Part 19
-
SNMP Monitoring

Version: 2020-02-20

Monitoring the Raspberry Pi with SNMP

As most of my Raspberry Pi cards are used headless - i.e. without a keyboard or monitor - I wanted to monitor what was happening remotely. In addition to monitoring the NTP Server operation, I've added general SNMP support (Simple Network Management Protocol) which allows monitoring of the network I/O, disk space and CPU temperature

1. Adding SNMP support for general remote monitoring

First make sure your Pi is up to date

```
sudo apt-get –y update
```

Next install the required modules for the SNMP server and client software

```
sudo apt -y install snmpd snmp
sudo apt –y install libsnmp-dev snmp-mibs-downloader
```

You need to edit the configuration if you want to monitor your Raspberry Pi from across the network rather than just locally. The configuration below allows anyone to read the SNMP data using the communitystring `public` - you may want a more secure configuration by restricting agentAddress to a specific IP address and by changing the community string `public` to a more secure name

```
sudo nano /etc/snmp/snmpd.conf
```

Change the agentAddress from:

```
# Listen for connections from the local system only
agentAddress udp:127.0.0.1:161
# Listen for connections on all interfaces (both IPv4 *and* IPv6)
#agentAddress udp:161,udp6:[::1]:161
```

to:

```
# Listen for connections from the local system only
#agentAddress udp:127.0.0.1:161
# Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:161,udp6:[::1]:161
```

Change the line

```
#rocommunity public localhost
```

to

```
rocommunity public
```

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file with the same name.

Restart the SNMP daemon

```
sudo /etc/init.d/snmpd restart
```

To fully test your SNMP install, run

```
snmpget -Os -c public -v 1 localhost SNMPv2-MIB::sysDescr.0
```

This will show you the description of your Raspberry Pi as mentioned in SNMP data. Eg.

```
sysDescr.0 = STRING: Linux MeRasPi4B-Test 4.19.97-v7l+ #1294 SMP Thu
Jan 30 13:21:14 GMT 2020 armv7l
```

The same can be optained with this

```
snmpget -Os -c public -v 1 localhost 1.3.6.1.2.1.1.1.0

iso.3.6.1.2.1.1.1.0 = STRING: "Linux MeRasPi4B-Test 4.19.97-v7l+ #1294
SMP Thu Jan 30 13:21:14 GMT 2020 armv7l"
```

The difference is that you get some numbers instead of text in the output
Listing several entries at the same time can be done using snmpwalk

```
snmpwalk -Os -c public -v 1 localhost UCD-SNMP-MIB::dskTable
```

or

```
snmpwalk -Os -c public -v 1 localhost 1.3.6.1.4.1.2021.9.1
```

This will show you the disk table as installed on your Raspberry Pi and

```
snmpwalk -Os -c public -v 1 localhost IF-MIB::ifTable
```

will show you the interface table as installed on your Raspberry Pi while

```
snmpwalk -Os -c public -v 1 localhost UCD-SNMP-MIB::laTable
```

This will show you the average Load table.

Wondering where these numbers come from? This document will not go into the details of it. Google for ‘SNMP OID explained’ and read all about it. :-)

2. Adding SNMP support for CPU temperature monitoring

To monitor the CPU temperature remotely, SNMP is the obvious medium for me. By chance I cam across a command to get the CPU temperature of the CPU on the Raspberry Pi in either milli-degrees:

```
cat /sys/class/thermal/thermal_zone0/temp
```

or in human readable format:

```
/opt/vc/bin/vcgencmd measure_temp
```

To get this information into SNMP we can create an SNMP "pass" extension, whereby SNMP will call a specific program or shell script to get a particular object ID (OID) value. The script needs to write to standard output three lines, the OID, the data type (integer/string etc.), and the actual data. To control the execution of the program or script, the snmpd program will include a parameter when the script is called, to indicate whether the request is a "get" command (-g), a "get-next" command (-n), or a "set" command (-s). It is important that the program or script not respond to a "get-next" command if it is returning only one variable and, of course, the set command would do nothing when measuring a temperature. Use the nano editor to create a script named snmp-cpu-temp (or whatever) in your default directory:

```
sudo nano snmp-cpu-temp
```

The script should contain

```
#!/bin/bash
if [ "$1" = "-g" ]; then
  echo 1.3.6.1.2.1.25.1.8
  echo gauge
  cat /sys/class/thermal/thermal_zone0/temp
fi
exit 0
```

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file.

