|  |  |
| --- | --- |
| \\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!”* |

Ultrasonic Distance Alarm Detection System

ECE 4220 Final Report

May 9, 2014

— By —

Jacob Starr

**Objectives and Project Description:**

The goal of this project is to use at least 2-5 ultrasonic distance sensors placed throughout a room to detect if a person enters the room. The ultrasonic distance sensors will be able to show that a person is in the room and communicate that information to a client through a network. By displaying the IP address of the board that is picking up information, a user will be able to determine where in the room that an object is being detected by seeing the change in the distance displayed by the client.

**Implementation:**

For my implementation, I used the TS-7250 board from class and connected my ultrasonic distance sensors to the board. For every sensor, I used a TS-7250 board in which the board ran a kernel module and a server. I also used an additional TS-7250 board to act as my client program. The ultrasonic distance sensors that I used for my project are SainSmart HC-SR04’s as displayed in the figure below:



Figure : SainSmart HC-SR04

This particular sensor has four pins. It has a 5VDC VCC pin, a ground, a trigger, and an echo pin. I would have used a sensor that communicated through serial data or USB, but they were way too expensive for me to afford without a budget. The trigger pin is an input pin in which you can set how fast you want the sensor to trigger. The trigger has to be greater than 10 microseconds for this particular sensor and for my project I set the trigger to 100 milliseconds. My kernel module implemented a real time task that acted as the trigger. My trigger was mapped to output pin DIO\_7 on the TS-7250 and I would use a rt\_task\_wait\_period every time after I set the pin high or low. I also mapped the output Echo pin from the sensor to two different input pins (DIO\_0 and DIO\_4 on the TS-7250) and I used interrupts to trigger the input pins on the TS-7250. I triggered pin DIO\_0 on the rising edge of the Echo output and sent a timestamp through a real-time FIFO and I triggered pin DIO\_4 on the falling edge of the Echo output and sent a second timestamp through a second real-time FIFO.

For my client program, I used a UDP broadcast approach in which I would broadcast the message “DATA\n” as an infinite loop throughout the entire network. It also does a check to make sure that it only prints data that is in the accurate range of the sensor (which is up to 5 meters away). The client program also checks to see if it is expecting any messages back from any of the servers that are running. If it receives a message back, it will display that message to the user. In this case, the message contained the distance of the object from the sensor in meters and the server’s IP address. By having the IP address of the server, it helps the client know approximately where in the room a moving object is and is able to detect the moving object. For my server program, I used a UDP server approach in which it would perform calculations if it received the message “DATA\n” from the client and it would send back a message to the client if there was proper data being detected from my calculations. In my server program, I used a pthread to perform the calculations because I had to read from two real-time FIFOs. Since reading from a FIFO is considered a “blocking function,” I had to implement the calculations using a pthread. For my pthread, if my server properly received the message “DATA\n”, one FIFO would read in the first time-stamp and one FIFO would read in the second time-stamp. My pthread would then proceed to get the difference of the two timestamps in seconds and covert it to a float by doing a very simple calculation. That calculation is: ((newtime.tv\_sec – oldtime.tv\_sec) \* 1000000 + (newtime.tv\_usec – oldtime.tv\_usec)) / 1000000. I then continue to use the number of seconds that it took for the ultrasound wave to bounce off the object and then return to the sensor by using the calculation: (time in seconds \* 341 m/s) / 2. 341 meters per second is approximately how fast the speed of sound travels, so that is how you can determine the distance of an object away from the sensor. From there, the distance of the object away from the sensor along with the IP address of the board is broadcasted back to the client so that the client knows if the sensor is picking up any data. Below is a flow chart of my overall implementation showing the flow of information between the three programs and then diagrams for the individual programs:

