Infrared Security System

By

Clark Buckles

Instructor

Dr. Guilherme DeSouza

ECE 4220 Real Time Embedded Computing

Department of Electrical and Computer Engineering

University of Missouri – Columbia

Columbia, Missouri

May 10, 2012

## Abstract

The following documentation describes an Infrared Security System. This device detects infrared movement through passive infrared sensors. It relays the information to embedded devices which communicate with a central control room. This device has four programs: a kernel module with real time task, a client, multiple servers, and a kernel module to check each passive infrared sensor. The real time task sends a signal to the client program every second. The client program reads the signal from the real time task and broadcasts a message to all the server programs. The client program also displays the security system interface. The server programs receive the message from the client and determine what status the sensor is in then sends the status back to the client. The last kernel module checks the status of the passive infrared sensor and sends that status to the server program.

## Introduction

This device provides an easy to use security system that allows the user to monitor multiple rooms from a single location or control room. Small and big businesses alike require use of security to monitor not just the perimeter of their residence, but also the interior of the building. Establishments like universities also use security devices inside and out. This system uses passive infrared sensors to detect changes in movement of the surrounding area. The user will be able to individually turn on and off the sensors for each room and monitor the status of the rooms from a single interface. The user will use the auxiliary board push buttons to provide input into the system. These push buttons can deactivate and activate sensors in a given room. For this project I simulate having two sensors in two separate rooms.

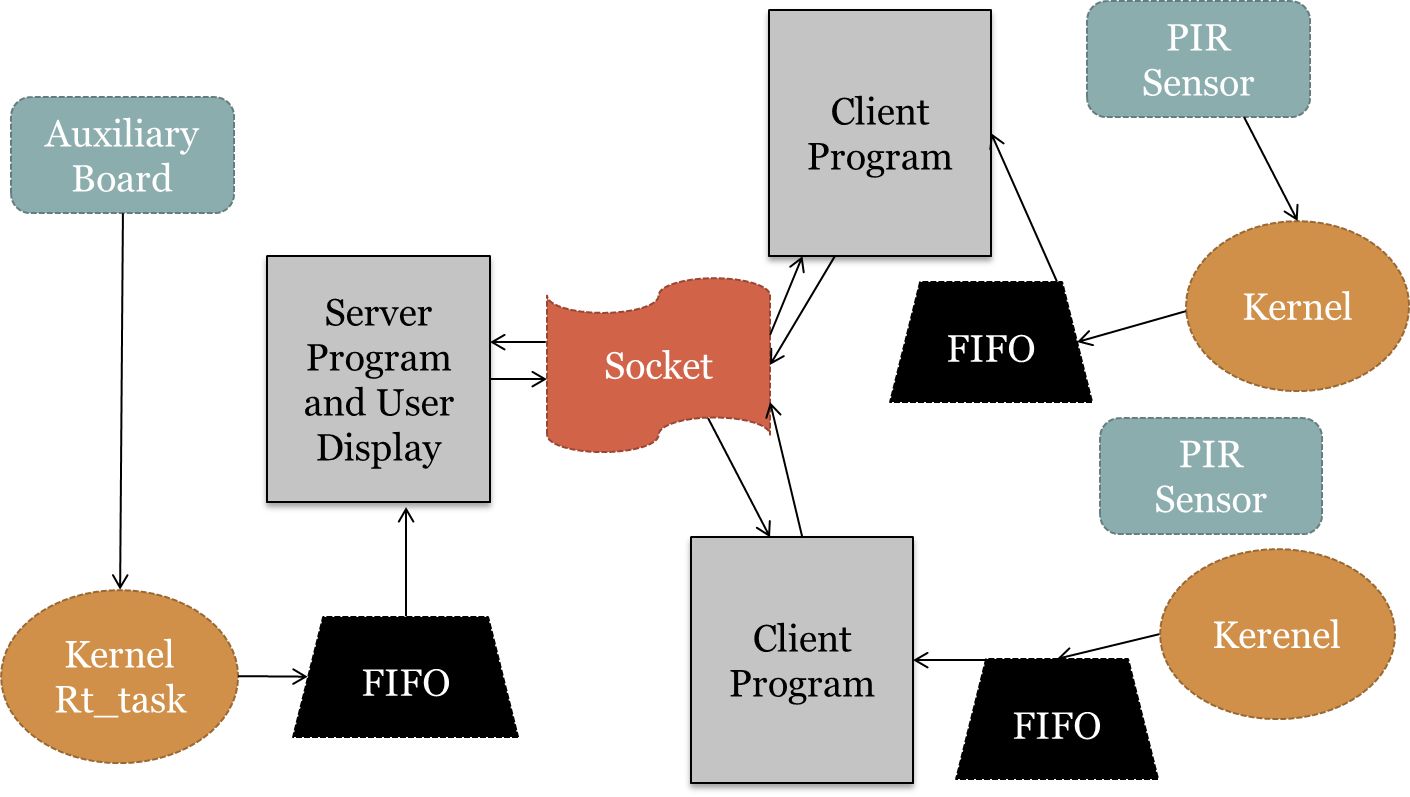
The parts used for this lab were the TS-7250, the auxiliary boards provided in lab, and passive infrared sensors. The digital input, output port was used on the 7250 to connect the auxiliary board. The auxiliary boards have five pushbuttons, three light emitting diodes (LEDs), and a single speaker. For this project I used two of the five pushbuttons and two of the LEDs. The pushbuttons are used to turn the sensors on and off and the LEDs are used to show the status of the sensors with the green button representing room one and the yellow button room two. The PIR Sensor is a pyroelectric device that detects motion by measuring changes in the infrared levels emitted by surrounding objects. When motion is detected the PIR Sensor outputs a high signal on its output pin. When no motion is detected it outputs a low signal. A high signal is represented by 3.3 V and a low signal 0 V. The PIRs have an operational voltage range of 3 V to 6 V.

