Simple User Space File I/O for Sensors

Abstract

Having a filesystem entity for sensor data fits with the Unix philosopy of “everythng is a file”. This document intends to describe a system of doing this that is fairly portable across Unix-like systems.

Sensors

There are far too many sensors to try to describe here, so, except where I need to be specific, I will try to speak generally. Sensors are hardware and can be interfaced in many ways. The goal of this framework will be to describe a general overview.

Targeting the Application

Since this lays out a simple framework, it does not get into the details of the underlying hardware. Formatting the data is the responsibility of the sensor collector and the application which is using this data.

Sensor Filesystem

Use of small ram disks provide for non-persistent data that does not stress the devices storage.

On Linux, I have a ramdisk on /sensor:

```
tmpfs /sensor   tmpfs
nosuid,noexec,nodev,noatime,uid=1000,gid=1000,size=5M 0 0
```

This creates a 5M ramdisk with my uid/gid as owner.

Limitations

Often sensors are used for controlling other devices and need a fine degree of timing. In cases like this it’s likely better to work with the physical device.

This framework is more about data presentation than acquisition.

Collectors

Collectors are programs which collect sensor data from hardware and write to the sensor filesystem.

Collectors can also read files on that filesystem, use of the modification time can represent new data for example. That data can be used to control a device.

```
#Example collector:
    while True:
        s = sensi7020-iio.sensor('si7021')
        sensorfs.dataToFs(s.read())
        time.sleep(1)
```

Collectors are not limited to hardware Collectors can be subscribers to MQTT brokers and write remote sensor data to the sensor filesystem.

```
# MQTT Data to sensorfs
while True:
    getMqttMessage('pi3/tempdata')
    sensorfs.dataToFs(data)
    time.sleep(1)
```

Collectors can perform write operations. An exampe would be for a collector to make a fifo and wait for input. This input could be passed along to the hardware. This could be extended to work using MQTT (or other machine to machine mechanism)

Sensor FS Organization

To accomodate sharing sensor data with other hosts, each host, including the local host, a path on the root. Sensor data will be written to a class directory which contains the items for that sensor. The format is > /sensor/host/class/item(s)

Use Case

Consider a collector receives a message with the following json data

```
{"temp": 74.18, "tempc": 23.43, "humidity": 65.07, "time": 1693112929}
```

The collector I am running collects data in JSON format. The data then gets written to:

```
/sensor/system/tempdata/module
/sensor/system/tempdata/temp
/sensor/system/tempdata/tempc
/sensor/system/tempdata/humidity
/sensor/system/tempdata/time
/sensor/system//tempdata/tempdata.json
```

The temp,tempc,humidity files contain the values, time comes from the sender or sensor at the time the reading was made, tempdata.json contains the json object the other files are derived from.

Running Example

Currently, I have two Raspberry Pi devices. One, a Pi 3B+ and the hub, a Pi 4B+. Each have a Silicon Labs Si7021 sensor on I2C bus 1.

Each of these devices publish messges with their data to a MQTT broker.

On the Pi 4, the data local sensor data is written to /sensor/pi4/tempdata while a collector that listens to the MQTT messages write the remote data from pi3 and stores it in /sensor/pi3/tempdata. Since each device uses the same interface to collect the actual sensor data their data folers all contain the same elements, albeit, with different data.

My /sensor filesystem looks like:

```
/sensor/
├── pi3
│   └── tempdata
│       ├── humidity
│       ├── modinfo
│       ├── temp
│       ├── tempc
│       ├── tempdata.json
│       └── time
└── pi4
    └── tempdata
        ├── humidity
        ├── modinfo
        ├── temp
        ├── tempc
        ├── tempdata.json
        └── time
```

Sample Files

File	Use
collector.py	Example collector. I use this to collect si7021 data and export to sensorfs
sensor.py	Base class for a sensor for a consistent interface
sensi7021.py	Derived class from Sensor for si7021
sensorfs.py	Sensor filesytem utility functions (just one for now)
fstab.snippet	Example fstab line for a ramdisk