

2014

University of Missouri



[REAL-TIME EMBEDDED SYSTEM FOR SPORTS WATCH APPLICATION]

Abstract

This project illustrates an embedded system for a sports watch application. This system will provide real-time feedback of user name, time-stamp, GPS, heart-rate, and speed. The system will also log these metrics every 5 seconds, storing them in a database, and providing the user with valuable historic workout data. Finally, the system will make the real-time data of other users available through this database. This system has the potential to greatly improve the efficiency and experience of running or exercising.

Contents

Abstract.....	0
Table of Figures.....	2
Introduction	1
Background	1
TS7250.....	1
Auxiliary board	1
Hardware Interrupts	2
Interrupt Service Routine (ISR)	2
FIFO	2
Real-Time Tasks.....	3
Faircom Database	3
Method	3
Kernel Module.....	3
Init_module()	3
Cleanup_module()	4
Harware_interrupt_routine().....	4
User Space.....	4
Main().....	4
Get_time().....	5
Get_GPSI().....	5
Get_heart().....	6
Get_speed().....	6
Write_database()	6
Get_btn().....	7
Basic Flowchart	9
Results and Experiments.....	11
Results.....	11
Experiments	15
Discussions and Conclusions.....	15
Appendix	16

Table of Figures

Figure 1: Auxiliary Board – Pushbuttons 0 – 4 from right to left	2
Figure 2: User Instructions.....	4
Figure 3: Real-time thread initialization	5
Figure 4: Thread creation.....	5
Figure 5: Run_data struct.....	5
Figure 6: Store GPS Data	6
Figure 7: Speed Calculation.....	6
Figure 8: Button 0 pressed – code	7
Figure 9: Button 1 pressed – Code.....	7
Figure 10: Button 2 pressed – Code.....	8
Figure 11: Button 2 pressed - User prompt	8
Figure 12: Button 3 pressed - Code	9
Figure 13: Basic Flowchart	9
Figure 14: Advanced Flowchart	10
Figure 15: User program flow	10
Figure 16: Install Kernel Module	11
Figure 17: Program instructions.....	11
Figure 18: Instantaneous data - User 1.....	12
Figure 19: Logged data - User 1	12
Figure 20: Clear logged data	13
Figure 21: New logged data	13
Figure 22: New User.....	14
Figure 23: Instantaneous data - User 2.....	14
Figure 24: Searched user data	15

Introduction

When running or exercising, there are a few simple things that technology could add to make the workout much more effective. For one, the ability for the user to have a real-time feedback of certain metrics during the exercise can help immensely in building efficient workouts. It is also invaluable to be able to look back and see historical data from past workouts. This can help the user gain insight into their progress as well as helping them build their future routines. Finally, one of the best ways to maintain a running schedule or workout routine is to be held accountable by a friend who is also working out. Interacting with other users can help create this accountability even when the users are not necessarily in the same location.

This project proposes a sports watch application using real-time embedded programming to address these three criteria. The system will provide the user with real-time data including: user name, timestamp, GPS data, heart rate, and speed. In this application the data is simulated, and every time a button in the system is pressed, this data will be displayed giving the user instantaneous feedback of their current status. The data is updated at different periods in order to simulate different hardware constraints that might occur in an actual system of this sort. The system will also allow the user to look back at data from past workouts. The same metrics mentioned above are written to a table in a database that is specific to the user name selected by the user. This data is written to the database every 5 seconds and remains until the user decides to clear the log. This data log can be accessed by pressing a button in the system and can be cleared using another button. Finally, the system allows the user to search the database for someone else's username. This lets the user search for a friend's name which allows them to see, in real-time, the data points that their friend is logging.

The objective of this project is to deliver a system that is easy to use, and delivers on all of the functionality outlined above.

Background

TS7250

This project will be developed on the TS7250 which is a compact single board computer based on an ARM CPU. This board supports high level embedded operating systems such as Linux, Windows, and others. In our case, we use the Linux operating system. The board provides a real-time extension that allows for systems with strict timing requirements. The system also offers several peripherals such as an Analog-to-Digital Converter (ADC), General Purpose Input and Output pins (GPIO), hardware and software interrupts, and serial communication, among many others. In my case, I will be taking advantage of the real-time scheduling capabilities, the GPIO pins, and the hardware interrupts.

Auxiliary board

The auxiliary board shown in Figure 1 provides some additional hardware peripherals for the board. These include a speaker, three LEDs, and 5 pushbuttons. For this project, I will be using the first four

pushbuttons to replicate pushbuttons that you might find on an actual sports watch. I will number them as pushbutton 0 – 4.