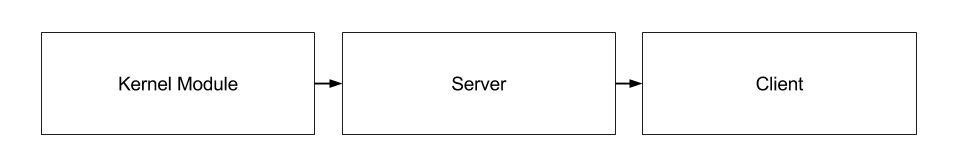


Figure : Data Flow of my Project

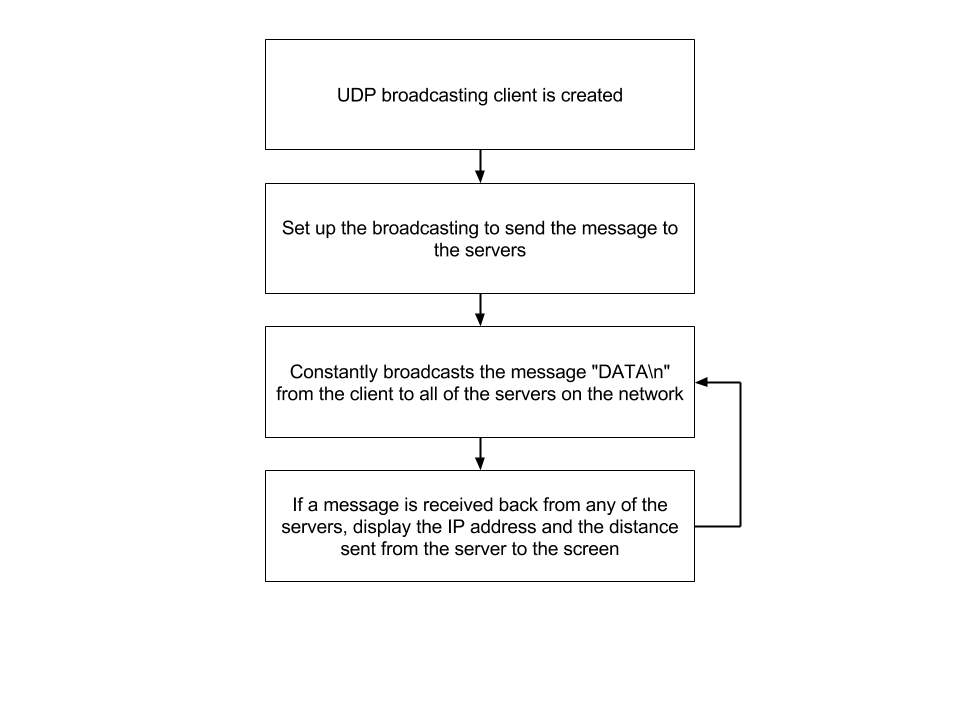


Figure : Client Program

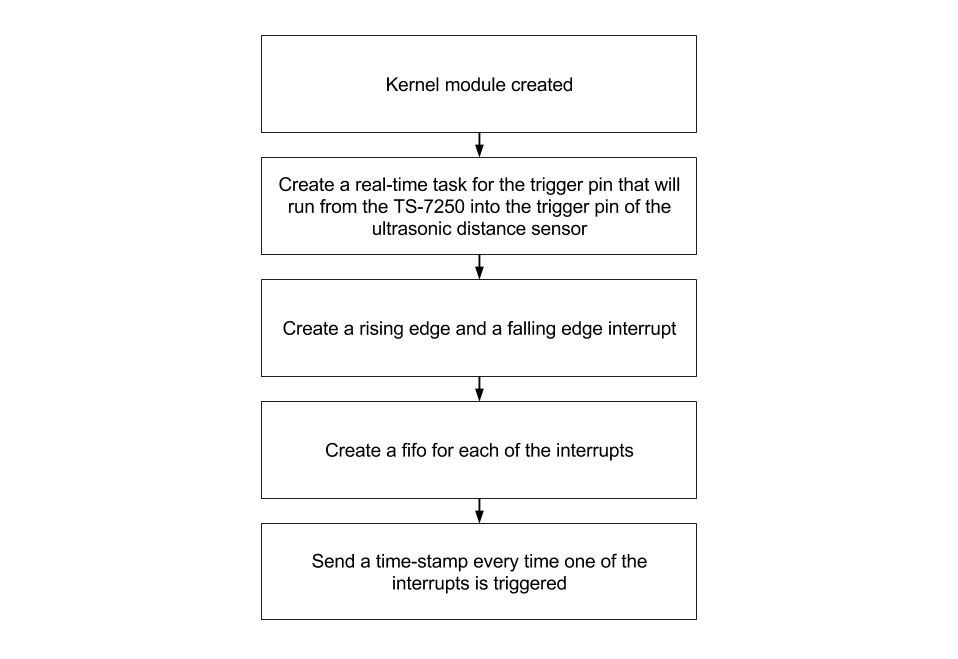


Figure : Kernel Module

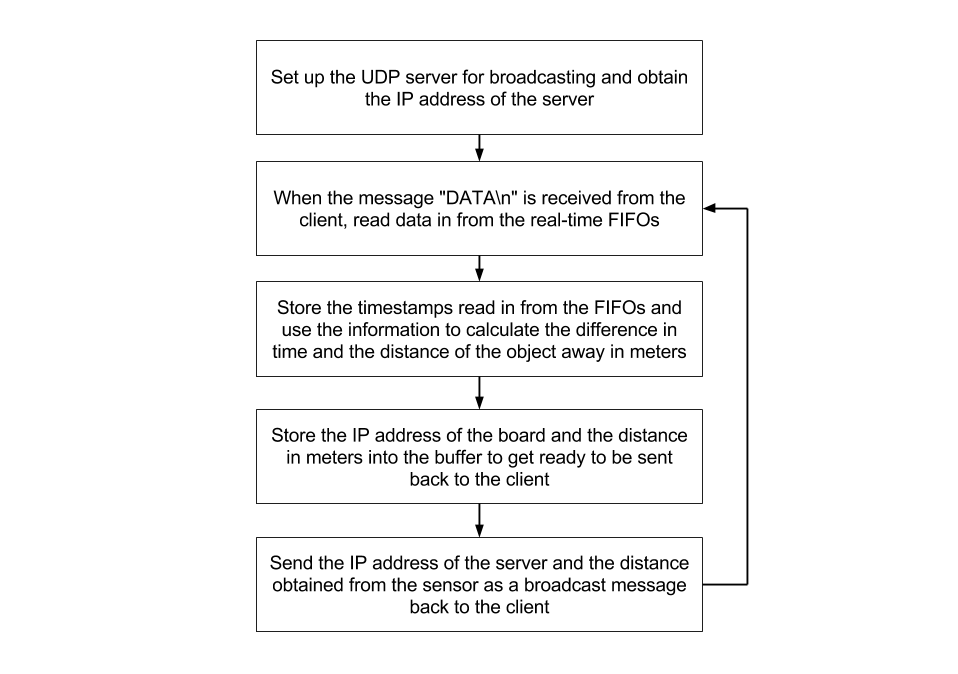


Figure : Server

**Experiments and Results:**

To ensure that my program was running properly, I experimented with several tests. The very first test that I performed was making sure that my kernel module would properly load to the board. Once my kernel module properly uploaded, I tried using the auxiliary board provided in the lab (since my inputs and outputs were mapped to port B) to verify that my pins were mapped correctly. My next test was triggering the green LED to turn on and off since that would end up being the trigger for my ultrasonic distance sensor. Once that worked, I used the red and yellow LEDs so that I could test to see whether or not I properly set up my interrupts as rising edge or falling edge. I mapped the red LED to turn on and the yellow LED to turn off for one of the interrupts and I mapped the yellow LED to turn on and the red LED to turn off for the other interrupt. Button one on the auxiliary board was mapped for the rising edge and button five was mapped for the falling edge. Once I verified that the interrupts were in fact being triggered properly on either the rising or falling edge, I then used buttons one and five to send the timestamps through the real-time FIFO and see if the server and client properly sent and received the data. Obviously this approach is not capable of triggering the data fast enough for accurate readings, but it would simulate that the timestamps were in fact being sent across the FIFOs properly. Once data was being sent through the FIFOs properly, I finally connected my ultrasonic distance sensor to the TS-7250. Once I was able to get the sensor to send