ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

Implementación de un agente extensible SNMP sobre la plataforma Raspberry Pi para la simulación de MIB propietarias

(Implementation of an extensible SNMP agent on the Raspberry Pi Platform for the simulation of proprietary MIBs)

Para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Autor: Adrián García Estébanez

Julio - 2015

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Adrián García Estébanez
Director del PFC: José Ángel Irastorza Teja
Título: "Implementación de un agente extensible SNMP sobre la plataforma Raspberry Pi para la simulación de MIB propietarias"
Title: "Implementation of an extensible SNMP agent on the Raspberry Pi Platform for the simulation of proprietary MIBs"

Presentado a examen el día: 30 de Julio de 2015

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Alberto E. García Gutierrez
Secretario (Apellidos, Nombre): Jose Angel Irastorza Teja
Vocal (Apellidos, Nombre): Marta García Arranz

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

Resumen

En este proyecto se describe el diseño e implementación de un agente de gestión SNMP extensible y autónomo sobre un computador de placa reducida Raspberry Pi. El agente se ha implementado utilizando el software Net-SNMP, siendo posible su extensión gracias al estándar AgentX. Para conseguirlo, se han desarrollado los programas necesarios para que el dispositivo soporte todas las funciones que definen un agente extensible autónomo, así como aquellos que simulan los escenarios de prueba del funcionamiento correcto de una plataforma de gestión SNMP completa en la que el dispositivo actúe como agente. Todo el código fuente del proyecto es abierto, ya que deriva de licencias BSD y GNU. Además, existe la intención de utilizar el dispositivo con fines docentes, en concreto en las prácticas de la asignatura del Grado en Ingeniería de Tecnologías de Telecomunicación de la Universidad de Cantabria *Gestión y Operación de Redes*; por tanto, su funcionamiento correcto se ha comprobado en el entorno del laboratorio donde se realizarán dichas prácticas.

Palabras clave: SNMP, AgentX, agente extensible, plataforma de gestión, Raspberry Pi, Net-SNMP, *open-source*.

Summary

This draft describes the design and implementation of an autonomous SNMP extensible agent over a SBC (Single Board Computer) Raspberry Pi. The agent has been implemented using the SNMP software suite Net-SNMP, while its extension capabilities rely on AgentX standard. In order to achieve these goals, the necessary programs for the device to support required SNMP extensible agent functions have been developed; and also those who simulate the test scenarios used to prove the correct operation of the autonomous agent within a full SNMP management platform. This draft code is completely open, as it derives from BSD and GNU licenses. In addition, it´s planned to use the device with educational purposes, specifically during the course *Gestión y Operación de Redes,* included in *Grado en Ingeniería de Tecnologías de Telecomunicación* (*Universidad de Cantabria*) syllabus; so its operation has also been tested in the enviroment where these course´s classes will be held.

Keywords: SNMP, AgentX, extensible agent, management plataform, Raspberry Pi, Net-SNMP, *open-source*.

Índice general

Índice de figuras

1. Introducción

En este primer capítulo se tratan las cuestiones preliminares relativas al proyecto: las motivaciones que han dado lugar a su realización, los objetivos planteados y, finalmente, un resumen breve explicando la estructura de esta memoria.

Con el comienzo de la era de internet, a principios de los años 90, se observó la necesidad imperiosa de gestionar los recursos de una red que crecía a gran velocidad y de manera relativamente caótica por estar descentralizada y apoyada en recursos de red, hardware y software con fiabilidades y capacidades muy diversas. Comenzó por tanto la creación de estándares que cubrieran esta necesidad; en concreto, se desarrollaron dos de forma paralela:

- Un estándar de gestión OSI (*Open System Interconnection, ISO/IEC 7498-1*) propuesto por la ITU-T y basado principalmente en dos recomendaciones: CMIP (recomendación X.711) y CMIS (recomendación X.710). Partiendo de estas recomendaciones ITU-T definió el estándar TMN, que proporciona las herramientas necesarias para incluir redes y sistemas heterogéneos en un mismo ecosistema de gestión.
- Un estándar de gestión Internet, basado en el protocolo SNMP, desarrollado por la IETF (*Internet Engineering Task Force*). SNMP se encarga de la gestión de Internet, por lo tanto es más concreto que los estándares OSI.

Los estándares anteriores se apoyan en el paradigma gestor-agente, que describe los papeles de dos actores principales en el escenario de gestión. El gestor está compuesto por los elementos de la red o sistema con los que interacciona el operador humano, y se comunica con los agentes para ordenarles que lleven a cabo las acciones requeridas. Los agentes se encargan de llevar a cabo dichas acciones.

Además, existe otra funcionalidad posteriormente añadida a la gestión de Internet sobre el estándar SNMP: los agentes extendidos. Se trata de una evolución lógica del estándar, que permite añadir subagentes a un agente principal. Estos subagentes permiten la modularidad del agente y un reparto más efectivo de tareas dentro del mismo, así como una mayor facilidad y velocidad de desarrollo de la plataforma de gestión. El estándar principal que cubre este tipo de subagentes es AgentX, desarrollado por la IETF en el documento RFC 2741.

1.2. Motivación

Las funciones proporcionadas por la gestión de red y los agentes extendidos son realmente interesantes para el control distribuido de cualquier tipo de sistema implementado en una red de comunicaciones, ya que posibilitan la gestión distribuida y remota de cualquier dispositivo conectado a dicha red utilizando las infraestructuras de la misma. Por eso la motivación de este proyecto va más allá de la gestión de internet, o de redes comerciales o institucionales, y se basa en la posibilidad de usar esta tecnología en proyectos educativos, personales y de pequeñas empresas. Para ello se necesita una plataforma que destaque por su bajo coste, su

facilidad de mantenimiento y sobre la que sea posible desarrollar nuevas aplicaciones de la forma más sencilla y directa posible.

En este punto entra en escena la irrupción en el mercado de la más popular SBC (*Single Board Computer*) *open source* de bajo coste, Raspberry Pi. Como consecuencia de su éxito, se ha producido un gran auge de desarrollo de aplicaciones y proyectos informáticos y electrónicos de todo tipo sin necesidad de inversiones económicas significativas, lo cual ha creado una comunidad amplia de aficionados y desarrolladores. Además, es destacable el espíritu de conocimiento libre que rodea a esta comunidad, de forma que gran cantidad de desarrollos son expuestos al conocimiento público con diversos grados de detalle.

Esta aparición de computadores de bajo coste y la existencia de las tecnologías de gestión presentadas permite la creación de plataformas de gestión modulares, baratas y fácilmente extensibles. Este proyecto pretende aunar todo lo descrito hasta ahora y crear un dispositivo autónomo que actúe como agente extensible en cualquiera de estas redes.

Con más detalle, se busca la implementación de un agente SNMP extensible mediante el estándar AgentX, de forma que se soporte la gestión de cualquier dispositivo o conjunto de datos accesible desde una red. Es decir, se amplía la funcionalidad de gestión de red de SNMP a gestión de elementos conectados a la red, utilizando para ello el concepto de agente extensible. Este agente estará completamente implementado sobre un SBC *Rapberry Pi*.

1.3. Objetivos

Se plantearon los siguientes objetivos:

- Investigar las posibilidades existentes en cuanto a hardware y software para crear un agente extensible autónomo SNMP implementado sobre una Raspberry Pi. Elegir los elementos que más se ajusten a las necesidades del proyecto y del dispositivo.
- Diseñar el agente extensible, detallando el uso de los elementos hardware y software disponibles e identificando el software que se debe desarrollar para su implementación definitiva.
- Implementar el dispositivo, que debe ser completamente autónomo una vez iniciado.
- Implementar un prototipo de simulación de gestión de un entorno realista, de forma que sirva de muestra de las funcionalidades del dispostivo.
- Toda la plataforma será un sistema abierto, de forma que cualquier desarrollador o estudiante pueda acceder al software desarrollado en este proyecto y ampliarlo o mejorarlo.
- Se debe comprobar el funcionamiento del dispositivo final en el entorno del Laboratorio de Telemática de la ETSIIT de la Universidad de Cantabria, dado que se va a aplicar para uso educativo, concretamente en las prácticas de la asignatura *Gestión y Operación de Redes*, perteneciente al programa del Grado en

Ingeniería de Tecnologías de Telecomunicación.

1.4. Estructura de la memoria

Este documento está estructurado en cuatro bloques principales, organizados de la siguiente manera:

- Una parte teórica, cubierta en el capítulo 2. En ella se introducen y explican todos los conceptos necesarios para la realización del proyecto, principalmente los referentes a la gestión de redes SNMP y a los agentes SNMP extendidos.
- Una parte que cubre los aspectos prácticos, correspondiente al capítulo 3, en la que se tratan todos los aspectos relativos al dispositivo *Raspberry Pi* (hardware, software, configuración), a la aplicación *Net-Snmp* (que soporta la gestión SNMP) y a la creación de agentes extendidos mediante el estándar AgentX.
- Una parte de desarrollo, el capítulo 4, en la que se describe de forma detallada el proceso de implementación del manejo de varios objetos gestionados de ejemplo (escalares, tablas y *traps*) a través de subagentes extendidos.
- Una descripción del prototipo final del proyecto, en el capítulo 5, que describe la simulación de un entorno de gestión completo de un generador eléctrico diésel ficticio que alimenta una sala de servidores. Incluye las pruebas definitivas de su funcionamiento.

Para finalizar, se incluyen varios anexos, por si se desean consultar más en detalle ciertos aspectos del proyecto. El anexo A1 incluye la estructura detallada en forma de árbol de los objetos gestionados descritos en esta memoria; el anexo A2 contiene el código completo de la MIB que da lugar a la estructura de A1. El anexo A3 contiene el código completo del archivo de configuración del *demon* de *Net-SNMP*. En el anexo A4 se encuentra recogido el código completo de la MIB del prototipo, *GENSERV-MIB*.

2. Aspectos teóricos

En este capítulo se describen brevemente las bases teóricas sobre las cuales se desarrolla el proyecto. Se centra principalmente en el protocolo SNMP y los agentes extensibles AgentX.

2.2. Introducción a la gestión de redes

Este apartado resume los conceptos esenciales de la gestión de red y los estándares principales referidos a ella.

La gestión de red se puede definir como la operación, administración, mantenimiento y aprovisionamiento de servicios en una red de telecomunicaciones [1].

Como se ha introducido en el capítulo 1, destacan dos modelos de gestión principales: la gestión OSI y la gestión Internet. La gestión OSI es la más compleja y, al mismo tiempo, más completa. Concebida inicialmente para la gestión de las propias redes OSI, está definida en estándares de ITU-T e ISO (X.700/ISO 7498-4, X.701/ISO 10040, X.710/ISO 9595, X.711/ISO 9596, X.720/ISO 10165-1, X.721/ISO 10165-2, X.722/ISO 10165- 4, X.73x-4x/ISO 10164-x). Este tipo de gestión es amplio, y se define como gestión de sistemas; evolucionó para permitir realizar la gestión de entornos más heterogéneos, como el estándar TMN, que proporciona las herramientas necesarias para incluir redes y sistemas heterogéneos en un mismo ecosistema de gestión.

La gestión Internet, base de este proyecto, es un modelo más concreto de gestión centrado en redes TCP/IP. Se basa en varios *Request for Comments* (RFC) de la *Internet Engineering Task Force* (IETF), principalmente en los documentos 1157, 1213 y 1155. Se trata de un modelo más sencillo que el de gestión OSI, lo que ha posibilitado su gran implantación. Es además la base sobre la que se apoya este proyecto, por lo que en el apartado 2.1.3 se desarrolla más ampliamente.

2.2.1. Bases de la gestión de red

De forma más específica, la gestión de red engloba el **conjunto de tareas que cubre todas las precauciones y actividades que aseguran el uso eficiente y efectivo de los procesos y recursos distribuidos que forman una red de telecomunicaciones**.

Por su parte, una arquitectura de gestión describe un marco general de un modelo de gestión integrado en un entorno heterogéneo, como una red de telecomunicaciones cualquiera. La arquitectura de gestión arquetípica considera varios submodelos: el modelo de información, el modelo organizativo, el modelo de comunicación y el modelo funcional.

El **modelo de información** engloba todos los datos e información que facilitan el uso y funcionamiento de una red de comunicaciones. El almacén conceptual donde está recogida esta información se denomina

Management Information Base (**MIB**), y está constituido por los objetos gestionados (MO o *Managed Objects*), que representan los recursos que deben gestionarse. Los objetos gestionados tienen definidas varias propiedades, como su identificación, comportamiento, relaciones, etc.

El **modelo organizativo** establece los papeles o funciones dentro del sistema de gestión y su distribución espacial. Como se verá más adelante, estos papeles son principalmente el de gestor y agente; o el de agente proxy, que se encarga de acceder a dispositivos propietarios mediante la traducción de protocolos.

El **modelo de comunicación** define los esquemas para el intercambio de información entre los diferentes componentes dentro del sistema de gestión. Cubre aspectos como la especificación de los elementos que intervienen en la comunicación, los protocolos y servicios que se ocupan de la aplicación de la gestión o la sintaxis y semántica de los datos de la comunicación.

El **modelo funcional** divide las tareas de gestión en un conjunto de áreas funcionales: configuración, fallos, contabilidad, etc.

2.2.2. El paradigma gestor-agente

Un concepto de gran importancia es la pareja gestor-agente. En la mayoría de sistemas de gestión, incluido el que se implementa en este proyecto, se consideran dos componentes principales: el gestor y el agente.

El **gestor** es la entidad que se encarga de dar las órdenes y realizar las peticiones necesarias en el entorno de gestión, y con él interaccionan los operadores humanos. El **agente** es la entidad que se encarga de cumplir dichas órdenes y peticiones y de manejar la información de gestión de los recursos de la red (los objetos gestionados). En la figura 2.1 se presenta un esquema simplificado de esta asociación.

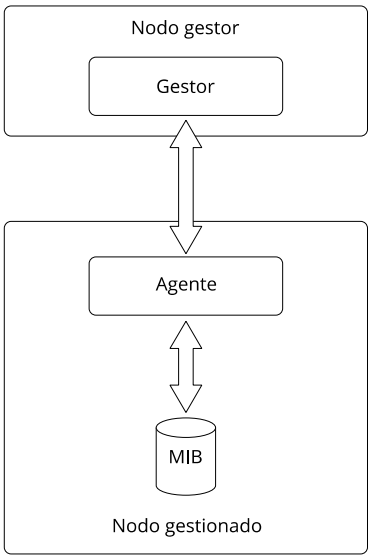


Figura 2.1. Esquema de la dupla gestor-agente

Se definen como **nodos gestores** a los nodos de una red en los que existe un gestor; los **nodos gestionados**,

por su parte, son aquellos en los que existe un agente. En estos últimos, el agente controla la información de todos sus recursos asociados (objetos gestionados, contenidos en la MIB), de forma que el gestor puede pedirla en cualquier momento, monitorizarla y ordenar que se modifique.

Además, el agente envía notificaciones en caso de algún evento especial relacionado con uno de sus objetos gestionados. Estas notificaciones suelen ser mensajes asíncronos que sirven para avisar al gestor de algún hecho de importancia.

2.2.3. Gestión Internet

En este apartado se tratarán los aspectos principales de la gestión de internet, que es el modelo de gestión que se implementa en la plataforma desarrollada en este proyecto. La gestión de las redes TCP/IP sobre las que se asienta internet se basa en dos modelos: SNMP, que es el protocolo de gestión que define el modelo de comunicación (basado en los RFC 1157 [2], 1441 [3], y 1452 [4]) y MIB II, que define el modelo de información (RFC 1213 [5]); este modelo de información se completa con el estándar SMI (*Structure and Identification of Management Information for TCP/IP-based Internets*), que define el formato de las estructuras de información de gestión internet (RFC 1155 [6]).

2.2.3.1. Estándar SNMP

El estándar SNMP (*Simple Network Management Protocol*) fue diseñado para proporcionar los cimientos de la gestión de *routers*, servidores, estaciones de trabajo y otros recursos de red. Se desarrolló con dos requisitos en mente: facilidad de implementación y baja ocupación de los recursos de la red.

La especificación de SNMP define un protocolo cuya función es intercambiar información de gestión entre los dispositivos de una red TCP/IP (corresponde al modelo de comunicación), una infraestructura para formatear y almacenar la información de gestión (modelo de información) y una serie de variables (objetos gestionados) de propósito general [8].

SNMP considera un conjunto de estaciones de gestión de red y de elementos de red. Las estaciones de gestión de red ejecutan aplicaciones de gestión que controlan y monitorizan elementos de red (dispositivos como *hosts*, *gateways*, *switches*, *servidores*, etc.). Estos elementos tienen instalados agentes de gestión que son responsables de realizar las funciones de gestión de red requeridas por las estaciones de gestión de red. El protocolo SNMP (*Simple Network Management Protocol*) comunica la información de gestión entre las estaciones de gestión de red y los agentes incluidos en los elementos de la red [2].

Se deduce que la dupla *gestor-agente* que se ha presentado en el apartado 2.1.2 corresponde a la dupla de SNMP *estación de gestión-agente* [7]. De aquí en adelante se usarán indistintamente ambas denominaciones.

SNMP utiliza UDP (*User Datagram Protocol*) como protocolo de transporte para intercambiar datos entre agentes y gestores (en la figura 2.2). UDP, definido en el RFC 768, fue elegido frente a TCP (*Transmission Control Protocol*) porque no está orientado a conexión; es decir, no se establece una conexión punto a punto entre el agente y el gestor. Este aspecto hace que UDP no sea fiable, ya que no existen confirmaciones (*acknowledgements*) al envío de mensajes y estos se pueden perder sin que ni el agente o el gestor lo sepan. Por ello, la aplicación SNMP debe vigilar estás pérdidas mediante reconocimientos a nivel de aplicación y realizar retransmisiones si se producen [7].

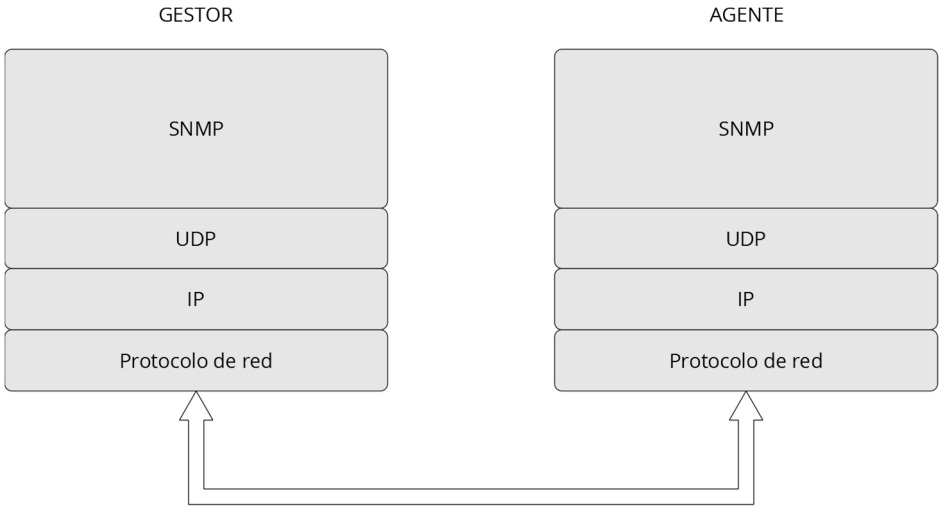


Figura 2.2. SNMP sobre el modelo de comunicación TCP/IP

La principal razón por la que se eligió UDP frente a TCP fue porque su impacto sobre el rendimiento de la red es mucho menor. En una red con alto volumen de tráfico y tendencia a congestionarse, SNMP sobre TCP es una mala idea, ya que contribuye a aumentar aún más la congestión, sobre todo teniendo en cuenta que su uso principal es gestionar los recursos de la propia red y en momentos de apuro los mensajes de gestión aumentan [7]. SNMP usa por defecto el puerto UDP 162 para el envío de *traps* y el 161 para el envío del resto de mensajes.

Los mensajes SNMP contienen tres campos principales: una identificación de la versión del protocolo utilizada (v1, v2c o v3), la comunidad y la PDU. La versión de SNMP utilizada a lo largo del desarrollo del proyecto es en todo momento v2c, ya que v1 ha quedado obsoleta y v3 incluye una seguridad que no se considera necesaria en el marco de este proyecto, orientado a la docencia. En la figura 2.3 se muestra el formato básico de un mensaje SNMP.

En **el campo *community*** (comunidad) se define la política de acceso a los objetos gestionados a los que se refiere el mensaje SNMP. SNMPv2c usa la noción de comunidades para establecer la relación de confianza entre gestores y agentes. Un agente siempre es configurado con tres nombres (*strings*) de comunidad: *read-only, read-write*, y *trap*. Los *strings* de comunidad actúan básicamente como contraseñas, y controlan diferentes tipos de actividad. El *string* de comunidad *read-only* permite leer los datos de los objetos gestionados; la comunidad *read-write* permite leer y escribir sobre los objetos gestionados. Finalmente, el string de comunidad *trap* permite recibir *traps* desde el agente. Existen *strings* de comunidad por defecto: *public* para

read-only y *private* para *read-write* [7]. Cuando una red es pública, es aconsejable cambiar estos *strings*, ya que actúan como contraseñas, pero en este proyecto se usarán los nombres de comunidad por defecto, ya que no se gestiona ninguna red real y tiene carácter educativo.

Por su parte, **el campo PDU** (*Protocol Data Unit*) contiene las operaciones SNMP. La comunicación SNMP entre el agente y el gestor es bidireccional, y sus PDUs se engloban en tres grupos: petición de información al agente por parte del gestor, modificación de información de un objeto gestionado por parte del gestor y envío de notificaciones desde el agente hacia el gestor.

Se definen las siguientes PDUs:

- *GetRequest*: Es una petición iniciada por el gestor, que se la envía al agente. Éste la recibe y la procesa si es posible, en cuyo caso envía una primitiva *GetResponse* al gestor, con los datos pedidos. Si el agente sostiene una carga excesiva y no puede devolver una respuesta, simplemente se descarta el mensaje [7]. En la figura 2.3 [8] se muestra su formato. El campo *request-id* corresponde al identificador de la petición, y se utiliza para relacionar peticiones y respuestas. El campo *variable-bindings* contiene una lista de variables junto a sus valores asociados.