## Class Topics Implemented

This system uses four topics that were discussed in class. These topics are real time process, hardware interrupt, FIFOS, and sockets. Real time tasks where used to check the status of the PIR sensors and to make sure the system completes one iteration every one second. A hardware interrupt causes the process to save its current state of execution and executes an interrupt handler. This program uses the four pushbuttons as hardware interrupts. When one is pushed the interrupt handler changed the message to be sent to the fifo. The FIFOs were used to communicate between the user space and kernel modules. In this project I implement two first in first out data structures. One stores the next message to be sent to the client program and the other stores the state of the passive infrared sensor. Finally, socket communication is used to communicate between the central computer or client program and the server programs that are running on separate TS-7250 boards.

## Implementation

Here is a block diagram of the security system. A detailed explanation is shown below.



In the above figure each enclosed section represents a different TS-7250 board. As you can see in this case there is a client program and two identical server programs. The first task is the kernel real time task. This task creates a real time task and makes it periodic giving it a period of one second. It then creates a FIFO with index zero. Next the program maps the interrupt registers necessary for the hardware interrupt. The program uses bit masking to set the two pushbuttons as hardware interrupts. The interrupts are generated by the falling edge of the pushbuttons by setting the necessary GPIOBIntType2 register bits to zero. The real time task writes the value of a variable called adata to a FIFO. Adata will always be 1 unless an interrupt is triggered. If an interrupt is triggered the interrupt handler is started. The interrupt handler checks the status of the interrupt register to determine which push button has been pressed and changes the value of adata so that the next time the real time process writes to the fifo it writes the new value. After this value is written the value of adata is changed back to 1.

The next program is the client. The client program creates a connectionless socket or UDP. The connectionless approach is not as reliable as the streaming style of TCP but the connectionless style is faster since it does not wait for a direct connection to be established. For this program the messages sent and received had a set length of twenty. The client program reads the value of adata from the FIFO and determines what message to send to the server programs. It then sends the message and waits for the servers to reply. When the reply messages are read the client program then changes the status’ of the rooms accordingly and updates the display screen.

The server programs start by creating their part of the connectionless socket. The program then receives the message from the client program. It takes this message and determines the message that it needs to send back to the client. In some cases it will need to read from the FIFO to determine the status of the PIR. The final program is the real time kernel module for the server programs. This program uses the check\_button() function to determine the input of the PIR sensor and stores the status into a FIFO where it can be read by the server program.

The next section explains the inputs and resulting outputs of the security system. By default the PIR sensors are set to off. The default message that the client sends is ‘STATUS’. The servers will both respond with ‘OFF’ and this will be displayed. If push button one is pressed an ‘ACTIVATE1’ message is sent to the servers. Server in room one returns with a ‘WARMINGUP’ message for forty seconds while the sensor gets acclimated to its surroundings. After forty seconds the room one status will display ‘NORMAL’ if nothing is detected or ‘DETECTION’ if movement is detected. This is the same for room two.

## Results

The result of this system was fairly successful. The real time interrupts performed well with every push of the button sending the appropriate message to the client. However, the auxiliary board push buttons would have a bouncing effect and the program would think that the push button was pressed multiple times even though the interrupt was disabled. The client program read from the FIFO and determined the message to be sent correctly. It also displayed the interface properly. The server programs successfully received the client’s message and were able to determine what message to send back. The real time kernel module was unable to determine the PIR sensors output. When trying to read the output the program uses the check\_button() function. I do not believe this function to be compatible with the PIR sensor output. Without changing the program I connected an auxiliary board in place of the PIR sensor. When pressing the appropriate push button that took place of the PIR sensor output the check\_button() function was able to show when the button was pressed. In the video demonstration of this project I pushed the button of the auxiliary board to simulate the sensor outputting a logic high.

## Conclusion

In conclusion this system was successful when the auxiliary board was used to simulate the input of the PIR sensor. Future work of this project could include a program that when the alarm system is off, turns on the lights of the room when someone enters. Another tool that might be useful is to add a camera that records data of the activity in the room when motion is detected.