data from the kernel module to the server and then to the client properly, I then tried connecting multiple boards to the network to make sure that the client could receive more than one server’s data. After being able to received data from multiple servers, I then used measuring tape to verify the accuracy of the sensor. I ran the program at least one hundred times before I was able to get the ultrasonic distance sensor to talk properly to the board. When dividing up the work for my project, I first wrote the kernel module, then the client, and then the server module. Once I wrote those programs, then I performed all of the experiments/steps listed above to verify the validity of the program.

**Discussion:**

The analysis of my results could not have gone any better than they did. I was properly able to do the calculations of the data and send the proper data to the client along with the board’s IP address so that the client would have an idea of where a moving object is in the room. The results were exactly what I was expecting. The main issues that I had in this project were connecting my hardware components to the board. The biggest issue I had was the ultrasonic distance sensor expects 5 VDC, but the IO pins of port B expect only 3.3VDC. The issue was that the Echo output pin from the ultrasonic distance sensor outputs 5 VDC, but the input pins of input pins DIO\_0 and DIO\_4 expect 3.3 VDC. At first I tried using a simple voltage divider in which I ran the 5 VDC echo pin through a 10kΩ resistor and then have that 3.3 VDC node run to the two input pins and run a 20kΩ resistor run to ground. This unfortunately did not work properly for me. I had to get an oscilloscope to verify that the 3.3 VDC signal was not properly going high and low. Then I tried connecting my circuit using a 2N3904 transistor and that worked properly. The circuit for it is below:

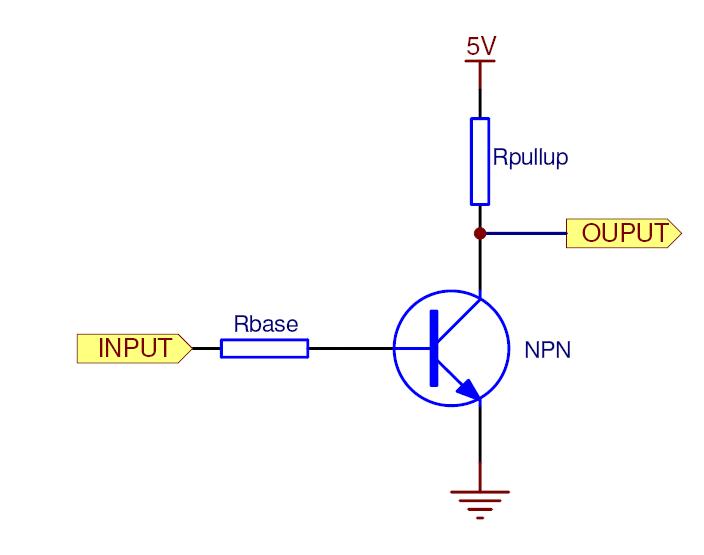


Figure : NPN Circuit from 5VDC to 3.3 VDC

Instead of feeding it a 5 VDC signal through the collector (as in Figure 6), I instead fed a 3.3 VDC signal from the TS-7250 board. The input on Figure 6 is the 5 VDC Echo pin from the sensor and for the Rbase and Rpullup, I used 1kΩ resistors to limit the current. Once I connected my sensor using this circuit, I was finally able to properly trigger the input pins on the TS-7250 so that they could read data. I also had some issues for a while in figuring out the difference in timestamps and the calculations for the data. I used an approach similar to lab 3 to solve my problem and I eventually got the calculations finished properly. Another issue was figuring out the mapping for the interrupt triggers. To solve this, I used a similar approach from lab 6 to solve this. My last major issue was trying to read the data properly in my server from the FIFOs. To solve this, I used a similar approach as lab 6 bonus to put the data inside of a pthread since the reading of a FIFO is considered a blocking function.

I learned several things from this lab. Another thing that I learned is how to create a UDP broadcasting client program that is able to broadcast a message to all of the boards on a network and receive any replies since we never had to create our own client server in any of the labs. After having trouble sending data through FIFOs in the past, I learned how to properly send and read data through a FIFO so that the program will run efficiently. I learned how to trigger interrupts to be rising edge, falling edge, or level-sensitive. I also learned that using a real-time task to create a trigger is a very effective way to implement a trigger since it is consistent. The last vital thing that I learned from this project was having the ability to connect my own hardware components to the board by using the pin outs provided in the owner’s manual of the TS-7250 board.

**Code Section:**

**finalprojectkernel.c:**

/\*

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

Name : finalprojectkernel.c

Author : Jacob Starr

Version :

Copyright : Mine

Description : Hello World in C, Ansi-style

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

\*/

/\*

So far I have created a hardware interrupt that is supposed to trigger when

the ECHO pin (DIO\_0 & DIO\_4) changes states. In my configuration, the TRIGGER

pin will be connected to DIO\_7.

\*/

#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>

#include <linux/time.h>

MODULE\_LICENSE("GPL");

typedef struct timeval timestamp\_t;

timestamp\_t time\_stamp;

static RT\_TASK mytask;

RTIME period;

unsigned long\* RawIntStsB;

unsigned long\* VIC2SoftIntClear;

unsigned long\* btnDR;

unsigned long\* spkDR;

unsigned long\* IntStsB;

unsigned long\* GPIOBEOI;

//real time task that creates the trigger for the sensor

static void rt\_process(int t){

while(1){

\*btnDR |= (1 << 7);

rt\_task\_wait\_period();

\*btnDR &= ~(1 << 7);

rt\_task\_wait\_period();

}

}

//hardware interrupt handler

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

//disables the irq handling

rt\_disable\_irq(irq\_num);

//trigger set for rising edge, gets timestamp and sends

//through FIFO 0

if (\*IntStsB & 0x01){

do\_gettimeofday(&time\_stamp);

rtf\_put(0, (char\*)&time\_stamp, sizeof(timestamp\_t));