Figure 1: Auxiliary Board – Pushbuttons 0 – 4 from right to left

Hardware Interrupts

The TS7250 board supports 32 hardware interrupts. Any of the GPIO pins can be configured as hardware interrupt. In my case, there were dedicated interrupt registers that needed to be set. `GPIOIntType1` and `GPIOIntType2` are the registers that control when interrupts are generated. Configuring these two registers determined that the interrupts would be generated on the falling edge of the signal on the GPIOs of Port B. This means that the interrupt is generated on the button press, not the release. `GPIOIntEnable` was used to control which pins of Port B would be able to produce an interrupt. For bits 0-3, a value of 1 was written to this register allowing the four push buttons of the auxiliary board to generate an interrupt. `GPIOBEOI` was the register designed to clear interrupts occurring on Port B. Finally, `RawIntStsB` was the register that is read by the interrupt handler to determine which pin of Port B generated the interrupt.

Interrupt Service Routine (ISR)

This is the routine that is entered when an interrupt occurs. It stores a pointer to the current Task Control Block (TCB) as well as the next TCB so that once the routine is finished executing it can return to the main program where it left. Within the ISR it is important to disable interrupts so that the ISR itself does not get interrupted. Once the routine is finished executing the interrupt is cleared and interrupts are re-enabled.

FIFO

A FIFO, also known as a named pipe, is a method for inter-process communication. Two separate processes will be able to access the pipe by name and read from it or write to it. In order to create a real-time FIFO, the command `rtf_create()` is used. The unique ID of the FIFO and the structure of the

data are passed as parameters. To write to the FIFO, `rtf_put()` is used. The ID of the FIFO, the pointer to the data, and the structure of the data are passed as parameters. To read from the FIFO, it must first be opened, and then the read command is used.

Real-Time Tasks

Real-Time tasks are tasks that have periods or deadlines that must be met. They require a scheduler to allocate CPU time in order for the system to run effectively. The following commands are used to initialize real-time tasks:

```
pthread_t thread_name  
  
ret = pthread_create(&thread, NULL, time, NULL);  
  
if (pthread_join(thread, NULL))  
  
base_period = start_rt_timer(nano2count(1000000));  
  
RT_TASK* task1 = rt_task_init(nam2num(" "), x, y, y);  
  
rt_task_make_periodic(task1, rt_get_time(), base_period / 5);
```

Faircom Database

The Faircom c-treeACE is a cross-platform database that offers high speed indexing. The system is broken down into four functions for simple use:

`Init()` – used to log on to the server

`Define()` – used to define the form of the tables in the database

`Manage()` – contains functions to add to records, delete, records, and display records

`Done()` – used to log off of the server.

Method

This project was broken down into two parts, the kernel module and the user space program.

Kernel Module

The Kernel module was broken down into the `init_module()`, the `cleanup_module()`, and the `hardware_interrupt_routine()`.

Init_module()

All of the initializations needed were handled in this module. The registers needed for the GPIO pins were mapped and the GPIO pins were configured. All of the configurations to set up the hardware

interrupt and make it a falling edge trigger were also handled here. Finally, the FIFO for inter-task communication was created here.

Cleanup_module()

This module was responsible for closing the FIFO and clearing/disabling all interrupts when the kernel module is removed.

Hardware_interrupt_routine()

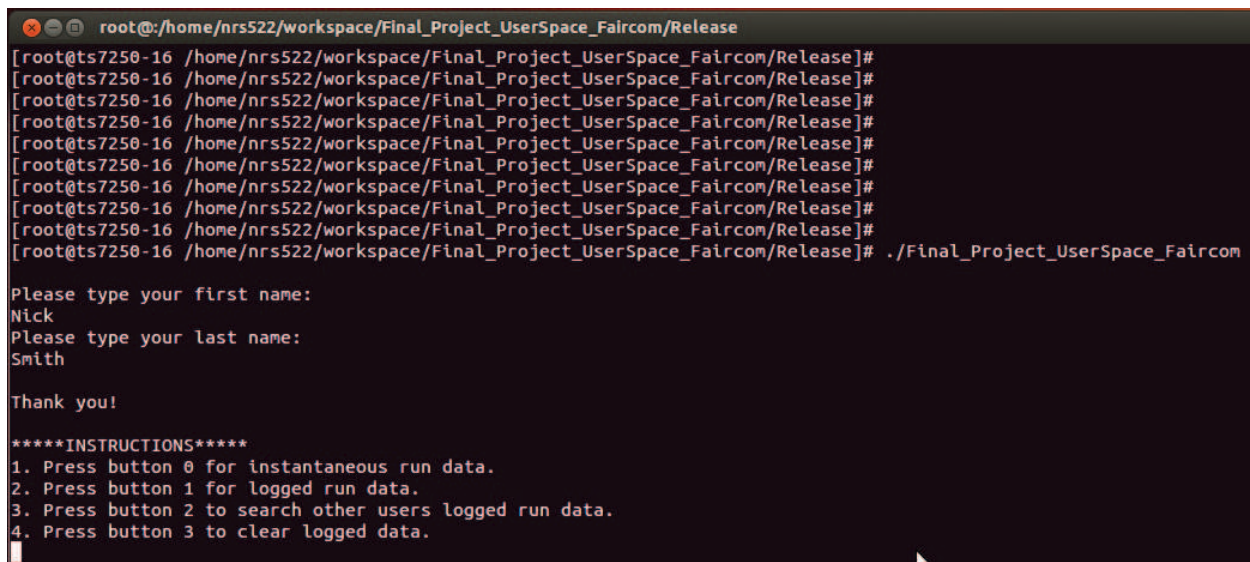
This function was the routine entered when a hardware interrupt occurred. First, the routine disabled the interrupts. Next, the system read the RawIntStsB register to determine what push button was pressed. It then wrote the an integer corresponding to the button number to the FIFO in order to signal that a button had been pressed and communicate what button it was.

