Project 4: Embedded Systems

PROJECT GOALS

This project will familiarize you with some simple distributed systems characteristics and tools. It will also give you some experience in working with hardware for embedded systems and provide some exposure to basic tools for securing distributed systems. The goals are:

- primary: build a simple client end of a client/server distributed system.
- primary: learn about working within the constraints of devices designed to support embedded systems.
- secondary: build a program to a specified protocol interface.
- secondary: obtain experience working with simple peripheral devices.
- secondary: obtain experience with simple networking debugging.

ASSIGNMENT OVERVIEW

The assignment is divided into two general parts.

1. Building an application that supports the use of a sensor to gather data on an embedded device.
2. Convert the application that interacts with the sensor to become a client using a

predefined network protocol to interact with a remote server program.

In this assignment, you will:

- Learn how to perform basic operations on the Intel Edison.
- Learn how to connect simple sensor devices to the Edison and access them from a local application.
- Implement and demonstrate an application that uses sockets to communicate with a shared server application on a remote machine.

Your deliverables for this assignment will include:

- A program that integrates a sensor into the basic Edison platform and gathers readings from that sensor.
- A test run in which your program interacts with the remote server and performs all supported operations specified in the design for the application. This test run will store a log file on the remote server machine, which will be used in grading your assignment.

To perform this assignment, you will need to learn about the Edison platform. There are several useful tutorials on working with the Edison that you might find helpful available on line:

Edison tutorials

Also, you will need to use a temperature sensor in the Grove sensor kit for this project. Here's a link that provides some information on using this sensor:

Temperature sensor information

PART 1 - Building a sample Edison embedded device

Summary of Deliverables

- the source for a C program and *Makefile* that cleanly (no warnings) builds using *gcc* on an Edison Linux system, implementing the functionality specified below.
- The contents of a log file showing the program operating on your Edison for at least 60 seconds.

Detailed Instructions

Write a program that uses the Edison to access the temperature sensor included in the Grove sensor kit. The program should read the sensor once per second and output its reading (in Fahrenheit) to a shell attached to the serial port, as the tutorials indicate. Also output these readings to a log file, in the format:

Timestamp Temperature

(Use a space between the timestamp and temperature, not a tab or multiple spaces.) The timestamp should be obtained by running the time() system call on the Edison, and should be converted to an HH:MIN:SEC format. The temperature should be in the format ##.#. (For example, 98.6.) Take measures to ensure that you measure several different temperatures. (Hint: holding your finger on the sensor is likely to produce a hotter reading than the air temperature.)

Optional extra credit: Also connect the display device that comes in the Grove kit to the Edison and display all temperature readings on the display, as well as sending them to the shell and saving them in a log. To obtain the extra credit, you must indicate in your submission that you have added this functionality, and it must properly display the temperatures in a format matching the log file's temperature column.

PART 2 - Integrate your Edison sensor device into a client/server system

Summary of Deliverables

- the source for a C module and *Makefile* that cleanly (no warnings) builds using *gcc* on an Edison system and implements the functionality specified below.
- A log file showing the temperatures your device sent to the server and all commands received from the server.

Detailed Instructions

Here is the specification for the device you are to build using the Edison and how it should interact with a remote server. Much of the functionality was already built in part 1, but you must add all additional functionality related to networking.

Remote temperature sensor device

BASIC DEVICE DESCRIPTION

The device connects to a server through a wireless network. In basic form, it does not use any encryption of its own. (If the wireless network you are using has 802.11 link encryption, that is fine. You should not need to do anything special in your device code to use such encryption.)

Every T seconds (T=3 by default), the device takes a temperature reading. It sends that reading to the server in the following format:
"[DevID] TEMP=##.#" where "##.#" is replaced by a reading to one decimal place. [DevID] should be a unique 9 decimal digit device identifier. This identifier should be your 9 digit student ID, expressed as a string, not as an integer.

For example,

123456789 TEMP=98.6
789123456 TEMP=32.0

(Use a space between the device ID and TEMP, not a tab or multiple spaces.) Readings are sent even if they are repetitions of the previous reading, by default.

Temperature can be measured and reported in either Fahrenheit or Centigrade. Fahrenheit is used by default. Every reading should be stored in a log file, on a separate line, in the following format:

Timestamp Temperature F/C

(Use a space between the timestamp, temperature, and scale fields, not a tab or multiple spaces.) The timestamp should be obtained by running the time() system call on the Edison, and should be converted to an HH:MIN:SEC format. The temperature should be in the format of "##.#". If the device is currently reporting Fahrenheit measurements to the server, the final column should contain "F". If it is currently reporting Centigrade measurements to the server, the final column should contain "C".

