Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de agente SNMP para sensores de temperatura en Raspberry

Autora: Desirée García Soriano
Tutor: Antonio José Estepa Alonso

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de agente SNMP para sensores de temperatura en Raspberry

Autora:
Desirée García Soriano

Tutor:
Antonio José Estepa Alonso
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Grado: Desarrollo de agente SNMP para sensores de temperatura en Raspberry

Autora: Desirée García Soriano

Tutor: Antonio José Estepa Alonso

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mis abuelos

Agradecimientos

En primer lugar, agradezco a mis padres y a mi hermana por el apoyo que he recibido continuamente, animándome siempre en los momentos duros a continuar estudiando y no rendirme nunca.

También agradecer a Julio, un genio de la programación, quien me ha ayudado siempre que lo he necesitado.

Gracias a mis amigas y compañeras de estudio, con las que he pasado horas tanto dentro como fuera de la escuela.

Por último, agradecer a mi tutor Antonio por orientarme en este trabajo y a los demás profesores y maestros, de los que he aprendido en toda mi etapa educativa.

Desirée García Soriano

Sevilla, 2018

Resumen

Hoy en día, con el desarrollo de la tecnología y la creación del Internet de las Cosas (IoT), es muy común encontrarnos redes que obtienen información de sensores y que, por lo tanto, necesitan ser monitorizadas.

En este trabajo se describe el diseño y la implementación de un agente SNMP en una Raspberry Pi para la monitorización de sensores de temperatura y humedad conectados a la misma.

Este agente se ha implementado utilizando el software Net-SNMP, gracias a su herramienta de agente extendido mediante AgentX. Para ello, se han desarrollado los programas necesarios en lenguaje de programación C y se ha definido la correspondiente MIB (Management Information Base) con el modelo de datos implementado.

Abstract

Nowadays the development of technology and the creation of the Internet of Things (IoT) makes very common to find networks capable of obtain information from sensors that need to be monitored.

In this paper we describe the design and implementation of an SNMP agent in a Raspberry Pi for monitoring temperature and humidity sensors connected to it.

This agent has been implemented using the Net-SNMP software, thanks to its agent extended tool through AgentX. To do this, the necessary programs have been developed in C programming language and the corresponding MIB (Management Information Base) has been defined with the implemented data model.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En este trabajo se lleva a cabo la gestión de sensores de temperatura y humedad en un entorno doméstico. Actualmente, el software existente dedicado a la monitorización de redes suele estar pensado para gestionar grandes redes, no para un uso doméstico y particular.

Es por este motivo por el que surge la necesidad de la creación de un software que permita realizar las mismas funciones que estos programas, ofreciendo además la posibilidad al usuario de personalizarlo creando su propio catálogo de objetos a monitorizar.

Para ello, se utilizará una Raspberry Pi [1], ya que es un dispositivo de bajo coste con un buen rendimiento a bajo consumo, a la cual se le conectarán unos sensores que midan la temperatura y la humedad.

Para la monitorización se ha optado por utilizar un protocolo de gestión de red normalizado como es SNMP, ampliándolo con el protocolo de Agente Extendido (AgentX) [2], el cual facilitará la opción de añadir nuevos dispositivos y de dar soporte a cualquiera que se añada a la red.

1.1. Objetivos

El objetivo de este trabajo es realizar un agente de gestión remota para la lectura de sensores de temperatura y humedad conectados a una Raspberry Pi. El agente de gestión utilizará el protocolo SNMP para el envío y recepción de mensajes con un gestor a través del cual se podrá consultar la lectura del valor de un sensor, así como programar alertas ante ciertos valores límite. Se desarrollará el software del agente en el lenguaje de programación C y se definirá la correspondiente MIB (Management Information Base) con el modelo de datos implementado.

1.2. Estructura del Documento

El documento está estructurado en 7 bloques, organizados de la siguiente forma:

- Sección 1: Introducción. Se describe el contexto del trabajo, sus objetivos y la estructura del documento.
- Sección 2: Planificación Temporal. Se describe la metodología que se ha seguido para llevar a cabo el trabajo y el diagrama de Gantt correspondiente.
- Sección 3: Estado del Arte. Se analizan los diferentes programas que existen para crear un agente con soporte para MIB propia.
- Sección 4: Diseño del Modelo de Datos. Se detalla cómo se ha implementado la MIB de este trabajo.

- Sección 5: Implementación. Se explica cómo se ha llevado a cabo el desarrollo del agente SNMP.
- Sección 6: Pruebas. Se describen las pruebas que se han realizado para comprobar el buen funcionamiento del agente.
- Sección 7: Conclusiones y Líneas de Continuación. Se finaliza con las conclusiones obtenidas como resultado del trabajo y posibles mejoras de este.

2 PLANIFICACIÓN TEMPORAL

En este capítulo se identifican todas las fases necesarias para la realización del trabajo y se indica la duración de cada una de ellas. También se ha diseñado un diagrama de Gantt en el que se han señalado los hitos más importantes del trabajo.

2.1. Metodología del trabajo

Para el desarrollo del sistema de gestión necesitaremos diseñar una MIB, obtener las medidas de los sensores e implementar un agente SNMP en una Raspberry Pi.

Para ello, en la primera fase del trabajo, se investigará y se buscará un software para el agente SNMP que se adecúa a nuestras necesidades.

En la segunda fase del trabajo, se programarán los sensores para obtener las medidas de temperatura y humedad de estos.

En la tercera fase se realizará una lista con los objetos que se quieran gestionar y se diseñará una MIB a partir de ella.

En la cuarta fase se realizará el desarrollo del agente. Para ello, se utilizará el lenguaje de programación C y se hará uso de la programación multihilo para la realización de varias tareas de forma paralela, además de colas de mensajes para la comunicación de datos entre programas.

En la quinta fase se realizarán todas las posibles pruebas para comprobar el correcto funcionamiento del agente.

En la sexta fase se llevará a cabo un proceso de mejora tanto de la MIB como de la programación del agente.

En la última fase, se redactará la memoria del trabajo.

2.2. Hitos del trabajo

- Software del agente – 28/09/2017
- Programas de los sensores – 20/10/2017
- Primera versión de TEMPHUM-MIB – 27/10/2017
- Programa del agente – 12/12/2017
- Corrección de fallos – 19/12/2017
- Versiones mejoradas de TEMPHUM-MIB y del programa del agente – 20/01/2018
- Memoria del trabajo – 05/03/2018

2.3. Diagrama de Gantt

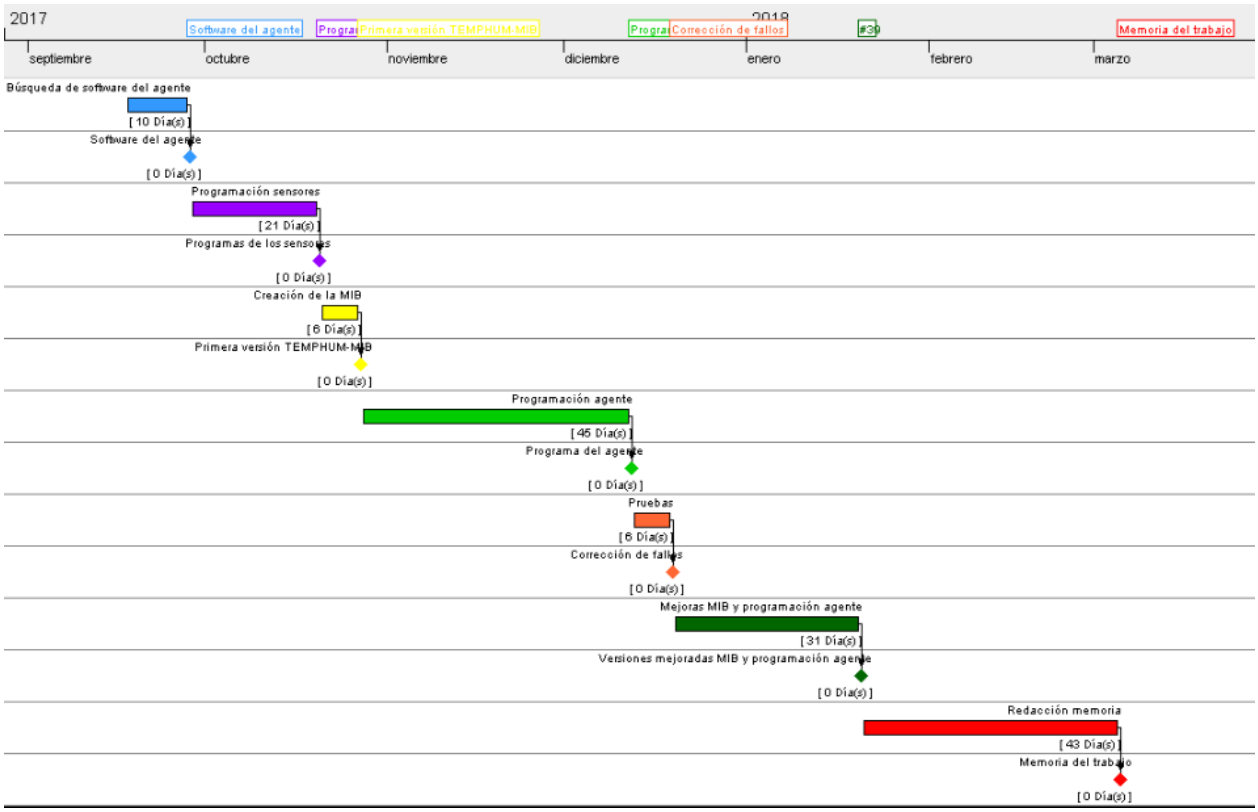


Figura 2-1. Diagrama de Gantt del trabajo.

3 ESTADO DEL ARTE

Tal y como se dijo anteriormente, el software existente hoy en día está dedicado, en la mayoría de los casos, a monitorizar y gestionar grandes redes, principalmente aplicaciones y servicios que están instalados en los dispositivos de la red. Esto conlleva a que un usuario particular que quiera monitorizar y configurar otros tipos de parámetros, como pueden ser la temperatura, la humedad, la cantidad de luz o el ruido en un entorno doméstico, no pueda llevarlo a cabo con alguno de estos programas.

En esta sección vamos a analizar las diferentes opciones que existen para realizar un agente SNMP que ofrezca soporte a MIBS propias y así poder crear un agente que se adapte a las necesidades del usuario.

3.1. Net-SNMP

Net-SNMP es una suite de aplicaciones de código abierto usadas para implementar SNMP (versiones 1, 2c, y 3) usando IPv4 e IPv6. Este software trae una API muy flexible y extensible. La API permite crear nuestros propios comandos, agregar extensiones al agente para dar soporte a MIBs propias y realizar un procesamiento especializado de notificaciones. La extensión del agente se puede realizar en varios lenguajes de programación como pueden ser Perl, Shell Script o C [3].

Para este trabajo se va a utilizar este software, ya que las herramientas que contiene nos proporcionan todas las funciones relativas de SNMP y facilita la implementación de los objetos de una MIB mediante la extensión del agente. El lenguaje de programación utilizado para ello será C.

3.2. PySNMP

PySNMP es una implementación de un motor SNMP puramente escrito en Python. Este motor puede funcionar en los roles de Agente/Manager/Proxy, con las versiones v1/v2c/v3 de SNMP sobre los protocolos IPv4/IPv6. Es una herramienta open-source, su código fuente está disponible en GitHub y ofrece soporte a MIBs propias [4].

3.3. SNMP4J

SNMP4J es una implementación de SNMP de código abierto para Java SE. Soporta las versiones v1, v2 y v3 de SNMP. Junto con el software AgenPro permite desarrollar un agente SNMP a partir de una MIB propia. Para crearlo, se debe ampliar la clase abstracta BaseAgent, la cual define un framework para escribir agentes SNMP usando la API SNMP4J-Agent [5].

4 DISEÑO DEL MODELO DE DATOS

En este capítulo se va a proceder a explicar cómo se ha llevado a cabo el diseño de la MIB (Management Information Base). Una MIB es un tipo de base de datos que contiene información plasmada de forma jerárquica, en forma de árbol, de todos los parámetros gestionables en un dispositivo. Está compuesta por una serie de objetos que supervisan y controlan un dispositivo en red. Cada objeto tiene un identificador único e incluye el tipo de objeto (entero, contador, cadena de caracteres), el nivel de acceso (lectura, escritura), restricciones de tamaño e información del rango del objeto.

Los objetos se representan en ASN.1 (Abstract Syntax Notation One), una sintaxis que se utiliza para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas [6].

4.1. Arquitectura del sistema de gestión

En una aplicación cliente/servidor distribuida de un sistema de gestión necesitamos un dispositivo que actúe como agente y otro que actúe como gestor.

El agente es el nodo instrumentalizado que se encarga de gestionar los datos de gestión. Recibe comandos que actúan sobre esos datos. Puede incluso generar notificaciones o alertas ante la ocurrencia de eventos para informar al gestor de posibles cambios de estado de estos datos.

El gestor es el nodo que envía comandos al agente y recibe sus respuestas. También recibe las notificaciones que genera el agente.

En nuestro sistema de gestión, el papel del agente lo tiene la Raspberry Pi y el del gestor, cualquier dispositivo que esté conectado a la misma red. Ambos deben compartir un catálogo de objetos, la MIB. El gestor tiene que saber a qué acceder y el agente debe mantener los valores de los objetos de forma estática o dinámica. El acceso a un objeto podrá ser de lectura, de escritura o de ambos, según el nivel de acceso correspondiente definido en la MIB.

