

Brad's Raspberry Pi Blog

Tuesday, July 15, 2014

C# (Mono) Code to Read an TMP102 I2C Temperature Sensor on a Raspberry Pi

Mono is an open source implementation of the Microsoft .NET framework that allows you to compile and run .NET programs on non-Windows platforms. The Raspberry Pi. It actually proved to be quite easy to install Mono (v. 3.2.8) on the RPi and get some C# code running. This example shows how to call the i2c C# program to read the temperature from a TMP102 and print the results in degrees Celsius and Fahrenheit in the terminal window.

To install Mono, run the following apt-get command as root:

```
apt-get install mono-complete
```

This will get you the complete Mono package.

For this example, I am using the TMP102 on a breakout board from Sparkfun.

Connect the SDA and SCL pins between the TMP102 and the RPi. Connect VCC to 3.3V and connect the ground. With the solder jumper on the TMP102 in default) the I2C device address is 0x48.

The i2cget utility is included with the i2c-tools package. If you don't already have it installed, you can run the following command as root to install it -

```
apt-get install i2c-tools
```

C# Code

Use your favorite text editor to save this code to a file called Tmp102.cs.

```
using System;
using System.Diagnostics;
using System.Threading;

public class Tmp102
{
    private string i2cgetExe = "/usr/sbin/i2cget";
    private string i2cgetCmdArgs = "-y 1 0x48 0 w";
    private string hexString = "";
    private Process p;

    public Tmp102()
    {
        p = new Process();
    }

    public double tempC
    {
        get { return readRawTempData() * 0.0625; }
    }

    public double tempF
    {
        get { return this.tempC * 1.8 + 32; }
    }

    private int readRawTempData()
    {
        // Don't raise event when process exits
        p.EnableRaisingEvents = false;
        // We're using an executable not document, so UseShellExecute false
        p.StartInfo.UseShellExecute = false;
        // Redirect StandardError
        p.StartInfo.RedirectStandardError = true;
        // Redirect StandardOutput so we can capture it
        p.StartInfo.RedirectStandardOutput = true;
        // i2cgetExe has full path to executable
        // Need full path because UseShellExecute is false

        p.StartInfo.FileName = i2cgetExe;
```

```

        // Pass arguments as a single string
        p.StartInfo.Arguments = i2cgetCmdArgs;
        // Now run i2cget & wait for it to finish
        p.Start();
        p.WaitForExit();
        // Data returned in format 0xa017
        // Last 2 digits are actually most significant byte (MSB)
        // 2 digits right after 0x are really least significant byte (LSB)
        string data = p.StandardOutput.ReadToEnd();
        // Get LSB & parse as integer
        hexString = data.Substring(2, 2);
        int lsb = Int32.Parse(hexString,
            System.Globalization.NumberStyles.AllowHexSpecifier);
        // Get MSB & parse as integer
        hexString = data.Substring(4, 2);
        int msb = Int32.Parse(hexString,
            System.Globalization.NumberStyles.AllowHexSpecifier);
        // Shift bits as indicated in TMP102 docs & return
        return ((msb << 8) | lsb) >> 4;
    }

    public static void Main()
    {
        Tmp102 t = new Tmp102();
        while(true)
        {
            // Print temp in degrees C and F to console
            Console.WriteLine("{0}°C {1}°F", t.tempC , t.tempF);
            Thread.Sleep(1000);
        }
    }
}

```

Compiling & Running the Code

To compile a Mono executable, run the following command:

```
gmcs Tmp102.cs
```

Then execute the program by running

```
mono Tmp102.exe
```

Press Ctrl-C to terminate the program.

Posted by Brad Berkland at 7:15 PM No comments:

 Recommend this on Google

Labels: C#, I2C, Mono

Saturday, July 12, 2014

Displaying Current Temperature on a Web Page Using a TMP102 & PHP on a Raspberry Pi

Here is an example of using PHP to read the current temperature from a TMP102 I2C temperature sensor and display it (in degrees Fahrenheit) in a Web page.

Prerequisites

Make sure that you have installed the i2c-tools package because this package provides i2cget to read the registers on the TMP102. The following apt-get command will install it for you:

```
sudo apt-get install i2c-tools
```

Install the Apache2 Web server:

```
apt-get install apache2
```

Then install PHP5:

```
apt-get install php5
```

Granting Access to I2C System Device

By default, and for the sake of security, the account used to run the Apache2 Web server doesn't have permission to access the `/dev/i2c-1` device. To fix this, edit the `/etc/passwd` configuration file that controls the file permissions for `/dev/i2c-1`.