User Space

The user space program was responsible for all of the user output, the data collection, and the database functions. It accomplished these tasks through a main thread and 6 real-time threads.

Main()

The main task was responsible for printing initial user instruction about how to interact with the program. This can be seen in Figure 2.

A terminal window with a dark background and light text. The title bar shows the path "/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release". The terminal content shows a series of shell prompts and user input. The user enters "Nick" for their first name and "Smith" for their last name. After a "Thank you!" message, a section of instructions is displayed, followed by a numbered list of four options for interacting with the program.

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]# ./Final_Project_UserSpace_Faircom

Please type your first name:
Nick
Please type your last name:
Smith

Thank you!

*****INSTRUCTIONS*****
1. Press button 0 for instantaneous run data.
2. Press button 1 for logged run data.
3. Press button 2 to search other users logged run data.
4. Press button 3 to clear logged data.
```

Figure 2: User Instructions

The main function takes input from the user in order to establish the username and then prints some instructions on how to use the program. Next, the function created all 6 of the real time threads required. The base period for these threads was initialized as 1 msec. The initialization code can be seen in Figure 3.

```

int ret;
// Initialize pthreads
pthread_t heart_thread, time_thread, GPS_thread, speed_thread, checkBtn_thread,
        faircom_thread;

// Base period is 1 msec
base_period = start_rt_timer(nano2count(1000000));

```

Figure 3: Real-time thread initialization

Figure shows an example of a thread being created.

```

// Create pthreads
ret = pthread_create(&time_thread, NULL, get_time, NULL);
if (ret != 0)
{
    printf("Error creating thread\n");
}

```

Figure 4: Thread creation

Get_time()

This function was responsible for updating a timestamp every 200 msec ($200 * \text{base_period}$). On every update, the timestamp was written to the `run_data` struct. The form of this struct can be seen in Figure 5.

```

//Initialize struct for run data
struct data
{
    char usr_name[50];
    char heart_rate[50];
    char speed[50];
    char timestamp[50];
    char GPS[50];
};

// Initialize run_data
struct data run_data;

```

Figure 5: Run_data struct

Get_GPS()

This function was responsible for updating the GPS location of the user every 200 msec ($200 * \text{base_period}$). On every update the GPS data was written to the `run_data` struct. The simulated data was stored for every 10 iterations to be used in the speed calculation. The loop to achieve this is shown in Figure 6.


```
// Randomly generate GPS value
int x = randr(1000,5000);
int y = randr(1000,5000);

// Store 10 values of GPS in order to calculate speed
GPS_x[idx] = x;
GPS_y[idx] = y;
if (idx < 10)
{
    idx++;
}
else
{
    idx = 0;
}
```

Figure 6: Store GPS Data

Get_heart()

This function was responsible for updating the heart rate of the user every 100 msec (100 * base_period). On every update the heart_rate data was written to the run_data struct.

Get_speed()

This function was responsible for updating the speed of the user every 2 sec (2000 * base_period). This longer period was used because updating the speed too often could result in invalid data. This is why the GPS data was stored. The speed calculation waited for 10 samplings of the GPS data and then used GPS data sample 1 and sample 10 to calculate the user speed. The code for the speed calculation is shown in .

```
//Calculate speed using GPS data of 9 and 0 and divide by period
double speed;
speed = abs(sqrt((GPS_x[9] - GPS_x[0])*(GPS_x[9] - GPS_x[0])
+ (GPS_y[9] - GPS_y[0])*(GPS_y[9] - GPS_y[0])) / 1200);
sprintf(run_data.speed, "%.21f", speed);
```

Figure 7: Speed Calculation

On every update the speed data was written to the run_data struct.

Write_database()

This function was responsible for writing the data in the run_data struct to the database every 5 sec (5000 * base_period). This was accomplished using the four Faircom functions, initialize(), define(), manage(), and done() as described in the background section.

Get_btn()

This function was responsible for the main user-facing functionality of the program. This function read the FIFO to determine what button was pressed and then acted accordingly. If button 0 was pressed the function would print out instantaneous data for the user. The code for this is shown in Figure 8.

```
// If button 0 is pressed print instantaneous data from local struct
else if (button == 1)
{
    printf("\nInstantaneous Run Data:");
    printf("\nNAME: %s",run_data.user_name);
    printf("\nDATE/TIME: %s",run_data.timestamp);
    printf("GPS: %s",run_data.GPS);
    printf("HEART RATE: %s BPM\n",run_data.heart_rate);
    printf("SPEED: %s MPH\n",run_data.speed);
}
```

Figure 8: Button 0 pressed – code

If button 1 was pressed the logged database from the database was displayed. The code for this is shown in Figure 9.