- *GetNextRequest*: Obtiene el valor del objeto siguiente al objeto referenciado que tenga un valor asociado, según el orden lexicográfico (ver apartado 2.1.3.3). Por lo demás, es idéntica a la primitiva *GetRequest*. En la figura 2.3 se puede ver su formato.

- *GetBulkRequest*: Obtiene el valor varios objetos y sirve para recorrer objetos gestionados heterogéneos. El propósito de esta PDU es pedir la transmisión de cantidades de datos potencialmente grandes, incluyendo la obtención rápida de tablas [19]. De nuevo, en la figura 2.3 se puede observar su formato. En el campo *variable-bindings* están contenidas las variables (nombre y valor); el campo *non-repeaters* especifica el número de variables que se deben obtener de forma individual, mientras que el campo *max-repetitions* indica el número de sucesores lexicográficos (ver apartado 2.1.3.3) del resto de variables. Si L es el número de variables totales, N el número de variables que se deben obtener de forma individual (*non-repeaters*) y R las variables de las que se tienen que obtener sucesores lexicográficos, se obtienen *N+R*(max-repetitions)* variables como máximo. Hay ocasiones en las que el campo *max-repetitions* es mayor que el número de sucesores lexicográficos, por lo que se obtiene un número menor al indicado por la expresión anterior; en este caso, el agente devuelve el valor *endOfMibView* con el nombre del último sucesor encontrado. En resumen, GetBulkRequest pide una serie de variables, las primeras de las cuales contenidas en variable-bindings son únicas, mientras que el resto sirven de punto de partida para acceder a una serie de datos ordenados según el orden lexicográfico (ver apartado 2.1.3.3).

- *GetResponse*: Confirmación a la recepción de las primitivas *GetRequest*, *GetNextRequest*, *GetBulkRequest* y *SetRequest*, acompañada del valor de las variables y errores si los hubiera (ver figura 2.3). Las variables y sus valores asociados, como en los casos anteriores, están contenidos en *variable-bindings*, mientras que los campos *error-status* y *error-index* se refieren a los errores que se hayan podido producir (*error-status*

identifica el tipo de error y *error-index* proporciona información adicional respecto a dicho error).

- *SetRequest:* El commando *set* se utiliza para cambiar el valor de un objeto gestionado o para crear una nueva fila en una tabla. Como *GetRequest*, es una petición iniciada por el gestor, que se la envía al agente. Éste la recibe y la procesa si es posible, en cuyo caso asigna el valor indicado al objeto referenciado y envía un mensaje al gestor que indica que no se han producido errores. Si éstos se producen, devuelve un *GetResponse* con información sobre los errores. Su formato se refleja en la figura 2.3; se puede comprobar que es análogo al de las primitivas *GetRequest* y *GetNextRequest*.
- *Trap:* El agente envía una notificación asíncrona al gestor, relacionada con un evento programado en el agente. Cuando las condiciones de este evento se cumplen, se envía esta primitiva hacia el puerto 162 del gestor. Los *traps* de la versión v2c de SNMP tienen un formato idéntico al de las primitivas *GetRequest*, *GetnextRequest* y *SetRequest*, como se muestra en la figura 2.3.

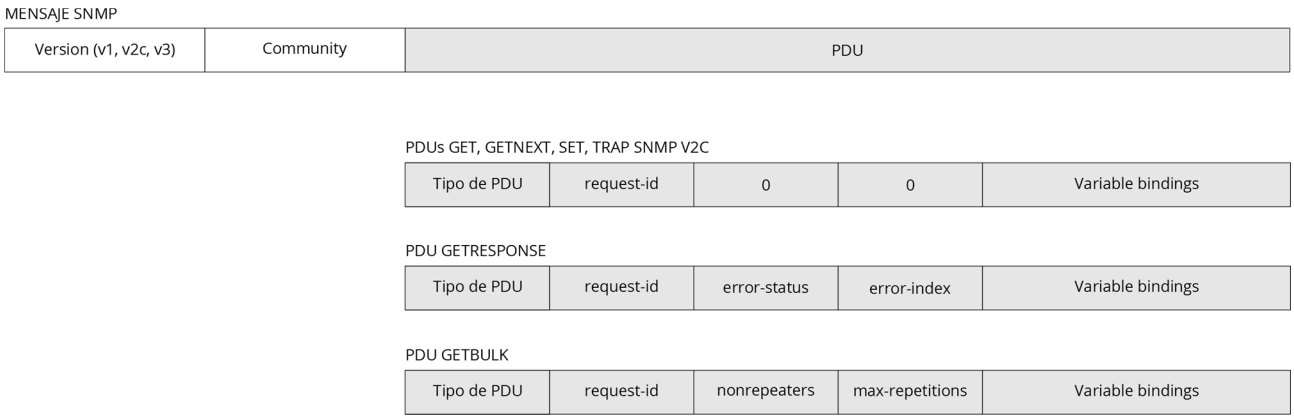


Figura 2.3. Formato de los mensajes y PDUs SNMP

En la figura 2.3 se muestran los formatos de un mensaje genérico SNMP y de las PDUs que puede contener, tal y como se ha explicado anteriormente. Se ha de tener en cuenta que cualquier implementación válida de agente o gestor SNMP v2c debe soportar todos los mensajes que se han descrito en este apartado.

2.2.3.2. Modelo de información: SMI y MIB II (MIB internet)

El modelo de información de la gestión internet está definido en el estándar MIB II (RFC 1213 [5]), que recoge los grupos de gestión internet. A su vez, este estándar utiliza las estructuras definidas en estándar SMI (*Structure and Identification of Management Information for TCP/IP-based Internets*) (RFC 1155 [6]) para definir cada uno de los objetos gestionados.

El estándar SMI describe las estructuras comunes y los esquemas de identificación de la información de gestión de redes TCP/IP. Define además tipos genéricos de objetos usados para describir información de gestión, basándose en el estándar ASN.1 [9].

La definición de objetos gestionados puede ser reducida a tres atributos: el nombre, el tipo/sintaxis y la

codificación.

El primero, el nombre u OID (*object identifier*), define de forma única un objeto gestionado. Los nombres aparecen en dos formatos: uno numérico y otro textual.

El segundo, el tipo/sintaxis, especifica cómo son representados y transmitidos los datos entre gestores y agentes dentro del contexto de SNMP. Las descripciones de los objetos dentro de la MIB se realizan usando la sintaxis ASN.1, que es una notación independiente del sistema.

El tercero es la codificación. Cada instancia de un objeto SNMP está codificada en un string de octetos utilizando las reglas básicas de codificación BER (*Basic Encoding Rules*). BER define cómo los objetos son codificados y decodificados de forma que puedan ser transmitidos sobre un medio (como, por ejemplo, Ethernet) [7].

Los objetos gestionados están organizados en una estructura de árbol. Esta estructura supone la base de las reglas utilizadas para nombrar a los objetos gestionados, es decir, formar las OIDs (tanto las numéricas como las textuales). El identificador de un objeto en su forma numérica está formado por una serie de enteros que sigue en orden en los nodos de la estructura en forma de árbol de la MIB, separados por puntos. La forma textual es muy similar, ya que solo cambia dichos números por los nombres a los que están asociados; esta asociación se define en la MIB en la que está contenida el objeto.

Un extracto del árbol de la MIB SNMP se muestra en la figura 2.4, y servirá como ejemplo para ilustrar la obtención de OIDs.

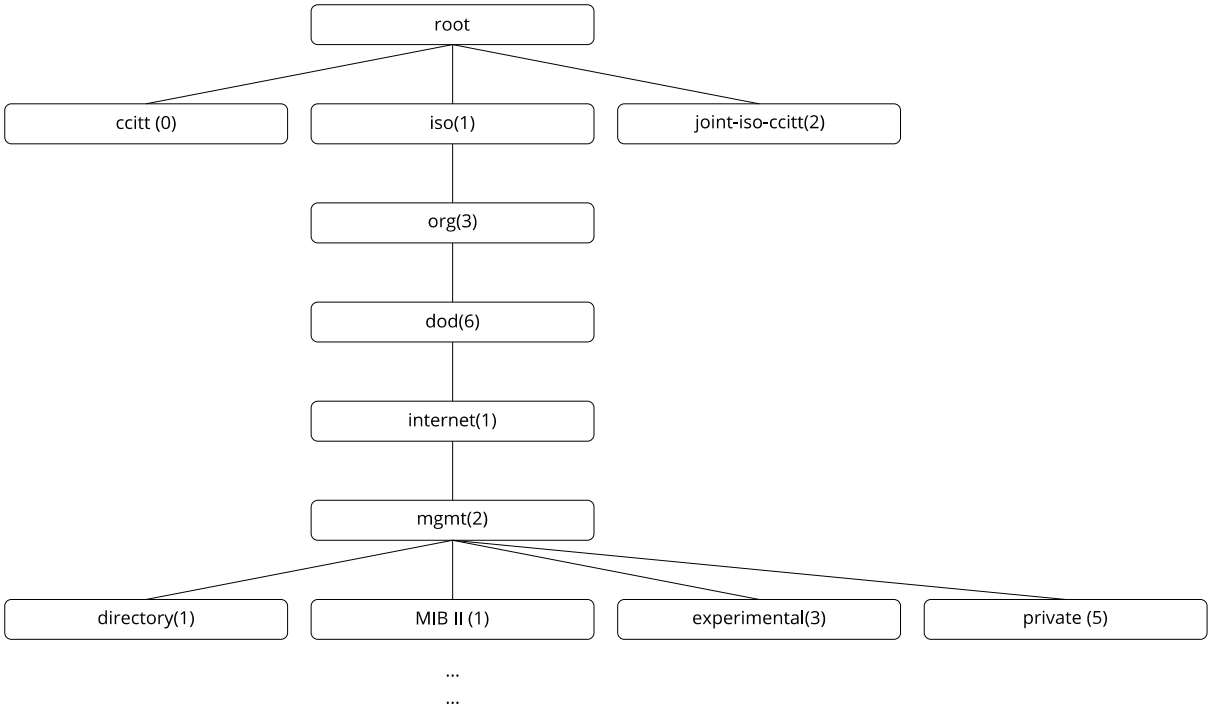


Figura 2.4. Árbol SNMP

En la figura anterior, por ejemplo, el OID numérico del nodo *mgmt* es *.1.3.6.1.2*, obtenido al descender a través

de los objetos gestionados del árbol global (el 1 corresponde a *iso*, el 3 a *org*, el 6 a *dod*, etc.); por tanto, su forma textual es *iso.org.dod.internet.mgmt.mib-2*.

El estándar SMI también describe cómo se define un nodo de la MIB; en este caso la definición ASN.1 es: *mgmt OBJECT IDENTIFIER ::= { internet 2 }*. Se puede observar que es una referencia al nodo anterior en la MIB (*internet*).

Éste estándar también especifica el formato que se debe usar para definir un objeto gestionado completo. Cada uno de ellos consta de cinco campos, según la estructura mostrada en la figura 2.5 [6].

```
OBJECT:

-------

   A textual name, termed the OBJECT DESCRIPTOR, for the object type,

   along with its corresponding OBJECT IDENTIFIER.


Syntax:

   The abstract syntax for the object type.  This must resolve to an

   instance of the ASN.1 type ObjectSyntax (defined below).


Definition:

   A textual description of the semantics of the object type.

   Implementations should ensure that their instance of the object

   fulfills this definition since this MIB is intended for use in

   multi-vendor environments.  As such it is vital that objects have

   consistent meaning across all machines.


Access:

   One of read-only, read-write, write-only, or not-accessible.


Status:

   One of mandatory, optional, or obsolete.
```

Figura 2.5. Estructura no formal de un objeto SNMP

Por lo tanto, además de lo descrito anteriormente, también es necesario definir una definición textual, los **permisos de acceso** y el **status**. La definición textual no es más que una descripción del significado y comportamiento del objeto gestionado, mientras que el campo *access* se refiere a las acciones que se pueden

llevar a cabo sobre dicho objeto (lectura, escritura, lectura-escritura o no accesible). Finalmente, el campo *status* puede tomar los valores *current*, *obsolete*, y *deprecated* y se refiere a la vigencia del objeto.

A todo lo anterior es necesario darle un formato estándar, o dicho de otra manera, una estructura formal. Ésta está definida en el estándar mediante la macro OBJECT-TYPE, cuya forma simplificada se muestra en la figura 2.6.

```
OBJECT-TYPE MACRO ::=
    BEGIN
      TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                        "ACCESS" Access
                        "STATUS" Status
      VALUE NOTATION ::= value (VALUE ObjectName)

      Access ::= "read-only"
                 | "read-write"
                 | "write-only"
                 | "not-accessible"
      Status ::= "mandatory"
                 | "optional"
                 | "obsolete"
      END
```

Figura 2.6. Extracto de la macro OBJECT-TYPE

Esta macro da lugar a las definiciones formales definitivas de los objetos gestionados SNMP. En la figura 2.7 se incluyen algunos ejemplos extraídos de las MIBs desarrolladas para este proyecto a modo ilustrativo.

```
-- Definición de un entero.
combustible OBJECT-TYPE
   SYNTAX      Integer32
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
   "Objeto SNMP con una variable que corresponde a la cantidad de combustible restante expresada en litros."
```

```
    ::= { genservMIBObjects 1 }
…

--Definición de una TABLA.
serversTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ServersEntry
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Esta tabla contiene los datos de consumo de cada uno de los servidores."
    ::= { genservMIBObjects 2 }

serversEntry OBJECT-TYPE
    SYNTAX      ServersEntry
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Fila de la tabla que contiene el nombre del servidor y su valor de potencia consumida en watts."
    INDEX   { server }
    ::= {serversTable 1 }

ServersEntry ::= SEQUENCE {
    server  OCTET STRING,
    potencia   Integer32,
    }

server OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
```

```
        "Cada uno de los servidores en los que se mide la potencia."

        ::= { serversEntry 1 }

    …
```

Figura 2.7. Ejemplos de definición formal de objetos gestionados SNMP

Se observa que el primer objeto gestionado, *combustible*, es un entero que puede ser leído y escrito. El segundo objeto, *serversTable*, es una tabla SNMP, las cuales se forman como secuencias (arrays) de objetos de tipo *entry*. En este caso, *serversTable* es una secuencia de *serversEntry*, que como se puede ver, es a su vez una secuencia de objetos escalares. Se incluye el primero de estos objetos en la figura, server; se trata de un objeto de tipo *octet string* y contiene el nombre de un servidor.

Finalmente, basándose en el estándar SMI, el RFC 1213 [5] define la MIB que rige la gestión de internet. Está dividida en varios grupos, que implementan las funciones fundamentales de gestión internet. A continuación se incluye un breve resumen de cada uno, sin entrar en detalles.

El grupo *system* proporciona información general sobre el sistema gestionado (identificador del fabricante).

El grupo *interfaces* contiene información sobre los interfaces físicos de la entidad, así como información sobre la configuración y estadísticas de los eventos ocurridos en cada uno de ellos.

El grupo *at* (*address translation*) tiene funciones de compatibilidad con la MIB I, el estándar anterior (ya obsoleto).

El grupo *ip* contiene información sobre la operación e implementación de *IP* en el nodo gestionado.

El grupo *icmp* está formado únicamente por contadores de los diferentes tipos de mensajes *icmp* enviados y recibidos.

El grupo *tcp* contiene información sobre la operación e implementación de *TCP* en el nodo gestionado.

El grupo *udp* contiene información sobre la operación e implementación de UDP en el nodo gestionado.

El grupo *egp* contiene información sobre la operación e implementación de *egp* en el nodo gestionado.

El grupo *transmission* mantiene información sobre el sistema físico de cada interfaz del sistema.

Para finalizar, en la figura 2.8 se incluye el inicio de la estructura de árbol de la MIB II, que continúa el árbol de la MIB SNMP mostrado en la figura 2.2.

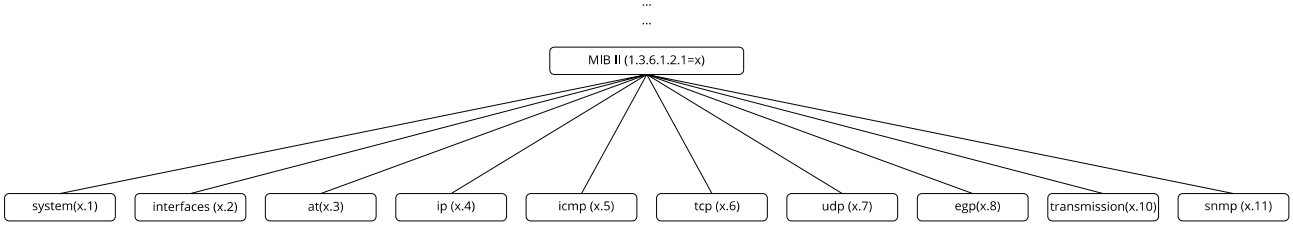


Figura 2.8. Extracto del árbol MIB II

Se observan todos los grupos descritos y su posición jerárquica en el árbol. Por ejemplo, el OID del nodo *interfaces* de la MIB II es .1.3.6.1.2.2.

2.2.3.3. El orden lexicográfico y la primitiva GetNext

Para finalizar con la parte teórica de la gestión internet, en este apartado se introduce el concepto de orden lexicográfico. Éste se refiere al ordenamiento lógico que siguen los objetos gestionados dentro de la MIB.

Como se ha explicado en el apartado anterior, cada nodo de la MIB tiene asociado un OID numérico compuesto por una serie de cifras separadas por puntos. El orden lexicográfico define la ordenación de dichos OIDs, comparando cada cifra de izquierda a derecha; considerando dos OIDs hipotéticos, *a.b.c.d* y *e.f.g.h*, para ordenarlos según este método se compararía el elemento *a* con el *e*, el *b* con el *f* si los anteriores fueran iguales, y así sucesivamente.

Como ejemplo ilustrativo, en la figura 2.9 se representa el orden lexicográfico de los nodos de una hipotética MIB sencilla compuesta por cinco elementos.

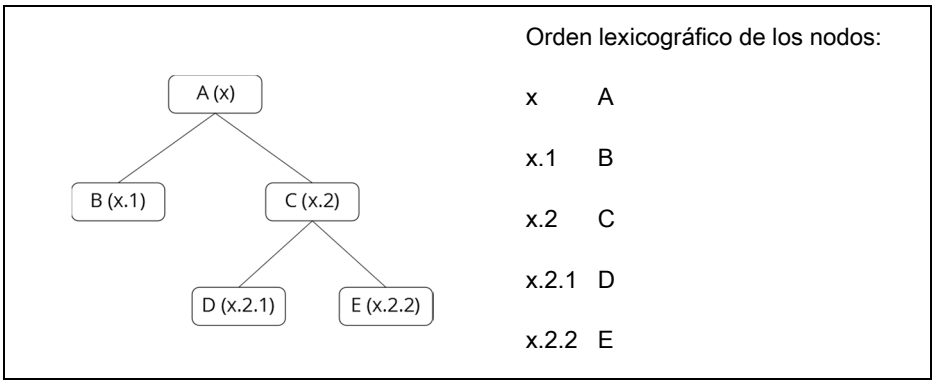


Figura 2.9. Orden lexicográfico de una estructura MIB sencilla

En la figura 2.9 se muestra la correspondencia entre la estructura en árbol de los nodos de una MIB y su orden lexicográfico. Se observa que los elementos de la OID numérica se comparan uno a uno (de izquierda a derecha) para realizar el ordenamiento.

Este orden tiene una aplicación directa cuando se usa la primitiva *GetNext*. Ésta accede al valor siguiente en orden lexicográfico; es decir, ignora todos los objetos que no tienen valor (nodos) y obtiene el primero que sí lo tiene. De esta manera, si por ejemplo se envía una primitiva *GetNext* con referencia al nodo A de la MIB de la figura 2.9 y B y C son nodos sin valor asociado, el valor devuelto corresponde al objeto D. Sabiendo esto, se puede recorrer una MIB al completo conociendo tan sólo el OID del nodo inicial.

Esto es especialmente útil a la hora de recorrer una tabla SNMP, como se verá más adelante.

2.3. Agentes extendidos

En este capítulo se introduce el concepto de agente extendido. Además, se explican someramente los métodos para implementar este tipo de agente y, finalmente, se detallan los aspectos más importantes del estándar AgentX.

2.3.1. Concepto, funciones y métodos

Extender un agente, en esencia, significa implementar en dicho agente el soporte de nuevos objetos gestionados, ya sea modificando las MIBs soportadas o añadiendo otras nuevas [7]. Ésta es una definición básica de agente extendido, de la cual se muestra un esquema funcional en la figura 2.10.

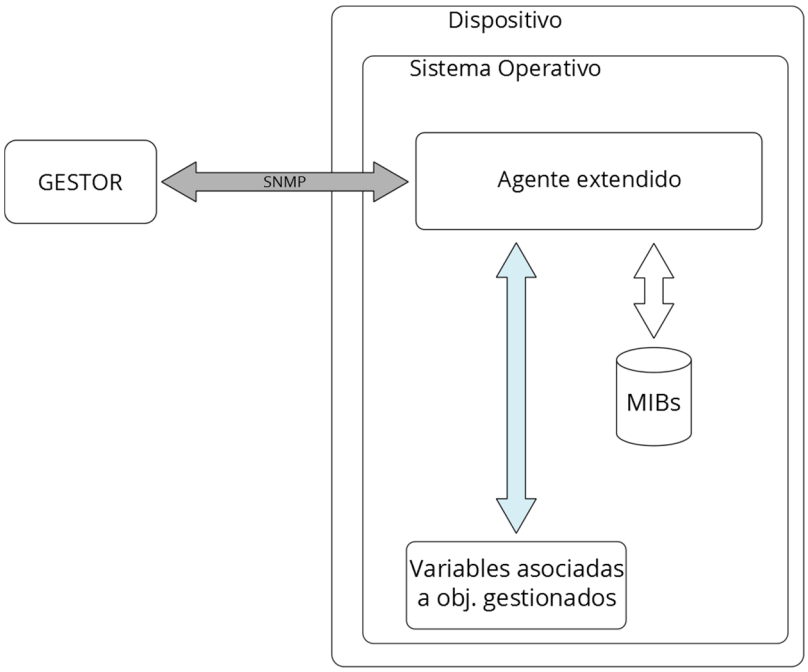


Figura 2.10. Esquema de un agente extendido SNMP básico.