Use nano or your favorite text editor to edit `/lib/udev/rules.d/60-i2c-tools.rule`. Replace the contents of the file with these two lines.

```
KERNEL=="i2c-0", GROUP="i2c", MODE="0660"
KERNEL=="i2c-1", GROUP="i2c", MODE="0666"
```

This leaves the i2c-0 device with the default permissions and changes the access to i2c-1 to allow a non-privileged user to read the device.

PHP Code

Here is a simple PHP page that reads the temperature data using i2cget, converts it to degrees Fahrenheit and displays it. Put this code in `/var/www/temp`

```
<?php
$output = `/usr/sbin/i2cget -y 1 0x48 0 w`;
$msb = hexdec(substr($output, 4, 2));
$lsb = hexdec(substr($output, 2, 2));
$tempF = (((($msb << 8) | $lsb) >> 4) * 0.0625 * 1.8 + 32);
echo "Temp: $tempF&deg;F";
?>
```

The backticks (```) tell PHP to run the enclosed command in a (Linux) command shell. The backtick is on the key to the left of the 1 key near the top of the keyboard.

In the command `i2cget -y 1 0x48 0 w`

- `-y` cancels the user prompt that asks if you really want to read from the device
- `1` actually refers to the 2nd I2C bus, `/dev/i2c-1`
- `0x48` is the default I2C device address for the TMP102
- `0` is the register with the temperature data
- `w` means we want to read a word (rather than just a byte).

Note that when i2cget reads the data, the byte order is reversed. The first byte is actually the least significant byte (LSB) and the second byte is the most significant (MSB). The string returned by i2cget is something like `0xb01d`. The code uses `substr()` to pull out the two digits for each byte, and then `hexdec()` to convert it to a decimal numeric value.

From a computer on the same network, access the URL for the page (where the IP address will vary):

`http://192.168.0.108/temp.php`

Posted by Brad Berkland at 8:35 PM 1 comment:

 Recommend this on Google

Labels: I2C, PHP

Sample Java Code to Read Temperature from an TMP102 via I2C & Save Data to a MySQL Database Table

The Java code below reads current temperature data from the TMP102 temperature sensor using the `i2cget` command line utility to read the TMP102's register. The reading is taken every second and the result is saved to a MySQL database table. The table contains a column for the date and time and a column for the temperature in degrees Fahrenheit. The code below should work with Java 6, 7, or 8. You will need to have the complete JDK installed to compile the code.

Prerequisites

The `i2cget` command line utility is part of the I2C Tools package. You can install this package on the Raspberry Pi using the following `apt-get` command:

```
apt-get install i2c-tools
```

To install MySQL, use the following command and follow the prompts:

```
apt-get install mysql-server
```

The following command installs the Java database (JDBC) driver that allows your code to access the database:

```
apt-get install java-mysql-connector
```

Database Structure

```
CREATE TABLE tmp102 (
  date_time      datetime NOT NULL,
  temp_f         double,
  PRIMARY KEY (date_time)
);
```

