: to "clean up" the buffer. The messages aren't always the same length...

bzero(buffer,MSG\_SIZE); // sets all values to zero. memset() could be used

fall = 0;

if (!fp) printf("Error Reading");

else{

//read in data

fscanf(fp,"%f %f %f",&data[i].xvalue,&data[i].yvalue,&data[i].zvalue);

//do calculation

total[i]=sqrt(data[i].xvalue\*data[i].xvalue +data[i].yvalue\*data[i].yvalue +data[i].zvalue\*data[i].zvalue );

printf("Read in Data:%f %f %f %f\n",data[i].xvalue,data[i].yvalue,data[i].zvalue,total[i]);

//judging whether the fall occurs

if(total[i]>380){fall = 1; printf("%d",fall);} else fall =0;

sprintf(buffer, "# %f %f %f %d\n",data[i].xvalue,data[i].yvalue,data[i].zvalue,fall);

if(fall){

write(fifo\_0, &fall, sizeof(fall));//write to the fifo 0 for the communication with kernel

\*VIC2SoftInt |= 0x80000000;//software interrupt flag set up

//print the message

printf("!!!!Fall Occurs!!!\n");

printf("!!!!Fall Occurs!!!\n");

printf("!!!!Fall Occurs!!!\n");

printf("!!!!Fall Occurs!!!\n");

printf("!!!!Fall Occurs!!!\n");

printf("!!!!Fall Occurs!!!\n");

printf("!!!!Fall Occurs!!!\n");

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

}

// send message to anyone there...

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

if (n < 0) error("Sendto");

i++;

}

}while(!feof(fp) && i!=Maxsize && fall != 1);

pthread\_exit(0);

}

1. Server Side

#ifdef \_WIN32\_WCE

#undef UNICODE

#undef \_UNICODE

#define main my\_main

#endif

/\* Preprocessor definitions and includes \*/

#include "ctdbsdk.h" /\* c-tree headers \*/

#define END\_OF\_FILE INOT\_ERR /\* INOT\_ERR is ctree's 101 error. See cterrc.h \*/

// header file

#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

/\* Global declarations \*/

CTHANDLE hSession;

CTHANDLE hDatabase;

CTHANDLE hTable;

CTHANDLE hRecord;

char xacc[47],yacc[47],zacc[47],fall[10];

/\* Function declarations \*/

void error(const char \*msg)

{

perror(msg);

exit(0);

}

#ifdef PROTOTYPE

VOID Initialize(VOID), Define(VOID), Manage(VOID), Done(VOID);

VOID Add\_Records(VOID), Display\_Records(VOID);

VOID Delete\_Records(CTHANDLE), Check\_Table\_Mode(CTHANDLE);

VOID Handle\_Error(CTSTRING);

#else

VOID Initialize(), Define(), Manage(), Done();

VOID Add\_Records(), Display\_Records();

VOID Delete\_Records(), Check\_Table\_Mode();

VOID Handle\_Error();

#endif

/\*

\* main()

\*

\* The main() function implements the concept of "init, define, manage

\* and you're done..."

\*/

#ifdef PROTOTYPE

NINT main (NINT argc, pTEXT argv[])

#else

NINT main (argc, argv)

NINT argc;

TEXT argv[];

#endif

{

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.

char \*token\_s;

if (argc < 2)

{

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

exit(0);

}

Initialize();

Define();

Manage();

// Display\_Records();

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

if (sock < 0)

error("Opening socket");

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(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

while (1)

{

// bzero: to "clean up" the buffer. The messages aren't always the same length...

bzero(buffer,MSG\_SIZE); // sets all values to zero. memset() could be used

// receive from a client

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

if (n < 0)

error("recvfrom");

printf("Received a datagram. It says: %s", buffer);

// To send a broadcast message, we need to change IP address to broadcast address

// (.255). If we don't change it (with the following line of code), the message

// would be transmitted to the address from which the message was received.