En la figura se observa como el agente extendido es una entidad que se encarga del soporte de la comunicación SNMP, de la información de gestión (MIBs) y de las funciones externas a SNMP de los objetos gestionados.

Los diversos sistemas de extensión de agentes que se han implementado a lo largo del tiempo han llevado a un modelo algo más complejo y flexible de agente extendido, el cual se describe a continuación.

En primer lugar, hay que tener en cuenta que el estándar de gestión de internet, como se ha visto en el apartado 2.1.2, define las comunicaciones entre el gestor y el agente, así como el modelo de información que rige los objetos gestionados. Pero no incluye la implementación de las funciones que soportan y manejan dichos objetos dentro del sistema operativo, lo cual queda en manos del desarrollador. Esto implica que si se desea añadir el soporte de gestión a un nuevo objeto (es decir, extender el agente), se deben proporcionar tanto las funciones de gestión SNMP como el soporte y manejo de dichos objetos dentro del sistema, como se ha

explicado anteriormente.

En segundo lugar, se requiere que la extensión del agente sea una tarea que se pueda realizar de forma modular. Es decir, el soporte completo de los nuevos objetos gestionados se debe poder añadir sin modificar el sistema completo.

Con estos dos requisitos, se llega a un concepto más avanzado de agente extendido, que permite gran modularidad y la separación de las tareas complejas de las que se encarga el agente extendido básico anterior: se define un agente extendido como un agente SNMP formado por el conjunto de un agente maestro y uno o más subagentes, de forma que el agente maestro se encarga de las funciones del protocolo de gestión y los subagentes del manejo de la información de gestión y de la comunicación del agente con otros procesos externos al entorno de gestión [10].

La figura 2.11 representa un esquema funcional de un agente extendido utilizando subagentes.

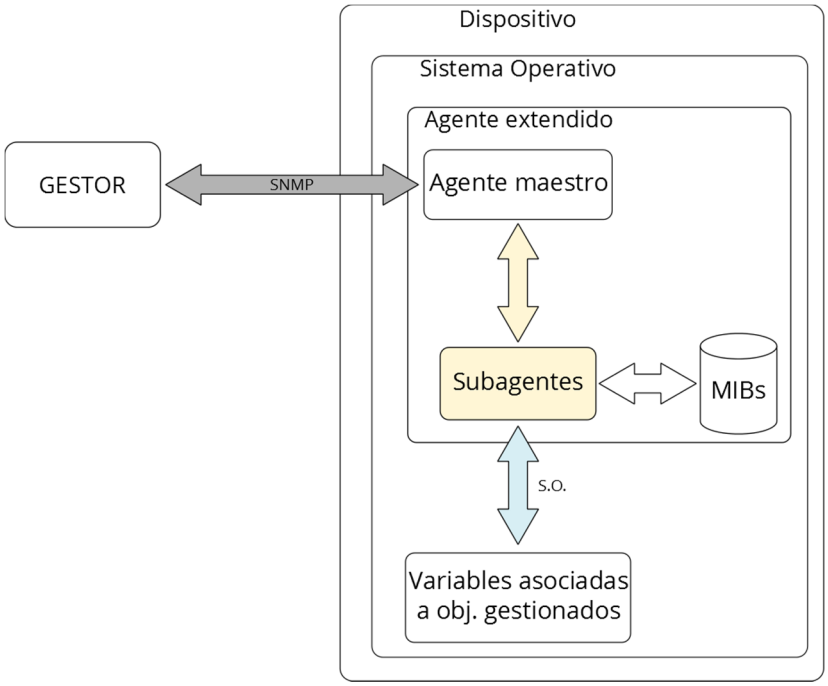


Figura 2.11. Esquema de un agente SNMP extendido con subagentes

Se observa la independencia del agente de las funciones *no SNMP,* así como de los subagentes de las funciones SNMP. Además, los subagentes manejan la información de gestión (MIBs)

En la práctica, los subagentes son *demons* que soportan las funciones externas a SNMP de uno o varios objetos gestionados de forma modular e independiente, así como su información de gestión. Proporcionan la identificación y manejo de dicho objeto por parte del sistema operativo del agente, lo que define uno de los aspectos más importantes del agente extendido: los objetos gestionados se tratan como cualquier variable. Este proyecto se basa en la utilidad que proporciona este hecho: extender las funciones de gestión de red a cualquier dispositivo conectado a ella. En resumen, de esta forma no sólo es posible gestionar los recursos de la propia red, sino cualquier sistema accesible desde la misma.

El método original propuesto por la IETF para extender los agentes SNMP fue SMUX [11], pero fue abandonado y hoy es considerado obsoleto. Su sustituto, AgentX, es el estándar vigente hoy en día, y el utilizado en este proyecto para el desarrollo del agente extendido dentro de la suite de gestión Net-SNMP. Este estándar fue propuesto en 1998, siendo actualizado s su versión actual en 2000. Como se verá de forma más detallada en el siguiente apartado (2.2.2), AgentX responde al modelo de agente extendido mostrado en la figura 2.11.

Además de los estándares de la IETF, existen dos métodos privados de extensión del agente integrados en paquetes completos de gestión: SystemEDGE de CA Technologies y OpenView de Hewlett Packard. SystemEdge responde al modelo básico mostrado en la figura 2.10, mientras que OpenView está implementado según el modelo más avanzado de la figura 2.11. [7].

Finalmente, se debe mencionar que Net-SNMP también permite otros métodos de extensión: las primitivas *pass* y *extend.* Pero ambos son muy limitados para los requisitos de este proyecto; se implementan con la forma de la figura 2.10 [10].

2.3.2. AgentX

En este apartado se describirá brevemente el estándar AgentX, el principal método estándar para la implementación de agentes extendidos.

2.3.2.1. Definición y motivación

Como se ha explicado en el apartado anterior, AgentX es un estándar que define la creación de agentes extendidos SNMP. Responde al esquema funcional de la figura 2.11, de forma que incluye una entidad llamada *agente maestro*, que envía y recibe mensajes SNMP pero que, por lo general, no tiene acceso a la información de gestión y una o varias entidades, llamadas s*ubagentes*, que están blindadas de los mensajes SNMP que son procesados por el agente maestro, pero que manejan la información de gestión [12]. En la figura 2.12 se muestra un esquema que sirve para ilustrar las comunicaciones que se producen en una plataforma cualquiera AgentX.

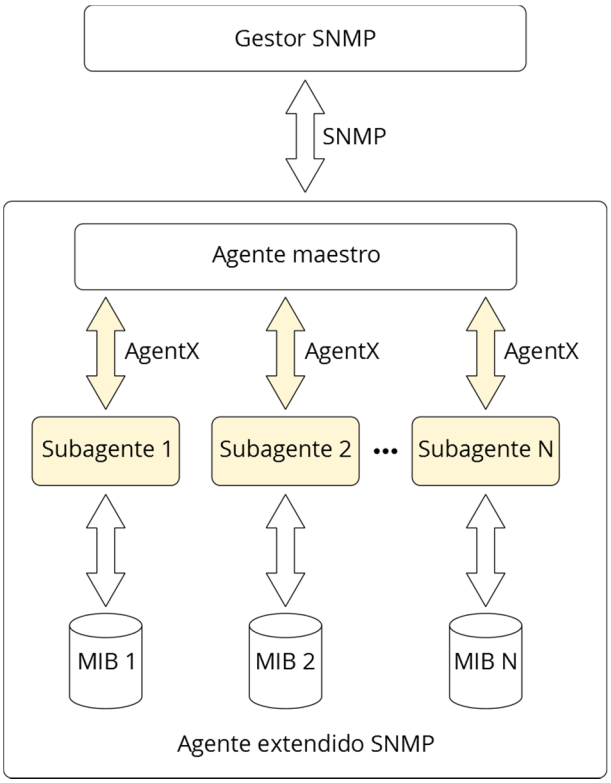


Figura 2.12. Esquema básico de un agente SNMP extendido mediante AgentX

Como se observa en la figura 2.12, los subagentes no contactan en ningún momento con los mensajes SNMP, y el agente maestro tampoco lo hace con la información de gestión (MIBs). Los subagentes se comunican con el agente maestro a través del protocolo AgentX; esta comunicación *agente maestro-subagentes* es completamente invisible para el gestor SNMP, que no puede distinguir un agente monolítico de un agente extendido mediante AgentX. Esta transparencia desde el punto de vista del gestor es uno de los requisitos principales descritos en el estándar [RFC2741]

Como nota aclaratoria y para evitar confusiones, conviene tener en cuenta que el término AgentX se puede referir tanto al protocolo de comunicación entre agente y subagente como al estándar completo desarrollado por la IETF.

Las motivaciones principales del estándar AgentX son separar el protocolo SNMP de la instrumentación relacionada con la MIB, hacer transparente la extensión a los gestores y proporcionar herramientas para extender de forma dinámica los objetos gestionados de la MIB de los subagentes

. Teniendo en cuenta las bases del modelo de información de internet, cada objeto gestionado SNMP está codificado en una instancia de una MIB; los grupos de trabajo de la IETF, las empresas y los particulares deben crear y editar esos objetos en la MIB del agente y también en la MIB del gestor e incluso en la MIB internet. Esto no es práctico en todas las situaciones, ya que se puede desear la gestión de objetos independientes de la MIB internet, de implementación inmediata, actualizables en tiempo real, de prueba, con incompatibilidades de cualquier tipo con otros objetos, etc.

AgentX además supone una ventaja técnica, ya que al separar de forma completa las funciones del agente y

del subagente se especializan las tareas y el sistema funciona y se amplía de forma modular, lo cual es útil en términos de gestión de rendimiento y de simplicidad de desarrollo.

2.3.2.2. Funciones del agente maestro y los subagentes

En este apartado se describen los detalles más relevantes de las funciones del agente maestro y de los subagentes.

El **agente maestro** debe aceptar el establecimiento de la sesión AgentX y el registro de regiones de una MIB por parte de los subagentes, enviar y aceptar mensajes SNMP en comunicación con el gestor y manejar los permisos referentes a los objetos gestionados, enviar y recibir mensajes AgentX (ver apartado 2.2.2.3) para acceder a la información de gestión y enviar notificaciones (*traps*) al gestor en nombre de los subagentes.

Un **subagente** debe iniciar la sesión AgentX con el agente maestro y registrar las regiones de la MIB que soporta, iniciar los objetos gestionados, relacionar dichos objetos gestionados con variables reales, ejecutar las operaciones de gestión sobre esas variables y enviar notificaciones (*traps*) al agente maestro para que éste las envíe al gestor.

Se debe destacar que en el estándar RFC 2741 están reflejadas características que no deben recaer en AgentX, y que deben ser tarea del desarrollador. En concreto se menciona una especialmente importante en lo referente a este proyecto:

"[No se debe implementar]… la habilidad de aceptar cualquier agente extensible imaginable. Éste fue el aspecto con más contenciosos en el desarrollo del estándar. En esencia, ciertas características existentes en algunos agentes extensibles comerciales no están incluidas en AgentX. Aunque éstas puedan ser útiles (o incluso vitales) en algunas implementaciones, el consenso mínimo resultó en que no son apropiadas para un estándar de internet o normalmente necesarias para que subagentes desarrollados de forma independiente coexistan."

Esto implica que los subagentes actúan como programas independientes, y el desarrollador tiene la capacidad y la responsabilidad de crear las funciones necesarias dentro de ese programa; no es tarea del estándar definir esas funciones, ya que se limitarían las opciones de personalización de los subagentes. En el desarrollo de este proyecto (capítulo 4) y en la creación del prototipo (capítulo 5) se observa la necesidad de esta característica, que permite total libertad a la hora de desarrollar cualquier objeto gestionado.

2.3.2.3. Bases del protocolo AgentX

En este apartado se explican brevemente las bases del protocolo AgentX y algunas de sus PDUs (*Protocol Data Units*).

AgentX es un protocolo de aplicación orientado a conexión, y se apoya en el protocolo TCP de transporte. Utiliza el puerto 705 para realizar las comunicaciones entre el agente maestro y el subagente.

Las PDUs de AgentX consisten en una cabecera común, seguida de un campo de longitud variable que contiene los datos específicos de cada PDU. La cabecera tiene un formato fijo, que consiste en una estructura de 20 octetos, tal y como se muestra en la figura 2.13 [12].

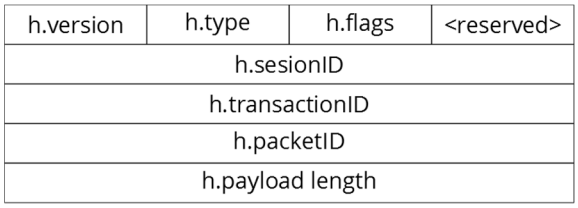


Figura 2.13. Cabecera de un mensaje AgentX

El campo más relevante de la cabecera para ilustrar el protocolo AgentX en lo referente a este proyecto es el que define el tipo de PDU que se envía en el mensaje, *h.type*. Puede contener uno de los siguientes valores, cada uno de los cuales corresponde a una PDU AgentX: *agentx-Open-PDU* (1), *agentx-Close-PDU* (2), *agentx-Register-PDU* (3), *agentx-Unregister-PDU* (4), *agentx-Get-PDU* (5), *agentx-GetNext-PDU* (6), *agentx-GetBulk-PDU* (7), *agentx-TestSet-PDU* (8), *agentx-CommitSet-PDU* (9), *agentx-UndoSet-PDU* (10), *agentx-CleanupSet-PDU* (11), *agentx-Notify-PDU* (12), *agentx-Ping-PDU* (13), *agentx-IndexAllocate-PDU* (14), *agentx-IndexDeallocate-PDU* (15), *agentx-AddAgentCaps-PDU* (16), *agentx-RemoveAgentCaps-PDU* (17), *agentx-Response-PDU* (18).

Las PDUs *agentx-Open* y *agentx-Close* se encargan respectivamente de establecer y cerrar la sesión AgentX con el agente maestro; para ello abren o cierran el socket correspondiente (utilizando el puerto TCP 705); agentx-Open, además, envía un OID que identifica al subagente que soporta [12].

- *agentx-Register-PDU* se encarga de registrar cada región de la MIB que se desee soportar, mientras que *agentx-Unregister* elimina ese soporte al finalizar la sesión AgentX.
- *agentx-Get-PDU*, *agentx-GetNext-PDU*, *agentx-GetBulk-PDU y agentx-Notify-PDU* se encargan de soportar en AgentX las PDUs correspondientes de SNMP (ver apartado 2.1.3.1).
- *agentx-TestSet-PDU*, *agentx-CommitSet-PDU*, *agentx-UndoSet-PDU* y *agentx-CleanupSet-PDU* soportan en AgentX la PDU *SetRequest* de SNMP (ver apartado 2.1.3.1).
- *agentx-Ping-PDU* es enviada por el subagente al agente maestro para monitorizar su capacidad de recibir y enviar PDUs AgentX durante la sesión AgentX.
- *agentx-IndexAllocate-PDU* es enviada por el subagente para asignar el valor de objetos de tipo índice. Por su parte, *agentx-IndexDeallocate-PDU* elimina esta asignación.
- *agentx-AddAgentCaps-PDU* es generada por el subagente para informar al agente maestro de sus capacidades. La PDU *agentx-RemoveAgentCaps-PDU* es generada por el subagente para pedir al agente

maestro que deje de exportar valores particulares referentes a esas capacidades [12].

- *agentx-Response-PDU* se encarga de manejar la correspondiente PDU SNMP (*GetResponse*, ver apartado 2.1.3.1).

3. Aspectos prácticos

Este capítulo describe las funciones y la configuración de la plataforma hardware y del software sobre los que se desarrolla el proyecto.

3.1. Plataforma

El esquema de la figura 3.1 muestra de forma resumida las bases funcionales de la plataforma completa de gestión. Se trata de una evolución de las figuras 2.8 y 2.10, adaptada a la plataforma utilizada en este proyecto.

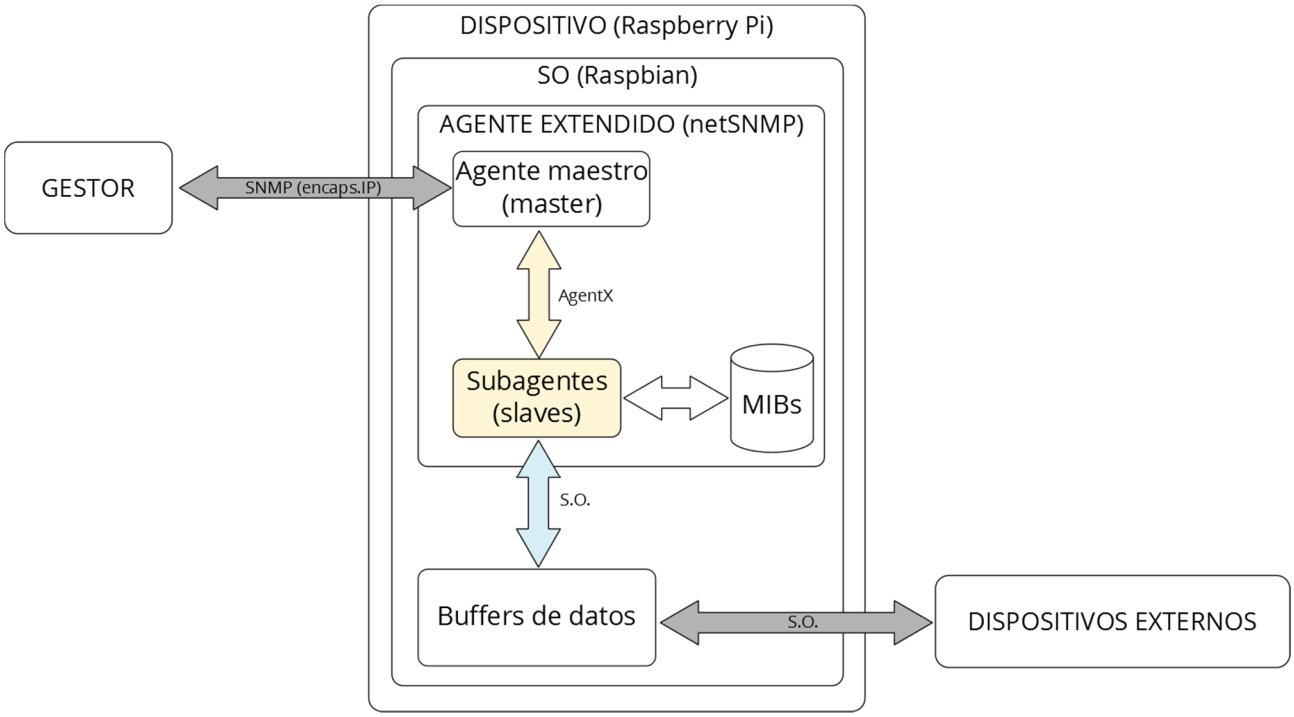


Figura 3.1. Esquema básico funcional de la plataforma de gestión.

El agente extendido está implementado sobre el hardware de la Raspberry Pi, de forma que puede gestionar cualquier dispositivo conectado a ella (por cable –USB, de red- o de forma inalámbrica –Wifi-) y el gestor humano puede acceder a él a través de una conexión de red o, si fuera necesario, directamente conectando una pantalla y dispositivos de entrada (teclado y ratón). El sistema operativo sobre el que se implementa el agente extendido es Raspbian (ver apartado 3.1.2.1) y la suite de gestión Net-SNMP soporta las funciones SNMP y AgentX del mismo. Además, hay que destacar que la comunicación de los subagentes con las funciones referidas a los objetos gestionados que sean externas a SNMP se produce a través de una serie de buffers de datos, cuyo desarrollo se explicará con detalle en los apartados correspondientes de los capítulos 4 y 5.

La suite Net-SNMP proporciona las librerías necesarias para la gestión básica SNMP y su extensión a través de AgentX; tanto los subagentes como su necesaria comunicación con el SO las debe proporcionar el desarrollador.

3.1.1. Hardware: Raspberry Pi Model B

El *hardware* utilizado es una SBC (Single Board Computer), en concreto la Raspberry Pi Model B [20]. Dispone de un SOC (*System on a Chip*) Broadcom BCM2835 que contiene la CPU (*ARM1176JZF-S*), la memoria principal (RAM de 512 MB) y la GPU (*VideoCore IV*).

El procesador ARM1176JZF-S es parte de la familia ARM11, la cual está construida alrededor de un núcleo ARM11. Se trata de una implementación de la arquitectura ARMv6. El núcleo funciona a una frecuencia de reloj básica de 750MHz, que puede ser llevada a más de 1GHz en caso de necesitarse más potencia. [13]. Este tipo de arquitectura se ha usado en la mayoría de dispositivos móviles en los últimos años, desde teléfonos móviles hasta *tablets*, por lo que está muy extendida.

Se ha elegido este dispositivo por varias razones. La primera, es completamente *open-source*, lo que proporciona libertad tanto de información como de modificación. La segunda, está ampliamente aceptado como el SBC más popular en la actualidad, lo cual garantiza documentación e información más accesibles y completas. La tercera, su bajo coste general: 35 euros de adquisición y un bajo consumo, de unos 3.5W [14]. La cuarta y última, su propósito educacional y de facilidad de desarrollo [14], dado que uno de los objetivos del proyecto es usar el dispositivo en las prácticas de la asignatura *Gestión y Operación de Redes*.

En la figura 3.2 se muestra un esquema simplificado del dispositivo con las conexiones disponibles [14].