Java Code

```
import java.io.*;
import java.util.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.PreparedStatement;

public class Tmp102 {
    // i2cget command to read raw data as a word (16-bit value)
    // from TMP102 sensor. -y tells command to run
    // without confirmation from user.
    // I2C address for TMP102 by default is 0x48.
    // Command is passed a separate String, unlike loadPinMapCmd
    // that has to be passed as an array of Strings.
    String runI2CGetCmd = "i2cget -y 1 0x48 0 w";

    Connection conn = null;
    PreparedStatement stmt = null;
    // Change gpsdb to name of your DB. Also change
    // DB user name & password
    String url = "jdbc:mysql://localhost/temperature?" +
        "user=YourDBUser&password=YourDBPassword";

    String insertSQL = "INSERT INTO tmp102(date_time, temp_f) " +
        "VALUES (now(), ?)";

    public Tmp102() {
        // Constructor sets up DB connection
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(url);
        }
        catch(Exception ex) { ex.printStackTrace(); }
    }

    // Read temperatures from both sensors & print it out to terminal
    public void readTemperature() {
        short tempData = 0;
        double temp = 0.00;
        tempData = runI2CGet(runI2CGetCmd);
        tempData = reverseBytes(tempData);
        temp = convertToDegreesF(tempData);
        try {
            stmt = conn.prepareStatement(insertSQL);
            stmt.setDouble(1, temp);
            stmt.execute();
            System.out.printf("%.3f\n", temp);
        }
        catch(Exception ex) { ex.printStackTrace(); }
    }

    // Need to reverse bytes in value returned by i2cget command
    private short reverseBytes(short tempData) {
        int b1 = (byte) tempData & 0xFF;
        int b2 = (byte) (tempData >> 8) & 0xFF;
        return (short) (b1 << 8 | b2);
    }

    // Convert raw temperature data to degrees Fahrenheit
    private double convertToDegreesF(short tempData) {
        return (tempData >> 4) * 0.0625 * 1.8 + 32;
    }

    private short runI2CGet(String cmd) {
        String line = "";
        short data = 0;
        try {
            Process p = Runtime.getRuntime().exec(cmd);
            p.waitFor();
            // Use Scanner to read results from i2cget command
            Scanner scan = new Scanner(
                new InputStreamReader(p.getInputStream()));
            // Shell command returns hex number as string like 0x6017.
            // Need to use substring to skip over 0x for parsing.
        }
    }
}
```

```

        // Using Short.parseShort() sometimes throws NumberFormatExceptions.
        if(scan.hasNext()) {
            data = (short) Integer.parseInt((scan.nextLine())
                .substring(2), 16);
        }
    }
    catch(Exception ex) { ex.printStackTrace(); }
    return data;
}

private void closeDBConn() {
    try {
        stmt.close();
        conn.close();
    } catch(Exception exc){ exc.printStackTrace(); }
}

public static void main(String[] args) {
    Tmp102 temp = new Tmp102();
    try {
        while(true) {
            temp.readTemperature();
            Thread.sleep(1000);
        }
    }
    catch(Exception ex) { ex.printStackTrace(); }
    finally { temp.closeDBConn(); }
}
}

```

Compiling & Running the Program

```
javac -cp /opt/mysql-connector-java-5.1.31/mysql-connector-java-5.1.31-bin.jar Tmp102.java
```

Adjust the -cp path for the MySQL-Connector .jar file to match your installation, if needed.


The following command runs the program. The temperature readings are printed to the terminal, while the current timestamp and temperature are saved in the database.

```
java -cp ./opt/mysql-connector-java-5.1.31/mysql-connector-java-5.1.31-bin.jar Tmp102
```

Again, adjust the -cp path to match the location of your MySQL-Connector .jar file.

Stop the program by pressing Ctrl-C.

Posted by Brad Berkland at 2:39 PM No comments:

 Recommend this on Google

Labels: I2C, Java, MySQL

Saturday, July 5, 2014

Note about External Connections to a MySQL Instance Running on the Raspberry Pi


If you want to connect to a MySQL DB server instance running on a Raspberry Pi, a couple bits of additional configuration are required.

On the Raspberry Pi, use nano or another text editor to edit the `/etc/mysql/my.conf` file. Under the `[mysqld]` section, edit the `bind-address` to match the IP address, if you wish to restrict access to one IP interface's address or set it to 0.0.0.0 to all access using the IP address for any available network interface.

To allow remote access via the MySQL root login, (re)run the `mysql_secure_installation` script and answer the relevant question about remote access for root. The following MySQL command (issued at the MySQL command line) has the same effect. Change the password as appropriate.

```
grant all privileges on *.* to 'root'@'%' identified by 'MyRootPassword' with grant option;
```

Posted by Brad Berkland at 4:56 PM No comments:

 Recommend this on Google

Labels: MySQL

Tuesday, June 24, 2014

Java Code to Read GPS Data from a Sparkfun Copernicus II DIP Module & Store Readings in JavaDB Database Table on a Raspberry Pi

This example uses Java 8 to connect to the Copernicus II GPS module via a serial connection. For instructions on how to install Java 8 on a Raspberry Pi, see [tutorial](#).

To install the RXTXComm serial library for Java, run the following apt-get command:

```
apt-get install librtx-java
```

Last year, I posted a similar example that uses MySQL, but since JavaDB (also known as Derby) comes with the Java 8 installation, it is very convenient and

Configuring JavaDB

While JavaDB comes with the Java 8 JDK, a small amount of configuration is needed. JavaDB can run in embedded or in network server mode. In this example, I'm running it as a network server so that it can be accessed by more than one application running in different Java virtual machines.