En la figura 4-1 se describe la arquitectura de nuestro sistema de gestión:

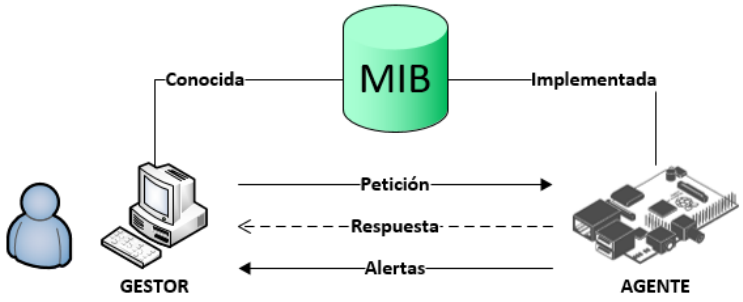


Figura 4-1. Escenario del sistema de gestión.

4.2. Objetos de la MIB

En nuestro sistema de gestión, vamos a llevar a cabo la lectura de sensores de temperatura y de humedad conectados a una Raspberry Pi. Para ello, mediante esquemas de los distintos componentes, se va a representar la información que necesitamos conocer de cada elemento de nuestro sistema. Esto nos ayudará posteriormente a definir los objetos de gestión de nuestra MIB.

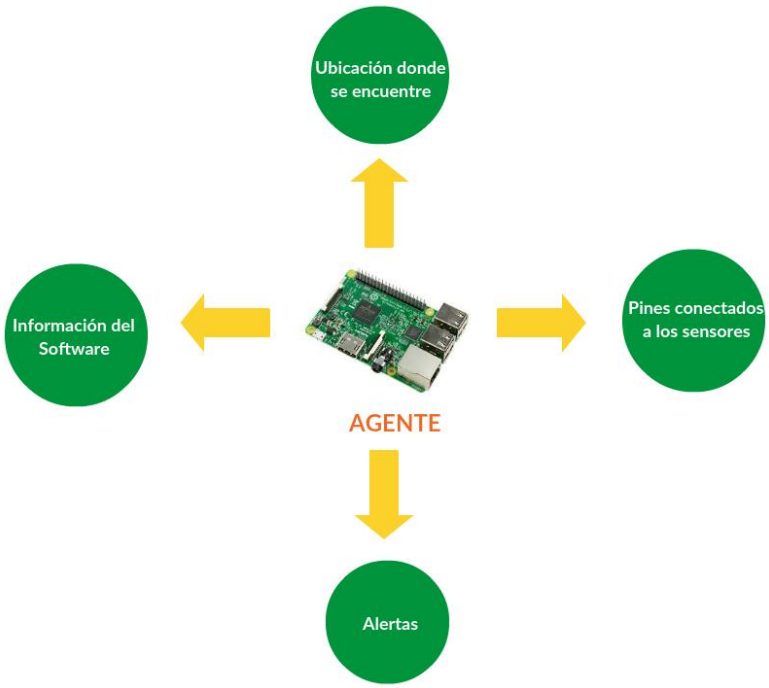


Figura 4-2. Esquema de objetos de gestión relacionados con el agente.

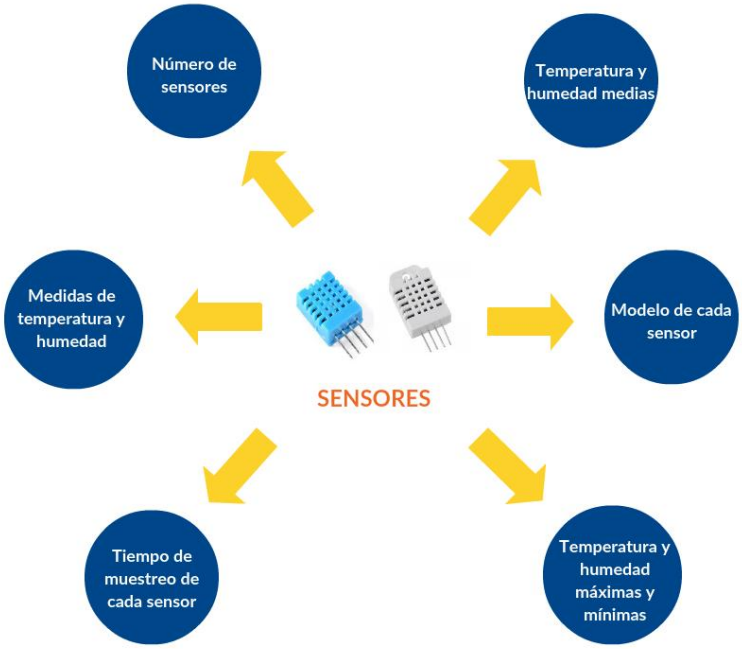


Figura 4-3. Esquema de objetos de gestión relacionados con los sensores.

4.3. Diseño de la MIB

En este punto se va a explicar el diseño de la MIB de nuestro sistema de gestión. Los objetos de una MIB se definen usando un subconjunto del ASN.1, la versión 2 de la estructura de la información de gestión (Structure of Management Information Version 2 o SMIv2) [7].

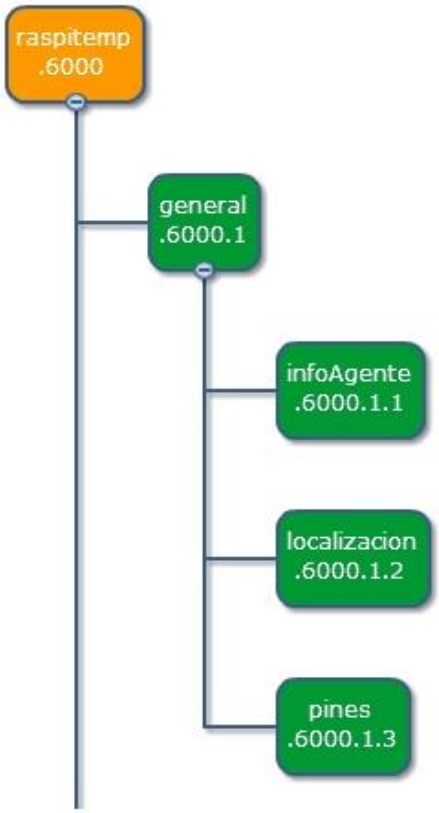
Todos los objetos gestionados en la MIB poseen un identificador único (OID). Todos los objetos de gestión existentes están organizados jerárquicamente y se pueden representar como un árbol con diferentes niveles, desde la raíz hasta las hojas.

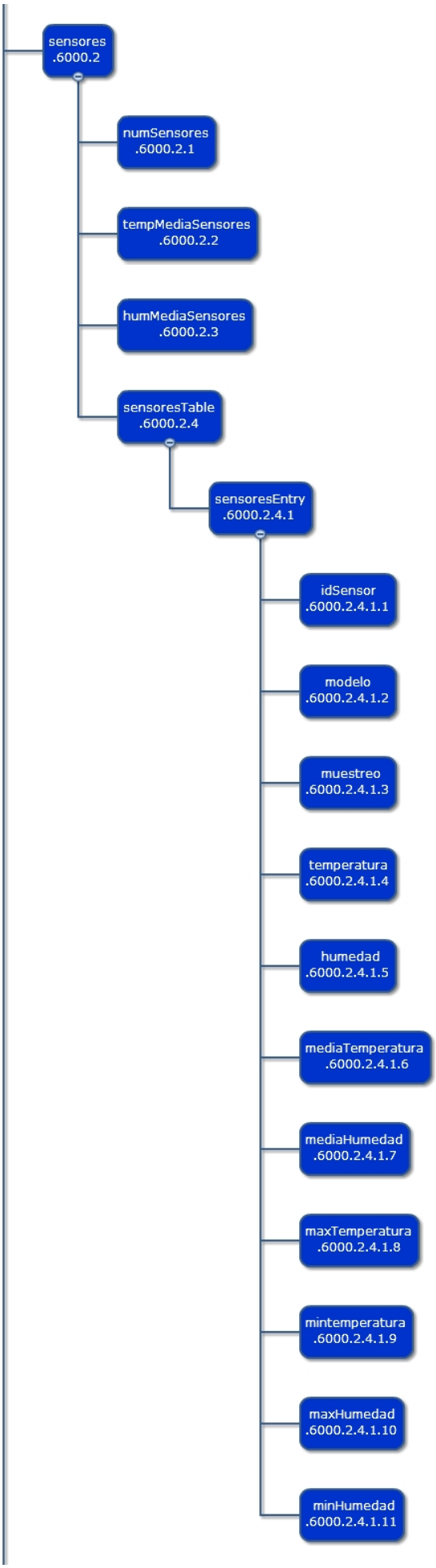
Estos objetos pueden ser de dominio público o privado. Las empresas definen sus objetos debajo de la rama privada, con OID iso.org.dod.internet.private.enterprises o 1.3.6.1.4.1 en forma numérica.

Nuestro OID se creará de forma ficticia para el uso de este trabajo, ya que la IANA es la encargada de incluir nuevos objetos en el árbol. La MIB llevará como nombre TEMPHUM-MIB y el nombre de la empresa será Raspitemp. Tendrá como OID 1.3.6.1.4.1.6000 (iso.org.dod.internet.private.enterprises.raspitemp), por lo que estará situada debajo de la rama privada y dentro de la rama de empresas.

4.3.1 Estructura del árbol

La estructura del árbol de nuestra MIB es la mostrada a continuación:




sensores
.6000.2
numSensores
.6000.2.1
tempMediaSensores
.6000.2.2
humMediaSensores
.6000.2.3
sensoresTable
.6000.2.4
sensoresEntry
.6000.2.4.1
idSensor
.6000.2.4.1.1
modelo
.6000.2.4.1.2
muestreo
.6000.2.4.1.3
temperatura
.6000.2.4.1.4
humedad
.6000.2.4.1.5
mediaTemperatura
.6000.2.4.1.6
mediaHumedad
.6000.2.4.1.7
maxTemperatura
.6000.2.4.1.8
mintemperatura
.6000.2.4.1.9
maxHumedad
.6000.2.4.1.10
minHumedad
.6000.2.4.1.11

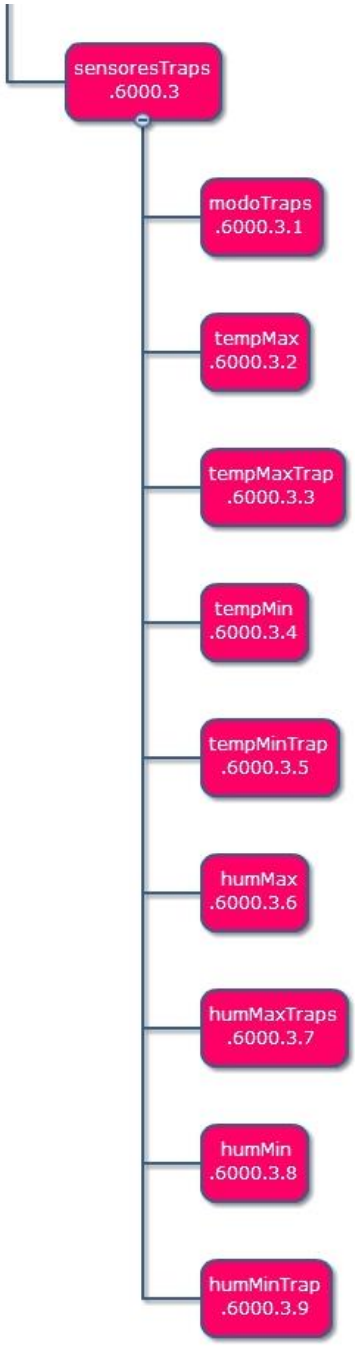


Figura 4-4. Estructura del árbol de la MIB.

4.3.1.1 Grupo General

En este grupo están definidos los objetos que tratan aspectos generales del agente. Estos objetos son:

- **infoAgente**: es una cadena de caracteres de sólo lectura que muestra las características del agente.
- **localizacion**: es una cadena de caracteres de lectura y escritura que muestra el lugar donde se encuentra el agente. Por defecto tiene el valor "default".
- **pines**: es una cadena de caracteres de sólo lectura que muestra los pines en los que están conectados los sensores.