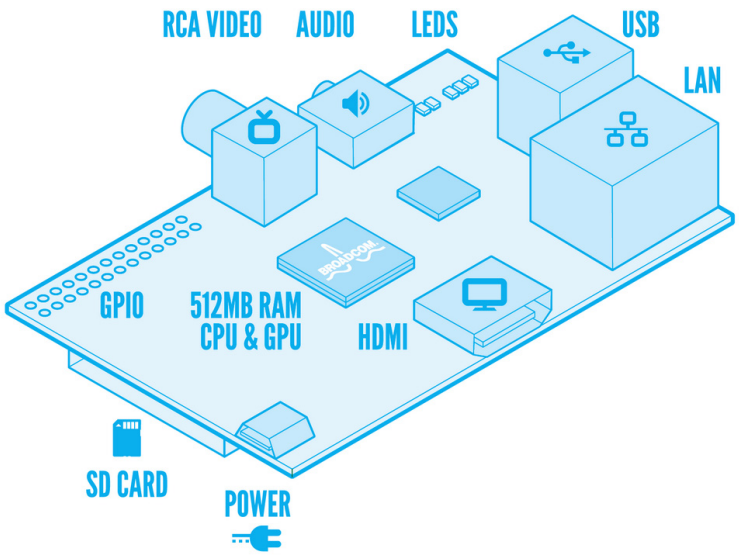


Figura 3.2. Esquema de la Raspberry Pi.

Se puede observar que, además de las conexiones estándar (HDMI, LAN Ethernet, USB, audio y RCA video), dispone de una conexión GPIO de pines, que permite la creación de conexiones personalizadas y dispone de 17 pines que pueden ser programados libremente como entradas o salidas.

3.1.2. Software

En este apartado se presentará el software utilizado para el soporte de las funciones requeridas y durante el desarrollo del proyecto.

3.1.2.1. Sistema operativo Raspbian: características básicas e instalación

El sistema operativo elegido es Raspbian. Se trata de un sistema GNU/Linux derivado de Debian (*fork* directo) y adaptado expresamente a la arquitectura ARMv6 y la potencia disponible en la Raspberry Pi.

Se ha elegido por ser, en la práctica, un sistema operativo Debian, una de las distribuciones GNU/Linux más extendidas. Esto garantiza el funcionamiento nativo de un buen número de aplicaciones, tanto las vitales para el proyecto (Net-SNMP, ver apartado 3.2) como las utilizadas durante el desarrollo (ver apartados 3.1.3 y 3.1.4). Esto permite conseguir un buen estado de funcionamiento de forma directa. Su popularidad también implica la disponibilidad de gran cantidad de documentación e información relacionada.

Además, se trata de un sistema operativo de propósito general completamente modificable. De esta manera se puede reducir a la mínima expresión necesaria una vez finalizado el desarrollo, dejando activos sólo las funciones y programas necesarios. Los objetivos de esta reducción son el máximo aprovechamiento de la capacidad del *hardware*, el mínimo consumo de energía y la máxima estabilidad general del sistema al tener que gestionar menos tareas.

La instalación es gráfica y está totalmente automatizada. En la propia página oficial de Raspberry Pi se facilita el instalador, por lo que sólo hay que descargar su imagen, grabarla en una tarjeta SD vacía que actuará como memoria de almacenamiento de la Raspberry Pi, iniciarla y seguir las instrucciones en pantalla.

3.1.3. Interacción usuario-dispositivo

Uno de los aspectos principales a nivel operativo, tanto para el desarrollo como para un uso posterior, es el acceso rápido y sencillo al dispositivo. La *Raspberry Pi* viene equipada con una salida de vídeo HDMI y conexiones USB, de forma que se le puede conectar directamente una pantalla, un teclado y un ratón. Pero en este caso se necesitará facilitar un acceso remoto al dispositivo, para tareas tanto de desarrollo como de configuración.

Este acceso remoto se puede realizar a través de la conexión LAN Ethernet o de forma inalámbrica utilizando una antena Wifi USB. En este proyecto se ha utilizado la conexión por cable Ethernet, ya que con la interfaz Wifi se produce cierta latencia en la conexión.

Para habilitar el acceso basta con un cable de red, que conecta directamente la Raspberry Pi a la red, y software

para el manejo remoto del escritorio de la Raspberry Pi desde un PC conectado a la misma red; también se realiza la transferencia de archivos utilizando el protocolo SFTP.

Se han utilizado dos métodos de comunicación con el dispositivo:

- **Escritorio remoto**, para manejar el escritorio de Raspbian desde cualquier PC. Se utiliza el protocolo de Windows para escritorios remotos, XRDP. La instalación de XRDP en el dispositivo garantiza que se pueda acceder al escritorio de Raspbian desde cualquier equipo con Windows o GNU/Linux, lo que facilitará posibles futuros desarrollos sobre la plataforma.
- **Cliente SFTP**, para el intercambio y modificación de ficheros de forma segura y rápida. Se ha utilizado FileZilla (filezilla-project.org), también de código libre.

3.1.4. Aplicaciones usadas en el proyecto

El entorno de desarrollo queda a elección del desarrollador, y en este caso se ha elegido el siguiente software:

- **Editores de texto**, para crear y modificar el código de los programas. Se han utilizado Notepad++ (notepad-plus-plus.org) para Windows y Leafpad (tarot.freeshell.org/leafpad) para Raspbian. Ambos son de código libre.
- **Compiladores y debuggers**. Se han utilizado exclusivamente los que se encuentran por defecto en cualquier sistema GNU/Linux: GCC y GDB.
- **Entornos gráficos de SNMP**, para visualizar estructuras, objetos, notificaciones, etc. Para realizar las primeras pruebas sencillas se ha utilizado el programa SnmpB, mientras que para las pruebas del prototipo final el programa elegido ha sido MG-SOFT MIB-Browser. Además, a lo largo del proyecto, muchas pruebas han sido realizadas desde línea de comandos, tanto desde la propia Raspberry en local como desde PCs gestores remotos.

SnmpB, tal y como está definido en su página oficial (*http://snmpb.sourceforge.net/*), es un navegador de MIBs SNMP que soporta SNMPv1, SNMPv2c y SNMPv3. SnmpB puede recorrer, editar, cargar y añadir archivos MIB y soporta todas las PDUs de SNMP. Soporta el descubrimiento de agentes conectados a la red, eventos generados por *traps* y la representación gráfica de valores asociados a los objetos gestionados alojados en cada agente. Se trata de un proyecto de software libre desarrollado y mantenido por la comunidad y actualmente se encuentra en fase *beta* avanzada (versión 0.8), de forma que funciona de forma aceptable aunque se pueden esperar ciertos errores y limitaciones. Es un software bajo la licencia GNU *General Public License version 2.0 (GPLv2)*[15]

MG SOFT MIB Browser [21], por su parte, es una solución de pago más avanzada, utilizada en el Laboratorio de Telemática de la ETSIIT de la UC. Como SnmpB, se trata de un navegador de MIBs SNMP. Permite monitorizar y gestionar cualquier agente SNMP utilizando los estándares SNMPv1, SNMPv2c y SNMPv3.

Soporta las PDUs Get, GetNext, GetBulk y Set. También permite capturar las *traps* enviadas desde cualquier dispositivo o aplicación SNMP de la red y representar gráficamente los valores de los objetos gestionados.

Ambos programas permiten realizar, en esencia, las mismas funciones, pero MG SOFT MIB Browser es considerablemente más potente y completo.

El paquete Net-SNMP, como se explicará en los siguientes apartados, se incluye en el sistema proporcionando las bases para el desarrollo del agente extendido SNMP.

3.2. Net-SNMP: descripción y configuración

Net-SNMP es la suite básica de gestión SNMP sobre la que funcionan tanto el protocolo SNMP básico como el de agente extendido AgentX. En este apartado se describirán sus funciones.

3.2.1. Descripción y componentes

Net-SNMP es una suite de aplicaciones usadas para implementar SNMP (versiones 1, 2c, y 3) usando IPv4 e IPv6. La suite incluye:

- Aplicaciones de línea de comando para obtener información de dispositivos con SNMP habilitado utilizando peticiones SNMP (*snmpget***,** *snmpgetnext***,** *snmpwalk***,** *snmptable*), manipular la información SNMP de un dispositivo (*snmpset*) y para pasar de la forma numérica a textual de los OID de una MIB y viceversa, así como mostrar el contenido y estructura de la MIB (*snmptranslate*).
- Un *demon* para recibir notificaciones SNMP (*snmptrapd*).
- Un agente extensible para responder a las peticiones de información de gestión (*snmpd*) […] Puede ser extendido de varias formas, incluyendo el uso del protocolo AgentX (Agent Extensibility).
- Una librería para desarrollar nuevas aplicaciones SNMP, con las APIs de C o Perl.

Net-SNMP está disponible para sistemas operativos Unix o basados en Unix y también para Microsoft Windows [18].

Se introduce el concepto de *daemon* o *demon,* que se trata de un tipo especial de proceso informático no interactivo, es decir, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Como se verá en adelante, los *demons* en un entorno de gestión se encargan de las tareas de monitorización y soporte de los objetos gestionados, que deben ser transparentes para el usuario y no requerir acciones por su parte.

También se deben destacar las dos funcionalidades de Net-SNMP en las que se basa este proyecto: proporciona y soporta todas las funciones relativas a SNMP y facilita la extensión del agente mediante varios métodos, incluido AgentX.

3.2.2. Instalación del agente SNMP en Raspberry Pi

El primer objetivo principal del proyecto es conseguir un dispositivo que actúe como agente SNMP. Para esto basta con instalar y configurar el paquete Net-SNMP.

La instalación de este paquete en Raspbian es directa, ya que se encuentra en los repositorios oficiales de Debian. Los paquetes de Debian son compatibles con Raspbian, siempre que la limitada potencia del *hardware* de la Raspberry Pi permita su funcionamiento; en este caso no hay problemas al respecto. Con el comando *sudo apt-get install net-snmp*, el gestor de paquetes de Raspbian (*apt-get*) se encarga de descargar, instalar y realizar la configuración preliminar de Net-SNMP.

3.2.3. Carga de una MIB

Como se detalla en el apartado 2.1.3.2, dedicado al modelo de información de la gestión internet, una MIB se comporta como una referencia, definiendo una estructura en la que se organizan los objetos gestionados. En este apartado se explica cómo se carga una MIB a través de *Net-SNMP*, es decir, como se informa al agente de la estructura de la misma.

Es importante recordar que el soporte completo de objetos SNMP implica también manejar las variables y funciones referidas a los objetos a los que se alude en la estructura de la MIB. Es decir, la carga de la MIB sólo es el primer paso en el soporte de objetos SNMP.

El primer paso es permitir que Net-SNMP cargue cualquier MIB desde los directorios que rastrea. Para ello, se debe modificar el archivo de configuración de Net-SNMP *snmp.conf*. En concreto, se debe añadir la línea "*mibs: mibs +ALL*".

Una vez habilitada la carga de nuevas MIBs, ésta se produce de forma automática una vez que los archivos de las MIBs están copiados en los directorios necesarios. En el tutorial de *Net-SNMP* [10] se indica que sólo es necesario que esté en un directorio, pero eso no resulta del todo correcto. Aun así, se indica el método para obtener la situación de las MIBs en el sistema [10]. Basta con introducir este comando, *net-snmp-config --default-mibdirs,* que devuelve los directorios desde los que Net-SNMP carga las MIBs (figura 3.3).

```
pi@raspberrypi ~ $ net-snmp-config --default-mibdirs
/home/pi/.snmp/mibs:/usr/share/mibs/site:/usr/share/snmp/mibs:/usr/share/mibs
/iana:/usr/share/mibs/ietf:/usr/share/mibs/netsnmp
pi@raspberrypi ~ $ _
```

Figura 3.3. Directorios desde los que Net-SNMP carga las MIBs.

Tras un proceso de puro ensayo-error, ya que no se encontró más documentación al respecto, se llegó a la conclusión de que las MIBs que se añadan deben estar situadas en estos dos directorios para que todo funcione correctamente: */usr/share/snmp/mibs* y */home/pi/.snmp/mibs*.

Una vez copiado el archivo ASN.1 de la MIB que se desea gestionar en ambos directorios, sólo hay que reiniciar el servicio *snmpd* para que el sistema reconozca la estructura, utilizando el comando *service snmpd restart*. Tras esto, se puede comprobar la estructura completa de la MIB utilizando el comando *snmptranslate*. En la figura 3.4 se muestra como ejemplo la MIB *SUBAGENT-1-MIB*; dicha MIB se usará más adelante para mostrar el desarrollo de subagentes.

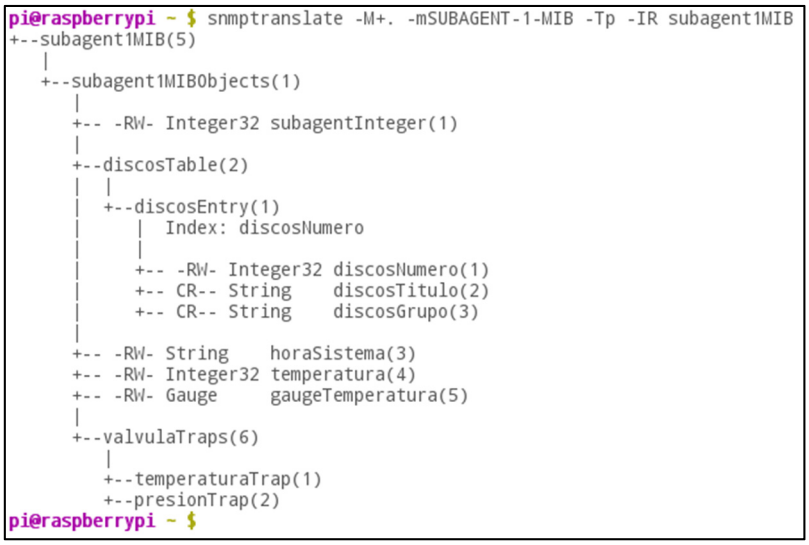


Figura 3.4. Estructura de árbol de SUBAGENT-1-MIB

Se observa que la MIB contiene varios objetos de tipo escalar (*subagentInteger*, *temperatura*), uno de tipo *string* (*horaSistema*), uno de tipo *gauge* (*gaugeTemperatura*) y una tabla (*discosTable*), formada por filas de tipo *entry* (*discosEntry*), que a su vez están compuestas por tres elementos columna (*discosNumero* –objeto de tipo escalar-, *discosTitulo* y *discosGrupo* –objetos de tipo *string*-). También contiene dos *traps* (*temperaturaTrap* y *presiónTrap*). Se puede comprobar que la estructura mostrada en la figura coincide con la definida en el código de *SUBAGENT-1-MIB* (ver anexo A3).

3.2.4. Utilidad mib2c

Como se ha indicado anteriormente, además de la carga de la estructura simbólica de la MIB, es necesario soportar y manejar las variables asociadas a cada uno de los objetos gestionados indicados en dicha estructura; el grueso del trabajo de desarrollo de este proyecto consiste precisamente en ello.

La utilidad mib2c, incluida en el paquete Net-SNMP, proporciona el punto de partida en esta tarea: su uso **genera de forma automática una serie de *documentos esqueleto*** formados por partes de código en lenguaje C y partes de pseudocódigo explicativas. Estos documentos generados tienen una gran utilidad como guía sobre la que programar las funciones de soporte y manejo de las variables asociadas a los objetos gestionados.

La utilidad recibe dos datos: el nombre del objeto gestionado y el nombre de un archivo de configuración (que está incluido en la propia utilidad). Con estas entradas, genera varios archivos de código fuente incompletos,

que el desarrollador debe usar como punto de partida para programar los *demons* del servicio de gestión. Los archivos de configuración corresponden a casos generales de objetos gestionados, de forma que, a base de investigar en la documentación y de pruebas, el desarrollador debe elegir el que mejor se adapte a sus necesidades.

Existen archivos de configuración para objetos de tipo escalar (*mib2c.scalar.conf, mib2c.int_watch.conf*), de tipo tabla (*mib2c.create-dataset.conf*, *mib2c.iterate.conf*, *mib2c.iterate-access.conf*, *mib2c.table_data.conf*, *mib2c.mfd.conf*) y de tipo notificación o *trap* (*mib2c.notify.conf*). En los capítulos 4 y 5 se explicarán y utilizarán algunos de ellos.

3.3. AgentX sobre Net-SNMP

Como se ha detallado en el apartado 2.2.2, AgentX es un estándar que describe la comunicación entre un agente maestro y uno o varios subagentes esclavos dentro de un agente extendido SNMP. Cada subagente consiste en un proceso (*demon*) que inicializa y maneja los objetos gestionados SNMP creados por el usuario, de forma que estos se comunican de forma automática y transparente con el agente maestro. En este punto, es recomendable repasar la figura 3.1, que representa un esquema funcional de un agente sobre Net-SNMP extendido mediante AgentX.

Bajo la plataforma proporcionada por Net-SNMP, un subagente AgentX consiste en un proceso (*demon*) iniciado por un ejecutable compilado a partir del código desarrollado para uno o varios nodos de una MIB (objetos gestionados). Este código es el desarrollado a partir del esqueleto proporcionado por *mib2c* tal y como se ha comentado en el apartado 3.2.4.

Como ya se ha indicado, la comunicación entre el agente maestro y los subagentes es automática y transparente, a través del protocolo AgentX. Net-SNMP se encarga de proporcionar las funciones para que esto sea posible, por lo que el trabajo de desarrollo se centra en crear los *demons* que soportan los subagentes.

3.3.1. Configuración

A continuación se describe el proceso completo a través del cual se configura el dispositivo para que actúe como un agente extendido SNMP con la capacidad de aceptar y gestionar de forma automática un número indefinido de subagentes de tipo AgentX. La figura 3.1, incluida al principio de este capítulo, representa un esquema conceptual del dispositivo (el color amarillo en esta figura corresponde a funcionalidades exclusivamente SNMP y el azul a funciones de comunicación del subagente con procesos externos -*kernel* del sistema operativo, archivos, dispositivos, etc.-).

Se debe conseguir que el agente se comunique correctamente con sus subagentes a través del protocolo AgentX y con los gestores a través de SNMP (ver apartado 2.2.2). Es importante reseñar que este proyecto se centra

exclusivamente en la versión 2c de SNMP, por lo que no existe soporte para la versión 3 (sí para la 1, ya que hay retrocompatibilidad).

Existe un archivo de configuración de Net-SNMP que gestiona las funciones necesarias para el soporte de agentes SNMP y subagentes AgentX: *snmpd.conf*. Se trata del archivo de configuración del *demon* de *Net-SNMP* (proceso residente en memoria de Net-SNMP). A continuación se detallan todos los cambios realizados y su explicación; el código completo se puede consultar en el anexo A4. Como se puede observar a continuación, no sólo es necesario configurar el soporte de AgentX, sino también alguna otra función básica.

En la figura 3.6 se incluyen todos los cambios necesarios en el archivo *snmpd.conf*.

```
...
#  Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:161
...
rocommunity public  localhost
rwcommunity private localhost
rocommunity public default
rwcommunity private default
…
#   send SNMPv2c traps
trap2sink    localhost public
trap2sink    default public
trap2sink    192.168.1.33 public
trap2sink    192.168.4.5 public
…
#  AgentX Sub-agents
#  Run as an AgentX master agent
master          agentx
…
```

Figura 3.5. Configuración de *snmpd.conf*

Como se puede observar, **el primer paso es habilitar el puerto UDP** sobre el que se produce la comunicación, el 161. Como se explica en el apartado 2.1.3.1, se trata del puerto de comunicación de las funciones SNMP (excepto de los *traps*). A continuación, **se configuran las comunidades** (ver apartado 2.1.3.1), de forma que la comunidad *public* tiene acceso de lectura desde cualquier dispositivo (remotos y *localhost*) y la comunidad

private, de lectura y escritura. Posteriormente, **se habilita el soporte de *traps*** para la versión 2c; se observa que es necesario añadir manualmente las IPs de todos los posibles gestores para que reciban los *traps* (por ejemplo, 192.168.4.5 corresponde a la IP privada de un equipo del laboratorio de Telemática de la ETSIIT que actúo como como gestor en las pruebas de funcionamiento). Finalmente, **se indica al *demon* de SNMP que se tiene que comportar como un maestro AgentX**, lo que permitirá que los subagentes AgentX se conecten.

Con estos cambios, el dispositivo se comportará como un agente SNMP extensible a través de subagentes AgentX, lo cual es el primer objetivo del proyecto.

Para activar el soporte de todas las funciones de agente extendido, incluida la comunicación automática y transparente con los subagentes AgentX, basta con poner en marcha el *demon* de Net-SNMP mediante el comando *sudo snmpd.*

3.3.2. Carga de un subagente AgentX

Este apartado sirve como introducción al soporte completo de objetos gestionados SNMP mediante subagentes AgentX. Como se ha explicado anteriormente, la MIB es tan solo una estructura simbólica en la que se definen los objetos, pero no se crean las variables ni las funciones relacionadas con ellos. Para un soporte completo, se requiere lo siguiente:

- Cargar la MIB y que ésta sea *reconocida* por Net-SNMP, para que la aplicación sea capaz de *entender* su estructura.
- Dotar a los objetos SNMP de las funcionalidades que se espera de ellos, tanto las relacionadas el soporte de funciones SNMP (*get*, *set*, *getnext*, etc.) como las relacionadas con la gestión de las variables implicadas por parte del sistema operativo (por ejemplo, el enlace en tiempo real del valor de un objeto SNMP con un valor proporcionado por un periférico). De esto se debe encargar por completo el desarrollador, y las funciones necesarias están incluidas en el *demon* AgentX; éste es un programa residente en memoria que, sin la interacción del usuario, proporciona todas las funciones necesarias automáticamente.

A partir de este momento, el trabajo del proyecto se centrará en la creación de *demons* que actúen como subagentes AgentX.

4. Desarrollo de agentes extendidos mediante AgentX

4.1. Introducción

En este capítulo se describen las tareas realizadas para la implementación completa de un agente SNMP extendido mediante AgentX. El trabajo consiste en el desarrollo de subagentes AgentX adaptados a distintos requisitos, de forma que supongan una base desde la cual desarrollar con facilidad subagentes útiles en escenarios concretos (realistas y de simulación).