Make sure that your JAVA_HOME environment variable is set. If you used the Adafruit tutorial above and have the JDK installed in /opt/jdk1.8.0, add the following line to your .bashrc file. If you have your JDK installed in a different location, adjust as needed.

```
export JAVA_HOME=/opt/jdk1.8.0
```

Then add the following lines to set DERBY_HOME and adjust your path to include the JavaDB executables.

```
export DERBY_HOME=$JAVA_HOME/db
export PATH=$PATH:$JAVA_HOME/db/bin
```

Run `source ~/.bashrc` to load the settings from your edited .bashrc file.

Edit your java.policy file to allow access to port 1527. If you have the JDK installed in /opt/jdk1.8.0, your policy file should be /opt/jdk1.8.0/jre/lib/security/java.policy. Add the following line to this file before the closing bracket of the grant block -

```
permission java.net.SocketPermission "localhost:1527", "listen";
```

Now start the JavaDB server with the following command:

```
/opt/jdk1.8.0/db/bin/startNetworkServer &
```

It may take a moment to start, but the output should indicate that the security policy has been applied and that the server is now running on port 1527.

For this example, I used the ij client utility to connect to JavaDB, create the gpsdb database, and create the gps_readings table. I won't go into a lot of detail; you can find good documentation online. With the path to the JavaDB/Derby binaries included in your path, you can simply run the ij command to start the command-line client.

To create the database, I used the following connect statement at the ij> prompt:

```
connect 'jdbc:derby://localhost:1527/gpsdb;create=true';
```

I ran the following SQL at the ij> prompt to create the gps_readings table:

```
create table gps_readings (
  utc_time_date timestamp not null,
  lat varchar(15) not null,
  long varchar(15) not null,
  constraint pk_gps_readings_utc_time_date primary key(utc_time_date)
);
```

Connecting the Copernicus II GPS Module

GPS Module	Raspberry Pi (Rev. B)
VCC	3.3V
GND	GND
TX-B	GPIO15
RX-B	GPIO14

Java 8 Code

Here is the source code from Gps.java -

```
import gnu.io.*;
import java.io.*;
import java.util.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.time.LocalDate;
import java.time.ZoneId;
```

```
public class Gps {
    private static String port = "/dev/ttyS80";
    private InputStream inStream;
    private OutputStream outStream;
    // NMEA command to set Copernicus II to output $GPGLL every second.
    private static String nmeaString = "$PTNLSNM,0002,01*55\r\n";

    private Connection connect = null;
    private Statement statement = null;
    private static String driverClass = "org.apache.derby.jdbc.ClientDriver";
    private String jdbcURL = "";
    private static String sql = "INSERT INTO GPS_READINGS(UTC_TIME_DATE, LAT, LONG) VALUES(?,?,?)";

    // Constructor takes JDBC URL for JavaDB server as argument
    public Gps(String url) { jdbcURL = url; }

    public void recordGPS() {
        try {
            // RXTXComm library uses /dev/ttyS80, so symbolic link needed
            String lnPortCmd = "ln -s /dev/ttyAMA0 /dev/ttyS80";
            Process p = Runtime.getRuntime().exec(lnPortCmd);
            p.waitFor();
            CommPortIdentifier portId = CommPortIdentifier.getPortIdentifier(port);
            SerialPort serialPort = (SerialPort) portId.open("GPS", 5000);
            // Change serial port speed as needed
            serialPort.setSerialPortParams(19200, SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
            serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
            inStream = serialPort.getInputStream();
            outStream = serialPort.getOutputStream();
            byte[] nmeaCmd = nmeaString.getBytes();
            String gpsData = "";
            outStream.write(nmeaCmd, 0, nmeaCmd.length);
            Class.forName(driverClass).newInstance();
            connect = DriverManager.getConnection(jdbcURL);
            PreparedStatement statement = connect.prepareStatement(sql);
            while(true) {
                if(inStream.available() > 0) {
                    int b = inStream.read();
                    if(b != 13) {
                        gpsData += (char)b;
                    }
                    else {
                        System.out.println(gpsData);
                        gpsData = gpsData.trim();
                        String[] datum = gpsData.split(",");
                        gpsData = "";
                        // Check for valid $GPGLL NMEA sentence
                        if(datum.length < 8 || !("$GPGLL").equals(datum[0]) || datum[1] == null ||
                            !("A").equals(datum[6])) {
                            continue;
                        }
                        else {
                            LocalDate todayUTC = LocalDate.now(ZoneId.of("UTC"));
                            String t = datum[5].substring(0,2) + '.';
                            t += datum[5].substring(2,4) + '.';
                            t += datum[5].substring(4,6);
                            statement.setString(1, todayUTC.toString() + '-' + t);
                            statement.setString(2, datum[1] + ' ' + datum[2]);
                            statement.setString(3, datum[3] + ' ' + datum[4]);
                            statement.executeUpdate();
                        }
                    }
                }
            }
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
        finally {
            try {
                statement.close();
                connect.close();
            }
            catch (Exception exc) {
                exc.printStackTrace();
            }
        }
    }
}
```

```

    }
}

public static void main(String[] args) {
    Gps copernicus = new Gps("jdbc:derby://localhost:1527/gpsdb");
    copernicus.recordGPS();
}
}