I selected the OID as the next unused in a sequence. You now need to mark the script executable and run it as follows

```
sudo chmod +x snmp-cpu-temp
sudo ./snmp-cpu-temp -g
```

and you should get a response something like (for a CPU temperature of 46°C):

```
1.3.6.1.2.1.25.1.8
gauge
46002
```

Having got that part working, you need to copy the script to a location where the SNMP software can find it. I used /usr/local/bin but I don't know Linux well enough to know whether that's the most appropriate directory.

```
sudo cp snmp-cpu-temp /usr/local/bin
```

Now we need to edit the SNMP daemon configuration to tell it how to handle this OID. In the snmpd.conf file, you will find a couple of examples of the "pass" command. Edit the file:

```
sudo nano /etc/snmp/snmpd.conf
```

and just below the sample ""Pass-through" MIB extension command" lines, insert a new active "pass" command

```
pass 1.3.6.1.2.1.25.1.8 /bin/sh /usr/local/bin/snmp-cpu-temp
```

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file.

Restart snmpd

```
sudo /etc/init.d/snmpd restart
```

and you can then test that it's all working with the command

```
/usr/bin/snmpget -v 2c localhost -c public 1.3.6.1.2.1.25.1.8
```

I got different responses on the two Raspberry Pi cards where I tested, but both contained the line

```
iso.3.6.1.2.1.25.1.8 = Gauge32: 46540
```

Extra's

I refer to a similar document about troubleshooting your Pi for more in depth info on the following.

To monitor power issues you can add in the same way as above the following

```
sudo nano snmp-cpu-power

#!/bin/bash
if [ "$1" = "-g" ]; then
  echo 1.3.6.1.2.1.25.1.9
  echo gauge
  val=$( vcgencmd get_throttled | sed  "s/throttled=0x//" | sed "s/000//" )
  echo $val
fi
exit 0

sudo chmod +x snmp-cpu-power
sudo ./snmp-cpu-power -g

1.3.6.1.2.1.25.1.9
gauge
0

sudo nano /etc/snmp/snmpd.conf

pass 1.3.6.1.2.1.25.1.9 /bin/sh /usr/local/bin/snmp-cpu-power

sudo /etc/init.d/snmpd restart

/usr/bin/snmpget -v 2c localhost -c public 1.3.6.1.2.1.25.1.9

iso.3.6.1.2.1.25.1.9 = Gauge32: 0
```

To monitor zombie tasks you can add in the same way as above the following

```
sudo nano snmp-zombie

#!/bin/bash
if [ "$1" = "-g" ]; then
  echo 1.3.6.1.2.1.25.1.10
  echo gauge
  ps axo stat,ppid,pid,comm | grep -w defunct  | wc -l
fi
exit 0

sudo chmod +x snmp-zombie
sudo ./snmp-zombie -g

1.3.6.1.2.1.25.1.10
gauge
0

sudo nano /etc/snmp/snmpd.conf

pass 1.3.6.1.2.1.25.1.10 /bin/sh /usr/local/bin/snmp-zombie

sudo /etc/init.d/snmpd restart

/usr/bin/snmpget -v 2c localhost -c public 1.3.6.1.2.1.25.1.10

iso.3.6.1.2.1.25.1.10 = Gauge32: 0
```

3. Monitoring Ambient Temperature using 1-wire protocol

I extended the SNMP pass function to allow monitoring of ambient temperature using the DS18B20 "single-wire" device. This is well written up for use with the Raspberry Pi. Multiple devices can all be connected, using just a ground, 3.3V and signal wire connection, with the signal wire going to GPIO 4 (pin 7).

The 1-Wire drivers are not loaded by default when the Raspberry Pi boots. You can load them with the following commands from a command prompt

```
sudo modprobe wire
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

But it's easier to add them into `/etc/modules`.

```
sudo nano /etc/modules
```

Add the lines

```
wire
w1-gpio
w1-therm
```

to the end of the file, taking care not to delete any lines already there.

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file.

A reboot will be required.

```
sudo reboot
```

Connect the sensor hardware to the Raspberry Pi and check it is detected by seeing if a device is listed on the 1-Wire bus

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slave_count
1
```

This will print the number of sensors detected, 1 - for my hardware at the moment. You can get the sensor ID (a hexadecimal string stored in ROM on the sensor chip) by reading the w1_master_slaves file

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slaves
10-00080234149b
```

Note: your device may start with a 28, not a 10. In fact, it seemed to vary between a 28 and a 10. A quick check for correct operation of the sensor is to "read" the sensor file, you'll need the hex ID of the sensor from earlier commands:

```
cat /sys/bus/w1/devices/10-00080234149b/w1_slave
37 00 4b 46 ff ff 07 10 1e : crc=1e YES
37 00 4b 46 ff ff 07 10 1e t=27312
```

The number after 't=' is the temperature in thousandths of a degrees Celsius.

Note: for later versions of Raspbian when setting up more 1 wire sensors on a raspberry pi B (one of the early ones), running Raspbian wheezy 2015-02-16, I founded that I need to add `dtoverlay=w1-gpio` to the bottom of `/boot/config.txt`, before the devices were recognised. I am only mentioning it in case you get queries about:

```
cat /sys/bus/w1/devices/w1_bus_master1/w1_master_slave_count
```

reporting that there is no such file or directory.

4. Monitoring Sea-Level Air pressure using I2C

There are a couple of very low cost devices for monitoring ambient air pressure, and it's very easy to add one of these devices to a Raspberry Pi as it already supports the I2C two-wire communication protocol used by the devices, and software support is readily available from Adafruit (to whom, many thanks). The devices are the older BMP085 and the more recent BMP180, which are pin- and software-compatible. You will need to connect the board-mounted device to the Raspberry Pi as described in the Adafruit notes. I assume that you followed their tutorial and have got as far as running the example program. Do note that if you are using a recent version of Raspbian the instructions for adding modules may have changed.

To use the sensor with SNMP and hence MRTG, you need to create a program which will respond with the pressure, create a shell script which SNMP will call to get the value of the pressure, and add a line to snmpd.conf telling SNMP where to find the script.

Simple program to read the pressure `pressure.py` - you can put this in your home directory: `/home/pi/`. Note that the 2175 is the adjustment required to convert readings at my height (183 m) to sea-level pressure. Yes, you can use a more accurate formula if you wish. Do *not* rely on the value from the Adafruit example program where the height appears to be derived from assuming the sea-level pressure to be 1015 hPa, if my reading of their note is correct.

```
#!/usr/bin/python
import Adafruit_BMP.BMP085 as BMP085
sensor = BMP085.BMP085(mode=BMP085.BMP085_ULTRAHIGHRES)
pressure = sensor.read_pressure() + 2175
print '{0:0.0f}'.format(pressure)
```

Simple shell script which can be called by SNMP pass command: snmp-air-pressure

```
#!/bin/bash
if [ "$1" = "-g" ]
then
  echo .1.3.6.1.2.1.25.1.20
  echo gauge
  python /home/pi/pressure.py
fi
exit 0
```

Create the script in your home directory, make it executable:

```
chmod a+x snmp-air-pressure
```

and copy it to a directory on the path such as:

```
sudo cp snmp-air-pressure /usr/local/bin
```

Now edit the snmpd configuration file, to add the pass command:

```
sudo nano /etc/snmp/snmpd.conf
```

add

```
pass .1.3.6.1.2.1.25.1.20 /bin/sh /usr/local/bin/snmp-air-pressure
```

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file.

A problem ...
As you can see from the previous examples, I expect to run a command like

```
/usr/local/bin/snmp-air-pressure -g
```

and get a result, but all my attempts gave a result of 0.

```
.1.3.6.1.2.1.25.1.20
gauge
0
```

... and the solution
Running the command as root (i.e. with sudo) it worked as expected, which suggested a rights issue. It emerged that there is a Linux group called "i2c" and if the current user has access to that group they can run the I2C software without requiring the sudo prefix. So I added my user name "pi" to that group

```
sudo adduser pi i2c
```

You can check a group's membership with the "members" command, but you may need to apt-get install that command first. Further checks shows that when snmpd ran a "pass" command it did so with the user name "snmp", so you would think that all that was required would be

```
sudo adduser snmp i2c
```

so that the user had access to the i2c group. However, it turns out that when snmpd is started it does not claim all the groups to which it should have access when it starts, and that one line needs to be added to /etc/snmp/snmpd.conf to set the group it uses by default. After the line

```
agentAddress 161
```

add

```
agentgroup i2c
```

When you have finished press [Ctrl] + X. This will ask if you want to save the modified files. Press 'Y' and then hit [Return] to save the file.

Once this is done, the pass script above works correctly when called by snmpd on receipt of an snmp get request.

5. SNMP and Python

Make sure you have the right and latest version on your Pi

```
apt -y install gcc python-dev
pip3 install easysnmp
```

Now create this sample Python script `snmptest.py`

```
nano snmptest.py
```

and add these lines

```
from easysnmp import Session

# Create an SNMP session to be used for all our requests
session = Session(hostname='localhost', community='public', version=2)

# You may retrieve an individual OID using an SNMP GET
location = session.get('1.3.6.1.2.1.1.6.0')
print(location.value)

# You may also specify the OID as a tuple (name, index)
# Note: the index is specified as a string as it can be of other types
than
# just a regular integer
contact = session.get(('1.3.6.1.2.1.1.4', '0'))
print(contact.value)