El desarrollo se ha realizado en orden de complejidad creciente, y cubre el manejo de datos simples y no enlazados con variables externas (apartado 4.2.1), el manejo de datos simples enlazados a variables cualesquiera (apartado 4.2.2), el manejo de tablas de datos dinámicas (apartado 4.3) y el manejo de *traps* asociados a objetos escalares con valores dinámicos (apartado 4.4).

A continuación, se incluye un esquema de creación de un objeto gestionado SNMP (figura 4.1). El color amarillo corresponde a funcionalidades exclusivamente SNMP y el azul a funciones de comunicación de SNMP con "el exterior" (*kernel* del sistema operativo, archivos, dispositivos, etc.).

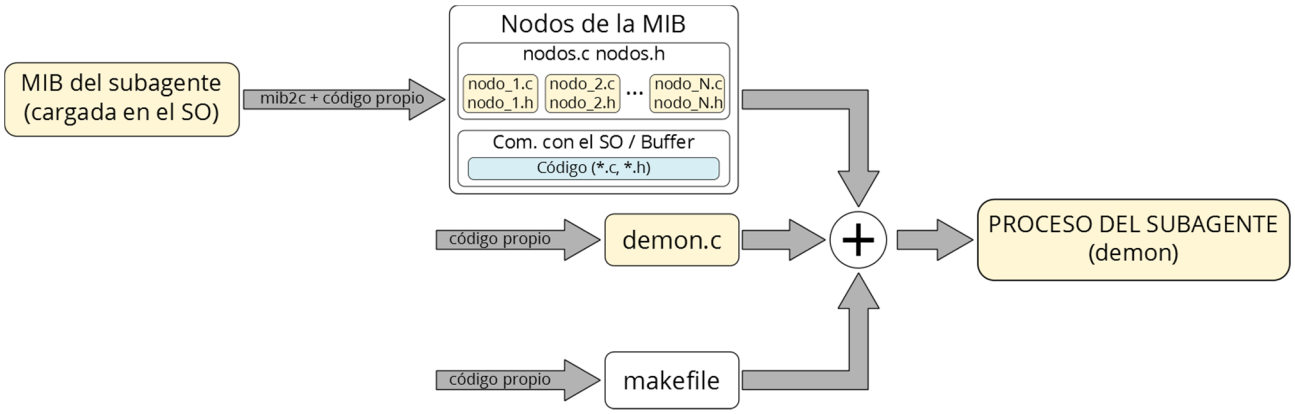


Figura 4.1. Esquema general para el desarrollo de un objeto gestionado SNMP.

Como se observa, se parte de la MIB del subagente, que contiene los objetos gestionados que se vayan a desarrollar y debe estar cargada en el sistema (ver apartado 3.2.3). Partiendo de la MIB se debe desarrollar el código fuente de los programas que manejarán los objetos. El primer paso es generar un esqueleto de código utilizando la utilidad *mib2c* (ver apartado 3.2.4), y completarlo con las funciones necesarias. Además, se debe implementar un *demon*, es decir, un proceso residente en memoria que soporte las funciones del objeto gestionado en todo momento.

4.2. Manejo de objetos escalares

En este apartado se tratará el soporte de objetos gestionados escalares; SNMP acepta varios tipos de escalares, como valores enteros (*integer*), *strings* (*octet string*), direcciones IP, OIDs (*object identifier*), contadores (*counter*), umbrales (*gauge*), etc. En su forma más sencilla, estos objetos tienen un valor sólo modificable usando funciones SNMP (apartado 4.2.1). En el apartado 4.2.2 se ha incluido una funcionalidad, que consiste en la asociación en tiempo real del valor del objeto con un dato externo al agente.

4.2.1. Escalares modificables a través de funciones SNMP.

En este apartado se describe el desarrollo del software que soporta un objeto escalar simple.

4.2.1.1. Desarrollo

Se utiliza un objeto escalar básico como punto de partida, tomado directamente del tutorial de Net-SNMP. Se trata del primer objeto definido en la MIB *SUBAGENT-1-MIB*, *subagentInteger* (ver figura 3.4). En la figura 4.2 se incluye la definición de este objeto.

```
…
subagentInteger OBJECT-TYPE
   SYNTAX      Integer32
   MAX-ACCESS  read-write
   STATUS      current
   DESCRIPTION
     "This is an object that simply supports a writable integer
      when compiled into the agent.  See
      http://www.net-snmp.org/tutorial-5/toolkit/XXX for further
      implementation details."
   DEFVAL { 0 }
   ::= { subagent1MIBObjects 1 }
…
```

Figura 4.2. Definición de objeto *subagentInteger* dentro de la MIB

Se observa que se trata de un objeto gestionado de tipo escalar (valor entero), que puede ser leído y escrito, con un valor por defecto de 0 (que toma una vez se inicializa) y cuyo objeto precedente en el árbol MIB es subagent1MIBObjects, lo que da lugar a su OID: 1.3.6.1.4.1.8072.2.5.1.1 (ver apartado 2.1.3.2 sobre el modelo de información internet y anexos A1 y A2 para tener una referencia completa de SUBAGENT-1-MIB).

Para obtener el *esqueleto* del código se usa la aplicación mib2c, descrita en el apartado 3.2.4. En este caso, se parte del archivo de configuración *mib2c.scalar.conf* para generarlo; esta configuración es la más sencilla de todas y la única que no sólo proporciona una plantilla, sino el código completo. Por lo tanto, una vez generado, tan solo es necesario compilarlo para obtener el programa ejecutable que soportará el objeto *subagentInteger* (*demon*).

El comando *sudo mib2c -c mib2c.scalar.conf SUBAGENT-1-MIB::subagentInteger* genera dos archivos: *subagentInteger.c* y *subagentInteger.h*. Se puede observar que en el comando se utiliza el *superusuario* de GNU/Linux (*sudo*), y de ahora en adelante será una práctica común para realizar la mayor parte del desarrollo, ya que son necesarios permisos de administrador para acceder a la mayoría de funciones SNMP.

Además, es necesario obtener código que soporte las funciones básicas del *demon*, para lo cual se parte de una plantilla proporcionada por el tutorial del proyecto Net-SNMP y un Makefile. En este caso, el archivo de código del *demon* del subagente es *subagent1_demon.c*. En la figura 4.3 se muestran fragmentos relevantes de dicho código.

```
...
init_agent("subagent1_demon");
…
init_subagentInteger();
...
int agentx_subagent=1
...
If (agentx_subagent)
{
    netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID, NETSNMP_DS_AGENT_ROLE, 1);
}
…
```

Figura 4.3. Código que inicia el subagente y la variable asociada al objeto gestionado

El primer cometido del *demon* es inicializar el agente SNMP y la variable asociada al objeto concreto que vayamos a gestionar, como se observa en los dos primeros fragmentos de código. Otras de las funciones clave del *demon* son la que lo habilita como subagente Agentx y la que inicia el soporte AgentX, que corresponden a los dos fragmentos restantes.

El archivo de código generado por *mib2c subagentInteger.c* inicia las funciones SNMP del objeto gestionado y registra su OID (ver apartados 2.1.2.1 y 2.1.2.2) con el código *static oid subagentInteger_oid[] = { 1,3,6,1,4,1,8072,2,5,1,1 }*. Se puede comprobar que la OID definida por este código (1.2.6.1.4.1.8072.2.5.1.1)

corresponde a la representada en *SUBAGENT-1-MIB* (ver anexos A1 y A2).

Finalmente, con el comando *sudo make subagent1_demon* se obtiene el ejecutable del *demon* que soporta este primer objeto gestionado.

Todo el proceso seguido para obtener el *demon* se ajusta a lo representado en la figura 4.1, como se puede comprobar.

4.2.1.2. Puesta en marcha

En este apartado se pone en funcionamiento el soporte de un objeto escalar.

Como primer paso, se debe inicializar en el agente maestro el soporte de subagentes AgentX, con el comando *sudo snmpd*.

Posteriormente, se lanza el *demon* del subagente descrito en el apartado anterior, utilizando el comando *sudo ./subagent1_demon*. A partir de este momento, el objeto SNMP *subagentInteger* esta inicializado y soporta las funciones SNMP. Para comprobarlo, en el siguiente apartado se incluyen una serie de pruebas sencillas realizadas desde un equipo remoto.

4.2.1.3. Pruebas y resultados

En este primer caso, se debe comprobar que el objeto soporta las funciones básicas de SNMP: GET, WALK y SET (ver apartado 2.1.3.1). En la captura siguiente (figura 4.4) se muestran los resultados obtenidos desde un equipo gestor remoto con sistema operativo Windows 8.1 a través de la línea de comando.

```
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\adrian>snmpget -v 2c -c public 192.168.1.42 .1.3.6.1.4.1.8072.2.5.1.1.0

NET-SNMP-MIB::netSnmp.2.5.1.1.0 = INTEGER: 0

C:\Users\adrian>snmpset -v 2c -c private 192.168.1.42 .1.3.6.1.4.1.8072.2.5.1.1.
0 i 89
NET-SNMP-MIB::netSnmp.2.5.1.1.0 = INTEGER: 89

C:\Users\adrian>snmpget -v 2c -c public 192.168.1.42 .1.3.6.1.4.1.8072.2.5.1.1.0

NET-SNMP-MIB::netSnmp.2.5.1.1.0 = INTEGER: 89

C:\Users\adrian>snmpwalk -v 2c -c public 192.168.1.42 .1.3.6.1.4.1.8072.2.5.1.1.
0
NET-SNMP-MIB::netSnmp.2.5.1.1.0 = INTEGER: 89

C:\Users\adrian>snmpgetnext -v 2c -c public 192.168.1.42 .1.3.6.1.4.1.8072.2.5.1
.1
NET-SNMP-MIB::netSnmp.2.5.1.1.0 = INTEGER: 89

C:\Users\adrian>
```

Figura 4.4. Comandos SNMP para comprobar el funcionamiento de un objeto escalar básico.

Se puede observar que el valor inicial del objeto es 0, para ser posteriormente modificado mediante un SET (ver apartado 2.1.3.1), quedando fijado en 89. Los siguientes tres comandos muestran el acceso al objeto básico

a través de tres métodos: GET; WALK, que recorre todos los objetos de la MIB a partir de la OID introducida; y GETNEXT, que accede a la instancia siguiente a la introducida según el orden lexicográfico.

4.2.2. Escalares asociados a una variable dinámica

En este apartado se introduce la comunicación de las variables asociadas a los objetos gestionados con el sistema operativo, de forma que puedan ser modificadas y leídas utilizando métodos externos al agente. Es similar al apartado 4.2.1, aunque tiene un añadido: el objeto gestionado toma su valor en tiempo real desde un archivo de texto.

4.2.2.1. Desarrollo

En este caso, se desarrollan las funciones necesarias para gestionar el objeto SNMP ***temperatura***. De nuevo, se parte del "código esqueleto" proporcionado por mib2c (ver apartado 3.2.4), en este caso con el archivo de configuración *mib2c.scalar.conf.* El comando utilizado es *sudo mib2c -c mib2c.scalar.conf SUBAGENT-1-MIB::temperatura*.

Este comando genera dos archivos: *temperatura.c* y *temperatura.h.* Es necesario también el código del *demon* (*temp_demon.c*) y un *makefile*. Además, se debe añadir el código de la función que realiza la lectura del archivo de texto (*valor.txt*) en el que se almacena el valor de temperatura (en el fichero *temp_demon.c*). En la figura 4.5 se muestra este código.

```
…
while(keep_running) {   /*Mientras se esté ejecutando el programa
    lee de un archivo .txt y extrae el valor de la temperatura*/
    FILE * fp;
    fp = fopen ("valor.txt", "r");
    if (fscanf(fp, "%d", &temperatura)==1){
            printf("%d",temperatura);
            }
            else{printf("Error al leer la temperatura.\n");
    }
    fclose(fp);
}
```

Figura 4.5. Código que extrae un valor de un archivo *buffer*

Se observa que el *demon* lee de forma constante el valor del archivo de texto, asignándoselo al valor del objeto gestionado *temperatura* en tiempo real. De esta forma, si se modifica el archivo de texto, el valor del objeto gestionado se actualiza inmediatamente.

Se compila el *demon* de forma análoga al apartado 4.2.1.1 y se obtiene el ejecutable del subagente AgentX, utilizando el comando *sudo make temp_demon.*

4.2.2.2. Puesta en marcha

Se debe iniciar en el agente el soporte de subagentes AgentX (ver apartado 4.2.1.2) y también el *demon* del subagente, con el comando *sudo ./temp_demon*.

A partir de ese momento, el objeto SNMP temperatura está inicializado y operativo, y su valor está enlazado con el contenido del archivo *valor.txt*.

4.2.2.3. Pruebas y resultados

Se debe comprobar que el objeto puede ser obtenido a través de peticiones SNMP tanto desde el dispositivo (en local) como desde un gestor remoto, y también que su valor está enlazado en tiempo real con el contenido de *valor.txt*. En la figura 4.6 se muestra la obtención del valor del objeto desde un gestor remoto, mientras que en la figura 4.7 se muestran las pruebas en local.

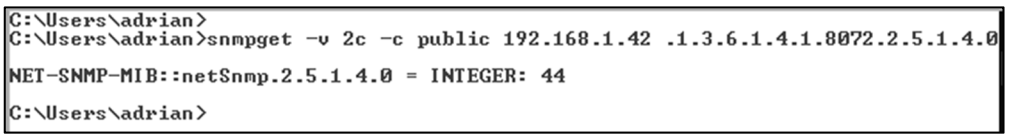


Figura 4.6. Comprobación del objeto desde un gestor remoto.

Únicamente se comprueba el valor desde el mismo equipo remoto que en el apartado 4.2.1.3.

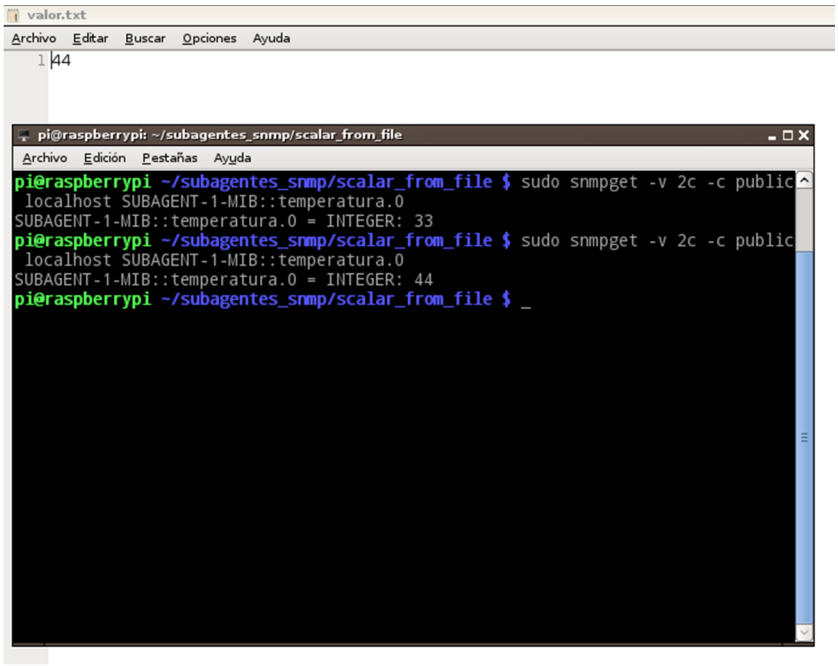


Figura 4.7. Valor asociado a un archivo de texto en tiempo real.

Se observa que el objeto se ha actualizado en consonancia al valor contenido en el archivo *valor.txt*.

4.3. Manejo de tablas dinámicas

En este apartado se desarrollan las funciones necesarias para el soporte de un objeto gestionado SNMP de tipo tabla. Los datos de esta tabla están almacenados en un archivo externo, y las funciones desarrolladas permiten que el valor de cada uno de los elementos de la tabla-objeto SNMP coincida en todo momento con el valor que toman en este archivo. Es decir, ambas tablas están enlazadas en tiempo real, y el cambio de cualquier elemento en el origen de datos se refleja casi instantáneamente en el objeto.

4.3.1. Introducción: tablas en SNMP

Las tablas SNMP se almacenan de una forma que puede resultar poco intuitiva en un primer momento, aunque este ordenamiento responde a una razón que se detalla más adelante. La organización de sus elementos sigue el llamado orden lexicográfico (ver apartado 2.1.3.3), como cualquier objeto SNMP dentro de la estructura de una MIB.

Las tablas SNMP están definidas como una secuencia de elementos de tipo *entry*. Una *entry* está compuesta por una fila completa, formando un *array* de elementos. Cada elemento de esa fila corresponde a una columna, y (al menos) uno de ellos debe estar identificado como índice.

Este índice tiene la clave de la ordenación de la tabla y del acceso a los elementos de la misma al utilizar

funciones SNMP. El índice forma parte del identificador del objeto (OID), de forma que si la tabla tiene un elemento X, el OID se estructura de la siguiente manera: **X.(NºColumna).(índice)**. Por ejemplo, el OID del elemento número 3 de la fila cuyo índice tiene un valor de 23 es X.3.23; si el valor del siguiente índice es 44 y la columna continúa, el OID del siguiente elemento sería X.3.44.

Lo anterior implica que conociendo el OID de la tabla (X) ésta se puede recorrer entera aun siendo desconocidos el resto de datos referentes a ella utilizando primitivas como *walk* o *get-bulk*. Esta es la razón principal del orden lexicográfico.

En resumen: la tabla se recorre columna a columna (de la primera a la última según el orden de los elementos de la *entry*), y el orden de recorrido de la columna lo marcan los índices. Es decir, se empieza por el índice más bajo y se recorre por orden hasta el más alto:

$$X.1.i_1 \rightarrow \ldots \rightarrow X.1.i_{M-1} \rightarrow X.2.i_1 \rightarrow \ldots \rightarrow X.2.i_{M-1} \rightarrow \ldots \rightarrow X.(N-1).i_1 \rightarrow \ldots \rightarrow X.(N-1).i_{M-1}$$

Siendo N el número total de columnas y M el número de filas.

En la figura 4.8 se incluye una explicación gráfica.

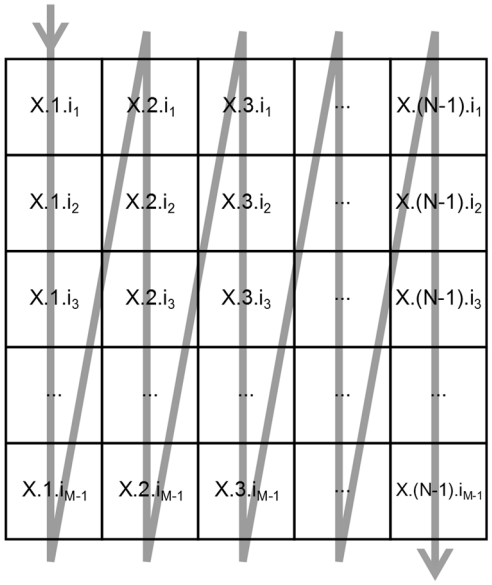


Figura 4.8. Recorrido de una tabla SNMP

Se observa que la línea gris translúcida recorre la tabla según el orden lexicográfico explicado en este apartado.

4.3.2. Consideraciones sobre el subagente

En este caso, se creará un subagente que inicializará y gestionará una estructura de tabla SNMP. Ésta consta de tres columnas, y simula una base de datos que contiene la lista de discos en una tienda de música

El número de serie de cada disco se encuentra reflejada en la columna *discosNumero*, que además actúa como índice de la tabla. Las dos columnas restantes, *discosTitulo* y *discosGrupo*, completan la información sobre cada disco.

Como se ha explicado en el apartado 4.3.1, se necesita un nodo de tipo *table*, sin contenido, del que *colgarán* uno o varios nodos (también sin contenido) de tipo *entry*, cada uno de los cuales corresponde a una fila. De esta manera queda definida la estructura de la tabla. A modo ilustrativo, se incluye el código relevante de la MIB en la figura 4.9 (ésta se puede consultar al completo en el anexo A3).

```
discosTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF DiscosEntry
    MAX-ACCESS  read-write
    …          ::= { subagent1MIBObjects 2 }
discosEntry OBJECT-TYPE
    SYNTAX     DiscosEntry
    INDEX   { discosNumero }
    …          ::= {discosTable 1 }
DiscosEntry ::= SEQUENCE {discosNumero Integer32, discosTitulo OCTET STRING, discosGrupo OCTET STRING}
discosNumero OBJECT-TYPE
    SYNTAX     Integer32
    …          ::= { discosEntry 1 }
discosTitulo OBJECT-TYPE
    SYNTAX     OCTET STRING
    …          ::= { discosEntry 2 }
discosGrupo OBJECT-TYPE
    SYNTAX     OCTET STRING
    …          ::= { discosEntry 3 }
```

Figura 4.9. Código de definición de *discosTable* en *SUBAGENT-1-MIB*

En este fragmento del código se puede observar cómo se definen cada uno de los elementos que componen la tabla: la fila (*entry*), como una estructura de objetos simples; el indice (*index*), asignado a un objeto escalar (*discosNumero*); la tabla, como una secuencia de estructuras fila (*DiscosEntry*).

En la figura 4.10 se muestra la estructura de árbol de la tabla, accesible desde el momento en el que se carga la MIB y obtenida mediante el comando *snmptranslate -M+. -mSUBAGENT-1-MIB -Tp -IR subagent1MIB*.