```

Compiling & Running the Program

I have tested compiling and running the program as root (superuser).

Use the following command to compile the program -


```
javac -cp /usr/share/java/RXTXcomm.jar:$JAVA_HOME/db/lib/derby.jar Gps.java
```

After compiling, use the following command to run the program. Note that when running the code you need to use derbyclient.jar.

```
java -Djava.library.path=/usr/lib/jni/ -cp /usr/share/java/RXTXcomm.jar:$JAVA_HOME/db/lib/derbyclient.jar:. Gps
```

When the program runs, the GPS data read from the Copernicus II is printed out in the terminal window and records are inserted into the `gps_readings` table. to query the data in the database. Since JavaDB is running in network server mode, you can access it using `ij` while the Java Gps program is running.

Posted by Brad Berkland at 9:54 PM No comments:

 Recommend this on Google

Labels: GPS, Java, JavaDB, Serial

Monday, June 9, 2014

A Python Program for the Raspberry Pi to Read TSIP GPS Data from a Copernicus II via USB

The Copernicus II GPS module can output data in binary TSIP (Trimble Standard Interface Protocol) format in addition to text-based NMEA format. This exar Python program that reads and parses TSIP data to obtain current position data.

The Copernicus II has two serial interfaces with two sets of RX/TX pins. You can connect a set to the GPIO 14/GPIO 15 pins on the Raspberry Pi or you can 3.3V FTDI adapter to connect a pair of RX/TX pins to the USB port on the RPi. The GPIO pins connect to `/dev/ttyAMA0`. The USB bus connects to `/dev/ttyU` `/dev/ttyAMA0`, don't forget that you may need to do some reconfiguration to free this serial port. I have found that it is possible to have a program that accesss interfaces at the same time, but I don't have a use case for this scenario. The example below uses an FTDI adapter to connect the Copernicus II to the USB. The Raspbian distribution should already have the FTDI-SIO driver needed to use `/dev/ttyUSB0`.

Connections

Copernicus II	FTDI Adapter (3.3V)
GND	GND
TX-A	RXI
RX-A	TXO

Connect the VCC and GND on the Copernicus II to the 3.3 volt power and ground on the Raspberry Pi.

Python Code

The following code reads the current latitude, longitude, and altitude and prints them to the terminal. In my experience, the altitude isn't very accurate.

```

import serial
import struct
import math

ser = serial.Serial("/dev/ttyUSB0", baudrate=38400)
tsip = []
last = ''
start = 0
dle_cnt = 0
id = 0

DLE = '\x10'
ETX = '\x03'

while True:
    data = ser.read()
    # Test for data frame start marker DLE (0x10)
    if start == 0 and data == DLE:

```



```

start = 1
# To avoid confusion, when the frame marker DLE (0x10) occurs in the sequence of data
# bytes in the data frame, it is doubled. We need to drop the extra 0x10 by skipping
# the rest of the loop for the current iteration. Count DLEs to track whether
# it is even or odd. The count DLE that marks the end of the frame - before ETX (0x03)
# - is always odd. See p. 122 of the Copernicus II manual.
elif start == 1 and data == DLE:
    dle_cnt = dle_cnt + 1
    if last == DLE:
        continue
# If the last byte was DLE (0x10) and the current byte is not ETX (0x03),
# then the current byte is the packet ID
elif start == 1 and data != ETX and last == DLE:
    id = data
# If the current byte is 0x03 (ETX), has come right after DLE (0x10), and the
# DLE count is odd, we have reached the end of the data frame.
elif start == 1 and data == ETX and last == DLE and dle_cnt % 2 == 1:
    dle_cnt = 0
    last = ''
    tsip.append( data )
# Packet 0x84 has the position data we need.
# See p. 163 of the Copernicus II manual for structure of this packet
if id == '\x84':
    # Join bytes into string without added spaces and unpack as big-endian
    # double. struct.unpack returns a tuple. Our value is in the first
    # element.
    lat_rad = struct.unpack('>d', "".join(tsip[2:10]))[0]
    lat_deg = lat_rad * 180.0 / math.pi
    lat_dir = 'N' if lat_deg > 0 else 'S'
    long_rad = struct.unpack('>d', "".join(tsip[10:18]))[0]
    long_deg = long_rad * 180.0 / math.pi
    long_dir = 'E' if long_deg > 0 else 'W'
    alt_m = struct.unpack('>d', "".join(tsip[18:26]))[0]
    print "%02.6f %s %03.6f %s %4.1fm\n" % (abs(lat_deg), lat_dir,
        abs(long_deg), long_dir, alt_m)
    id = 0
    tsip = []
    start = 0
    continue
last = data
tsip.append(data)

```

Note about the output: The values for latitude and longitude are in degrees. The value to the left of the decimal point represents whole degrees and the value the decimal point represents a fraction of a degree. This is different from the values in NMEA sentences, where the value to the left of the decimal point represents degree and whole minutes, and the value to the right of the decimal represents a fraction of a minute.

Posted by Brad Berkland at 3:25 PM No comments:



Recommend this on Google

Labels: GPS, Python, Serial

Thursday, June 5, 2014

C Program to Read Multiple DS18B20 1-Wire Temperature Sensors & Save Data to a Sqlite Database on a Raspberry Pi

The C program below reads the temperature from multiple DS18B20 1-Wire devices and saves the data to a Sqlite3 database. This example has a couple of features: It uses a linked list rather than arrays to keep track of the attached sensors and it uses signal() with an event handler to detect when the user presses the program. The handler allows for the current round of readings to complete and closes the database safely.

To use 1-Wire devices with the Raspberry Pi, you will need to load a couple kernel modules by issuing the following commands before running the program:

```

modprobe wl-gpio
modprobe wl-therm

```

Connections

The DS18B20s are hooked up in non-parasitic power mode. Looking at the flat side of the head of the sensor, the left pin is connected to ground, the right pin to 3V3, and the center pin is attached to the center pin of the next sensor. There is a 4.7k Ohm pull-up resistor on the connection to the Raspberry Pi's (rev. E

Sqlite3 Database Table

The code below assumes a Sqlite3 database in a file called ds18b20_temp_data.db in the same directory as the program. Here is the schema for the table:

```

CREATE TABLE ds18b20_temp_data (

```

```

    date_time    date        not null,
    dev_id       varchar(16) not null,
    temp_c       number(7,3) not null,
    constraint pk_ds18b20_temp_data primary key(date_time, dev_id)
);