4.3.1.2 Grupo Sensores

En este grupo están definidos los objetos relacionados con los sensores y sus correspondientes mediciones. Estos objetos son:

- **numSensores**: es un entero de sólo lectura que muestra el número de sensores conectados al agente. Por defecto vale 2, ya que en este trabajo se han utilizado dos sensores.
- **tempMediaSensores**: es un entero de sólo lectura que muestra la media de temperatura medida por los sensores en grados centígrados.
- **humMediaSensores**: es un entero de sólo lectura que muestra la media de humedad medidas por los sensores en tanto por ciento.
- **tablaSensores**: es una tabla que almacena la siguiente información de los sensores:
    - **idSensor**: es un entero de sólo lectura que identifica el sensor para diferenciarlo de otro.
    - **modelo**: es una cadena de caracteres de sólo lectura que muestra el modelo del sensor.
    - **muestreo**: es un entero de lectura y escritura que muestra el tiempo de muestreo del sensor en segundos.
    - **temperatura**: es un entero de sólo lectura que muestra la temperatura medida por el sensor en grados centígrados.
    - **humedad**: es un entero de sólo lectura que muestra la humedad medida por el sensor en tanto por ciento.
    - **mediaTemperatura**: es un entero de sólo lectura que muestra la media de las últimas 30 mediciones de la temperatura medida por el sensor en grados centígrados.
    - **mediaHumedad**: es un entero de sólo lectura que muestra la media de las últimas 30 mediciones de la humedad medida por el sensor en tanto por ciento.
    - **maxTemperatura**: es un entero de sólo lectura que muestra la temperatura máxima medida por el sensor en grados centígrados.
    - **minTemperatura**: es un entero de sólo lectura que muestra la temperatura mínima medida por el sensor en grados centígrados.
    - **maxHumedad**: es un entero de sólo lectura que muestra la humedad máxima medida por el sensor en tanto por ciento.
    - **minHumedad**: es un entero de sólo lectura que muestra la humedad mínima medida por el sensor en tanto por ciento.

4.3.1.3 Grupo sensoresTraps

En este grupo están definidos los objetos relacionados con las alertas (traps) enviadas por el agente. Estos objetos son:

- **modoTraps**: es un entero de lectura y escritura que sólo puede tomar dos valores, 0 o 1. Cuando vale 0 los traps funcionan teniendo en cuenta la media de los sensores y cuando vale 1 funcionan teniendo en cuenta las medidas individuales de cada sensor. En este último caso, se envía la información del sensor que haya obtenido la mayor/menor medida.
- **tempMax:** es un entero de lectura y escritura que muestra el límite máximo de temperatura permitido en grados centígrados. Por defecto vale 99 para que no se active el correspondiente trap.
- **tempMaxTrap:** trap que se envía cada 5 segundos cuando la temperatura sobrepasa el límite máximo establecido.
- **tempMin:** es un entero de lectura y escritura que muestra el límite mínimo de temperatura permitido en grados centígrados. Por defecto vale 0 para que no se active el correspondiente trap.

- **tempMinTrap:** trap que se envía cada 5 segundos cuando la temperatura no alcanza el límite mínimo establecido.
- **humMax**: es un entero de lectura y escritura que muestra el límite máximo de humedad permitido en tanto por ciento. Por defecto vale 99 para que no se active el correspondiente trap.
- **humMaxTrap**: trap que se envía cada 5 segundos cuando la humedad sobrepasa el límite máximo establecido.
- **humMin**: es un entero de lectura y escritura que muestra el límite mínimo de humedad permitido en tanto por ciento. Por defecto vale 0 para que no se active el correspondiente trap.
- **humMinTrap**: trap que se envía cada 5 segundos cuando la humedad no alcanza el límite mínimo establecido.

4.4. Validación de la MIB

Para validar la MIB se ha utilizado una herramienta web proporcionada por SimpleWeb [8]. Para ello, hay que seleccionar el archivo de texto donde esté definida la MIB y elegir un nivel de severidad. El nivel 1 es el más tolerante a fallos y el nivel 6 el más estricto. Para que una MIB sea válida es suficiente con que no existan errores de niveles del 0 al 3, ambos incluidos. Los errores de niveles superiores sólo son recomendaciones.

Tras terminar con la MIB, debemos copiarla en los directorios donde el agente buscará todas las MIBs. Estos directorios son los siguientes:

- $HOME/.snmp/mibs
- /usr/local/share/snmp/mibs

5 IMPLEMENTACIÓN

Tras haber explicado el diseño del módulo vamos a detallar el procedimiento que se ha llevado a cabo para implementarlo en una Raspberry Pi.

5.1. Raspberry Pi

La Raspberry Pi utilizada en este trabajo posee las siguientes características:

- Raspberry Pi 3 Model B
- Procesador 1.2GHz Quad-Core ARM Cortex-A53
- 802.11 bgn Wireless LAN
- Bluetooth 4.1
- 1GB RAM
- Salida HDMI

Figura 5-1. Raspberry Pi 3 Model B.

5.2. Sensores

Los sensores utilizados en este trabajo permiten medir simultáneamente temperatura y humedad. Las características de estos son las siguientes:

- **DHT11**:
    - Presenta carcasa azul.
    - Medición de temperatura entre 0 a 50 ºC con precisión de 2 ºC.
    - Medición de humedad entre 20 a 80 % con precisión del 5 %.
    - Frecuencia de muestreo de 1 muestra por segundo.

- **DHT22**:
    - Presenta carcasa blanca.
    - Medición de temperatura entre -40 a 125 ºC con precisión de 0.5 ºC.

- Medición de humedad entre 0 a 100 % con precisión del 2 al 5 %.
- Frecuencia de muestreo de 2 muestras por segundo.

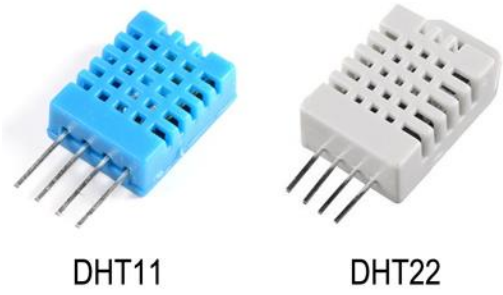


Figura 5-2. Sensores DHT11 y DHT22.

5.2.1 Conexión a Raspberry Pi

Cada sensor tiene 3 pines para conectarlo a la Raspberry Pi. Un pin hay que conectarlo a 3.3 voltios, otro a tierra y el restante a un pin de datos [9].

En la siguiente figura se detallan estas conexiones:

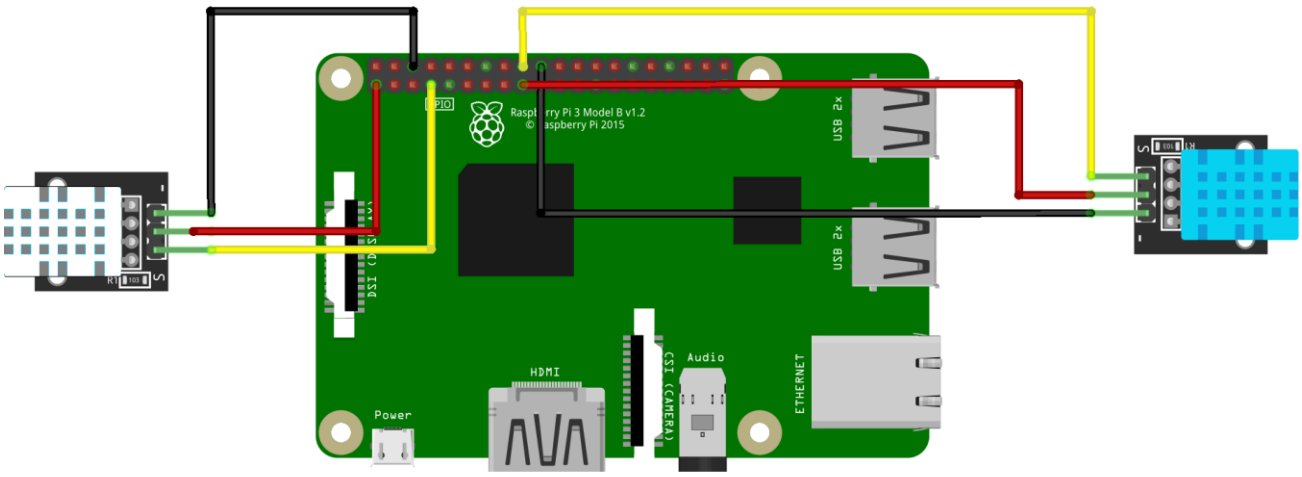


Figura 5-3. Esquema de conexión de sensores a Raspberry Pi.

Las conexiones con cables rojos corresponden a 3.3 voltios, con cables negros a tierra y con cables amarillos a datos.

5.2.2 Obtención de las medidas

Los sensores DHT11 y DHT22 son dispositivos analógicos cuyas tramas de datos son de 40 bits. El primer grupo de 8-bit es la parte entera de la humedad y el segundo grupo la parte decimal. Lo mismo ocurre con el tercer y cuarto grupo, la parte entera de la temperatura y la parte decimal. Por último, los bits de paridad para confirmar que no hay datos corruptos.

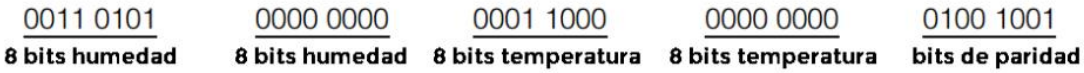


Figura 5-4. Trama de datos de los sensores DHT11 y DHT22.

Para obtener estos datos y decodificarlos, se han utilizado dos programas escritos en C (uno para cada sensor), que extraen de los pines de datos de los sensores estos bits y los convierte en decimal [10]. Cuando se obtiene las medidas de temperatura y humedad, se meten ambas en dos colas de mensajes y en ficheros de texto (para la comunicación de datos con otros programas). Además, estos programas leen de un fichero de texto el tiempo de espera para volver a obtener una nueva medida (tiempo de muestreo).

En la siguiente figura se muestra el diagrama de flujo de ambos programas:

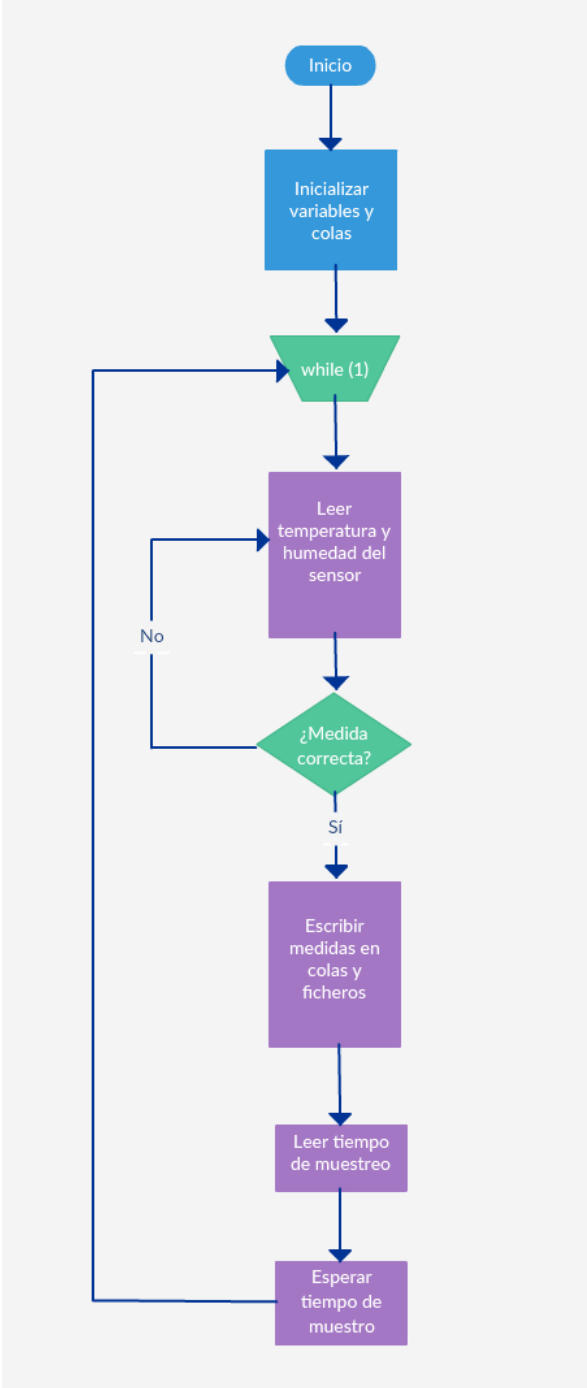


Figura 5-5. Diagrama de fujo de los programas de los sensores.

5.3. Software

En este trabajo se ha utilizado Net-SNMP en su versión más reciente (5.7.3) para el desarrollo del agente en Raspberry Pi. A continuación, se va a proceder a explicar la instalación, la herramienta usada y su puesta en marcha.

5.3.1 Instalación

Los paquetes para la instalación de Net-SNMP se encuentran en los repositorios de Debian. Lo primero que haremos será actualizar los repositorios:

```
sudo apt-get update
sudo apt-get upgrade
```

Lo siguiente que haremos es instalar el agente SNMP:

```
sudo apt-get install snmp
```

Cuando se haya instalado, instalaremos un paquete que permite mostrar el árbol de la MIB por línea de comandos y además nos permite acceder a las MIBs estándar:

```
sudo apt-get install snmp-mibs-downloader
```

Tras ello, procederemos a instalar el demonio, el cual nos permitirá interactuar con los objetos de la MIB:

```
sudo apt-get install snmpd
```

Por último, para la herramienta que utilizaremos para la programación de los objetos (mib2c), necesitaremos instalar dos librerías:

```
sudo apt-get install libsnmp-dev
sudo apt-get install libsnmp-perl
```

5.3.2 Configuración

Por defecto, Net-SNMP utiliza las MIBs por defecto descargadas anteriormente. Para que utilice la nuestra, deberemos modificar el fichero de configuración correspondiente. Este fichero se encuentra en /etc/snmp/snmp.conf. Dentro encontraremos una línea que pone “mibs”. Al cambiarla por “mibs +ALL”, Net-SNMP buscará todas las MIBs que existan en los directorios que utiliza por defecto.

```
  GNU nano 2.7.4                                        Fichero: /etc/snmp/snmp.conf

# As the snmp packages come without MIB files due to license reasons, loading
# of MIBs is disabled by default. If you added the MIBs you can reenable
# loading them by commenting out the following line.
mibs +ALL
```

Figura 5-6. Fragmento de código del fichero /etc/snmp/snmp.conf.