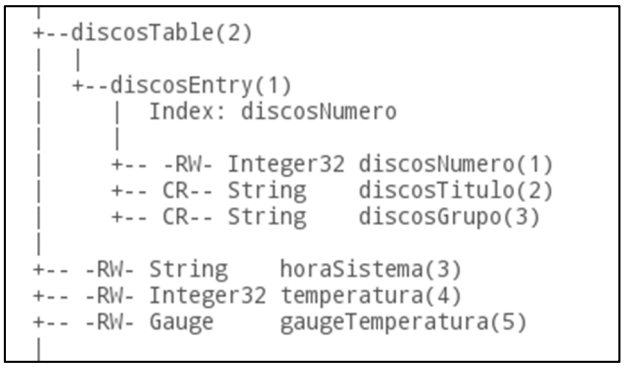


Figura 4.10. Resultado del comando *snmptranslate* para *discosTable*

Se puede observar que corresponde a la estructura definida por SUBAGENT-1-MIB (ver anexos A1 y A2).

4.3.3. Funcionamiento

Este es un caso más complejo que los del apartado 4.2, correspondientes a objetos escalares. Esto se debe a que el programa debe ser capaz de, además de gestionar las funciones propias de SNMP y AgentX de objetos simples y la actualización de un valor simple, manejar las funciones de tabla SNMP y su relación con una tabla de datos externa.

Los datos se deben poder obtener de una localización cualquiera dentro del SO del agente (archivos, I/O, *kernel*, etc.). Como ejemplo, se usa un archivo de texto en el que las filas están separadas por saltos de línea y las columnas por tabuladores. Este archivo podría a su vez ser utilizado como *buffer* en el que se introduzcan datos desde cualquier otro proceso o dispositivo.

La tabla SNMP se debe actualizar inmediatamente tras cualquier cambio realizado en el origen de los datos (archivo de texto en este caso), y debe estar en todo momento almacenada en orden lexicográfico (ver apartado 4.3.1), independientemente de cómo esté organizada la tabla origen de datos. Net-SNMP facilita esta tarea, tal y como se verá más adelante en este capítulo. Además, esta tabla SNMP debe ser de tamaño dinámico. Es decir, el *demon* del subagente AgentX también debe gestionar la creación y el borrado de filas.

4.3.4. Tareas iniciales de desarrollo

Como en los casos anteriores, se usa la aplicación *mib2c* para crear un *esqueleto* del código necesario. Este caso ha sido el más problemático en cuanto a la elección del archivo de configuración, ya que hay cuatro opciones principales en lo referente a tablas: *mib2c.mfd.conf*, *mib2c.create-dataset.conf*, *mib2c.iterate.conf* y *mib2c.table_data.conf*.

Finalmente, tras bastantes pruebas, la configuración que más se adapta a lo buscado en este proyecto es la proporcionada por el archivo *mib2c.table_data.conf* [16].

El método elegido, *table_data*, necesita punteros hacia cada elemento de la tabla, por lo tanto la ubicación de

los datos debe ser necesariamente conocida. Como se explicará más adelante, el funcionamiento de este método se basa en una cache interna de datos proporcionada por Net-Snmp que funciona como buffer intermedio entre los datos de la tabla externa y el objeto tabla SNMP.

Es momento de generar el código esqueleto en el que se basará el programa, con el comando *mib2c -c mib2c.table_data.conf SUBAGENT-1-MIB::discosTable*. En este punto, la aplicación mib2c pide ciertas decisiones al desarrollador, realizando varias preguntas. Es importante elegir la opción de habilitar la *cache*; esto hará que el código generado por mib2c contenga las funciones necesarias para extraer datos de tablas externas e introducirlos dentro de la tabla SNMP ordenadamente (según el orden lexicográfico).

El comando genera dos ficheros: *discosTable.c* y *discosTable.h*. Por lo tanto, también es trabajo del desarrollador proporcionar el código básico del *demon* y del *Makefile* correspondientes. En lo referente a estos dos últimos archivos, tan solo es necesario cambiar los nombres de las variables y funciones que se deben inicializar respecto a los de los apartados 4.2.1 o 4.2.2.

A partir de este momento, el grueso del trabajo se encuentra en modificar y ampliar convenientemente *discosTable.c*. En el tutorial de Net-SNMP se afirma que la modificación del código debería consistir principalmente en enlazar los datos externos de la tabla con la cache proporcionada por Net-SNMP.

Más concretamente, éste es el procedimiento que se debe desarrollar: existe una tabla almacenada en un archivo cualquiera, de la que hay que extraer todo su contenido y almacenarlo en una *estructura-buffer*; finalmente se almacenan los contenidos de esta estructura en la parte correspondiente de la estructura de *cache* proporcionada por Net-SNMP. Una serie de funciones proporcionadas por Net-SNMP se encargan de actualizar la tabla SNMP.

En los próximos apartados se describe en detalle el proceso completo.

4.3.5. Código y funciones principales

Para desarrollar estas funciones se ha utilizado principalmente la información aportada por el propio código de Net-SNMP y los mensajes de la *mailing list* del proyecto Net-SNMP, que contienen conversaciones aclaratorias entre desarrolladores y usuarios [17].

El objetivo principal que tiene el soporte de tablas externas al agente en Net-SNMP es la transferencia de toda la información contenida en la tabla externa al objeto gestionado tabla SNMP, en el cual las filas deben quedar ordenadas siguiendo el orden lexicográfico. Además, en este caso concreto, se exige la monitorización de la tabla externa de forma que cualquier cambio en ella debe ser reflejado inmediatamente en el objeto tabla SNMP; es decir, si se añade una fila en el archivo externo, ésta debe ser añadida al objeto tabla SNMP en su posición correcta según el orden lexicográfico, y si se modifica una de las filas existentes, el cambio se debe reflejar en la tabla SNMP.

La suite Net-SNMP proporciona una *cache* que funciona como estructura de intercambio de datos, a través de

la cual se transfiere la tabla almacenada en el archivo de texto a la tabla SNMP. En la figura 4.11 se muestra un esquema básico que refleja esto.

Figura 4.11. Esquema básico de carga de datos en una tabla SNMP

Se observa cómo se extraen los datos del buffer *tabla.txt*, se cargan en la cache y desde ésta se envían a la tabla SNMP *discosTable*.

Para realizar todo esto, Net-SNMP proporciona una serie de funciones que se detallan a continuación.

discosTable_load extrae los datos que llenarán la tabla SNMP desde cualquier ubicación que se le indique y les da el formato necesario para cargarlos en la *cache*. ***discosTable_createEntry*** *y* ***discosTable_removeEntry*** crean y eliminan nuevas filas en el objeto tabla SNMP, respectivamente. ***netsnmp_cache_create*** crea la *cache* a partir de los datos estructurados proporcionados por ***discosTable_load***. El contenido de esta *cache* es cargada de forma ordenada (según el orden lexicográfico) en la tabla SNMP *discosTable* por la función ***netsnmp_inject_handler***. Finalmente, la función ***discosTable_handler*** se encarga de la gestión de la estructura de la tabla frente a las primitivas SNMP (*GetRequest*, *SetRequest*, etc.).

Más en detalle, ***discosTable_load*** utiliza la función *discosTable_createEntry* para crear nuevas filas, extrayendo los valores de la tabla contenida en el archivo *tabla.txt* uno a uno y cargándolos en estructuras de tipo *discosTable_entry*. La función *discosTable_createEntry* devuelve una estructura de tipo *netsnmp_tdata_row*, que está definida por el código mostrado en la figura 4.12.

```
struct netsnmp_tdata_row_s {
    netsnmp_index oid_index;
    netsnmp_variable_list *indexes;
    void       *data;
}
```

Figura 4.12. Definición de la estructura *netsnmp_tdata_row*

La última parte de la estructura, *void *data*, contiene una fila *discosTable_entry* (que contiene a su vez los elementos de la fila), mientras que *netsnmp_index oid_index* y *netsnmp_variable_list *indexes* contienen el índice y la lista de variables de la fila. Esto da lugar a la estructura anidada que se muestra en la figura 4.13.

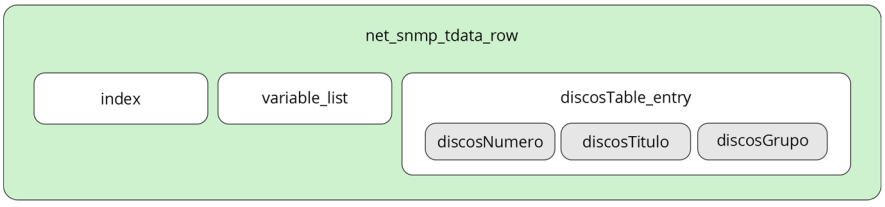


Figura 4.13. Estructuras anidadas de la cache de filas en Net-SNMP

Es decir, ***netsnmp_cache_create*** crea la *cache* a partir de los datos estructurados tipo *net_snmp_tdata_row* proporcionados por ***discosTable_load***. Finalmente, esta *cache* es cargada en la tabla SNMP *discosTable* por la función *netsnmp_inject_handler*.

La función que más trabajo de desarrollo implica es *discosTable_load*, la cual se encarga de cargar los datos de la tabla SNMP en la cache fila a fila. Su trabajo consiste en extraer y aislar los elementos columna de cada fila y asignarlos a su lugar correspondiente en la *cache*. A continuación, en la figura 4.14, se incluyen los fragmentos relevantes del código.

```
…
int discosTable_load( netsnmp_cache *cache, void *vmagic ) {
    netsnmp_tdata     *table = (netsnmp_tdata *)vmagic;
    netsnmp_tdata_row *row;
    struct discosTable_entry *this;
    FILE *fp;
    char buf[128];
    int i = 0;
    int numero_ind;
    char titulo[32];
    char grupo[32];
    fp = fopen( "tabla.txt", "r" );
    ...
    while ( fgets( buf, 128, fp )) {/* Ciclo para leer línea a línea el archivo de texto */
            sscanf(buf, "%d %s %s ", &numero_ind, titulo, grupo); /*Asignamos los valores columna*/
            /*El resto del ciclo carga los valores en sus lugares correspondientes*/
            row = discosTable_createEntry(table , numero_ind );
            this = row->data;
            while(i<32){
                    this->discosTitulo[i]=titulo[i];
                    this->discosGrupo[i]=grupo[i];
                    i++;}
            i=0;
    }
```

```
        fclose(fp);

        return 0;

}

...
```

Figura 4.14. Código de la función *discosTable_load*

En el ciclo *while* de la función se puede ver cómo se cargan los elementos fila de la tabla en la *cache*. Se lee el archivo *tabla.txt* fila a fila, de las cuales se extraen cada uno de los elementos columna. Finalmente, estos elementos se inyectan en una estructura de tipo *netsnmp_tdata_row* (ver figura 4.13).

En la figura 4.15 se muestra una evolución de la figura 4.11 teniendo en cuenta las funciones explicadas en este apartado.

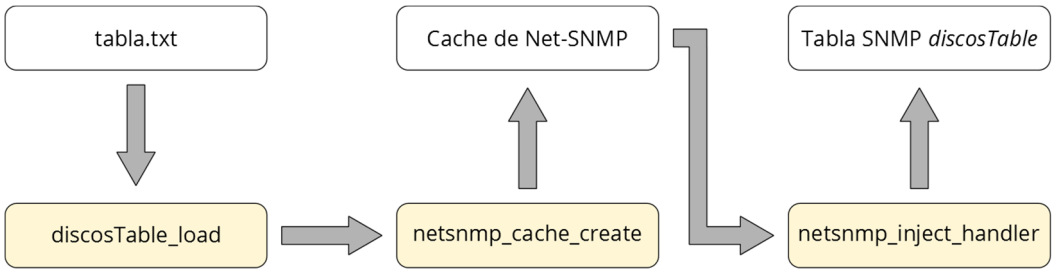


Figura 4.15. Esquema de carga de datos en una tabla SNMP

Se observan las funciones descritas a lo largo de este apartado y el flujo de datos implicado en la creación de un objeto tabla SNMP.

4.3.6. Puesta en marcha

Se inicia el soporte para AgentX como en los apartados anteriores (*sudo snmpd*) y se compilan los ejecutables del *demon* que soportará la tabla con el comando *sudo make discos_demon*.

Se inicia el *demon* con *sudo ./discos_demon*. El soporte de la tabla SNMP ya está en marcha.

4.3.7. Pruebas y resultados

A continuación se muestran los resultados de los comandos *snmptable* en local, para observar la tabla *discosTable* en un formato de tabla más intuitivo (figura 4.16) y de *snmpwalk* en el gestor remoto, para ilustrar el orden lexicográfico de tablas SNMP (figura 4.17).

```
pi@raspberrypi ~/subagentes_snmp/table_from_file $ snmptable -v 2c -c public -Os localhost discosTable
SNMP table: discosTable

 discosNumero           discosTitulo          discosGrupo
            4 "AmericanRecordingsIV"         "JohnnyCash"
           41    "DarkSideoftheMoon"          "PinkFloyd"
          129       "QueenOfDenmark"          "JohnGrant"
          233            "LeaveHome"            "Ramones"
          326      "Californication" "RedHotChiliPeppers"
pi@raspberrypi ~/subagentes_snmp/table_from_file $
```

Figura 4.16. Tabla presentada en formato *table*.

```
Microsoft Windows [Versión 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\adrian>snmpwalk -v 2c -c public 192.168.1.42 .1.3.6.1.4.1.8072.2.5..1.2

.1.3.6.1.4.1.8072.2.5..1.2:  (Sub-id not found: netSnmp -> .1.2)

C:\Users\adrian>snmpwalk -v 2c -c public 192.168.1.42 .1.3.6.1.4.1.8072.2.5.1.2
NET-SNMP-MIB::netSnmp.2.5.1.2.1.1.4 = INTEGER: 4
NET-SNMP-MIB::netSnmp.2.5.1.2.1.1.41 = INTEGER: 41
NET-SNMP-MIB::netSnmp.2.5.1.2.1.1.129 = INTEGER: 129
NET-SNMP-MIB::netSnmp.2.5.1.2.1.1.233 = INTEGER: 233
NET-SNMP-MIB::netSnmp.2.5.1.2.1.1.326 = INTEGER: 326
NET-SNMP-MIB::netSnmp.2.5.1.2.1.2.4 = STRING: "AmericanRecordingsIV"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.2.41 = STRING: "DarkSideoftheMoon"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.2.129 = STRING: "QueenOfDenmark"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.2.233 = STRING: "LeaveHome"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.2.326 = STRING: "Californication"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.3.4 = STRING: "JohnnyCash"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.3.41 = STRING: "PinkFloyd"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.3.129 = STRING: "JohnGrant"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.3.233 = STRING: "Ramones"
NET-SNMP-MIB::netSnmp.2.5.1.2.1.3.326 = STRING: "RedHotChiliPeppers"
```

Figura 4.17. Tabla presentada en orden lexicográfico, tal y como está almacenada en SNMP.

Ambas figuras muestran los mismos datos en diferente formato; en el apartado 4.3.1 se pueden consultar las bases del orden lexicográfico de almacenamiento de tablas.

Si se quiere cambiar el contenido de la tabla, basta con modificar el origen de los datos, el archivo *tabla.txt*. Una muestra, en la que se ve la tabla con una fila añadida y el resultado comparado con el anterior (nótese que las filas se han ordenado automáticamente en relación al índice), se presenta en la figura 4.18.

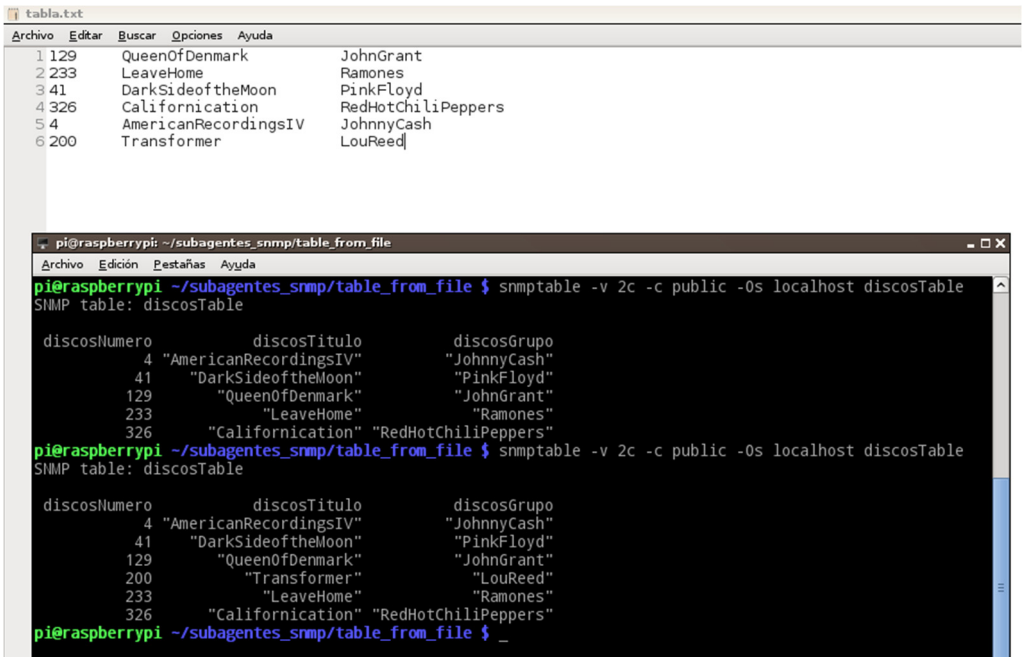


Figura 4.18. Tabla actualizada automáticamente respecto al contenido del origen de datos.

4.3.8. Problemas encontrados

En este caso no sólo ha existido una falta casi total de documentación al respecto, sino que además el código proporcionado por mib2c, del cual parte todo el proceso, tenía errores graves de partida.

Otra dificultad añadida es la de acceder al formato de las estructuras entre las que se realiza el intercambio de datos, ya que son volátiles y la depuración del código con *gdb* (depurador por defecto de GNU/Linux) desde la Raspberry Pi para este tipo de programas complejos no es sencilla.

Con esto no se pretende únicamente dejar patentes las dificultades, sino también proponer un consejo para desarrollar sistemas complejos de gestión *Net-SNMP* para *Raspberry Pi*: desarrollar los programas en un PC lo suficientemente potente (con depuración gráfica fluida) que corra bajo un sistema operativo GNU/Linux, de forma que tan solo sea necesario realizar la compilación del programa en la Raspberry Pi.

4.4. Manejo de *traps* asociados a objetos dinámicos

En este apartado, se desarrollará el código necesario para crear un subagente que soporte objetos gestionados de tipo *trap*, que a su vez estén asociados a otros objetos de tipo escalar y que envíen notificaciones al gestor cuando se cumplan ciertas condiciones fijadas.

4.4.1. Concepto

Como ya se ha descrito en el apartado 2.1.3.1, una de las PDUs de SNMP son las *traps*. Se trata de avisos que el agente envía a los gestores cuando se cumplen ciertas condiciones en los objetos gestionados monitorizados.

Estos mensajes, en la versión 2c de SNMP, transportan la siguiente información: el tiempo que lleva el sistema funcionando (*sysUpTime*), el identificador del objeto *trap* (OID) y, en el campo *variable bindings*, una serie de valores de objetos de la MIB que se considera necesario enviar al gestor. Se puede asociar el valor de cualquier objeto SNMP (como un umbral) a una *trap* para que el agente envíe una *trap* o notificación al gestor; de manera que se pueda monitorizar de forma automática.

A continuación se detalla el desarrollo completo de un escenario de monitorización de un objeto SNMP a través del envío de *traps*. Dicho escenario consta de un objeto escalar SNMP asociado a una variable que cambia con el tiempo, un sistema automático de aviso basado en *traps* SNMP v2c y un programa que simula la variación del objeto escalar gestionado.

En concreto, se trata de simular la gestión de un sensor de temperatura. Se ha creado un objeto que refleja el valor instantáneo de la temperatura que se está asociada a un objeto escalar SNMP (*temperatura*,

OID .1.3.6.1.4.1.8072.2.5.1.4). Este objeto escalar está a su vez asociado a un objeto de tipo *NOTIFICATION-TYPE*, *temperaturaTrap* (OID 1.3.6.1.4.1.8072.2.5.1.6.1). El subagente debe monitorizar el valor de la variable a través del objeto SNMP *temperatura* y encargarse de enviar avisos al gestor a través del agente maestro (ver figura 3.1) cuando su valor supere los 70ºC, en forma de *traps* de tipo *temperaturaTrap*. Estos *traps* deben informar del valor de la temperatura, función que se implementará asociando el objeto *temperatura* a *temperaturaTrap*.

En la figura 4.19 se incluye el extracto de la MIB *SUBAGENT-1-MIB* en la que se define el objeto de tipo *NOTIFICATION-TYPE* (se incluye también su objeto escalar asociado).

```
...
temperatura OBJECT-TYPE
    SYNTAX      Integer32
    …          ::= { subagent1MIBObjects 4 }
...
temperaturaTrap NOTIFICATION-TYPE
   STATUS  current
   OBJECTS {temperatura}
   DESCRIPTION
        "Trap asociado al objeto temperatura que representa la temperatura de la válvula."
   ::= { valvulaTraps 1 }
…
```

Figura 4.19. Definición de objetos de tipo *notify* y su escalar asociado en *SUBAGENT-1-MIB*

Se observa la asociación del *trap temperaturaTrap* con el objeto escalar *temperatura* (en el campo OBJECTS). En los siguientes apartados se detalla cómo se realiza esta asociación.

4.4.2. Funciones y desarrollo

A partir de estas premisas, es necesario crear tres procesos que funcionen en paralelo. A continuación se describen.

temp_demon es un proceso que gestiona un objeto de tipo escalar, ***temperatura***. Este objeto toma su valor en tiempo real de un archivo de texto, ***valor.txt***, que actúa de *buffer* de intercambio de datos. Se trata, por tanto, del mismo programa desarrollado en el apartado 4.2.2 (escalar asociado a una variable dinámica).

sim_temp es un programa que simula la variación de un valor de temperatura entre 20 y 100ºC de forma

periódica y lo inyecta en el archivo *valor.txt* (*buffer*) cada 5 segundos. Está compilado a partir del código de *sim_temp.c*, mostrado en la figura 4.20.

```
int main()
{   int i=20;   /*Temperatura de partida*/
    int j=0;
    FILE *fp;
    fp = fopen("valor.txt","w+");
    fprintf(fp, "%d", i);
    while(1){
       while(i<100){
                rewind(fp);
                fprintf(fp, "%d", i);
                i=i+5;
                sleep(5);
                j++;
       }
       while(i>20){
                rewind(fp);
                fprintf(fp, "%d\n", i);
                i=i-5;
                sleep(5);
                j++;
          }
    }
    fclose(fp)
}
```

Figura 4.20. Código de *sim_temp*

Se puede observar en la figura que la función incrementa el valor en pasos de 5ºC cada cinco segundos y lo inyecta en el archivo *valor.txt*, hasta que alcanza un valor de 100ºC. A partir de aquí, realiza el proceso inverso hasta alcanzar de nuevo los 20ºC, momento en el cual comienza el ciclo desde el principio.