COMMANDS

The device should accept the following commands from the server. Extraneous characters at the end of a command should result in the command being ignored. For commands accepting a parameter, non-valid values of the parameter should result in the command being ignored. No error messages will be sent to the server when commands are ignored. Each command received, whether valid or invalid, should be saved into the same log file as the temperature readings. The entire command received should be written to the log, each on its own line. Invalid commands should have the characters " I" (space I) appended to the end of the received command, to indicate they were invalid.

TURN OFF - The device should turn off, which means the program should stop taking

readings or responding to commands. When off, the device should restart only when a button on the device is pressed (or, alternately, by input from the command shell on the Edison). The program can actually be halted or merely put into a dormant mode, as the student prefers. When the program is turned on again via button press or command shell input, all parameters should be set to their default values, not the values they were in when the program was turned off. Nothing sent to the device while it is turned off should be logged or even read.

Format – “OFF”

STOP – The device should stop taking and reporting readings. However, the program should continue running and should continue to accept remote commands. If another STOP command is received while the device is in stop mode, it should be treated as a no-op, but should be logged as a normal command.

Format – “STOP”

START - If the device is in stop mode, the device should start taking readings and should report them. It should restart in whatever state it was in when it stopped, meaning the interval of measurement and reporting will remain the same, the choice of Fahrenheit or Centigrade should be consistent, and whether the temperature is displayed locally should be the same (if that extra credit functionality is implemented).

Format – “START”

CHANGE SCALE – The device should start using the scale specified in the command to report its temperature reading. It should continue using this scale across STOP and START commands. A TURN OFF command followed by a restart should start it in the Fahrenheit mode, the default scale. A CHANGE SCALE command requesting the same scale as is currently in use should be treated as a no-op, but should be logged and not treated as erroneous.

Format – “SCALE=[F/C]” where “F” means change to Fahrenheit and “C” means change to

Centigrade.

Examples: SCALE = F
SCALE = C

CHANGE FREQUENCY – The device should change the frequency at which it measures temperature. The frequency is measured in seconds and should be an integer value between 1 and 3600. If the requested frequency is the same as the current frequency, the command should be treated as a no-op, but should be logged.

Format - "FREQ=#####" where "#####" is a number between 1 and 3600. The number may have leading zeros, but should be properly interpreted whether it does or does not have them.

Examples: FREQ = 5
FREQ = 010
FREQ = 2400

Optional extra credit: You can also implement the display command (described below), if you performed the extra credit portion of Part 1. If you do not implement the display command, your program should treat it as an invalid command. To obtain the extra credit, you must indicate in your submission that you have added this functionality, and it must properly display the temperatures in a format matching the log file's temperature column. It should display only those temperatures measured while the display was on.

DISPLAY – Turn the use of the Edison display for showing the measured temperature on and off. It should be on by default. "DISP N" should turn the display off, and "DISP Y" should turn it on. When on, it should show the same temperature sent to the server after the last reading, until another temperature is measured, in the same scale, followed by either "F" (when using Fahrenheit) or "C" (when using Centigrade). When off, nothing should be displayed. A display command requesting the already current mode should be treated as a no-op, but should be logged and should not be considered erroneous. There will be a single space between the command name and the parameter, not a tab or multiple spaces.

Format – “DISP [Y/N]”
Examples: DISP Y
DISP N

PROTOCOL

The Edison should set up a TCP connection to the server (currently lever.cs.ucla.edu) using a socket mechanism. The device should initially contact the server on port 16000 and send a message containing “Port request XXXXXXXXX”, where “XXXXXXXXX” is the student’s nine digit student ID number. The server will respond with a message containing nothing but a new port number, in the form of an integer. The device should then disconnect from port 16000 and connect to a socket on that new port number and continue to use that new socket until either the device or the server is disconnected.

On top of TCP, the device should be prepared to accept a command at any time and act upon it when received. Commands will not require the device to send a response. Also, as long as the device is in measurement mode, it should send a message on that socket in the format described earlier reporting the temperature read from its sensor at the specified period. There will be no application-level acknowledgement of the measurement messages. The device should not send any messages to the server on this socket other than measurement messages.

The server will send several commands to the client, ending with a OFF command.