En el capítulo anterior se explicó que nuestra MIB tenía que estar en los directorios donde el agente buscaría por defecto. En nuestro caso, estos directorios son /home/pi/.snmp/mibs y /usr/local/share/snmp/mibs.

Para que Net-SNMP detecte las nuevas MIBs incluidas en estos directorios, hay que reiniciar el servicio snmpd:

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

Tras ello, nos aparecerá un aviso informándonos de que el servicio se ha reiniciado con éxito.

Para comprobar que la MIB se ha cargado correctamente, podemos utilizar un comando que nos muestra su árbol con información de los nodos:

```
snmptranslate -Tp -IR TEMPHUM-MIB::raspitemp
```

IR se utiliza para que se busque la MIB por su nombre y Tp para que se dibuje el árbol.

```
pi@raspberrypi:~ $ snmptranslate -Tp -IR TEMPHUM-MIB::raspitemp
+--raspitemp(6000)
   |
   +--general(1)
   |  |
   |  +-- -R-- String    infoAgente(1)
   |  +-- -RW- String    localizacion(2)
   |  +-- -R-- String    pines(3)
   |
   +--sensores(2)
   |  |
   |  +-- -R-- Integer32 numSensores(1)
   |  +-- -R-- Integer32 tempMediaSensores(2)
   |  +-- -R-- Integer32 humMediaSensores(3)
   |  |
   |  +--sensoresTable(4)
   |     |
   |     +--sensoresEntry(1)
   |        |  Index: idSensor
   |        |
   |        +-- -R-- Integer32 idSensor(1)
   |        +-- -R-- String    modelo(2)
   |        +-- -RW- Integer32 muestreo(3)
   |        +-- -R-- Integer32 temperatura(4)
   |        +-- -R-- Integer32 humedad(5)
   |        +-- -R-- Integer32 mediaTemperatura(6)
   |        +-- -R-- Integer32 mediaHumedad(7)
   |        +-- -R-- Integer32 maxTemperatura(8)
   |        +-- -R-- Integer32 minTemperatura(9)
   |        +-- -R-- Integer32 maxHumedad(10)
   |        +-- -R-- Integer32 minHumedad(11)
   |
   +--sensoresTraps(3)
      |
      +-- -RW- Integer32 modoTraps(1)
      |        Range: 0..1
      +-- -RW- Integer32 tempMax(2)
      +--tempMaxTrap(3)
      +-- -RW- Integer32 tempMin(4)
      +--tempMinTrap(5)
      +-- -RW- Integer32 humMax(6)
      +--humMaxTrap(7)
      +-- -RW- Integer32 humMin(8)
      +--humMinTrap(9)
```

Figura 5-7. Árbol de la MIB obtenido con el comando snmptranslate.

5.3.3 AgentX

Net-SNMP contiene una implementación del agente AgentX, el cual es un protocolo de extensibilidad definido en la RFC 2741 [2]. Este protocolo permite gestionar objetos SNMP definidos por diferentes procesos a través de un solo maestro. Los agentes que exportan los objetos a través del AgentX al agente maestro se denominan subagentes. Los subagentes se comunican con el agente maestro a través del protocolo AgentX. Esta comunicación es invisible para el gestor SNMP, ya que desde su lado sólo ve a un agente SNMP como otro cualquiera.

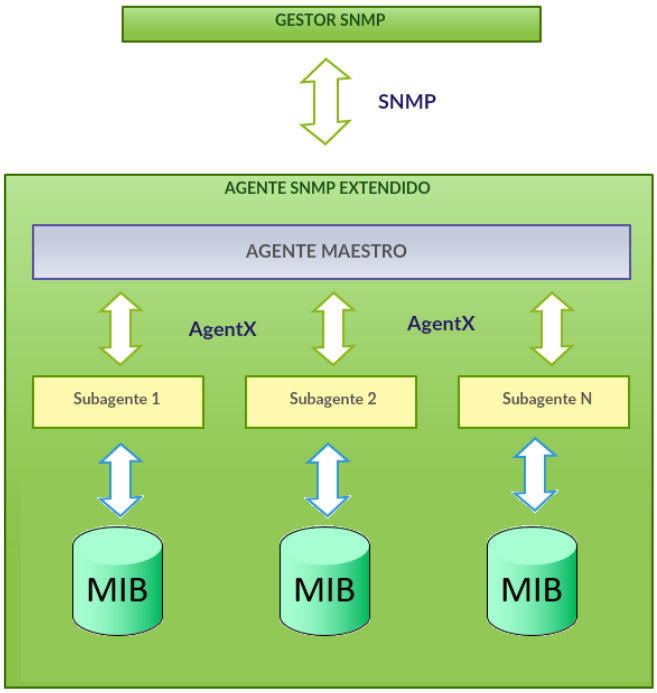


Figura 5-8. Esquema de AgentX.

AgentX es un protocolo de la capa de aplicación, orientado a conexión y que se apoya en el protocolo TCP. Utiliza el puerto 705 para comunicarse con el agente maestro.

La estructura de la cabecera de su PDU es la siguiente:

h.version	h.type	h.flags	<reserved>
h.sessionID			
h.transactionID			
h.packetID			
h.payload_length			

Tabla 5-1. Cabecera de la PDU de AgentX.

El campo más importante es h.type, el cual informa del tipo de PDU que se envía.

5.3.3.1 Configuración

El agente maestro se comunicará con el gestor mediante SNMP y con los subagentes mediante el protocolo AgentX. Para que esto sea posible hay que modificar el archivo /etc/snmp/snmpd.conf. Se trata del archivo de configuración del demonio de Net-SNMP.

Por defecto, el puerto que aparece para las conexiones SNMP es el 161.

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

Figura 5-9. Fragmento de código del fichero /etc/snmp/snmpd.conf con el puerto utilizado.

En el apartado ACCESS CONTROL, añadiremos nuestra MIB y habilitaremos que la comunidad "public" tenga acceso de sólo lectura y la "private" de lectura y escritura desde cualquier dispositivo.

```
#  ACCESS CONTROL
#

                                          #  system + hrSystem groups only
view   systemonly  included   .1.3.6.1.2.1.1
view   systemonly  included   .1.3.6.1.2.1.25.1


view   all         included   .1.3.6.1.4.1.6000

                                          #  Full access from the local host
 rocommunity public  localhost
                                          #  Default access to basic system info
#rocommunity public  default    -V systemonly
                                          #  rocommunity6 is for IPv6
#rocommunity6 public  default   -V systemonly


 rocommunity public default
 rwcommunity private default
```

Figura 5-10. Fragmento de código del fichero /etc/snmp/snmpd.conf con apartado ACCESS CONTROL.

En el apartado ACTIVE MONITORING, añadiremos las IPs a las que se enviarán los traps generados por el agente.

```
###############################################################################
#
#  ACTIVE MONITORING
#

                                  #   send SNMPv1  traps
 trapsink     localhost public
                                  #   send SNMPv2c traps
 trap2sink    localhost public

 trap2sink    192.168.1.90 public

                                  #   send SNMPv2c INFORMs
#informsink   localhost public
```

Figura 5-11. Fragmento de código del fichero /etc/snmp/snmpd.conf con el apartado ACTIVE MONITORING.

Por defecto, el agente funcionará como maestro.

```
#  AgentX Sub-agents
#
                                        #  Run as an AgentX master agent
 master          agentx
                                        #  Listen for network connections (from localhost)
                                        #    rather than the default named socket /var/agentx/master
#agentXSocket    tcp:localhost:705
```

Figura 5-12. Fragmento de código del fichero /etc/snmp/snmpd.conf con la información del AgentX.

Para que los cambios surtan efecto, se debe reiniciar el demonio SNMP con el comando que se utilizó anteriormente:

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

5.3.4 Mib2c

Para poder manejar las variables asociadas a los objetos de la MIB necesitamos demonios para interactuar con cada uno de ellos. Para ello, nos apoyaremos en una herramienta incluida en el paquete de Net-SNMP llamada "mib2c" [11].

Esta herramienta genera unos documentos esqueleto que tienen partes en código C y otras en pseudocódigo, en las cuales se explica cómo completar el código para compilarlo.

Para que genere estos códigos, debemos indicarle el nombre del objeto de la MIB y el nombre de un archivo de configuración. Obtendremos 2 plantillas, una será un fichero de cabecera y otro un fichero de implementación. Estos códigos se obtienen para un objeto genérico, por lo que se debe adaptar a un objeto en concreto.

Los archivos de configuración vienen integrados en la propia herramienta, sólo hay que indicar cuál usaremos para cada objeto. En nuestro caso, utilizaremos "mib2c.scalar.conf" para los escalares, "mib2c.table_data.conf" para la tabla y "mib2c.notify.conf" para los traps.

Para hacer funcionar el demonio de cada objeto necesitaremos 4 plantillas. Las dos primeras son las obtenidas por la herramienta mib2c y que hay que adaptar al objeto. Las otras dos restantes nos las facilitan en la web de Net-SNMP. Se trata de una plantilla para iniciar el demonio y un makefile para compilar todos los ficheros [12].

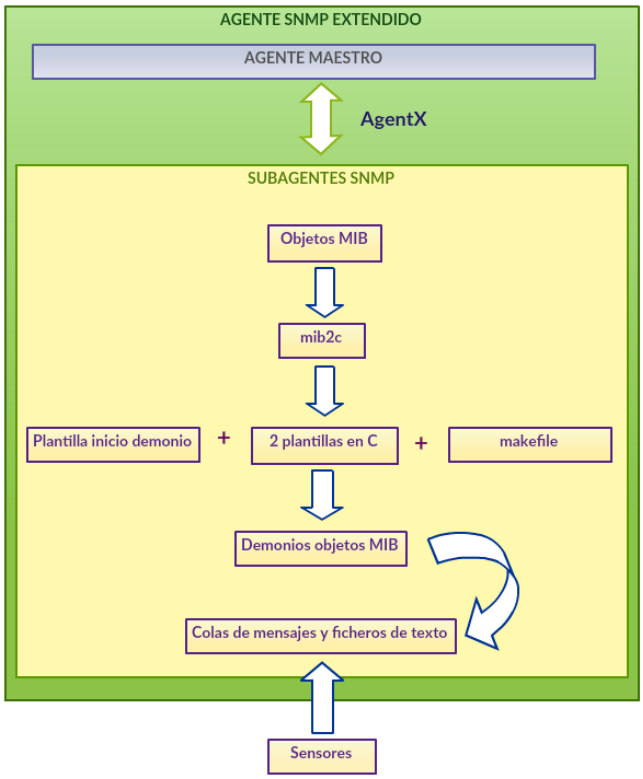


Figura 5-13. Esquema de AgentX con Mib2c.

5.3.4.1 Objetos escalares

En este apartado se llevará a cabo la implementación de los objetos escalares de nuestra MIB. Estos objetos son los siguientes:

- infoAgente (.1.3.6.1.4.1.6000.1.1)
- localización (.1.3.6.1.4.1.6000.1.2)
- pines (.1.3.6.1.4.1.6000.1.3)
- numSensores (.1.3.6.1.4.1.6000.2.1)
- tempMediaSensores (.1.3.6.1.4.1.6000.2.2)
- humMediaSensores (.1.3.6.1.4.1.6000.2.3)
- modoTraps (.1.3.6.1.4.1.6000.3.1)
- tempMax (.1.3.6.1.4.1.6000.3.2)
- tempMin (.1.3.6.1.4.1.6000.3.4)
- humMax (.1.3.6.1.4.1.6000.3.6)
- humMin (.1.3.6.1.4.1.6000.3.8)

Para obtener el código esqueleto de cada uno se ha utilizado el siguiente comando, cambiando "nombre_objeto" por uno de la lista anterior:

```
sudo mib2c -c mib2c.scalar.conf TEMPHUM-MIB::nombre_objeto
```

Este comando nos genera dos ficheros tal y como se dijo anteriormente: nombre_objeto.h y nombre_objeto.c.

En nombre_objeto.c se inicializan las funciones SNMP del objeto gestionado y se registra su oid.

```
/** Initializes the nombre_objeto module */
void
init_nombre_objeto(void)
{
    const oid nombre_objeto_oid[] = { 1,3,6,1,4,1,6000,X,X };
}
```

Figura 5-14. Fragmento de código del fichero nombre_objeto.c con la inicialización del objeto gestionado.

En este fichero podemos encontrar también un manejador para las peticiones SNMP que reciba el agente para ese objeto. En la parte de código correspondiente al mismo podemos encontrar un switch con varios case, cada uno con una posible petición SNMP.

En el manejador de los objetos que sean de sólo lectura sólo encontramos el case correspondiente a la petición GET, en el cual deberemos pasar un puntero donde se encuentre el valor actual de la variable correspondiente al objeto gestionado y su tamaño.

```
  switch(reqinfo->mode) {
        case MODE_GET:
            snmp_set_var_typed_value(requests->requestvb, ASN_OCTET_STR,
                                     &nombre_objeto,
                                     sizeof(nombre_objeto));
            break;
  }
```

Figura 5-15. Fragmento de código del fichero nombre_objeto.c con case GET.