El único propósito de este programa es comprobar el funcionamiento correcto de los objetos SNMP, y hacerlo

en un periodo de tiempo controlado y razonable. Si se quisiera hacer una simulación más compleja, basta con implementar la función deseada; incluso se puede asociar el valor de temperatura a un sensor real o a los datos enviados por un equipo de medición remoto.

trap_demon define y gestiona todas las funciones SNMP y AgentX asociadas a la *trap*; además, se encarga de su envío cuando la temperatura es mayor de 70ºC. Está enlazado al objeto *temperatura* como se observa en el extracto de la MIB de la figura 4.20.

Como en los apartados anteriores, se parte del "código esqueleto" generado por mib2c con el archivo de configuración apropiado, que se obtiene con el comando *sudo mib2c -c mib2c.notify.conf SUBAGENT-1-MIB::temperaturaTrap*. Es decir, se crea una estructura de código que maneja un objeto de tipo *notify*, correspondiente al nodo *temperaturaTrap* de la MIB *SUBAGENT-1-MIB*. De nuevo, se obtienen dos "archivos esqueleto": *temperaturaTrap.c* y *temperaturaTrap.h*. Este código tal cual es creado por *mib2c* proporciona además el soporte del envío de *traps* desde un subagente AgentX a través de un agente maestro SNMP, pero no realiza el propio envío. Debe ser el desarrollador el que cree el código dentro del *demon* que realice este envío del *trap* según las condiciones deseadas; además de inicializar las funciones necesarias referidas al soporte del objeto *trap* gestionado.

Se incluye los fragmentos más relevantes del código de *temperaturaTrap* en la figura 4.21.

```
int
send_temperaturaTrap_trap( void )
{
    netsnmp_variable_list  *var_list = NULL;
    oid temperaturaTrap_oid[] = { 1,3,6,1,4,1,8072,2,5,1,6,1 };
    oid temperatura_oid[] = { 1,3,6,1,4,1,8072,2,5,1,4, 0 };
    int temp=0;
    /*Lee el valor de temperatura de valor.txt y lo extrae*/
    FILE * fp;
    fp = fopen ("valor.txt", "r");
    if (fscanf(fp, "%d", &temp)==1){
        printf("%d",temp);
        }else{
        printf("Error al leer la temperatura.\n");
    }
```

```
snmp_varlist_add_variable(&var_list, snmptrap_oid, OID_LENGTH(snmptrap_oid), ASN_OBJECT_ID,
temperaturaTrap_oid, sizeof(temperaturaTrap_oid));

  …

snmp_varlist_add_variable(&var_list, temperatura_oid, OID_LENGTH(temperatura_oid), ASN_INTEGER,
(u_char*)&temp, sizeof(temp));

send_v2trap( var_list );
```

Figura 4.21. Código relevante de *temperaturaTrap*

Se puede observar que *temperaturaTrap* tiene como función principal crear el formato del mensaje trap y proporcionar las funciones necesarias para su envío. En primer lugar se definen las OIDs del trap y del valor de su objeto asociado (nótese que accede al valor de *temperatura* -1.3.6.1.4.1.8072.2.5.1.4.0-, no al objeto *temperatura* -1.3.6.1.4.1.8072.2.5.1.4.-; ver apartado 2.1.3.2), posteriormente se lee el valor de temperatura desde el archivo *valor.txt*. A continuación, se inicializan las variables asociadas al objeto *trap* con la función *snmp_varlist_add_variable*, siendo éstas la OID referida al valor del objeto *temperatura* y el propio valor. Finalmente, la función *send_v2trap* se encarga del envío de los *traps* que transportan las variables asociadas anteriormente inicializadas.

Para que se realice el envío de traps, se añade el fragmento de código que se muestra en la figura 4.22 al archivo *trap_demon.c*.

```
/*Envía un trap SNMP v2c cuando el valor de temperatura es superior a 70ºC*/

if(temp>70){

        send_temperaturaTrap_trap(); /*Envía un trap cada 5 segundos*/

        sleep(5);

}
```

Figura 4.22. Código de envío de *traps.*

Se puede observar que la función *send_temperaturaTrap_trap*, mostrada en la figura 4.21, es invocada cada 5 segundos siempre que el valor de la temperatura extraído del archivo de texto supera los 70ºC. La frecuencia de envíos se puede cambiar según lo deseado.

4.4.3. Puesta en marcha

Como primer paso, es necesario compilar los ejecutables de los programas anteriormente descritos, con los comandos sudo *make temp_demon*, *sudo make trap_demon*, *cc -o sim_temp sim_temp.c*.

Para que los *traps* se envíen a los gestores remotos, hay que habilitar las IPs de los mismos en el archivo */etc/snmp/snmpd.conf,* en el apartado *ACTIVE MONITORING* (ver anexo A3).

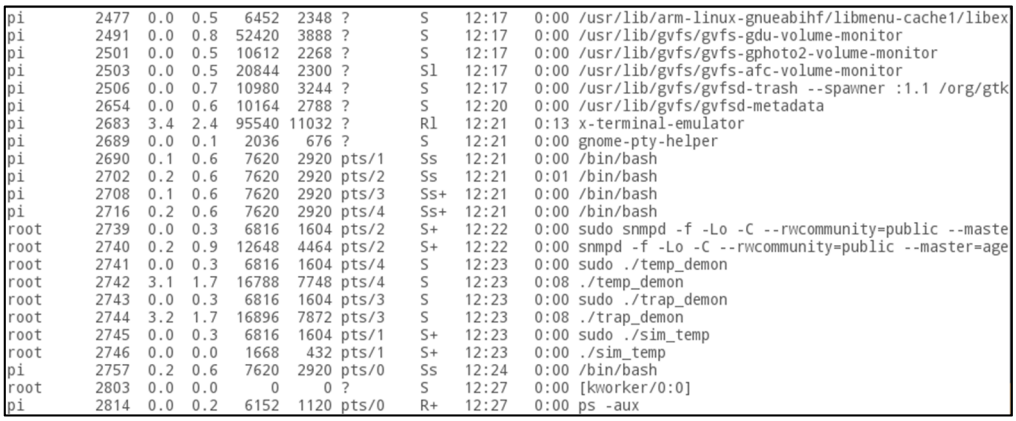
Después, se debe reiniciar la aplicación *Net-SNMP* para que los cambios en *snmpd.conf* se hagan efectivos, con el comando *sudo service snmpd restart*. Finalmente, como en los apartados anteriores, iniciamos el soporte para AgentX, los *demons* (comandos *sudo ./temp_demon* y *sudo ./trap_demon*) y el programa de simulación del sensor de temperatura, con el comando *sudo ./sim_temp.*

4.4.4. Pruebas y resultados

Con los tres procesos funcionando simultáneamente, se comprueba el funcionamiento del subagente en las condiciones más reales posibles. Como hasta el momento, el equipo gestor será un PC con sistema operativo Windows 8.1. De nuevo, se recuerda que se pueden obtener los mismos resultados en cualquier dispositivo con una aplicación gestora que proporcione funciones SNMP V2c.

Como novedad, en esta ocasión se utiliza la aplicación SnmpB (ver apartado 3.1.3), para observar en un entorno gráfico la relación entre la temperatura simulada y el envío de *traps*.

Se comprueba que los procesos necesarios están funcionando con el comando ps –aux, cuya salida se muestra en la figura 4.23.



```
pi        2477  0.0  0.5   6452  2348 ?        S    12:17   0:00 /usr/lib/arm-linux-gnueabihf/libmenu-cache1/libex
pi        2491  0.0  0.8  52420  3888 ?        S    12:17   0:00 /usr/lib/gvfs/gvfs-gdu-volume-monitor
pi        2501  0.0  0.5  10612  2268 ?        S    12:17   0:00 /usr/lib/gvfs/gvfs-gphoto2-volume-monitor
pi        2503  0.0  0.5  20844  2300 ?        Sl   12:17   0:00 /usr/lib/gvfs/gvfs-afc-volume-monitor
pi        2506  0.0  0.7  10980  3244 ?        S    12:17   0:00 /usr/lib/gvfs/gvfsd-trash --spawner :1.1 /org/gtk
pi        2654  0.0  0.6  10164  2788 ?        S    12:20   0:00 /usr/lib/gvfs/gvfsd-metadata
pi        2683  3.4  2.4  95540 11032 ?        Rl   12:21   0:13 x-terminal-emulator
pi        2689  0.0  0.1   2036   676 ?        S    12:21   0:00 gnome-pty-helper
pi        2690  0.1  0.6   7620  2920 pts/1    Ss   12:21   0:00 /bin/bash
pi        2702  0.2  0.6   7620  2920 pts/2    Ss   12:21   0:01 /bin/bash
pi        2708  0.1  0.6   7620  2920 pts/3    Ss+  12:21   0:00 /bin/bash
pi        2716  0.2  0.6   7620  2920 pts/4    Ss+  12:21   0:00 /bin/bash
root      2739  0.0  0.3   6816  1604 pts/2    S+   12:22   0:00 sudo snmpd -f -Lo -C --rwcommunity=public --maste
root      2740  0.2  0.9  12648  4464 pts/2    S+   12:22   0:00 snmpd -f -Lo -C --rwcommunity=public --master=age
root      2741  0.0  0.3   6816  1604 pts/4    S    12:23   0:00 sudo ./temp_demon
root      2742  3.1  1.7  16788  7748 pts/4    S    12:23   0:08 ./temp_demon
root      2743  0.0  0.3   6816  1604 pts/3    S    12:23   0:00 sudo ./trap_demon
root      2744  3.2  1.7  16896  7872 pts/3    S    12:23   0:08 ./trap_demon
root      2745  0.0  0.3   6816  1604 pts/1    S+   12:23   0:00 sudo ./sim_temp
root      2746  0.0  0.0   1668   432 pts/1    S+   12:23   0:00 ./sim_temp
pi        2757  0.2  0.6   7620  2920 pts/0    Ss   12:24   0:00 /bin/bash
root      2803  0.0  0.0      0     0 ?        S    12:27   0:00 [kworker/0:0]
pi        2814  0.0  0.2   6152  1120 pts/0    R+   12:27   0:00 ps -aux
```

Figura 4.23. Procesos activos, se observan los *demons* y el soporte para SNMP en funcionamiento.

Se puede observar que todos los procesos necesarios están en marcha (soporte de subagentes AgentX, *temp_demon*, *trap_demon*, y *sim_temp*).

En la figura 4.24 se muestra la función del valor *temperatura* que genera el programa de simulación de temperatura *sim_temp* en una gráfica obtenida con el programa SnmpB.

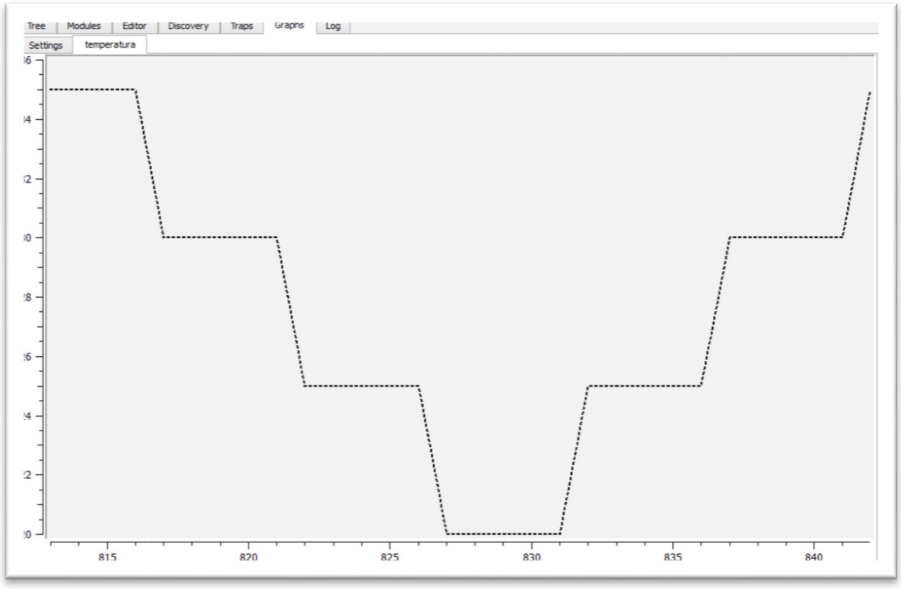


Figura 4.24. Gráfica correspondiente al valor del objeto *temperatura*.

Se puede comprobar que corresponde a la función definida en el apartado 4.4.2 (ver figura 4.20).

Finalmente se comprueba el envío correcto de *traps* utilizando la aplicación SnmpB, la cual muestra los resultados recogidos en la figura 4.25.

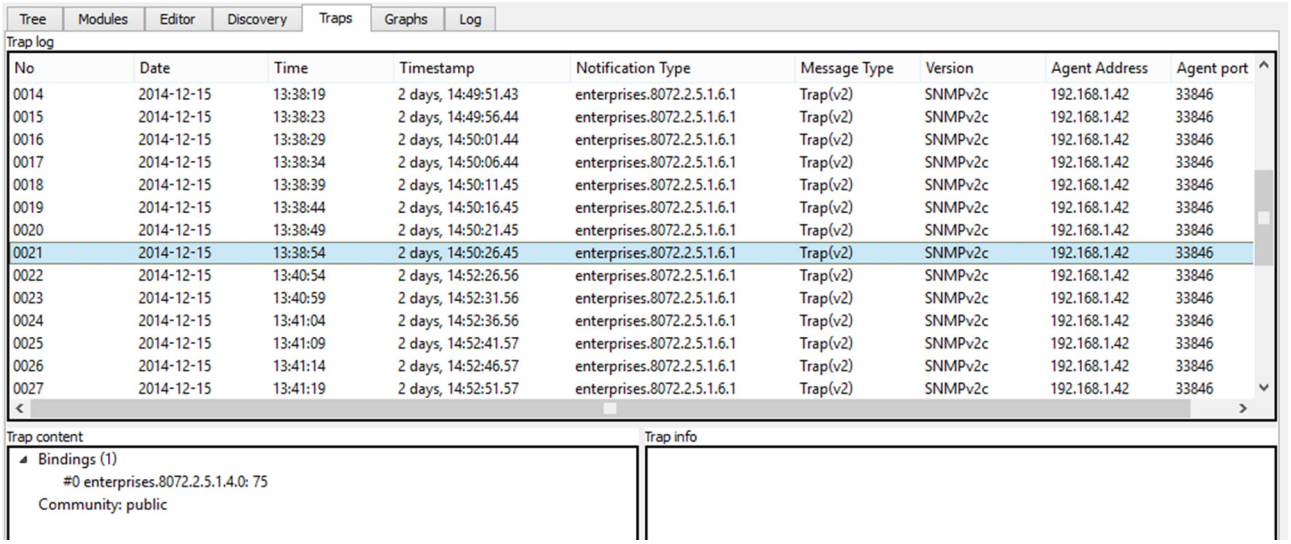


Figura 4.25. Captura de *traps* en SnmpB.

Se puede ver que se realiza un envío cada 5 segundos (campo *Timestamp*) y, en el caso concreto seleccionado en la figura 4.25, la *trap* corresponde a un valor de temperatura de 75ºC (ver campo *Trap content*, *Variable Binding* correspondiente a la OID del objeto *temperatura*). En este momento la función de simulación de temperatura está en descenso, por lo que el siguiente valor será de 70ºC y el agente dejará de enviar *traps* hasta que se superen de nuevo los 70ºC en el siguiente ciclo de subida (ver figura 4.22). En el campo *timestamp* se observan los dos minutos de pausa en el envío, que corresponden a los valores menores o iguales a 70ºC proporcionados por el programa *sim_temp*.

5. Desarrollo del prototipo

En este capítulo se aúna lo desarrollado hasta el momento en este proyecto, y se utiliza en una simulación de un sistema completo inspirado en la realidad.

En concreto, se implementará un prototipo de sistema de gestión de combustible para un generador diésel que alimenta una sala de servidores sin acceso a la red eléctrica; este sistema está compuesto por una serie de funciones que simulan los datos generados por un generador ficticio y por los servidores que alimenta, así como por un agente extendido SNMP. Este agente, a partir de los datos anteriores, se encarga de las funciones necesarias para garantizar que el generador entrega la potencia necesaria y tiene el suficiente combustible como para mantener la sala de servidores en marcha. Existe un programa de control (externo a SNMP) que proporciona datos clave al agente para que éste se encargue de informar (con envío de *traps*) al gestor de la falta de potencia o de la necesidad de rellenar el depósito de combustible.

5.1. Definición y funciones del prototipo

Se trata de un sistema de gestión SNMP completo de un generador diésel ficticio, cuyas principales misiones son gestionar el rellenado del depósito, vigilar el consumo de potencia instantánea del conjunto de los servidores y proporcionar al gestor diversos datos del sistema en tiempo real. En la figura 5.1 se muestra un esquema básico de funcionamiento.

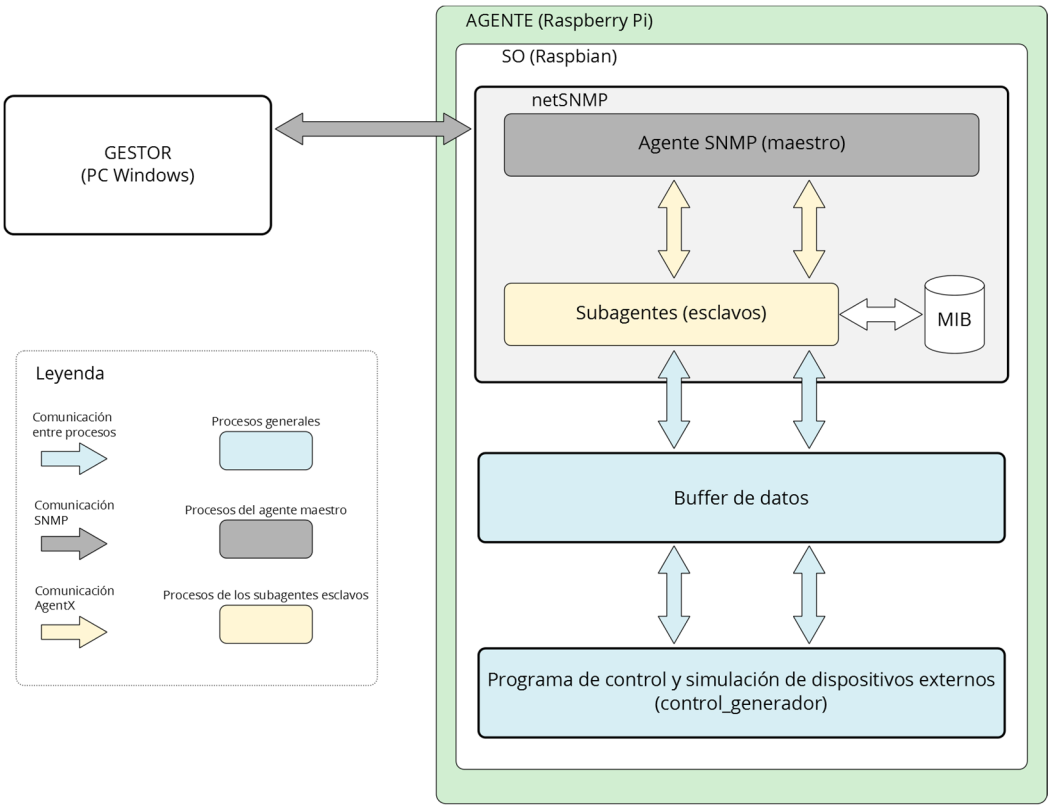


Figura 5.1. Esquema básico del prototipo.

En la figura 5.1 se observan los bloques lógicos del agente. La comunicación entre el gestor y el agente (a través del protocolo *SNMP*) es proporcionada por el software de *Net-SNMP*, al igual que la existente entre el agente maestro y los subagentes esclavos *AgentX* (a través del protocolo *AgentX*). Además, se observa que los subagentes toman su estructura de datos de una MIB contenida en el agente. Finalmente, existen funciones externas a *NetSNMP*, que son las que proporcionan los datos a los subagentes desde un *buffer* de datos que está conectado a un *stream* de información proporcionado por el dispositivo virtual que simula el dispositivo real a gestionar.

La tarea a partir de este momento es implementar este sistema de gestión autónomo en una *Raspberry Pi*, desarrollando un subagente *AgentX* y los programas de control y manipulación de datos necesarios.

5.2. El subagente y las funciones asociadas

En este apartado se propone una implementación de las funciones descritas anteriormente sobre un subagente *AgentX*. Para ello, se debe formar una base de información de gestión (MIB) que contenga los objetos gestionados y desarrollar las funciones necesarias del subagente y de obtención y manipulación de datos.

5.2.1. MIB del subagente (GENSERV-MIB)

El primer paso es crear una MIB que soporte los objetos gestionados. Tras analizar el caso, se ha decidido que son necesarios los siguientes:

- ***combustible***: Un objeto escalar que representa el combustible restante en el depósito.
- ***serversTable***: Una tabla con la información de consumo energético de todos los servidores en funcionamiento. Cada vez que se conecte o se desconecte un servidor la tabla es actualizada en tiempo real. Esta base de datos tiene dos funciones: informar al operador humano que desee consultarla y la obtención en tiempo real del consumo de potencia eléctrica de cada uno de los servidores.
- ***ptotal***: Un escalar que representa la potencia instantánea total consumida por todos los servidores, obtenido a partir de los datos de la tabla.
- ***frelleno***: Un factor de relleno que representa la urgencia de llenar el depósito, relacionado con el combustible restante y con la potencia consumida.
- ***trapcombustible***: Un *trap* que se encarga de notificar al gestor cuándo se llega a un nivel crítico de combustible y es necesario rellenar el depósito.
- ***trappotencia***: Un *trap* que se encarga de notificar al gestor que la potencia consumida se acerca al límite

de potencia instantánea que puede producir el generador.