//the next two commented lines lit up LEDs on the

//auxiliary board to test the interrupts

//\*btnDR &= ~(1 << 5);

//\*btnDR |= (1 << 6);

}

//trigger set for falling edge, gets timestamp and sends

//through FIFO 1

else if (\*IntStsB & 0x10){

do\_gettimeofday(&time\_stamp);

rtf\_put(1, (char\*)&time\_stamp, sizeof(timestamp\_t));

//the next two commented lines lit up LEDs on the

//auxiliary board to test the interrupts

//\*btnDR &= ~(1 << 6);

//\*btnDR |= (1 << 5);

}

//clears the interrupt

\*GPIOBEOI |= 0x1F;

//enables the irq handling once again

rt\_enable\_irq(irq\_num);

}

//the module that initializes everything

int init\_module(void) {

unsigned long \*ptr, \*GPIOBIntEn, \*GPIOBIntType1, \*GPIOBIntType2, \*GPIOBDB, \*btnDDR, \*spkDDR;

//real time function that attaches the irq

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

//maps Port B

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

//maps the hardware interrupt register memory

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

//enables the buttons

\*GPIOBIntEn |= 0x1F;

//disables the LED's

\*GPIOBIntEn &= ~0xE0;

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

//clears the interrupt flags

\*GPIOBEOI |= 0x1F;

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

//sets the interrupt for the falling edge

\*GPIOBIntType2 &= ~0x10;

//sets the interrupt for the rising edge

\*GPIOBIntType2 |= 0x01;

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

//sets the interrupt for the edge

\*GPIOBIntType1 |= 0x1F;

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

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

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

//maps the memory for Port B

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

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

//sets the LED's data direction register's output to 1 and 0 for input buttons

\*btnDDR |= 0xE0;

\*btnDDR &= ~0x1F;

//maps the speaker register memory

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

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

\*spkDDR |= 0x2;

//enables the irqs

rt\_enable\_irq(59);

rt\_set\_periodic\_mode();

//100ms to pull the trigger (at least 10 us) (50ms high, 50ms low)

period = start\_rt\_timer(nano2count(50000000));

\*btnDR &= ~0xE0;

//initializes the real time task and makes it periodic

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

rt\_task\_make\_periodic(&mytask, rt\_get\_time(), period);

//creates the real time fifos

rtf\_create(0, sizeof(time\_stamp));

rtf\_create(1, sizeof(time\_stamp));

return 0;

}

//deletes the real time task, stops the timer, destroys the fifos,

//releases the interrupts, and turns off the speakers

void cleanup\_module(void) {

rt\_task\_delete(&mytask);

stop\_rt\_timer();

rtf\_destroy(0);

rtf\_destroy(1);

rt\_disable\_irq(59);

rt\_release\_irq(59);

\*btnDR &= ~0xE0;

\*spkDR &= ~0x02;

}

**finalprojectclient.c:**

/\*

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

Name : finalprojectclient.c

Author : Jacob Starr

Version :

Copyright :

Description : My client program for the final project

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

\*/

/\*

\* In this program, I create a UDP broadcast client that will forever loop

\* the message "DATA\n" to everyone on the network. If the client receives

\* any data back from any of the servers, it will print the distance and the

\* IP address of the server received from the server to the client. The client

\* is able to tell approximately where in the room a person is based off of

\* which IP is received and the distance away the object is detected from.

\*/

#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>

//message size for the buffer

#define MSG\_SIZE 100

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]; //stores received messages or messages to be sent.

int boolval = 1;

char \*ipAddress;

double distance = 0, tempDistance = 0;

if (argc != 2)

{

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

exit(1);

}

//creates a connectionless socket

sock = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sock < 0)

error("socket");

//changes the permissions of the socket to allow broadcast

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

{

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

exit(-1);

}

//symbol constant for Internet domain, port field, and the broadcast address

anybody.sin\_family = AF\_INET;

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

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

//structure size

length = sizeof(struct sockaddr\_in);

do {

//sets all the bits of buffer back to zero since not every message

//will be the same size

bzero(buffer,MSG\_SIZE);

//sends a message to anyone there

strcpy(buffer, "DATA\n");

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

if (n < 0)

error("Sendto");

//receives the message

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

if (n < 0)

error("recvfrom");

//use strtok to retrieve the IP address and the distance from the buffer

tempDistance = distance;

ipAddress = strtok(buffer, " ");

distance = atof(strtok(NULL, " "));