```

Code

```

#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sqlite3.h>
#include <signal.h>

// Flag used by handler - changed to 0 when user presses Ctrl-C
// Loop that reads & records temperatures keeps running when
// keepRunning = 1
int8_t volatile keepRunning = 1;

// Pointer to Sqlite3 DB - used to access DB when open
sqlite3 *db = NULL;
// Path to DB file - same dir as this program's executable
char *dbPath = "ds18b20_temp_data.db";
// DB Statement handle - used to run SQL statements
sqlite3_stmt *stmt = NULL;

// struct to hold ds18b20 data for linked list
// 1-Wire driver stores info in file for device as text
struct ds18b20 {
    char devPath[128];
    char devID[16];
    char tempData[6];
    struct ds18b20 *next;
};

// Find connected 1-wire devices. 1-wire driver creates entries for each device
// in /sys/bus/w1/devices on the Raspberry Pi. Create linked list.
int8_t findDevices(struct ds18b20 *d) {
    DIR *dir;
    struct dirent *dirent;
    struct ds18b20 *newDev;
    char path[] = "/sys/bus/w1/devices";
    int8_t i = 0;
    dir = opendir(path);
    if (dir != NULL)
    {
        while ((dirent = readdir(dir))) {
            // 1-wire devices are links beginning with 28-
            if (dirent->d_type == DT_LNK &&
                strstr(dirent->d_name, "28-") != NULL) {
                newDev = malloc( sizeof(struct ds18b20) );
                strcpy(newDev->devID, dirent->d_name);
                // Assemble path to OneWire device
                sprintf(newDev->devPath, "%s/%s/w1_slave", path, newDev->devID);
                i++;
            }
            newDev->next = 0;
            d->next = newDev;
            d = d->next;
        }
    }
    (void) closedir(dir);
    }
    else {
        perror ("Couldn't open the w1 devices directory");
        return 1;
    }
    return i;
}

// Write data to DB (DB already opened in main())
int8_t recordTemp(char *devID, double tempC) {
    char *sql = "INSERT INTO ds18b20_temp_data(date_time, dev_id, temp_c) VALUES(datetime('now'), ?, ?)";
    sqlite3_prepare_v2(db, sql, strlen(sql), &stmt, NULL);
    sqlite3_bind_text(stmt, 1, devID, strlen(devID), 0);

```

```

        sqlite3_bind_double(stmt, 2, tempC);
        sqlite3_step(stmt); // Run SQL INSERT
        sqlite3_reset(stmt); // Clear statement handle for next use
    return 0;
}

// Cycle through linked list of devices & take readings.
// Print out results & store readings in DB.
int8_t readTemp(struct ds18b20 *d) {
    while(d->next != NULL){
        d = d->next;
        int fd = open(d->devPath, O_RDONLY);
        if(fd == -1) {
            perror ("Couldn't open the wl device.");
            return 1;
        }
        // 1-wire driver stores data in file as long block of text
        // Store file contents in buf & look for t= that marks start of temp.
        char buf[256];
        ssize_t numRead;
        while((numRead = read(fd, buf, 256)) > 0) {
            strncpy(d->tempData, strstr(buf, "t=") + 2, 5);
            double tempC = strtod(d->tempData, NULL);
            // Driver stores temperature in units of .001 degree C
            tempC /= 1000;
            printf("Device: %s - ", d->devID);
            printf("Temp: %.3f C ", tempC);
            printf("%.3f F\n\n", tempC * 9 / 5 + 32);
            recordTemp(d->devID, tempC);
        }
        close(fd);
    }
    return 0;
}

// Called when user presses Ctrl-C
void intHandler() {
    printf("\nStopping...\n");
    keepRunning = 0;
}

int main (void) {
    // Intercept Ctrl-C (SIGINT) in order to finish writing data & close DB
    signal(SIGINT, intHandler);
    struct ds18b20 *rootNode;
    struct ds18b20 *devNode;
    int rc = sqlite3_open(dbPath, &db);
    // If rc is not 0, there was an error
    if(rc){
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        exit(0);
    }
    // Handler sets keepRunning to 0 when user presses Ctrl-C
    // When Ctrl-C is pressed, complete current cycle of readings,
    // close DB, & exit.
    while(keepRunning) {
        rootNode = malloc( sizeof(struct ds18b20) );
        devNode = rootNode;
        int8_t devCnt = findDevices(devNode);
        printf("\nFound %d devices\n\n", devCnt);
        readTemp(rootNode);
        // Free linked list memory
        while(rootNode) {
            // Start with current value of root node
            devNode = rootNode;
            // Save address of next devNode to rootNode before
            // deleting current devNode
            rootNode = devNode->next;
            // Free current devNode.
            free(devNode);
        }
        // Now free rootNode
        free(rootNode);
    }
    sqlite3_close(db);
    return 0;
}

```

```
}
```

Compiling the Code

Assuming that the code above is saved in a file name w1mdb.c, compile it using the following command:

```
gcc -Wall -lsqlite3 -o w1mdb w1mdb.c
```

Remember to include the -l option so that the linker includes the Sqlite3 library.

Posted by Brad Berkland at 12:08 PM 2 comments:



Recommend this on Google

Labels: 1-Wire, C, Sqlite

[Home](#)

Subscribe to: Posts (Atom)

Simple template. Powered by Blogger.