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

token\_s = strtok(buffer, " ");

token\_s = strtok(NULL, " ");

// printf("%s\n",token\_s);

strcpy(xacc, token\_s); //Station ID

// data[i].xacc = atoi(xacc);

token\_s = strtok(NULL, " ");

strcpy(yacc, token\_s); //Station ID

token\_s = strtok(NULL, " ");

strcpy(zacc, token\_s); //Station ID

token\_s = strtok(NULL, " ");

strcpy(fall, token\_s); //Station ID

Add\_Records();

// Display\_Records();

}

Done();

printf("\nPress <ENTER> key to exit . . .\n");

#ifndef ctPortWINCE

getchar();

#endif

return 0;

}

/\*

\* Initialize()

\*

\* Perform the minimum requirement of logging onto the c-tree Server

\*/

#ifdef PROTOTYPE

VOID Initialize(VOID)

#else

VOID Initialize()

#endif

{

CTDBRET retval;

printf("INIT\n");

if ((retval = ctdbStartDatabaseEngine())) /\* This function is required when you are using the Server DLL model to start the underlying Server. \*/

Handle\_Error("Initialize(): ctdbStartDatabaseEngine()"); /\* It does nothing in all other c-tree models \*/

/\* allocate session handle \*/

if ((hSession = ctdbAllocSession(CTSESSION\_CTREE)) == NULL)

Handle\_Error("Initialize(): ctdbAllocSession()");

hDatabase = hSession; /\* database not used in this tutorial \*/

/\* connect to server \*/

printf("\tLogon to server...\n");

if (ctdbLogon(hSession, "FAIRCOMS", "ADMIN", "ADMIN"))

Handle\_Error("Initialize(): ctdbLogon()");

}

/\*

\* Define()

\*

\* Open the table, if it exists. Otherwise create and open the table

\*/

#ifdef PROTOTYPE

VOID Define(VOID)

#else

VOID Define()

#endif

{

CTHANDLE hField1, hField2, hField3, hField4;

printf("DEFINE\n");

/\* allocate a table handle \*/

if ((hTable = ctdbAllocTable(hDatabase)) == NULL)

Handle\_Error("Define(); ctdbAllocTable()");

/\* open table \*/

printf("\tOpen table...\n");

if (ctdbOpenTable(hTable, "custmast", CTOPEN\_NORMAL))

{

/\* define table fields \*/

printf("\tAdd fields...\n");

hField1 = ctdbAddField(hTable, "x", CT\_STRING, 47);

hField2 = ctdbAddField(hTable, "y", CT\_STRING, 47);

hField3 = ctdbAddField(hTable, "z", CT\_STRING, 47);

hField4 = ctdbAddField(hTable, "result", CT\_STRING, 10);

if (!hField1 || !hField2 || !hField3 || !hField4 )

Handle\_Error("Define(); ctdbAddField()");

/\* create table \*/

printf("\tCreate table...\n");

if (ctdbCreateTable(hTable, "custmast", CTCREATE\_NORMAL))

Handle\_Error("Define(); ctdbCreateTable()");

if (ctdbOpenTable(hTable, "custmast", CTOPEN\_NORMAL))

Handle\_Error("Define(); ctdbOpenTable()");

}

else

Check\_Table\_Mode(hTable);

}

/\*

\* Manage()

\*

\* This function performs simple record functions of add, delete and gets

\*/

#ifdef PROTOTYPE

VOID Manage(VOID)

#else

VOID Manage()//

#endif

{

printf("MANAGE\n");

/\* allocate a record handle \*/

if ((hRecord = ctdbAllocRecord(hTable)) == NULL)

Handle\_Error("Manage(): ctdbAllocRecord()");

/\* delete any existing records \*/

Delete\_Records(hRecord);

/\* populate the table with data \*/

// Add\_Records();

/\* display contents of table \*/

// Display\_Records();

}

/\*

\* Done()

\*

\* This function handles the housekeeping of closing tables and

\* freeing of associated memory

\*/

#ifdef PROTOTYPE

VOID Done(VOID)

#else

VOID Done()

#endif

{

printf("DONE\n");

/\* close table \*/

printf("\tClose table...\n");

if (ctdbCloseTable(hTable))

Handle\_Error("Done(): ctdbCloseTable()");

/\* logout \*/

printf("\tLogout...\n");

if (ctdbLogout(hSession))

Handle\_Error("Done(): ctdbLogout()");

/\* free handles \*/

ctdbFreeRecord(hRecord);

ctdbFreeTable(hTable);

ctdbFreeSession(hSession);

/\* If you are linked to the Server DLL, then we should stop our Server at the end of the program. \*/

/\* It does nothing in all other c-tree models \*/

ctdbStopDatabaseEngine();

}

/\*

\* Check\_Table\_Mode()

\*

\* Check if existing table has transaction processing flag enabled.