//since the sensor is only accurate up to 5 meters, only print data in that range

if ( distance > 0.0 && distance <= 5.0 ) {

printf("Object is detected %lf meters away from board %s.\n", distance, ipAddress);

}

}

while(1);

//closes the socket

close(sock);

return 0;

}

**finalprojectserver.c:**

/\*

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

Name : finalprojectserver.c

Author : Jacob Starr

Version :

Copyright : Mine

Description : My server program for the final project

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

\*/

/\*

In this server program, I do not care about whether or not a server is a

master or slave configuration. I will be reading in information from the

real-time FIFO and do calculations based off of the data that

I receive. Once I receive the data (if there is any valid data), I will

broadcast the IP address of that board along with the distance calculated

based off of my data to the client program.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <time.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <pthread.h>

#define MSG\_SIZE 100 //message size

//prints an error if there is an error detected

void error(const char \*msg)

{

perror(msg);

exit(0);

}

//pthread that performs the calculations

void\* pipeData();

// globals

int sock;

int portnum;

char masterIP[16], myIP[16];

typedef struct timeval timestamp\_t;

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

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

//variable declarations

int length, boolval = 1; //for a socket option

struct sockaddr\_in server;

char host[64];

//randomize the vote seed

srand(time(NULL));

//removes the module from the board if installed and install

//the module when the programs starts up

system("rmmod finalprojectkernel");

system("insmod /home/jas985/workspace/finalprojectkernel/Release/src/finalprojectkernel.o");

//error check command line

if (argc < 2) {

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

exit(0);

}

//creates connectionless socket

sock = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sock < 0) {

error("Opening socket");

}

//sets up the socket

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

bzero(&server,length); //sets all values to zero

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

portnum = atoi(argv[1]);

//gets the IP address of the server

gethostname(host, 64);

struct hostent \*p = gethostbyname(host);

strcpy(myIP, inet\_ntoa(\*((struct in\_addr \*)p->h\_addr)));

printf("My IP address is %s.\n\n", myIP);

//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);

}

//creates the pthread

pthread\_t tid;

pthread\_create(&tid, NULL, pipeData, NULL);

return 0;

}

//this function reads in timestamps and performs calculations based off

//of the speed of sound to calculate the distance of an object from the

//ultrasonic distance sensor

void\* pipeData(){

char buffer[MSG\_SIZE];

int n;

//variable declarations that set up the pipe and broadcast

int fd\_fifo\_old = open("/dev/rtf/0", O\_RDWR);

int fd\_fifo\_new = open("/dev/rtf/1", O\_RDWR);

timestamp\_t oldtime, newtime;

socklen\_t fromlen = sizeof(struct sockaddr\_in);

struct sockaddr\_in broadcast\_addr;

broadcast\_addr.sin\_family = AF\_INET;

broadcast\_addr.sin\_port = htons(portnum);

broadcast\_addr.sin\_addr.s\_addr = inet\_addr("10.3.52.255"); //for broadcasting

while(1){

//cleans up the buffer since the messages aren't always the same length

bzero(buffer,MSG\_SIZE);

//receive from a client

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

if (n < 0) {

error("recvfrom");

}

//reads in from the fifo

read(fd\_fifo\_old, &oldtime, sizeof(timestamp\_t));

read(fd\_fifo\_new, &newtime, sizeof(timestamp\_t));

//the message "DATA\n" is sent from the client

if (!strcmp(buffer, "DATA\n")) {

//calculations to get the difference in time and the distance based off the speed of sound

float time = ((newtime.tv\_sec - oldtime.tv\_sec) \* 1000000 + (newtime.tv\_usec - oldtime.tv\_usec))/1000000.0;

float distance = (time \* 341)/2;

//checks to make sure that the data is doing the calculations correctly

//this line could be removed

printf("The data is oldtime: %u.%u seconds, newtime: %u.%u seconds, difference: %f\n", oldtime.tv\_sec, oldtime.tv\_usec, newtime.tv\_sec, newtime.tv\_usec, time);

//stores the IP address of the server and the distance to the buffer

sprintf(buffer, "%s %.6f", myIP, distance);

//sends the IP address and the distance to the client program

if (sendto(sock, buffer, strlen(buffer), 0, (struct sockaddr \*)&broadcast\_addr, fromlen) < 0)

error("sendto");

}

else {

printf("Invalid message received: %s\n\n", buffer);

}

//}

}

pthread\_exit(0);

}