```
//If button 1 is pressed print logged data from faircom
else if (button == 2)
{
    printf("\nLogged Run Data:");
    Initialize();

    Define1();

    Manage2();

    Done();
}
```

Figure 9: Button 1 pressed – Code

If button 2 was pressed the program prompted the user to search the database for a username and then displayed the data of that user. The code for this is shown in Figure 10 and the user prompt is shown in Figure 11.

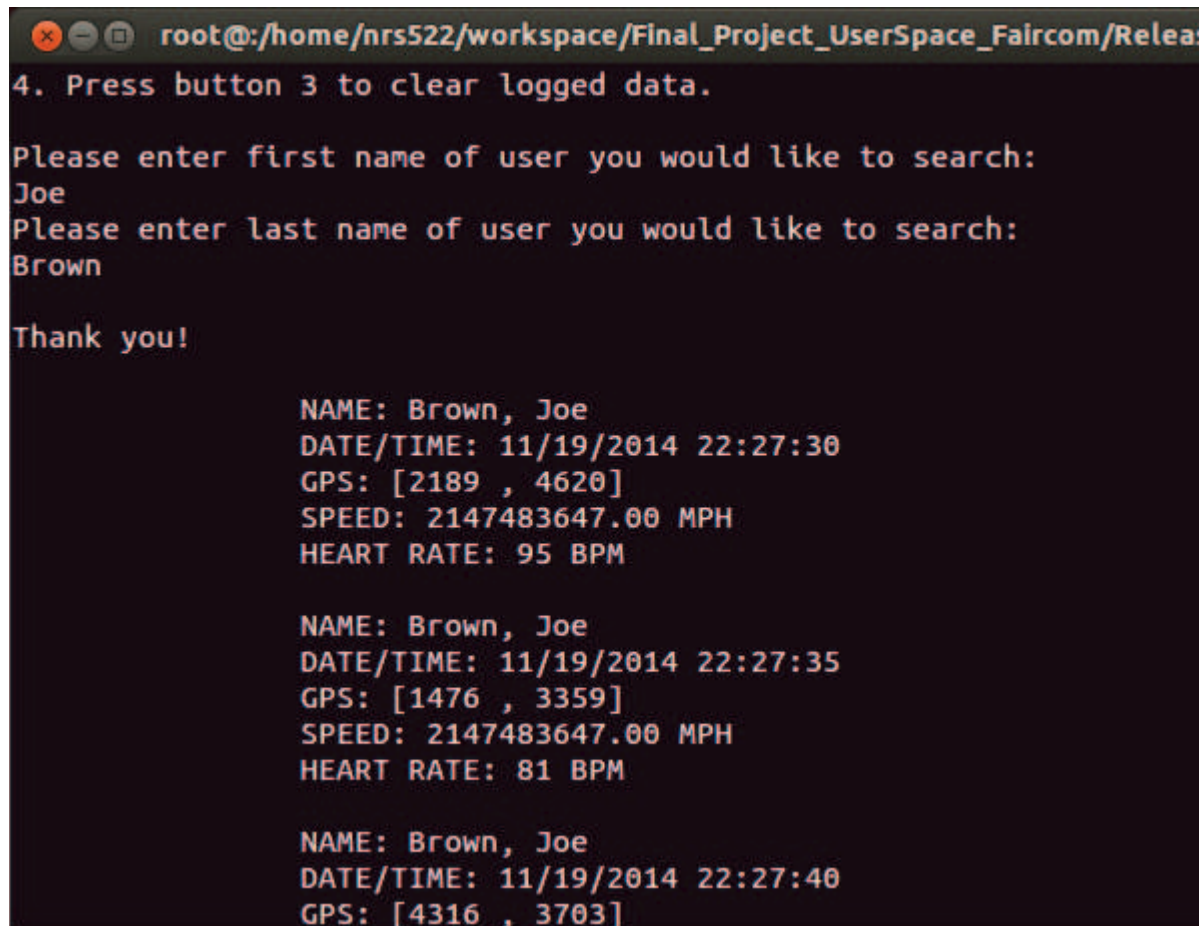
```
// If button 2 is pressed execute database search for other user
else if (button == 3)
{
    printf("\nPlease enter first name of user you would like to search:\n");
    scanf("%s", &search_firstname);
    printf("Please enter last name of user you would like to search:\n");
    scanf("%s", &search_lastname);
    sprintf(search_fullname, "%s, %s", search_lastname, search_firstname);

    printf("\nThank you!\n");

    fflush(stdout);