\* If a table is under transaction processing control, modify the

\* table mode to disable transaction processing

\*/

#ifdef PROTOTYPE

VOID Check\_Table\_Mode(CTHANDLE hTable)

#else

VOID Check\_Table\_Mode(hTable)

CTHANDLE hTable;

#endif

{

CTCREATE\_MODE mode;

/\* get table create mode \*/

mode = ctdbGetTableCreateMode(hTable);

/\* check if table is under transaction processing control \*/

if ((mode & CTCREATE\_TRNLOG))

{

/\* change file mode to disable transaction processing \*/

mode ^= CTCREATE\_TRNLOG;

if (ctdbUpdateCreateMode(hTable, mode) != CTDBRET\_OK)

Handle\_Error("Check\_Table\_Mode(); ctdbUpdateCreateMode");

}

}

/\*

\* Delete\_Records()

\*

\* This function deletes all the records in the table

\*/

#ifdef PROTOTYPE

VOID Delete\_Records(CTHANDLE hRecord)

#else

VOID Delete\_Records(hRecord)

CTHANDLE hRecord;

#endif

{

CTDBRET retval;

CTBOOL empty;

printf("\tDelete records...\n");

empty = NO;

retval = ctdbFirstRecord(hRecord);

if (retval != CTDBRET\_OK)

{

if (retval == END\_OF\_FILE)

empty = YES;

else

Handle\_Error("Delete\_Records(): ctdbFirstRecord()");

}

while (empty == NO) /\* while table is not empty \*/

{

/\* delete record \*/

if (ctdbDeleteRecord(hRecord))

Handle\_Error("Delete\_Records(): ctdbDeleteRecord()");

/\* read next record \*/

retval = ctdbNextRecord(hRecord);

if (retval != CTDBRET\_OK)

{

if (retval == END\_OF\_FILE)

empty = YES;

else

Handle\_Error("Delete\_Records(): ctdbNextRecord()");

}

}

}

/\*

\* Add\_Records()

\*

\* This function adds records to a table in the database from an

\* array of strings

\*/

#ifdef PROTOTYPE

VOID Add\_Records(VOID)

#else

VOID Add\_Records()

#endif

{

CTDBRET retval;

CTSIGNED i;

// CTSIGNED nRecords = sizeof(data) / sizeof(CUSTOMER\_DATA);

printf("\tAdd records...\n");

/\* add data to table \*/

// for (i = 0; i < Maxsize; i++)

// {

/\* clear record buffer \*/

ctdbClearRecord(hRecord);

retval = 0;

/\* populate record buffer with data \*/

retval |= ctdbSetFieldAsString(hRecord, 0, xacc);

retval |= ctdbSetFieldAsString(hRecord, 1, yacc);

retval |= ctdbSetFieldAsString(hRecord, 2, zacc);

retval |= ctdbSetFieldAsString(hRecord, 3, fall);

printf("\n%s %s %s %s",xacc,yacc,zacc,fall);

if (retval)

Handle\_Error("Add\_Records(): ctdbSetFieldAsString()");

/\* add record \*/

if (ctdbWriteRecord(hRecord))

Handle\_Error("Add\_Records(): ctdbWriteRecord()");

// }

}

/\*

\* Display\_Records()

\*

\* This function displays the contents of a table. ctdbFirstRecord() and

\* ctdbNextRecord() fetch the record. Then each field is parsed and displayed

\*/

#ifdef PROTOTYPE

VOID Display\_Records(VOID)

#else

VOID Display\_Records()

#endif

{

CTDBRET retval;

TEXT x[47+1];

TEXT y[47+1];

TEXT z[47+1];

TEXT result[10+1];

printf("\tDisplay records...");

/\* read first record \*/

retval = ctdbFirstRecord(hRecord);

if (retval != CTDBRET\_OK)

Handle\_Error("Display\_Records(): ctdbFirstRecord()");

while (retval != END\_OF\_FILE)

{

retval = 0;

retval |= ctdbGetFieldAsString(hRecord, 0, x, sizeof(x));printf("%d\n",sizeof(x));

retval |= ctdbGetFieldAsString(hRecord, 1, y, sizeof(y));

retval |= ctdbGetFieldAsString(hRecord, 2, z, sizeof(z));

retval |= ctdbGetFieldAsString(hRecord, 3, result, sizeof(result));

if (retval)

Handle\_Error("Display\_Records(): ctdbGetFieldAsString()");

printf("\n%s %s %s %s\n",x,y,z,result);

/\* read next record \*/

retval = ctdbNextRecord(hRecord);

if (retval == END\_OF\_FILE)

break; /\* reached end of file \*/

if (retval != CTDBRET\_OK)