En el manejador de los objetos que sean de lectura y escritura encontraremos, además del case anterior, otro para el SET, en el cual la variable asociada al objeto tomará el nuevo valor obtenido en la petición:

```
  case MODE_SET_ACTION:
            nombre_objeto = *requests->requestvb->val.integer;
            break;
```

Figura 5-16. Fragmento de código del fichero nombre_objeto.c con case SET.

Como se dijo anteriormente, utilizaremos un fichero con el código de inicio del demonio del correspondiente objeto.

Algunas partes relevantes de este código son las siguientes:

```
..
int agentx_subagent=1; /* change this if you want to be a SNMP master agent */
..

/* we're an agentx subagent? */
  if (agentx_subagent) {
    /* make us a agentx client. */
    netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID, NETSNMP_DS_AGENT_ROLE, 1);
  }
..

/* initialize the agent library */
init_agent("nombre_objeto_demon");

init_nombre_objeto();
..
```

Figura 5-17. Fragmento de código del fichero nombre_objeto_demon.c con inicialización del demonio.

Este código lo que hace es inicializar el subagente AgentX y la variable asociada al objeto.

Algunos de los demonios de los objetos escalares mencionados anteriormente necesitan realizar funciones cuando están en funcionamiento. Por ejemplo, el demonio del objeto humMediaSensores necesita leer de un fichero de texto el valor medio de la humedad para enviarlo en un GET y el demonio del objeto humMax necesita escribir en un fichero de texto el valor que recibe de un SET para hacer funcionar su correspondiente Trap.

Es por ello por lo que en estos códigos se hayan añadido estas funciones dentro de un bucle while que se repite infinitamente mientras el demonio esté en funcionamiento.

En el caso en el que se necesite escribir en un fichero se hará de la siguiente forma:

```
  while(keep_running) {

    //Escribe en un txt
    fp = fopen ("ruta_del_fichero.txt", "w");
    if(fp == NULL)
      exit(1);
    fprintf(fp, "%ld", nombre_objeto);
    fclose(fp);
  }
```

Figura 5-18. Fragmento de código del fichero nombre_objeto_demon.c con escritura en fichero.

En el caso en el que se necesite leer de un fichero se hará de la siguiente forma:

```
while(keep_running) {


   //Lee de un txt
   fp = fopen ("ruta_de_fichero.txt", "r");
   if(fp == NULL)
     exit(1);
   if(fscanf(fp, "%ld", &nombre_objeto) != 1)
     printf("%s\n", "Error al leer");
   fclose(fp);
}
```

Figura 5-19. Fragmento de código del fichero nombre_objeto_demon.c con lectura de fichero.

Por último, necesitamos el makefile, el cual tendrá el siguiente código:

```
#
# Warning: you may need more libraries
than are included here on the
# build line.  The agent frequently
needs various libraries in order
# to compile pieces of it, but is OS
dependent and we can't list all
# the combinations here.  Instead,
look at the libraries that were
# used when linking the snmpd master
agent and copy those to this
# file.
#

CC=gcc

OBJS1=snmpdemoapp.o
OBJS2=nombre_objeto_demon.o
nombre_objeto.o
OBJS3=asyncapp.o
TARGETS=nombre_objeto_demon
snmpdemoapp asyncapp

CFLAGS=-I. `net-snmp-config --cflags`
BUILDLIBS=`net-snmp-config --libs`
BUILDAGENTLIBS=`net-snmp-config --
agent-libs`

# shared library flags (assumes gcc)
DLFLAGS=-fPIC -shared
```

```
all: $(TARGETS)

snmpdemoapp: $(OBJS1)
        $(CC) -o snmpdemoapp $(OBJS1)
$(BUILDLIBS)

asyncapp: $(OBJS3)
        $(CC) -o asyncapp $(OBJS3)
$(BUILDLIBS)

nombre_objeto_demon: $(OBJS2)
        $(CC) -o nombre_objeto_demon
$(OBJS2)  $(BUILDAGENTLIBS)

clean:
        rm $(OBJS2) $(OBJS2)
$(TARGETS)

nstAgentPluginObject.so:
nstAgentPluginObject.c Makefile
        $(CC) $(CFLAGS) $(DLFLAGS) -c
-o nstAgentPluginObject.o
nstAgentPluginObject.c
        $(CC) $(CFLAGS) $(DLFLAGS) -o
nstAgentPluginObject.so
nstAgentPluginObject.o
```

Figura 5-20. Fragmento de código del fichero Makefile.

Para poner en marcha el demonio del objeto, nos situaremos en el directorio donde se encuentren los 4 ficheros necesarios, compilaremos y ejecutaremos:

```
sudo makefile nombre_objeto_demon
sudo ./nombre_objeto_demon
```

5.3.4.2 Objeto tabla

En este apartado se llevará a cabo la implementación del objeto tabla de nuestra MIB (sensoresTable .1.3.6.1.4.1.6000.2.4)

Para obtener el código esqueleto se ha utilizado el siguiente comando:

```
sudo mib2c -c mib2c.table_data.conf TEMPHUM-MIB::sensoresTable
```

Tras pulsar intro nos aparecerá una opción para habilitar una caché de datos, la cual seleccionaremos.

Con este comando se generan los ficheros sensoresTable.h y sensoresTable.c.

El archivo de configuración "table_data", está preparado para funcionar con una caché interna de datos que funciona como buffer intermedio entre los datos de una tabla externa y el objeto tabla. Esta tabla externa debe estar en una localización cualquiera dentro del sistema operativo del agente, como por ejemplo en un fichero de texto donde las filas están separadas por saltos de línea y las columnas por espacios. La tabla SNMP debe actualizarse inmediatamente después de cualquier cambio en la tabla externa (la situada en el fichero de texto) y debe estar ordenada lexicográficamente.

Net-SNMP extrae los datos del fichero de texto, los carga en la caché y actualiza la tabla SNMP. Además, no importa que la tabla externa no esté ordenada lexicográficamente, ya que se encarga de ordenarla al pasarla a la caché. Cuando se añaden nuevas filas a la tabla externa, también las añade a la misma.

Para realizar esto, en la plantilla generada por mib2c se proporcionan algunas funciones como las siguientes:

- **sensoresTable_load:** extrae los datos de la tabla externa y los carga en la caché.
- **sensoresTable_createEntry:** crea una nueva fila en la tabla SNMP
- **sensoresTable_removeEntry:** elimina una fila en la tabla SNMP
- **netsnmp_cache_create:** crea la caché a partir de los datos obtenidos por sensoresTable_load
- **netsnmp_inject_handler:** carga los datos en la tabla SNMP de forma ordenada.
- **sensoresTable_handler:** manejador que responde a las peticiones SNMP

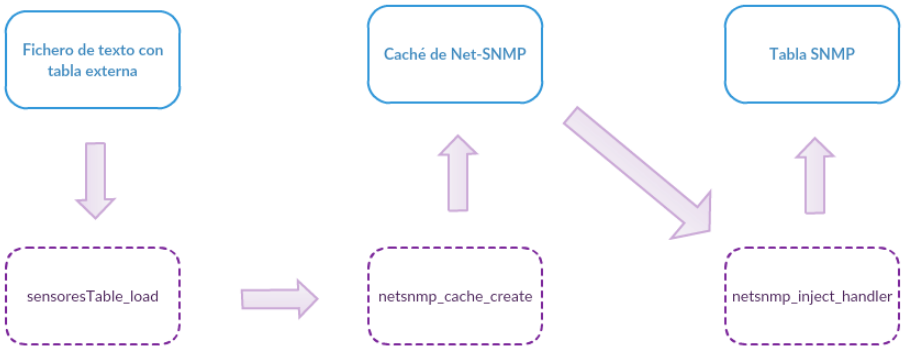


Figura 5-21. Esquema de funcionamiento del fichero sensoresTable.c.

La función más importante para nosotros es **sensoresTable_load**, ya que debemos programar la extracción de los datos de la tabla externa que se encuentra en el fichero de texto. Dentro del bucle while, se lee del fichero de la tabla externa por filas y se va guardando esa información en una estructura que posteriormente se utilizará para actualizar el objeto tabla SNMP.

```
int sensoresTable_load( netsnmp_cache *cache, void *vmagic ) {
    netsnmp_tdata     *table = (netsnmp_tdata *)vmagic;
    netsnmp_tdata_row *row;
    struct sensoresTable_entry *this;
    FILE * fp;
    char buf[255];
    …
    fp = fopen( "/home/pi/sensoresTable/sensoresTable.txt", "r" );
    ..
    while ( fgets( buf, 255, fp )) {
        sscanf(buf, "%ld %s %ld %ld %ld %ld %ld %ld %ld %ld %ld", &idSensor,
          modelo, &muestreo, &temperatura, &humedad, &mediaTemperatura,
          &mediaHumedad, &maxTemperatura, &minTemperatura, &maxHumedad,
          &minHumedad);
        row = sensoresTable_createEntry(table, idSensor);
        this = row->data;
        strncpy(this->modelo, modelo, strlen(modelo));
        this->muestreo = muestreo;
        this->temperatura = temperatura;
        this->humedad = humedad;
        this->mediaTemperatura = mediaTemperatura;
        this->mediaHumedad = mediaHumedad;
        this->maxTemperatura = maxTemperatura;
        this->minTemperatura = minTemperatura;
        this->maxHumedad = maxHumedad;
        this->minHumedad = minHumedad;
    }
    …
}
```

Figura 5-22. Fragmento de código del fichero sensoresTable.c.

Para escribir la tabla externa en un fichero de texto, se ha creado un fichero de código en C que extrae de las colas la información que se necesita conocer para rellenar la tabla SNMP (temperatura y humedad de cada sensor, tiempos de muestro…). Este fichero contiene varias funciones que recogen esta información de los sensores y para que esta recogida de datos se haga de forma paralela y sea más rápida, se ha procedido a crear varios hilos a los que se le ha asignado una función en bucle infinito, la cual obtiene de una cola de mensajes el parámetro correspondiente.

```
void * temp1_f(void * arg) {
  key_t clave = ftok("temphum", ENTERO_CLAVE);
  int msqid = msgget(clave, 0777 | IPC_CREAT);
  struct temphum_msg temp1_msg;
  int longitud = sizeof(temp1_msg) - sizeof(temp1_msg.tipo);

  while(1) {
    /*Lee de la cola y extrae el valor de la temperatura del sensor 1*/
    if (msgrcv(msqid, &temp1_msg, longitud, 1, 0) == -1)
      printf("%s\n", "Error al recibir temperatura del sensor 1");

    temp1 = temp1_msg.parametro;

    if(temp1_msg.parametro > temp1Max)
    temp1Max = temp1_msg.parametro;

    if(temp1_msg.parametro < temp1Min)
    temp1Min = temp1_msg.parametro;
  }

  return NULL;
}
```

Figura 5-23. Fragmento de código del fichero infoTabla.c con el ejemplo de una función.

La función main de este fichero lo que hace es crear los hilos y escribir en el fichero de texto la tabla externa que será cargada en la caché de Net-SNMP.

```
  …
  pthread_t temp1_h;
  pthread_t temp2_h;
  pthread_t hum1_h;
  pthread_t hum2_h;
  …
  pthread_create (&temp1_h, NULL, temp1_f, NULL);
  pthread_create (&temp2_h, NULL, temp2_f, NULL);
  pthread_create (&hum1_h, NULL, hum1_f, NULL);
  …
  while(1) {
    fp = fopen ("/home/pi/sensoresTable/sensoresTable.txt", "w");

    if(fp == NULL)
      exit(1);

    fprintf(fp, "%ld %s %ld %ld %ld %ld %ld %ld %ld %ld %ld\n", (long)1, "DHT11",
    muestreo1,  temp1,  hum1,  mediaTemp1,  mediaHum1,  temp1Max,  temp1Min,  hum1Max,
hum1Min);

    fprintf(fp, "%ld %s %ld %ld %ld %ld %ld %ld %ld %ld %ld", (long)2, "DHT22",
    muestreo2,  temp2,  hum2,  mediaTemp2,  mediaHum2,  temp2Max,  temp2Min,  hum2Max,
hum2Min);

    fclose(fp);

    sleep(4);
  }
  …
```

Figura 5-24. Fragmento de código del fichero infoTabla.c con función Main.

Por último, al ser la columna “muestreo” de la tabla un objeto de lectura y escritura, tendremos que recoger el cambio de su valor cuando haya una petición SET en la tabla SNMP. Para ello, tal y como se hizo anteriormente en los objetos escalares, utilizaremos el correspondiente bucle while del fichero de inicio del demonio para recoger ese nuevo valor e introducirlo en una cola, para que así el fichero anterior actualice el valor en la tabla externa y posteriormente se pase a la caché y a la tabla SNMP.

```
key_t clave = ftok("temphum", ENTERO_CLAVE);
int msqid = msgget(clave, 0777 | IPC_CREAT);
if(msqid == -1)
   exit(1);

struct temphum_msg muestreo;
int longitud = sizeof(muestreo) - sizeof(muestreo.tipo);

while(keep_running) {

  if(num_sensor != 0) {

  if(num_sensor == 1)
    muestreo.tipo = 5;
  if(num_sensor == 2)
    muestreo.tipo = 6;

  muestreo.parametro = var_muestreo;
  if(msgsnd(msqid, &muestreo, longitud, 0) == -1)
    printf("%s\n", "Error al enviar muestreo");
  }
}
```

Figura 5-25. Fragmento de código del fichero sensoresTable_demon.c.