# And of course, you may use the numeric OID too
description = session.get('.1.3.6.1.2.1.1.1.0')
print(description.value)

# Set a variable using an SNMP SET
#session.set('1.3.6.1.2.1.1.6.0', 'The SNMP Lab')
#location = session.get('1.3.6.1.2.1.1.6.0')
#print(location)

# Perform an SNMP walk
system_items = session.walk('1.3.6.1.2.1.1')
# Each returned item can be used normally as its related type (str or int)
# but also has several extended attributes with SNMP-specific information
for item in system_items:
    print('{oid}.{oid_index} {snmp_type} = {value}'.format(oid=item.oid,
oid_index=item.oid_index, snmp_type=item.snmp_type, value=item.value))
```

Save the script and run it

```
python3 snmptest.py
```

You should get an output similar to this one

```
Sitting on the Dock of the Bay
Me <me@example.org>
Linux MeRasPi3B-Test 4.19.97-v7+ #1294 SMP Thu Jan 30 13:15:58 GMT 2020 armv7l
iso.3.6.1.2.1.1.1.0. OCTETSTR = Linux MeRasPi3B-Test 4.19.97-v7+ #1294 SMP Thu Jan 30
13:15:58 GMT 2020 armv7l
iso.3.6.1.2.1.1.2.0. OBJECTID = .1.3.6.1.4.1.8072.3.2.10
iso.3.6.1.2.1.1.3.0. TICKS = 197104
iso.3.6.1.2.1.1.4.0. OCTETSTR = Me <me@example.org>
iso.3.6.1.2.1.1.5.0. OCTETSTR = MeRasPi3B-Test
iso.3.6.1.2.1.1.6.0. OCTETSTR = Sitting on the Dock of the Bay
iso.3.6.1.2.1.1.7.0. INTEGER = 72
iso.3.6.1.2.1.1.8.0. TICKS = 1
iso.3.6.1.2.1.1.9.1.2.1. OBJECTID = .1.3.6.1.6.3.11.3.1.1
```

```
iso.3.6.1.2.1.1.9.1.2.2. OBJECTID = .1.3.6.1.6.3.15.2.1.1
iso.3.6.1.2.1.1.9.1.2.3. OBJECTID = .1.3.6.1.6.3.10.3.1.1
iso.3.6.1.2.1.1.9.1.2.4. OBJECTID = .1.3.6.1.6.3.1
iso.3.6.1.2.1.1.9.1.2.5. OBJECTID = .1.3.6.1.6.3.16.2.2.1
iso.3.6.1.2.1.1.9.1.2.6. OBJECTID = .1.3.6.1.2.1.49
iso.3.6.1.2.1.1.9.1.2.7. OBJECTID = .1.3.6.1.2.1.4
iso.3.6.1.2.1.1.9.1.2.8. OBJECTID = .1.3.6.1.2.1.50
iso.3.6.1.2.1.1.9.1.2.9. OBJECTID = .1.3.6.1.6.3.13.3.1.3
iso.3.6.1.2.1.1.9.1.2.10. OBJECTID = .1.3.6.1.2.1.92
iso.3.6.1.2.1.1.9.1.3.1. OCTETSTR = The MIB for Message Processing and Dispatching.
iso.3.6.1.2.1.1.9.1.3.2. OCTETSTR = The management information definitions for the SNMP
User-based Security Model.
iso.3.6.1.2.1.1.9.1.3.3. OCTETSTR = The SNMP Management Architecture MIB.
iso.3.6.1.2.1.1.9.1.3.4. OCTETSTR = The MIB module for SNMPv2 entities
iso.3.6.1.2.1.1.9.1.3.5. OCTETSTR = View-based Access Control Model for SNMP.
iso.3.6.1.2.1.1.9.1.3.6. OCTETSTR = The MIB module for managing TCP implementations
iso.3.6.1.2.1.1.9.1.3.7. OCTETSTR = The MIB module for managing IP and ICMP implementations
iso.3.6.1.2.1.1.9.1.3.8. OCTETSTR = The MIB module for managing UDP implementations
iso.3.6.1.2.1.1.9.1.3.9. OCTETSTR = The MIB modules for managing SNMP Notification, plus
filtering.
iso.3.6.1.2.1.1.9.1.3.10. OCTETSTR = The MIB module for logging SNMP Notifications.
iso.3.6.1.2.1.1.9.1.4.1. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.2. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.3. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.4. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.5. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.6. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.7. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.8. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.9. TICKS = 1
iso.3.6.1.2.1.1.9.1.4.10. TICKS = 1
```