    Initialize();
    Define2();
    Manage2();
    Done();
}
```

Figure 10: Button 2 pressed – Code



A terminal window screenshot showing the execution of a program. The window title is "root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release". The prompt "4. Press button 3 to clear logged data." is shown. The user is prompted to enter a first name and last name, providing "Joe" and "Brown" respectively. The program then outputs "Thank you!" followed by three sets of sensor data for "NAME: Brown, Joe". The data includes DATE/TIME, GPS coordinates, SPEED, and HEART RATE, which change slightly between outputs.

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
4. Press button 3 to clear logged data.

Please enter first name of user you would like to search:
Joe
Please enter last name of user you would like to search:
Brown

Thank you!

NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:30
GPS: [2189 , 4620]
SPEED: 2147483647.00 MPH
HEART RATE: 95 BPM

NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:35
GPS: [1476 , 3359]
SPEED: 2147483647.00 MPH
HEART RATE: 81 BPM

NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:40
GPS: [4316 , 3703]
```

Figure 11: Button 2 pressed - User prompt

Finally, if button 3 was pressed the user's logged data was cleared from the database. The code for this is shown in Figure 12.

```
// If button 3 is pressed clear logged data from database
else if (button == 4)
{
    printf("\nClearing User Data\n");

    Initialize();
    Define1();
    Manage3();
    Done();
}
```

Figure 12: Button 3 pressed - Code

Basic Flowchart

A basic flowchart of the program structure is shown in Figure 13 to give a general idea of the program structures.

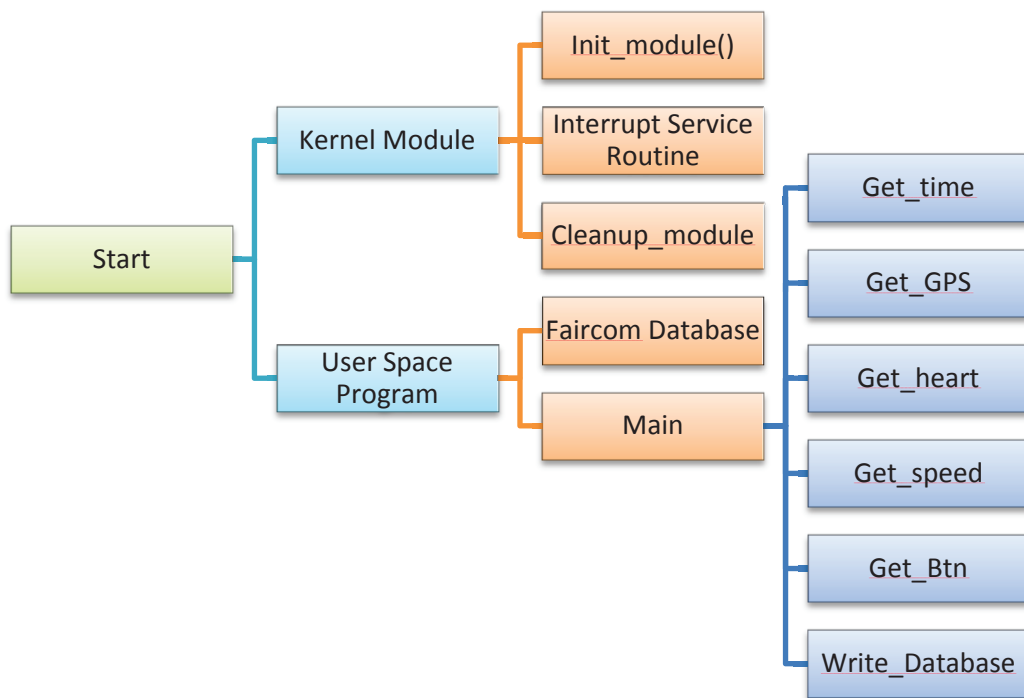


Figure 13: Basic Flowchart

A more detailed flowchart is shown in – to give specifics about the program functionality.

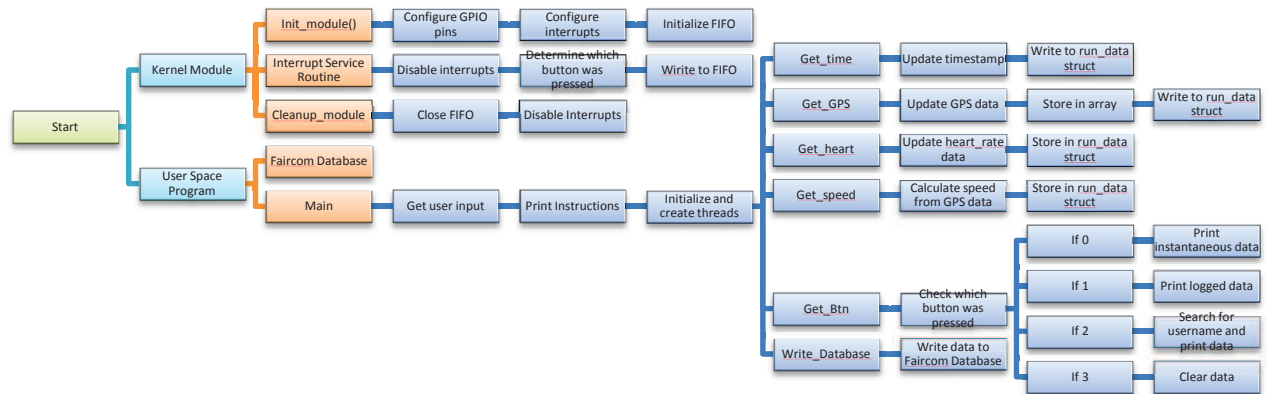


Figure 14: Advanced Flowchart

Figure 15 shows the program flow as the user would experience it.