Finalmente, adaptaremos el fichero makefile al objeto tabla y pondremos en marcha el demonio del objeto. Nos situaremos en el directorio donde se encuentren los 4 ficheros necesarios, compilaremos y ejecutaremos:

```
sudo makefile sensoresTable_demon
sudo ./sensoresTable_demon
```

5.3.4.3 Traps

Los traps son avisos que se envían a gestor SNMP para alertarle sobre un cambio en el valor de un objeto de gestión. En nuestro caso, se van a programar traps de versión 2c para que se envíen cuando se superen los límites de temperatura y de humedad o no se superen los mínimos establecidos.

Para ello, cada trap tiene asociado un objeto de tipo escalar, en el cual se establecen estos límites. Cuando se active uno de ellos, se enviará la siguiente información:

- Tiempo que lleva el agente en funcionamiento (sysUpTime)
- Identificador del objeto trap
- Identificador de la variable asociada al trap
- Valor de la variable asociada al trap

Para obtener el código esqueleto se ha utilizado el siguiente comando:

```
sudo mib2c -c mib2c.notify.conf TEMPHUM-MIB::nombre_objeto_trap
```

De nuevo se obtienen dos ficheros: nombre_objeto_trap.c y nombre_objeto_trap.h

En nombre_objeto_trap.c se encuentra la programación de la información que se enviará cuando se active el trap. Como se dijo anteriormente, dependiendo del valor que tomase el objeto "modoTraps", se tendrían en cuenta los valores individuales de cada sensor o la media de ambos.

Algunas partes relevantes del código son las siguientes:

```
int send_nombre_objeto_trap( void )
{
    …
    snmp_varlist_add_variable(&var_list,
        snmptrap_oid, snmptrap_oid_len,
        ASN_OBJECT_ID,
        nombre_objeto_trap_oid, sizeof(nombre_objeto_trap_oid));

     //Si el modo es 0, se envia la media de ambos sensores
    if(modoTraps == 0)
    snmp_varlist_add_variable(&var_list,
        parametroMediaSensores_oid, OID_LENGTH(parametroMediaSensores_oid),
        ASN_INTEGER,
        &parametro, sizeof(parametro));

    //Si el modo es 1, se envia el valor del sensor que marque un parametro mayor
    else if(modoTraps == 1) {
      //Si el sensor con mayor valor es el 1, se envia
      if(sensor == 1)
        snmp_varlist_add_variable(&var_list,
            parametro1_oid, OID_LENGTH(parametro1_oid),
            ASN_INTEGER,
            &parametro, sizeof(parametro));
      //Si el sensor con mayor valor es el 2, se envia
      else if(sensor == 2)
        snmp_varlist_add_variable(&var_list,
            parametro2_oid, OID_LENGTH(parametro2_oid),
            ASN_INTEGER,
            &parametro, sizeof(parametro)); }
```

Figura 5-26. Fragmento de código del fichero nombre_objeto_trap.c.

Para que se realice el envío del trap, en el bucle while deberemos de leer de un fichero el valor del objeto asociado al trap y enviarlo cuando esté fuera de los límites establecidos.

```
while(keep_running) {


    /*Lee de un txt y extrae el valor limite*/
    fp = fopen ("ruta_valor_limite.txt", "r");
    …
    /*Lee de un txt y extrae el valor del objeto asociado*/
      fp = fopen ("ruta_valor_objeto_asociado.txt", "r");
    …
    if (parametro > limite) {
        send_nombre_objeto_trap();
        /*Envia un trap cada 5 segundos*/
        sleep(5);
}
```

Figura 5-27. Fragmento de código del fichero nombre_objeto_trap_demon.c.

Finalmente, adaptaremos el fichero makefile al objeto trap y pondremos en marcha el demonio del objeto. Nos situaremos en el directorio donde se encuentren los 4 ficheros necesarios, compilaremos y ejecutaremos:

```
sudo makefile nombre_objeto_trap_demon
sudo ./nombre_objeto_trap_demon
```

6 PRUEBAS

Para este apartado, debemos tener en funcionamiento todo el sistema de gestión. Para ello, se ha procedido a crear un fichero shell script que pone en marcha los 7 demonios de los subagentes, el programa que recoge la información para la tabla y los dos programas que extraen la temperatura y la humedad de los sensores. También se ha creado otro fichero shell script para poner en funcionamiento los 9 demonios de los subagentes relacionados con los traps.

6.1. Escenario de pruebas

El escenario elegido para realizar las pruebas esté compuesto de los siguientes dispositivos:

- Ordenador con IP 192.168.1.90/24 que ejecutará el programa MIB Browser [13] para acceder a los objetos de gestión.
- Raspberry Pi 3 Model B con IP 192.168.1.50/24 que ejecutará el agente maestro Net-SNMP y los subagentes de los objetos de gestión.

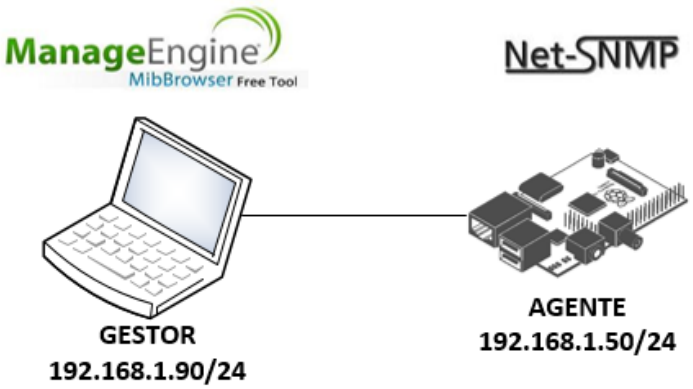


Figura 6-1. Esquema del escenario de pruebas.

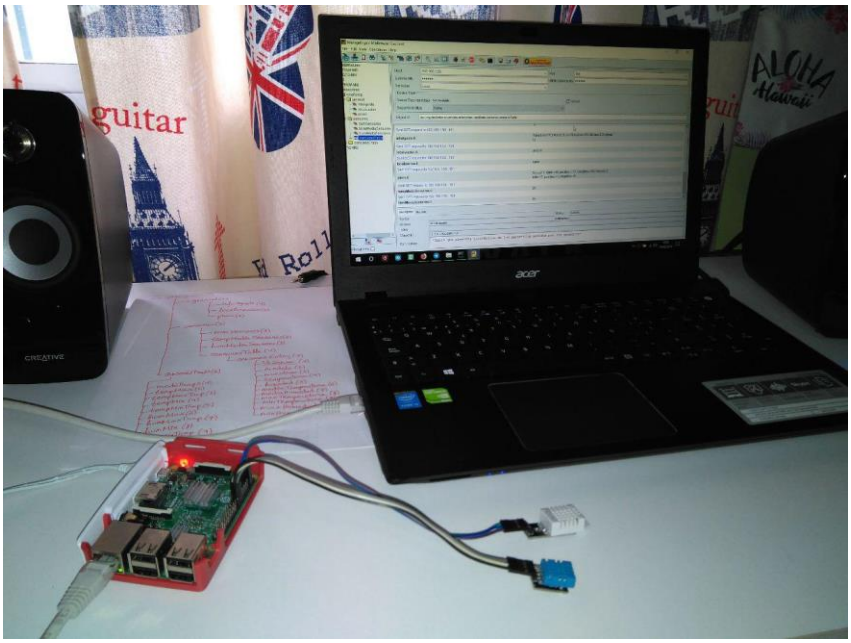


Figura 6-2. Escenario de pruebas.

6.2. Objetos escalares

6.2.1 infoAgente

Prueba	Petición GET al objeto infoAgente
Respuesta 	Sent GET request to 192.168.1.50 : 161 **infoAgente.0** Raspberry Pi 3 Model B con Raspbian GNU/Linux 9.3 (stretch)

Tabla 6-1. Prueba petición GET al objeto infoAgente.

6.2.2 localizacion

Prueba	Petición GET al objeto localizacion
Respuesta 	Sent GET request to 192.168.1.50 : 161 **localizacion.0** default

Tabla 6-2. Prueba petición GET al objeto localizacion.

Prueba	Petición SET al objeto localizacion
Respuesta 	Sent SET request to 192.168.1.50 : 161 localizacion.0 casa

Tabla 6-3. Prueba petición SET al objeto localizacion.

6.2.3 pines

Prueba	Petición GET al objeto pines
Respuesta 	Sent GET request to 192.168.1.50 : 161 pines.0 Sensor 1: data->18, positivo->17, negativo->20; Sensor 2: data->7, positivo->1, negativo->6

Tabla 6-4. Prueba petición GET al objeto pines.

6.2.4 numSensores

Prueba	Petición GET al objeto numSensores
Respuesta 	Sent GET request to 192.168.1.50 : 161 numSensores.0 2

Tabla 6-5. Prueba petición GET al objeto numSensores.

6.2.5 tempMediaSensores

Prueba	Petición GET al objeto tempMediaSensores
Respuesta 	Sent GET request to 192.168.1.50 : 161 tempMediaSensores.0 18

Tabla 6-6. Prueba petición GET al objeto tempMediaSensores.

6.2.6 humMediaSensores

Prueba	Petición GET al objeto humMediaSensores
Respuesta 	Sent GET request to 192.168.1.50 : 161 humMediaSensores.0 60

Tabla 6-7. Prueba petición GET al objeto humMediaSensores.

6.2.7 modoTraps

Prueba	Petición GET al objeto modoTraps
Respuesta 	Sent GET request to 192.168.1.50 : 161 modoTraps.0 0

Tabla 6-8. Prueba petición GET al objeto modoTraps.

Prueba	Petición SET al objeto modoTraps
Respuesta 	Sent SET request to 192.168.1.50 : 161 modoTraps.0 1

Tabla 6-9. Prueba petición SET al objeto modoTraps.

6.2.8 tempMax

Prueba	Petición GET al objeto tempMax
Respuesta 	Sent GET request to 192.168.1.50 : 161 tempMax.0 99

Tabla 6-10. Prueba petición GET al objeto tempMax.

Prueba	Petición SET al objeto tempMax
Respuesta 	Sent SET request to 192.168.1.50 : 161 tempMax.0 30

Tabla 6-11. Prueba petición SET al objeto tempMax.

6.2.9 tempMin

Prueba	Petición GET al objeto tempMin
Respuesta 	Sent GET request to 192.168.1.50 : 161 tempMin.0 0

Tabla 6-12. Prueba petición GET al objeto tempMin.

Prueba	Petición SET al objeto tempMin
Respuesta 	Sent SET request to 192.168.1.50 : 161 tempMin.0 15

Tabla 6-13. Prueba petición SET al objeto tempMin.

6.2.10 humMax

Prueba	Petición GET al objeto humMax
Respuesta 	Sent GET request to 192.168.1.50 : 161 humMax.0 99

Tabla 6-14. Prueba petición GET al objeto humMax.

Prueba	Petición SET al objeto humMax
Respuesta 	Sent SET request to 192.168.1.50 : 161 **humMax.0** 75

Tabla 6-15. Prueba petición SET al objeto humMax.

6.2.11 humMin

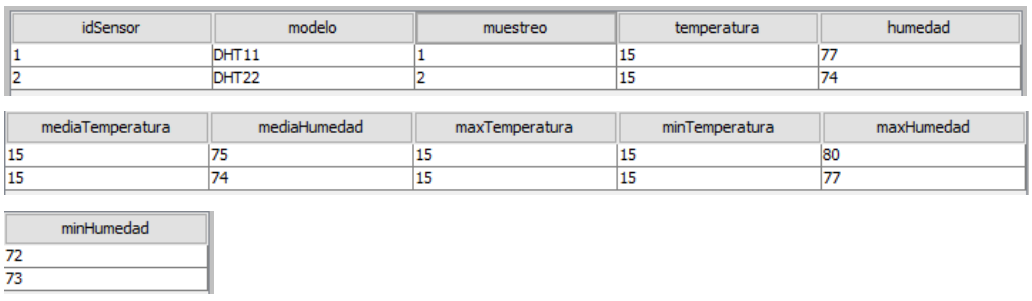
Prueba	Petición GET al objeto humMin
Respuesta 	Sent GET request to 192.168.1.50 : 161 **humMin.0** 0

Tabla 6-16. Prueba petición GET al objeto humMin.

Prueba	Petición SET al objeto humMin
Respuesta 	Sent SET request to 192.168.1.50 : 161 **humMin.0** 35

Tabla 6-17. Prueba petición SET al objeto humMin.

6.3. Objeto tabla

Prueba	Petición GET al objeto sensoresTable
Respuesta 	

idSensor	modelo	muestreo	temperatura	humedad
1	DHT11	1	15	77
2	DHT22	2	15	74

mediaTemperatura	mediaHumedad	maxTemperatura	minTemperatura	maxHumedad
15	75	15	15	80
15	74	15	15	77

minHumedad
72
73

Tabla 6-18. Prueba petición GET al objeto sensoresTable.