Handle\_Error("Display\_Records(): ctdbNextRecord()");

}

}

/\*

\* Handle\_Error()

\*

\* This function is a common bailout routine. It displays an error message

\* allowing the user to acknowledge before terminating the application

\*/

#ifdef PROTOTYPE

VOID Handle\_Error(CTSTRING errmsg)

#else

VOID Handle\_Error(errmsg)

CTSTRING errmsg;

#endif

{

printf("\nERROR: [%d] - %s \n", ctdbGetError(hSession), errmsg);

printf("\*\*\* Execution aborted \*\*\* \nPress <ENTER> key to exit...");

ctdbLogout(hSession);

ctdbFreeRecord(hRecord);

ctdbFreeTable(hTable);

ctdbFreeSession(hSession);

getchar();

exit(1);

}

1. Kernel Module

/\*

============================================================================

Name : Finalkernel.c

Author :

Version :

Copyright : Your copyright notice

Description : Hello World in C, Ansi-style

============================================================================

\*/

#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>

MODULE\_LICENSE("GPL");

static RT\_TASK mytask;

RTIME period[2];

#define irq\_num 59

unsigned long \*PBDR;

unsigned long \*PBDDR;

unsigned long \*PFDR;

unsigned long \*PFDDR;

unsigned long \*GPIOBIntEn;

unsigned long \*GPIOBIntType1;

unsigned long \*GPIOBIntType2;

unsigned long \*GPIOBEOI;//end of interrupt clear

unsigned long \*GPIOBDB;

unsigned long \*IntStsB;

unsigned long \*RawIntStsB;

unsigned long \*VIC2IntEnClear;

unsigned long \*VIC2IntEn;

unsigned long \*VIC2IntTrigger;

int status;

static void rt\_process(int t){

// light the RED light

\*PBDR |= 0x20;

while(1){

//sound the speaker

(\* PFDR) ^= 0x02;

rt\_task\_wait\_period();

}

}

static void software\_interrupt(unsigned irq, void \* cookie){

rt\_disable\_irq(63);//disabling software interrupt while this function executes

int sendfromuser;

rtf\_get(0,(int\*)&sendfromuser,sizeof(int));//get data from the fifo

rt\_task\_make\_periodic(&mytask, rt\_get\_time(), period[sendfromuser]);//set period with the value from other process

\*VIC2IntEnClear |= 0x80000000;//clear the register

rt\_enable\_irq(63);//re enable the interrupt

}

int init\_module(void){

rt\_request\_irq(63, software\_interrupt, 0, 1); //software interrupt

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

unsigned long \*ptr\_sft = (unsigned long \*)\_\_ioremap(0x800C0000,4096,0);//reading register address

//Port B

PBDDR = (unsigned long \*)((char \*)ptr+0x14);//assigning the address of PBDDR

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

(\* PBDDR) |= 0xE0 ;//set the pin of lights to be input

//Port FVIC2IntEn

PFDR =(unsigned long \*)((char \*)ptr + 0x30);

PFDDR = (unsigned long \*)((char \*)ptr + 0x34);

(\* PFDDR) |= 0x02;

(\* PFDR) ^=0x02 ;

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

(\*GPIOBIntEn) |= 0x1f;

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

(\*GPIOBIntType1) |= 0x1f;

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

(\*GPIOBIntType2) |= 0xE0;

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

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

\*GPIOBDB |= 0x1F;

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

VIC2IntEn =(unsigned long \*)((char \*) ptr\_sft +0x10);

\*VIC2IntEn |= 0x80000000;

VIC2IntEnClear =(unsigned long \*)((char \*) ptr\_sft +0x1C);

VIC2IntTrigger =(unsigned long \*)((char \*) ptr\_sft +0x18);

rt\_enable\_irq(63);//software interrupt enable

rt\_set\_periodic\_mode();/\*let real time task periodic\*/

period[0] = 0;

period[1]= start\_rt\_timer(nano2count(500000));

rt\_task\_init(&mytask,rt\_process, 0, 256, 0, 0, 0);

rtf\_create(0, sizeof(int)); //create fifo

return 0;

}

void cleanup\_module(void){

//releasing hardware interrupt

rt\_release\_irq(63);

\*PBDR &= 0xFFFFFF1F;// turn off all the lights

stop\_rt\_timer(); //stop timer

rtf\_destroy(0);

rtf\_destroy(1);

}