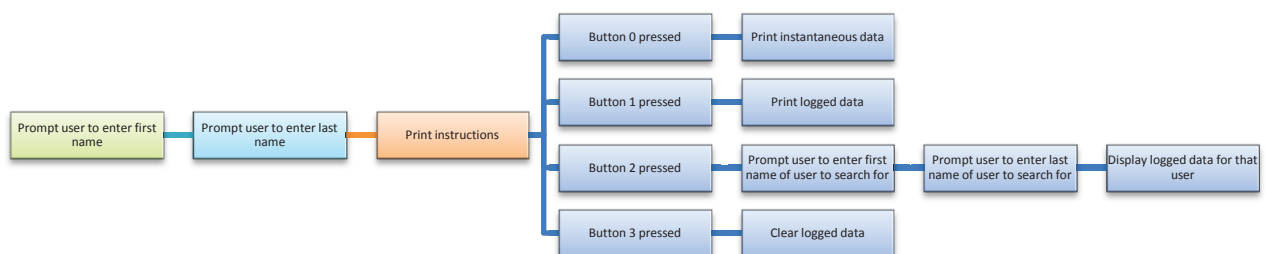


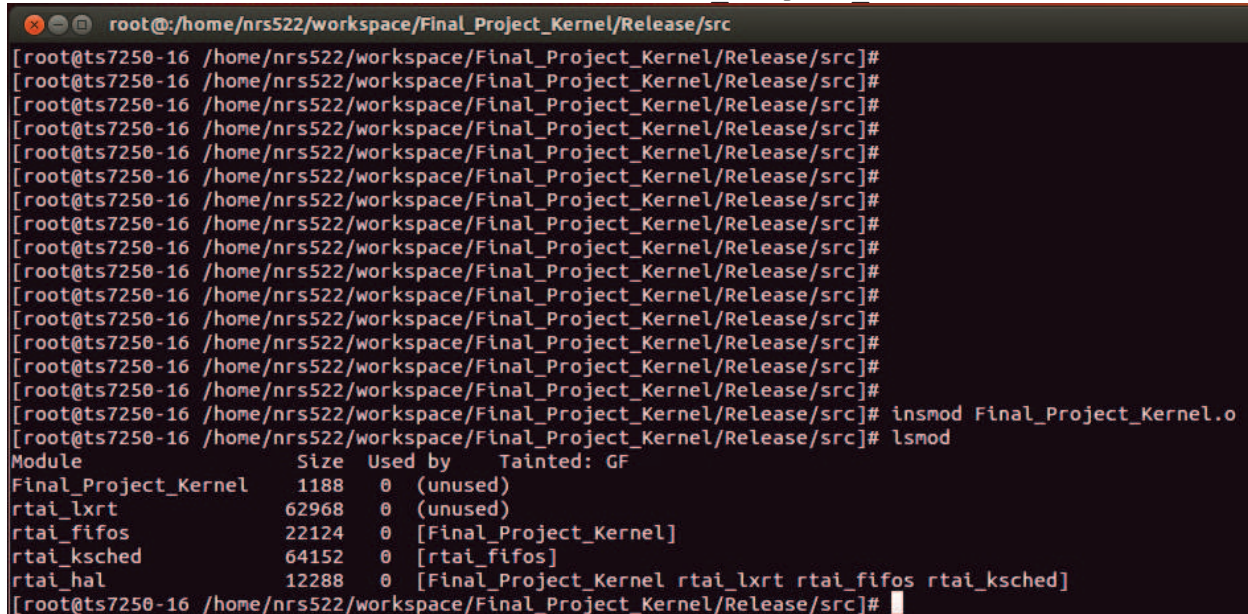
Figure 15: User program flow

Results and Experiments

Results

In order to present the results, I will follow the instructions in the README file that is included with this project. This will give a good idea of the functionality of the program.