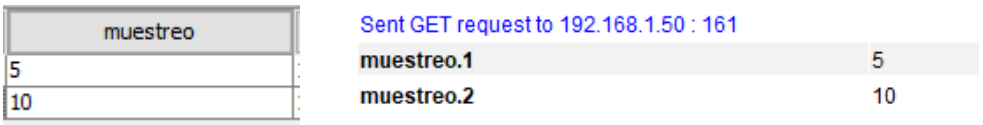
Prueba	Petición SET a la columna muestreo
Respuesta 	

Tabla 6-19. Prueba petición SET a la columna muestreo.

6.4. Traps

6.4.1 Con modoTraps igual a 0

6.4.1.1 tempMaxTrap

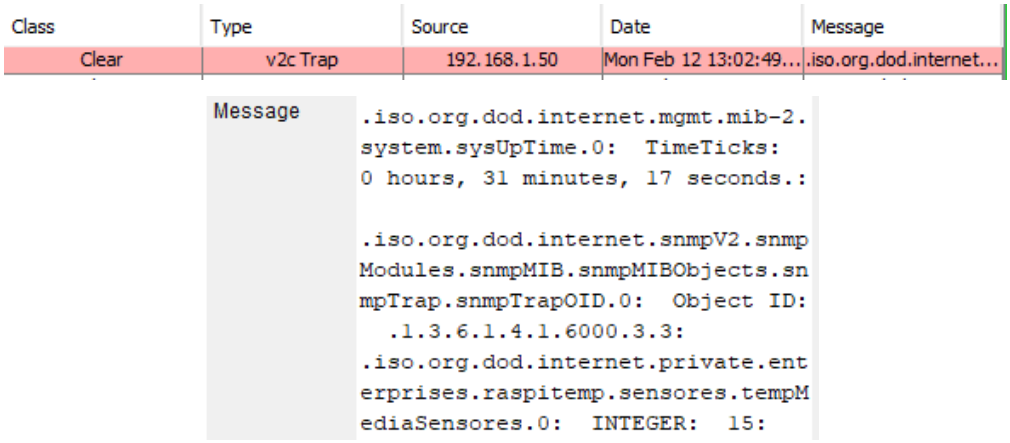
Prueba	Hacer que se envíe un trap cuando la temperatura supere el máximo establecido (14 ºC)
Respuesta 	

Tabla 6-20. Prueba trap tempMaxTrap con modoTraps igual a 0.

6.4.1.2 tempMinTrap

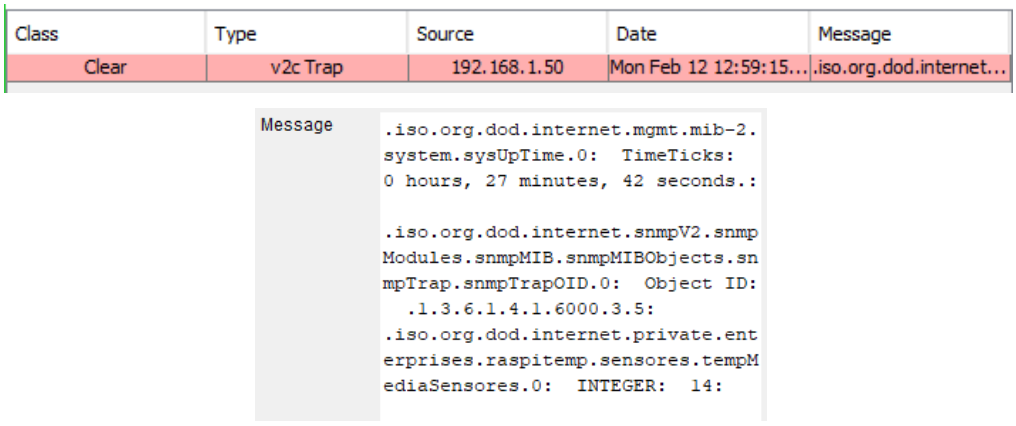
Prueba	Hacer que se envíe un trap cuando la temperatura no llegue al mínimo establecido (16 ºC)
Respuesta 	

Tabla 6-21. Prueba trap tempMinTrap con modoTraps igual a 0.

6.4.1.3 humMaxTrap

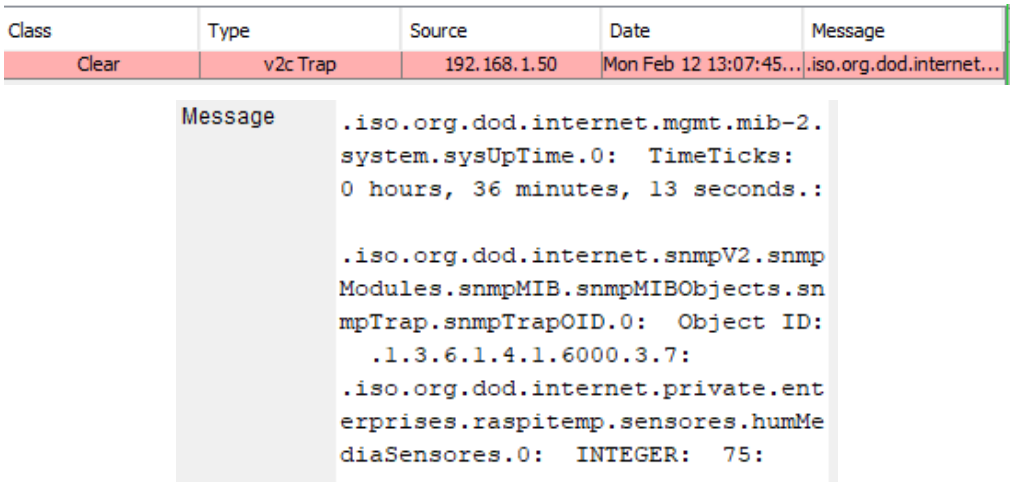
Prueba	Hacer que se envíe un trap cuando la humedad supere el máximo establecido (74 %)
Respuesta 	

Tabla 6-22. Prueba trap humMaxTrap con modoTraps igual a 0.

6.4.1.4 humMinTrap

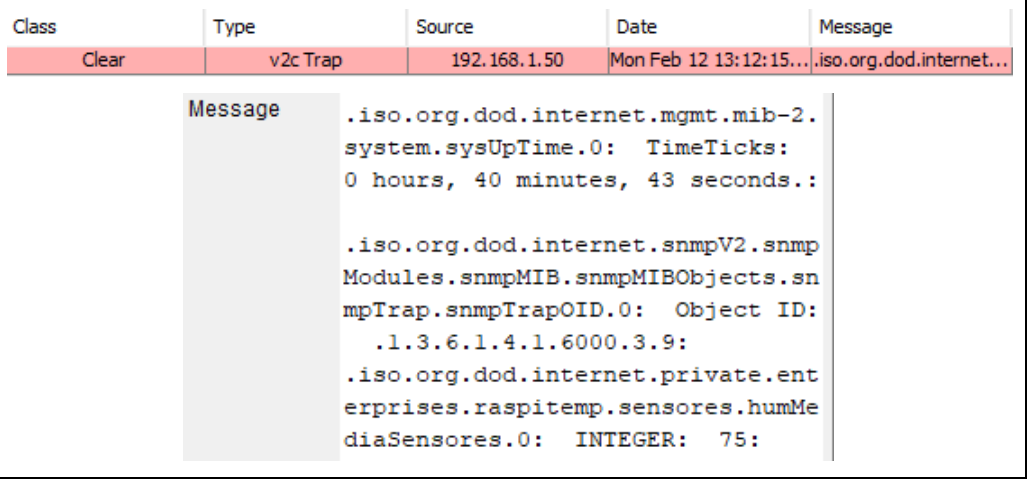
Prueba	Hacer que se envíe un trap cuando la humedad no llegue al mínimo establecido (76 %)
Respuesta 	

Tabla 6-23. Prueba trap humMinTrap con modoTraps igual a 0.

6.4.2 Con modoTraps igual a 1

6.4.2.1 tempMaxTrap

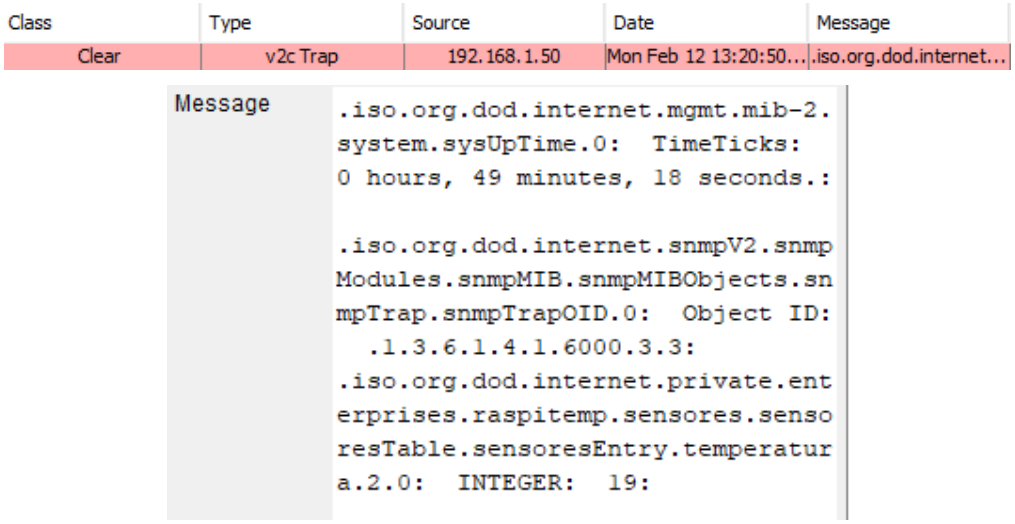
Prueba	Hacer que se envíe un trap cuando la temperatura supere el máximo establecido (15 ºC)
Respuesta 	

Tabla 6-24. Prueba trap tempMaxTrap con modosTraps igual a 1.

6.4.2.2 tempMinTrap

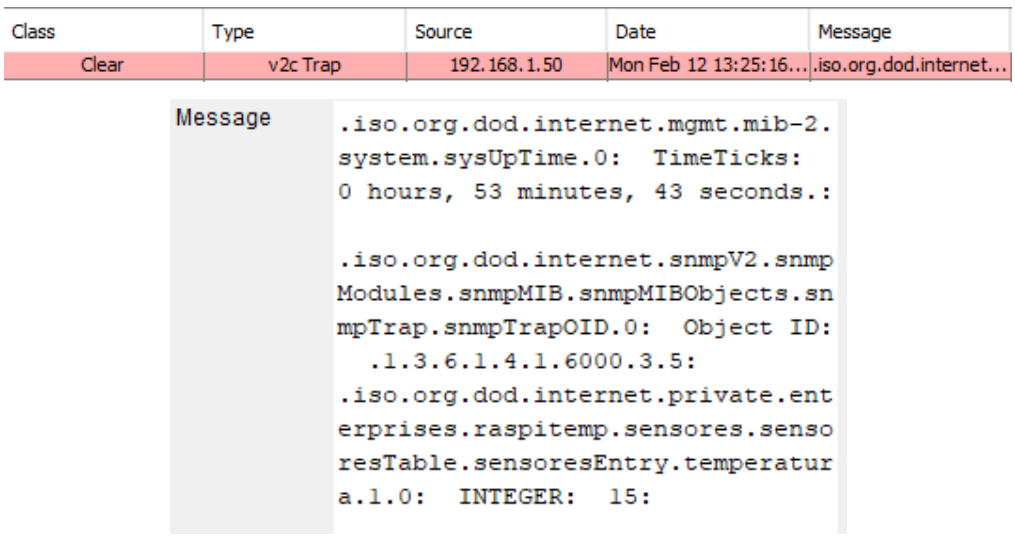
Prueba	Hacer que se envíe un trap cuando la temperatura no llegue al mínimo establecido (16 ºC)
Respuesta 	

Tabla 6-25. Prueba trap tempMinTrap con modosTraps igual a 1.

6.4.2.3 humMaxTrap

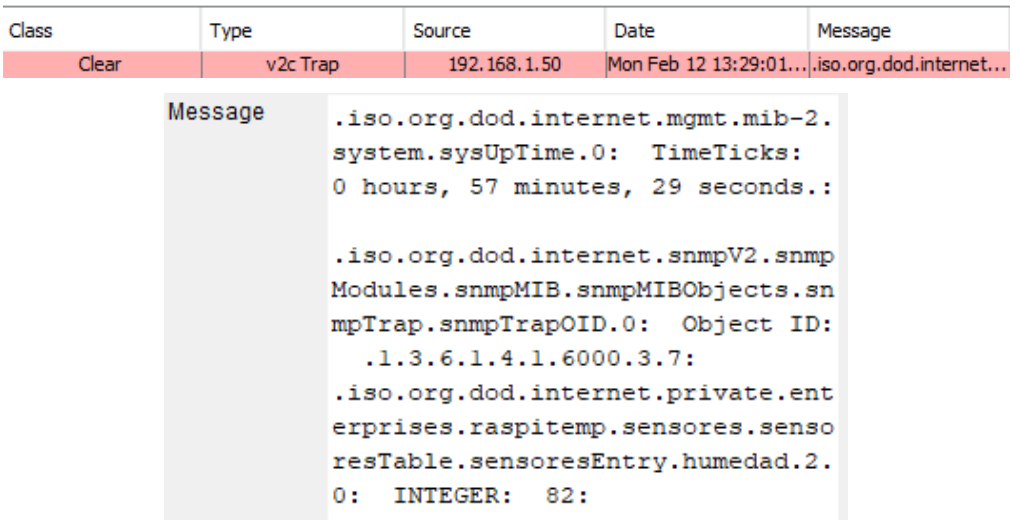
Prueba	Hacer que se envíe un trap cuando la humedad supere el máximo establecido (75 %)
Respuesta 	

Tabla 6-26. Prueba trap humMaxTrap con modosTraps igual a 1.