1. Install the Kernel Module from: Final_Project_Kernel/Release/src



```

root@:/home/nrs522/workspace/Final_Project_Kernel/Release/src
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]# insmod Final_Project_Kernel.o
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]# lsmod
Module                Size  Used by    Tainted: GF
Final_Project_Kernel   1188      0 (unused)
rtai_lxrt              62968     0 (unused)
rtai_fifos             22124     0 [Final_Project_Kernel]
rtai_ksched            64152     0 [rtai_fifos]
rtai_hal              12288     0 [Final_Project_Kernel rtai_lxrt rtai_fifos rtai_ksched]
[root@ts7250-16 /home/nrs522/workspace/Final_Project_Kernel/Release/src]#

```

Figure 16: Install Kernel Module

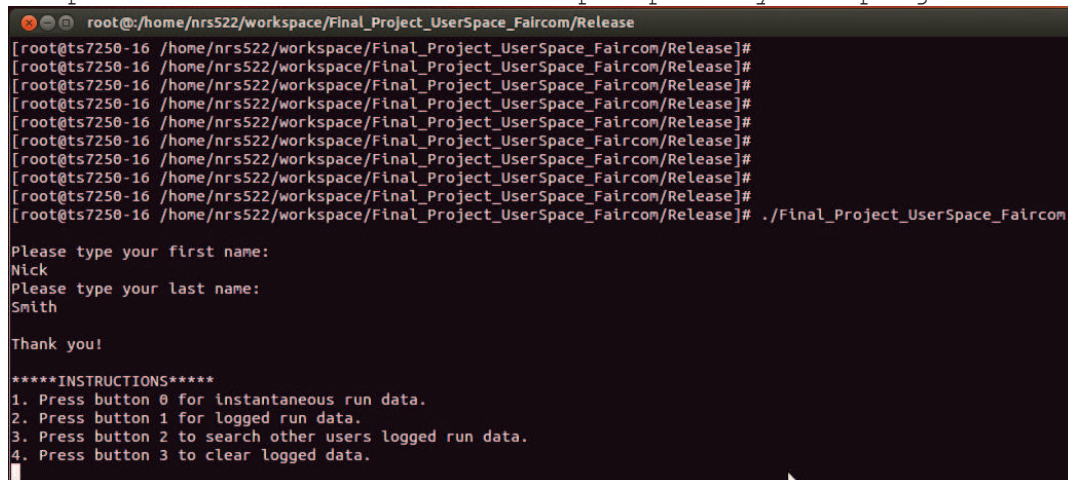
2. Run the server from: linux.v2.4.arm.32bit/bin/ace/isam

Use the command `./ctsrvr &`

3. Run the user space program from:

Final_Project_UserSpace_Faircom/Release

-Input a first and last name as prompted by the program



```

root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]# ./Final_Project_UserSpace_Faircom

Please type your first name:
Nick
Please type your last name:
Smith

Thank you!

*****INSTRUCTIONS*****
1. Press button 0 for instantaneous run data.
2. Press button 1 for logged run data.
3. Press button 2 to search other users logged run data.
4. Press button 3 to clear logged data.

```

Figure 17: Program instructions

-Click button 0 on the auxiliary board to see instantaneous data

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
HEART RATE: 92 BPM
SPEED: 15.00 MPH

Instantaneous Run Data:
NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:24:07
GPS: [3599 , 3209]
HEART RATE: 97 BPM
SPEED: 15.00 MPH

Instantaneous Run Data:
NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:24:08
GPS: [4239 , 3792]
HEART RATE: 88 BPM
SPEED: 15.00 MPH

Instantaneous Run Data:
NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:24:08
GPS: [4239 , 3792]
HEART RATE: 72 BPM
SPEED: 15.00 MPH
```

Figure 18: Instantaneous data - User 1

-Wait for 20 seconds and click button 1 to see logged data

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
DATE/TIME: 11/19/2014 22:26:30
GPS: [4316 , 3703]
SPEED: 15.00 MPH
HEART RATE: 67 BPM

Logged Run Data:
NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:26:20
GPS: [2189 , 4620]
SPEED: 15.00 MPH
HEART RATE: 95 BPM

NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:26:25
GPS: [1476 , 3359]
SPEED: 15.00 MPH
HEART RATE: 81 BPM

NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:26:30
GPS: [4316 , 3703]
SPEED: 15.00 MPH
HEART RATE: 67 BPM
```

Figure 19: Logged data - User 1

-Click button 3 to clear the logged data

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
GPS: [2252 , 4010]
SPEED: 15.00 MPH
HEART RATE: 72 BPM

NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:28:38
GPS: [4151 , 4786]
SPEED: 15.00 MPH
HEART RATE: 64 BPM

NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:28:43
GPS: [1422 , 4554]
SPEED: 15.00 MPH
HEART RATE: 64 BPM

NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:29:15
GPS: [4361 , 2577]
SPEED: 15.00 MPH
HEART RATE: 92 BPM

Clearing User Data
```

Figure 20: Clear logged data

-Wait for 20 more seconds and click button 1 again to see that logged data has been cleared and there is new data

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
Clearing User Data

Logged Run Data:
NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:29:30
GPS: [2189 , 4620]
SPEED: 15.00 MPH
HEART RATE: 95 BPM

NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:29:25
GPS: [4918 , 3975]
SPEED: 15.00 MPH
HEART RATE: 100 BPM

Clearing User Data

Logged Run Data:
NAME: Smith, Nick
DATE/TIME: 11/19/2014 22:29:35
GPS: [1476 , 3359]
SPEED: 15.00 MPH
HEART RATE: 81 BPM
```

Figure 21: New logged data

-Terminate the program

4. Run the user space program from:
Final_Project_UserSpace_Faircom/Release
- Input a new first and last name


```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
DATE/TIME: 11/19/2014 22:26:30
GPS: [4316 , 3703]
SPEED: 15.00 MPH
HEART RATE: 67 BPM

[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]#
[root@ts7250-16 /home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release]# ./Final_Project_UserSpace_Faircom

Please type your first name:
Joe
Please type your last name:
Brown

Thank you!

*****INSTRUCTIONS*****
1. Press button 0 for instantaneous run data.
2. Press button 1 for logged run data.
3. Press button 2 to search other users logged run data.
4. Press button 3 to clear logged data.
```

Figure 22: New User

-Click button 0 to see instantaneous data

```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Release
GPS: [1977 , 2304]
HEART RATE: 86 BPM
SPEED: 2147483647.00 MPH

Instantaneous Run Data:
NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:35
GPS: [3380 , 2448]
HEART RATE: 72 BPM
SPEED: 2147483647.00 MPH

Logged Run Data:
NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:30
GPS: [2189 , 4620]
SPEED: 2147483647.00 MPH
HEART RATE: 95 BPM

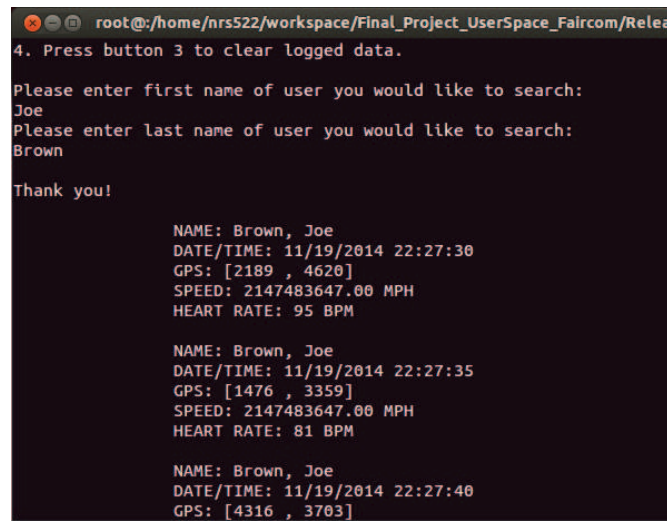
NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:35
GPS: [1476 , 3359]
SPEED: 2147483647.00 MPH
HEART RATE: 81 BPM
```

Figure 23: Instantaneous data - User 2

-Click button 2

-Follow board prompts and input first and last name used in step 3

-This will allow you to see data from the user created in step 3



```
root@:/home/nrs522/workspace/Final_Project_UserSpace_Faircom/Relea
4. Press button 3 to clear logged data.

Please enter first name of user you would like to search:
Joe
Please enter last name of user you would like to search:
Brown

Thank you!

NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:30
GPS: [2189 , 4620]
SPEED: 2147483647.00 MPH
HEART RATE: 95 BPM

NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:35
GPS: [1476 , 3359]
SPEED: 2147483647.00 MPH
HEART RATE: 81 BPM

NAME: Brown, Joe
DATE/TIME: 11/19/2014 22:27:40
GPS: [4316 , 3703]
```

Figure 24: Searched user data

Experiments

As far as debugging, I took many steps to ensure that my program was working. The first thing I did, and one of the most important, was to place print statements with timestamps in all of my real-time tasks. This allowed me to make sure that the tasks were running correctly and were adhering to the timing restrictions. This also allowed me to make sure that the variables being saved to the struct were the correct values.

Another test I did was over-inputting using the buttons. I would press the buttons very quick to see what the response of the program would be. This led me to discover a bug in the database write. If the buttons are pressed before data is logged and the program tries to read from an empty table in the database, the program will enter an error handler and terminate. This problem could be easily solved by setting certain conditions or using an error message and not terminating the program.

The final test I did was let my database log data for a long time (around 5 min. with data logging every 5 seconds). This allowed me to make sure that the historic data would in fact be available and there wouldn't be any errors when logging or reading the data.

Discussions and Conclusions

For the most part, the results of this program were what I expected. The implementation and debug were relatively straightforward. One suggestion that I received during the presentation was very beneficial. I would like to create a function that passes known values to variables. This way I could create a "curve" using the known values and a "curve" using the values from my program to validate that the output of my program was correct.

Most of the problems that occurred were from the Faircom Database. As I mentioned above, the most common problem that occurred was when my program tried to read from a database table that was

empty. Although this condition rarely occurs, it's important to handle it. This is a simple implementation problem that could be fixed with a condition that the table must exist before attempting to access it. It could also be fixed by removing the exit program condition from the error handler and just displaying an error message.

Overall this project helped me to gain good perspective on some of the practical uses of embedded programming. Through applying the techniques we learned in class to a real world problem, I gained a greater understanding for the importance of a solid theoretical background in embedded programming.

Appendix

Code is in the Final_Project folder in a PDF named Final_Project_Code.pdf.