6.4.2.4 humMinTrap

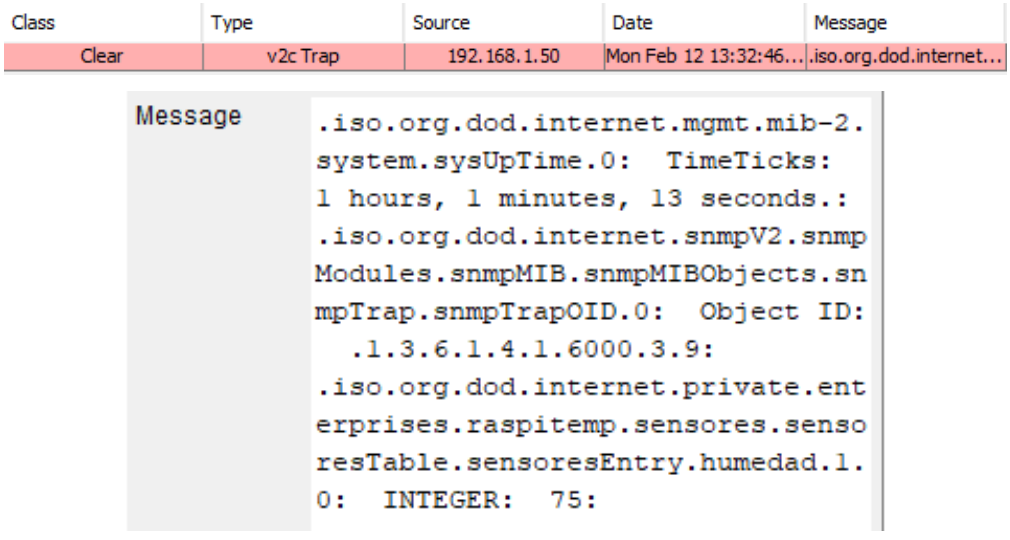
Prueba	Hacer que se envíe un trap cuando la humedad no llegue al mínimo establecido (80 %)
Respuesta 	

Tabla 6-27. Prueba trap humMinTrap con modosTraps igual a 1.

6 CONCLUSIONES Y LÍNEAS DE CONTINUACIÓN

En este trabajo se ha conseguido implementar un agente SNMP extensible en una Raspberry Pi, utilizando para ello software de código abierto y hardware de bajo coste. Como resultado, hemos obtenido un agente autónomo que responde a peticiones SNMP y envía traps.

Para ello se ha utilizado el protocolo AgentX de Net-SNMP, el cual nos ha permitido crear subagentes adaptados a nuestro trabajo partiendo de objetos genéricos para cualquier dispositivo y en el que se pueden añadir todos los objetos que se deseen gestionar. Por ello, en este trabajo pueden añadirse más sensores; sólo habría que añadir el correspondiente código de obtención de sus medidas y adaptar el código existente a más sensores.

Para poner en funcionamiento el agente, se han creado scripts que facilitan la tarea, compilando y ejecutando los programas de los sensores y de los demonios de los subagentes.

En el desarrollo del trabajo se han encontrado algunas dificultades. La mayoría se ha debido a la escasa información para rellenar las plantillas de código obtenidas con la herramienta mib2c. Además, algunos de esos códigos generados tenían errores, los cuales pudieron ser solucionados gracias a mensajes del Mailing Lists de desarrolladores de Net-SNMP [14].

Con este trabajo se ha podido comprobar la potencia de una Raspberry Pi 3, la cual es capaz de ejecutar muchos demonios de forma paralela que permiten gestionar varios sensores conectados a ella. Este trabajo puede ampliarse para gestionar sensores de diferentes tipos para, por ejemplo, monitorizar un invernadero, en el que se controle la temperatura, la humedad, la cantidad de CO_2 y la luz para un correcto cultivo de las plantas. Además, se puede crear un bot en Telegram para que el agente envíe traps directamente al móvil del gestor en el caso en el que se superen los límites establecidos.

REFERENCIAS

[1] Página oficial del proyecto Raspberry Pi. [En línea]. Disponible en: https://www.raspberrypi.org/

[2] RFC 2741 - Agent Extensibility (AgentX) Protocol. [En línea]. Disponible en: https://www.ietf.org/rfc/rfc2741.txt

[3] Net-SNMP. [En línea]. Disponible en: http://www.net-snmp.org/

[4] PySNMP. [En línea]. Disponible en: http://pysnmp.sourceforge.net/

[5] SNMP4J. [En línea]. Disponible en: https://www.snmp4j.org/

[6] Larmouth, J. *ASN.1 Complete.* Open Systems Solutions, 1999.

[7] Structure of Management Information Version 2 (SMIv2). [En línea]. Disponible en: https://tools.ietf.org/html/rfc2578

[8] The SimpleWeb - MIB module validation. [En línea]. Disponible en: https://www.simpleweb.org/ietf/mibs/validate/

[9] GPIO: Raspberry Pi Models A and B. [En línea]. Disponible en: https://www.raspberrypi.org/documentation/usage/gpio/

[10] Funcionamiento sensor DHT11. [En línea]. Disponible en: http://www.uugear.com/portfolio/dht11-humidity-temperature-sensor-module/

[11] Manual de mib2c. [En línea]. Disponible en: http://www.net-snmp.org/docs/man/mib2c.html

[12] NET-SNMP Tutorial -- Demon. [En línea]. Disponible en: http://www.net-snmp.org/tutorial/tutorial-5/toolkit/demon/

[13] Free SNMP MIB Browser. [En línea]. Disponible en: https://www.manageengine.com/products/mibbrowser-free-tool/

[14] Mailing Lists de Net-SNMP. [En línea]. Disponible en: https://sourceforge.net/p/net-snmp/mailman/message/28160638/

ANEXOS

Anexo A: TEMPHUM-MIB

```
TEMPHUM-MIB DEFINITIONS ::= BEGIN

IMPORTS
          OBJECT-TYPE, MODULE-IDENTITY, NOTIFICATION-TYPE, Integer32,
          enterprises FROM SNMPv2-SMI
          OBJECT-GROUP FROM SNMPv2-CONF;

raspitemp MODULE-IDENTITY
          LAST-UPDATED "201802211750Z"
          ORGANIZATION "US"
          CONTACT-INFO "Desiree Garcia Soriano
                        desireegarcia6@gmail.com
                        Sevilla, Espana"
          DESCRIPTION "MIB que recoge los OIDs necesarios para la gestion de
                       sensores de temperatura y humedad"
          REVISION "201802211750Z"
                  DESCRIPTION "Quinta version: modificaciones en algunos objetos"
          REVISION "201711061720Z"
                  DESCRIPTION "Cuarta version: pequenas mejoras"
          REVISION "201710231830Z"
                  DESCRIPTION "Tercera version: traps anadidos"
          REVISION "201710151330Z"
                  DESCRIPTION "Segunda version: tabla anadida"
          REVISION "201709301430Z"
                  DESCRIPTION "Primera version: objetos escalares basicos"
          ::= { enterprises 6000 }

-- grupos en TEMPHUM-MIB
general OBJECT IDENTIFIER ::= { raspitemp 1 }
sensores OBJECT IDENTIFIER ::= { raspitemp 2 }
sensoresTraps OBJECT IDENTIFIER ::= { raspitemp 3 }
```

```
-- grupo general
infoAgente OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Informacion relativa al agente"
        ::= { general 1 }

localizacion OBJECT-TYPE
        SYNTAX OCTET STRING (SIZE(0..255))
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          "Lugar donde se encuentra el agente"
        ::= { general 2 }

pines OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Pines en los que estan conectados los sensores"
        ::= { general 3 }


-- grupo sensores
numSensores OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Numero de sensores"
        ::= { sensores 1 }

tempMediaSensores OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
```

```
DESCRIPTION
          "Media de la temperatura medida por los sensores en grados centigrados"
        ::= { sensores 2 }

humMediaSensores OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Media de la humedad medida por los sensores en tanto por ciento"
        ::= { sensores 3 }

-- tablaSensores
sensoresTable OBJECT-TYPE
        SYNTAX SEQUENCE OF SensoresEntry
        MAX-ACCESS not-accessible
        STATUS current
        DESCRIPTION
          "Tabla que almacena informacion de los parametros medidos por los
          sensores"
        ::= { sensores 4 }

SensoresEntry ::=
        SEQUENCE {
              idSensor Integer32,
              modelo OCTET STRING,
              muestreo Integer32,
              temperatura Integer32,
              humedad Integer32,
              mediaTemperatura Integer32,
              mediaHumedad Integer32,
              maxTemperatura Integer32,
              minTemperatura Integer32,
              maxHumedad Integer32,
              minHumedad Integer32
        }

sensoresEntry OBJECT-TYPE
        SYNTAX SensoresEntry
        MAX-ACCESS not-accessible
```

```
        STATUS current
        DESCRIPTION
          "Informacion de los parametros de un sensor concreto, el cual debe ser
          distinguible por un identificador"
        INDEX { idSensor }
        ::= { sensoresTable 1 }

idSensor OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Identificador del sensor que lo diferencia de cualquier otro"
        ::= { sensoresEntry 1 }

modelo OBJECT-TYPE
        SYNTAX OCTET STRING
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Modelo del sensor"
        ::= { sensoresEntry 2 }

muestreo OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          "Tiempo de muestreo del sensor en segundos"
        ::= { sensoresEntry 3 }

temperatura OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Temperatura medida por el sensor en grados centigrados"
        ::= { sensoresEntry 4 }
```

```
humedad OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Humedad medida por el sensor en tanto por ciento"
        ::= { sensoresEntry 5 }

mediaTemperatura OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Media de las ultimas 30 mediciones de la temperatura medida por el
          sensor en grados centigrados"
        ::= { sensoresEntry 6 }

mediaHumedad OBJECT-TYPE
        SYNTAX Integer32
        UNITS "Tanto por ciento"
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Media de las ultimas 30 mediciones de la humedad medida por el
          sensor en tanto por ciento"
        ::= { sensoresEntry 7 }

maxTemperatura OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Maxima temperatura medida por el sensor en grados centigrados"
        ::= { sensoresEntry 8 }

minTemperatura OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
```

```
        DESCRIPTION
          "Minima temperatura medida por el sensor en grados centigrados"
        ::= { sensoresEntry 9 }

maxHumedad OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Maxima humedad medida por el sensor en tanto por ciento"
        ::= { sensoresEntry 10 }

minHumedad OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-only
        STATUS current
        DESCRIPTION
          "Minima humedad medida por el sensor en tanto por ciento"
        ::= { sensoresEntry 11 }


-- grupo sensoresTraps
modoTraps OBJECT-TYPE
        SYNTAX Integer32 (0..1)
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          "Parametro que vale 0 para que los traps funcionen teniendo en
          cuenta la media de los sensores y que vale 1 para que funcionen
          teniendo en cuenta las medidas individuales de cada sensor. En este
          ultimo caso, se enviara la informacion del sensor que haya obtenido la
          mayor/menor medida"
        ::= { sensoresTraps 1 }

tempMax OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-write
        STATUS current
```

```
        DESCRIPTION
          "Limite maximo de temperatura permitido en grados centigrados"
        ::= { sensoresTraps 2 }

tempMaxTrap NOTIFICATION-TYPE
        OBJECTS { temperatura }
        STATUS current
        DESCRIPTION
          "Trap que se envia cada 5 segundos cuando la temperatura sobrepasa el
          limite maximo establecido"
        ::= { sensoresTraps 3 }

tempMin OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          "Limite minimo de temperatura permitido en grados centigrados"
        ::= { sensoresTraps 4 }

tempMinTrap NOTIFICATION-TYPE
        OBJECTS { temperatura }
        STATUS current
        DESCRIPTION
          "Trap que se envia cada 5 segundos cuando la temperatura no alcanza el
          limite minimo establecido"
        ::= { sensoresTraps 5 }

humMax OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          "Limite maximo de humedad permitido en tanto por ciento"
        ::= { sensoresTraps 6 }

humMaxTrap NOTIFICATION-TYPE
        OBJECTS { humedad }
        STATUS current
```

```
        DESCRIPTION
          "Trap que se envia cada 5 segundos cuando la humedad sobrepasa el
          limite maximo establecido"
        ::= { sensoresTraps 7 }

humMin OBJECT-TYPE
        SYNTAX Integer32
        MAX-ACCESS read-write
        STATUS current
        DESCRIPTION
          "Limite minimo de humedad permitido en tanto por ciento"
        ::= { sensoresTraps 8 }

humMinTrap NOTIFICATION-TYPE
        OBJECTS { humedad }
        STATUS current
        DESCRIPTION
          "Trap que se envia  cada 5 segundos cuando la humedad no alcanza el
          limite minimo establecido"
        ::= { sensoresTraps 9 }

END
```