En la figura 5.2 se puede observar la estructura de árbol de la MIB GENSERV-MIB. Además, el código de la MIB está incluido al completo en el anexo A5.

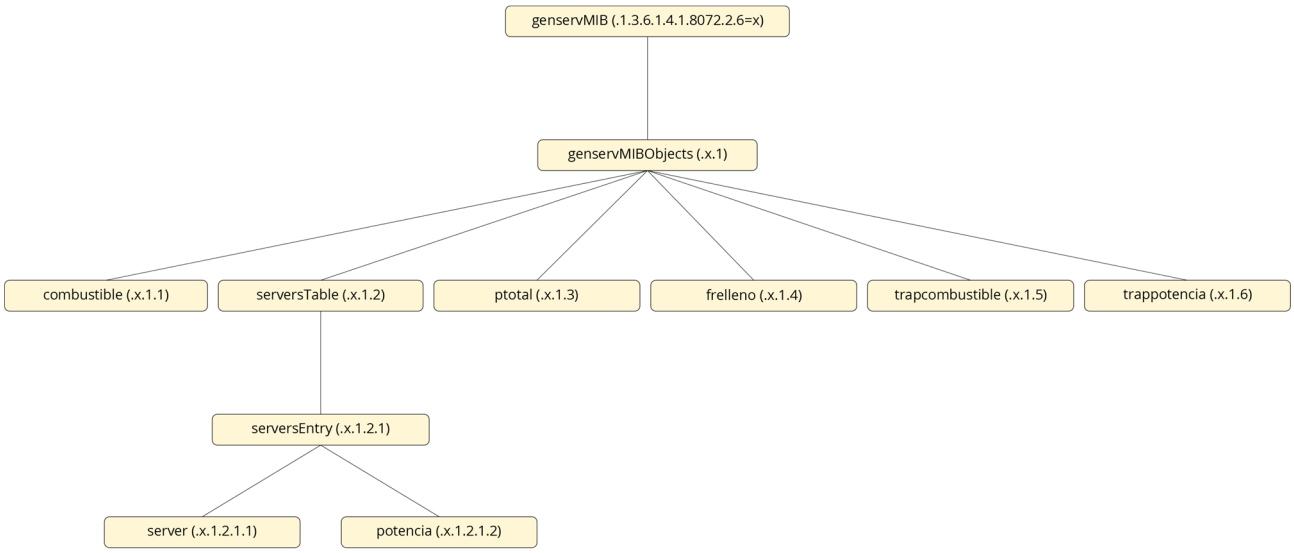


Figura 5.2. Esquema en árbol de *GENSERV-MIB*.

Hay que tener en cuenta que los únicos datos conocidos por el agente son el del nivel de combustible (se supone la existencia de un sensor que envía el dato periódicamente) y la tabla de potencia de los servidores activos (de nuevo, se supone que es enviada al agente). Por lo tanto, el subagente AgentX se debe encargar de obtener y manejar el resto de datos necesarios.

El siguiente paso es la carga de la MIB, proceso análogo al realizado en el apartado 3.2.4. (Carga de una MIB). Tras codificar la MIB en formato ASN-1, se carga en los directorios */usr/share/snmp/mibs* y */home/pi/.snmp/mibs*. Finalmente, se reinicia el servicio *snmpd* para que el sistema reconozca la estructura de la MIB. En la figura 5.3 se observa la estructura de la MIB *GENSERV-MIB*.

```
pi@raspberrypi ~ $ snmptranslate -M+. -mGENSERV-MIB -Tp -IR genservMIB
+--genservMIB(6)
   |
   +--genservMIBObjects(1)
      |
      +-- -RW- Integer32 combustible(1)
      |
      +--serversTable(2)
      |  |
      |  +--serversEntry(1)
      |     |  Index: server
      |     |
      |     +-- CR-- String    server(1)
      |     +-- -RW- Integer32 potencia(2)
      |
      +-- -RW- Integer32 ptotal(3)
      +-- -RW- Integer32 frelleno(4)
      +--trapcombustible(5)
      +--trappotencia(6)
```

Figura 5.3. Captura de la estructura de la MIB del prototipo.

Una vez cargada la MIB se deben desarrollar los programas que soporten los objetos gestionados y que, además, manejen y calculen los datos y *buffers* necesarios para el funcionamiento del prototipo.

5.2.2. Aplicaciones del subagente

En este apartado se definen todos los programas necesarios que deben ser desarrollados en el subagente. Algunos de ellos son análogos a los desarrollados en el capítulo 4 de este proyecto y otros introducen funciones nuevas.

5.2.2.1. Introducción

En la figura 5.4 se presenta el esquema más detallado del funcionamiento del prototipo, ampliando el mostrado en la figura 5.1.

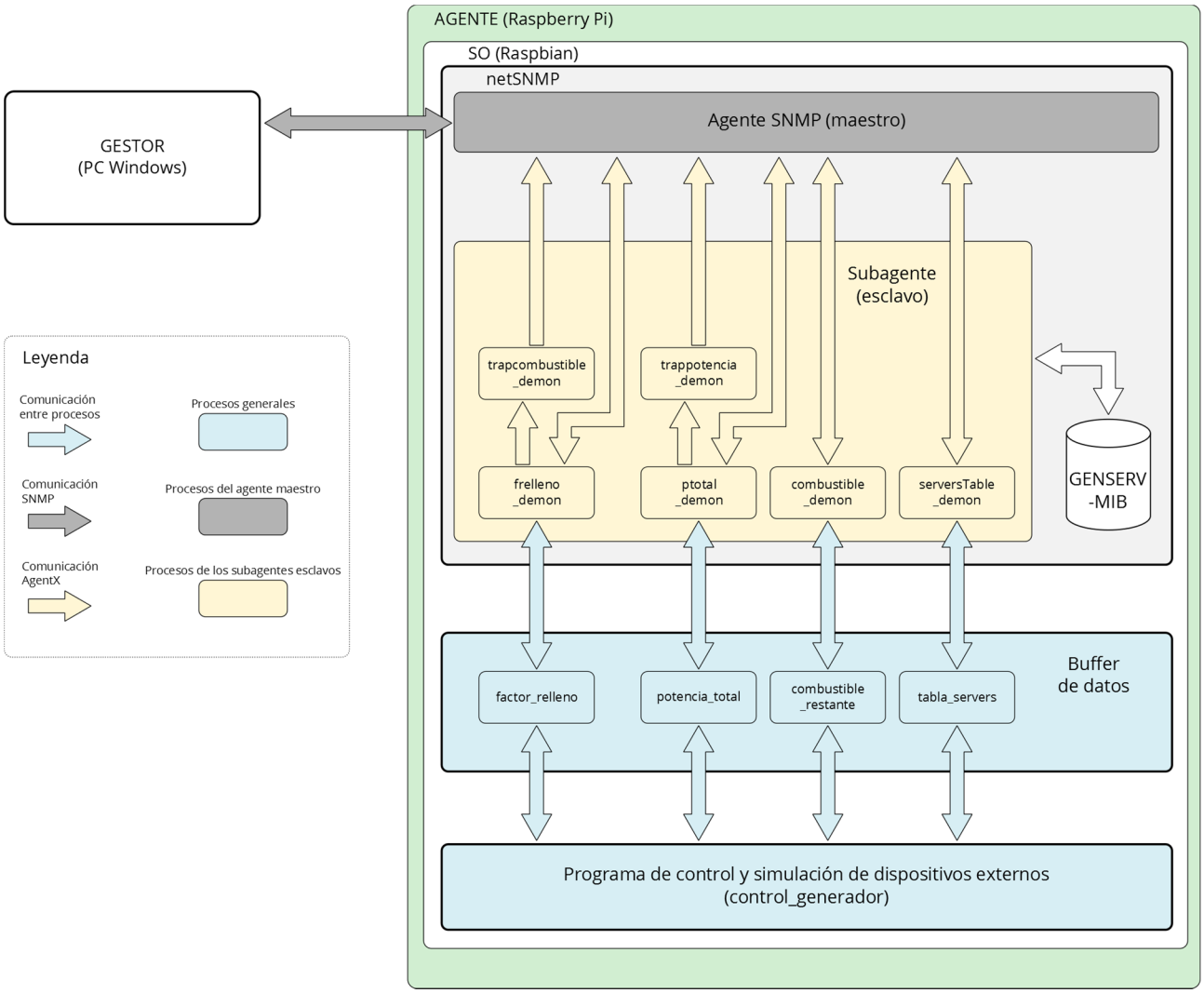


Figura 5.4. Esquema funcional detallado del prototipo.

Se observa que los subagentes esclavos (bloque amarillo) se encargan del soporte de la información de gestión, contenida en la MIB (*GENSERV-MIB*). En este bloque funcional se sitúan varias funciones. ***serversTable_demon*** soporta el objeto gestionado *serversTable*; se trata de una tabla actualizable en tiempo real y que toma sus valores de un archivo externo, *tabla_servers.txt*, que actúa como buffer de datos. Por su parte, ***combustible_demon*****,** ***ptotal_demon*** y ***frelleno_demon*** soportan los objetos escalares enteros *combustible*, *potencia_total* y *factor_relleno;* son actualizables en tiempo real y toman sus valores de los archivos de texto *combustible_restante.txt*, *potencia_total.txt* y *factor_relleno.txt*. ***trapcombustible*** **y** ***trappotencia*** soportan los *traps* asociados a los escalares *frelleno* y *ptotal*; se encargan de manejar el envío de *traps* en ciertas condiciones asociadas con los valores de dichos escalares (ver apartado 5.3.2.5).

Los bloques de color azul de la figura corresponden a las funciones y datos externos a los subagentes. El programa ***control_generador*** realiza las operaciones externas a los subagentes AgentX, y simula los datos de control de un generador ficticio. Por su parte, los archivos *factor_relleno*, *potencia_total*, *combustible_restante* y *tabla_servers* funcionan como buffers de intercambio de datos entre el programa *control_generador* y los programas del subagente AgentX correspondientes (*frelleno_demon*, *ptotal_demon*, *combustible_demon* y *serversTable_demon*, respectivamente).

5.2.2.2. Aplicación tablaServers_demon

Esta aplicación se encarga del soporte del objeto *serversTable*, una tabla con dos columnas que contienen el nombre de cada servidor y su potencia correspondiente.

Todo el proceso de desarrollo es análogo al del apartado 4.3. (Manejo de tablas), por lo que aquí se tratarán principalmente los cambios introducidos. La estructura del código se obtiene utilizando la aplicación *mib2c,* con el archivo de configuración *mib2c.table_data.conf*. El comando concreto utilizado es *$ mib2c -c mib2c.table_data.conf GENSERV-MIB::serversTable*.

Una vez obtenido el *esqueleto* del código de *serversTable.c*, el archivo principal creado por *mib2c*, se debe completar con ajuste a las necesidades concretas de este caso. En este punto se produce la novedad más significativa respecto a la tabla de ejemplo del apartado 4.3: el índice de la tabla es en este caso un *string* de caracteres, por lo que el proceso de carga de los datos en la cache de NetSNMP debe ser modificado en consecuencia. La función significativa en lo referente a este asunto es *serversTable_load*, cuyo código relevante se puede observar en la figura 5.5.

```
...
serversTable_load( netsnmp_cache *cache, void *vmagic ) {
    netsnmp_tdata     *table = (netsnmp_tdata *)vmagic;
    netsnmp_tdata_row *row;
```

```
    struct serversTable_entry *this;
    FILE *fp;
    char buf[128];
    int i = 0;
    char server_ind[32];
    int potencia_temp;

    fp = fopen( "tabla_servers.txt", "r" );
    if ( !fp ) { return -1; }
   while ( fgets( buf, 128, fp )) {
       sscanf(buf, "%s %d", server_ind, &potencia_temp);
       row = serversTable_createEntry(table, server_ind, strlen(server_ind));
       this = row->data;
       this->potencia=potencia_temp;
    }
    fclose(fp);
    return 0;  /* OK */
}
…
```

Figura 5.5. Fragmento de código de la función *serversTable_load*

Esta función se encarga de cargar secuencialmente los contenidos del archivo *tabla_servers.txt* en la cache de NetSNMP extrayendo las filas una a una, diferenciando y extrayendo cada elemento de la fila y, finalmente, cargando estos elementos en su lugar correspondiente de la cache. Como se vio en el apartado 4.3.5, la variable *data* del código, que se encuentra dentro de la estructura *row*, corresponde a la subestructura donde se deben alojar los datos de cada fila de la cache. Una vez completado el código, se debe compilar utilizando el comando *sudo make tablaServers_demon*. Con esto se obtiene finalmente la aplicación *tableServers_demon*, que soporta el objeto gestionado tal y como se pretendía.

5.2.2.3. Aplicaciones combustible_demon, ptotal_demon y frelleno_demon

Estás tres funciones soportan los objetos escalares *combustible*, *potencia_total* y *factor_relleno, respectivamente.* Los valores de estos objetos están almacenados en los archivos de texto

combustible_restante.txt, *potencia_total.txt* y *factor_relleno.txt*, que actúan como *buffers* de datos. Al modificar el contenido de los *buffers*, el objeto SNMP se actualiza instantáneamente.

Para obtener el código necesario se utilizan los comandos *mib2c -c mib2c.scalar.conf GENSERV-MIB::combustible*, *mib2c -c mib2c.scalar.conf GENSERV-MIB::potencia_total* y *mib2c -c mib2c.scalar.conf GENSERV-MIB::factor relleno*, que respectivamente, obtienen el código de los *demons combustible_demon*, *ptotal_demon* y *frelleno_demon*.

Una vez compilado el código, se obtienen los programas *combustible_demon*, *ptotal_demon* y *frelleno_demon*, que están asociados respectivamente con los archivos buffer *combustible_restante.txt*, *potencia_total.txt* y *factor_relleno.txt*. De estos archivos toman los valores asociados a sus variables de forma periódica; los valores contenidos en los archivos de texto pueden ser modificados en cualquier momento y los programas se encargan de actualizar el valor de las variables automáticamente. En la figura 5.4 se puede ver como se enmarcan estas funciones en el conjunto del prototipo.

5.2.2.4. Aplicación control_generador

Este programa es responsable de aportar la *inteligencia* a las funciones de gestión del subagente. Entre sus tareas se encuentran la lectura del archivo *tabla_servers.txt* (que contiene la tabla de potencias de los servidores conectados) y la obtención del valor de potencia total consumida (escribe el valor de potencia en el archivo *potencia_total.txt*); el cálculo de la velocidad con la que se consume el combustible y la simulación del propio consumo, actualizando periódicamente el buffer *combustible_restante.txt.* La tarea más compleja de la que se encarga es el cálculo del factor de relleno, desarrollada más adelante en este apartado. Todas estas funciones están representadas en la figura 5.4.

Con la intención de simplificar los cálculos, se supone un generador con las siguientes características: Una potencia máxima efectiva entregada de 10KW, un depósito con una capacidad de 100l y una velocidad de consumo de combustible representada por un factor que varía entre 1 (motor al ralentí) y 10 (motor a máximo funcionamiento, generando los 10KW máximos).

El cálculo de la potencia total máxima se realiza accediendo al archivo *tabla_servers.txt* y sumando los valores de la columna correspondiente (la segunda, donde están alojados los valores de potencia consumida de cada servidor). En la figura 5.6 se muestra el extracto de código relevante del programa.

```
...
        f_servers = fopen("tabla_servers_demon/tabla_servers.txt","r");
         potencia_total=0;
         while ( fgets( buf, 128, f_servers )){                    // Se lee línea a línea hasta fin del archivo.
```

```
            potencia_server=0;
            sscanf(buf, "%s %d", nombre_server, &potencia_server);   // Se extraen las columnas una a una.
            random=rand() % 40;
            potencia_server=(6*potencia_server/10)+(potencia_server*random/100;
            potencia_total=potencia_total+potencia_server;}
    fclose(f_servers);
    f_potencia = fopen("ptotal_demon_Trap/potencia_total.txt","w");
    fprintf(f_potencia, "%d", potencia_total);
    fclose(f_potencia);
    …
```

Figura 5.6. Código de la potencia total.

Para hacer la simulación más realista, se ha supuesto que el consumo de los servidores varía con el tiempo, debido a cambios en la carga de trabajo. Para simular esta situación se ha utilizado un número pseudoaleatorio, de forma que la potencia que entregan los servidores varía entre un 60% y un 100% de la potencia máxima que pueden consumir.

El valor de la potencia total consumida está asociado al envío de un *trap*, que se describe en el siguiente apartado (5.2.2.5).

Se puede suponer que la velocidad de consumo de combustible es directamente proporcional a la potencia instantánea de los servidores conectados; la simulación de consumo de combustible se realiza restando al combustible restante de forma periódica la cantidad que se consume en dicho periodo (obtenida de la velocidad de consumo). En la figura 5.7 se puede consultar el extracto de código relevante.

```
…
/*Cálculo de la velocidad de consumo de combustible.
Nota: Se acelera el consumo real que podría tener un generador por exigencias de la simulación.
Se supone que, a máxima potencia de funcionamiento (10KW), se consumen 10l/minuto, y a mínima potencia (0W), 1l/minuto (debido al ralentí).*/
      if(potencia_total>1000){
         velocidad_consumo=potencia_total/1000;}
         else{
            velocidad_consumo=1;
      }
```

```
/*Se simula el consumo de combustible decrementando el valor alojado en combustible_restante.txt*/
f_combustible = fopen("combustible_demon/combustible_restante.txt","w+");
 fscanf(f_combustible, "%d", &combustible_restante);
 …
 if (combustible_restante<=velocidad_consumo){
    combustible_restante=0;}   //Esto evita valores negativos de combustible.
    else{
        combustible_restante=combustible_restante-(velocidad_consumo);
    }
 fprintf(f_combustible, "%d", combustible_restante);
 fclose(f_combustible);
 …
```

Figura 5.7. Código correspondiente a la velocidad de consumo y al consumo de combustible.

Se observa que, básicamente, el programa extrae el valor del combustible restante del archivo *combustible_restante.txt*, resta a ese valor los litros proporcionados por el valor de la velocidad de consumo, y reescribe *combustible_restante.txt* con el valor obtenido.

A partir de los datos anteriores se obtiene el factor de relleno, que representa de forma cuantificable la urgencia en rellenar el depósito. Se ha creado un algoritmo partiendo de la tabla de la figura 5.8, que representa los valores del factor de relleno en relación al combustible restante en el depósito del generador y a la velocidad de consumo del mismo.

velocidad de consumo (l/min) \ combustible_restante (l)	100	90	80	70	60	50	40	30	20	10	
10	3	4	5	6	7	8	9	10	10	10	Tramo 1
9	3	4	5	6	7	8	9	10	10	10	
8	2	3	4	5	6	7	8	9	10	10	Tramo 2
7	2	3	4	5	6	7	8	9	10	10	
6	1	2	3	4	5	6	7	8	9	10	Tramo 3
5	1	2	3	4	5	6	7	8	9	10	
4	1	1	2	3	4	5	6	7	8	9	Tramo 4
3	1	1	2	3	4	5	6	7	8	9	
2	1	1	1	2	3	4	5	6	7	8	Tramo 5
1	1	1	1	2	3	4	5	6	7	8	

Figura 5.8. Tabla del factor de relleno

Se observa que se trata de un algoritmo dividido en tramos, que corresponden a diferentes valores de velocidad

de consumo de combustible. En figura 5.9 se muestra el algoritmo que se ha desarrollado para generar el valor del factor de relleno.

Se obtiene una fórmula por cada tramo indicado en la figura 5.8.

Tramo 1, velocidad de consumo entre 10 y 9 l/min.

$$factor\_relleno = \begin{cases} \frac{|140 - combustible\_restante|}{10}, & combustible\_restante > 30 \\ 10, & combustible\_restante \leq 30 \end{cases}$$

Tramo 2, velocidad de consumo entre 8 y 7 l/min.

$$factor\_relleno = \begin{cases} \frac{|130 - combustible\_restante|}{10}, & combustible\_restante > 20 \\ 10, & ombustible\_restante \leq 20 \end{cases}$$

Tramo 3, velocidad de consumo entre 6 y 5 l/min.

$$factor\_relleno = \frac{|120 - combustible\_restante|}{10}$$

Tramo 4, velocidad de consumo entre 4 y 3 l/min.

$$factor\_relleno = \begin{cases} \frac{|100 - combustible\_restante|}{10}, & combustible\_restante \neq 100 \\ 1, & combustible\_restante = 100 \end{cases}$$

Tramo 5, velocidad de consumo entre 2 y 1 l/min.

$$factor\_relleno = \begin{cases} \frac{|90 - combustible\_restante|}{10}, & combustible\_restante < 90 \\ 1, & combustible\_restante \geq 90 \end{cases}$$

Figura 5.9. Algoritmo del factor de relleno.

Se observa que el factor de relleno relaciona de forma lineal la velocidad de consumo con el combustible restante, que son los dos factores que influyen en la urgencia de relleno.

El algoritmo se incluye en el programa control_generador con el código mostrado en la figura 5.10.

```
…
/*Los valores del factor_relleno se encuentran entre 1 y 10, y el umbral para envío de traps se fija en el agente
(objeto trapcombustible, umbral por defecto:6)*/
      switch(velocidad_consumo){
         case 10:
         case 9:
            if(combustible_restante<=30){
               factor_relleno=10;}
```

```
    else{
        factor_relleno=(130-combustible_restante)/10;
        break;
    }

case 8:
case 7:
  if(combustible_restante<=20){
     factor_relleno=10;
     break;}
   else{
        factor_relleno=(120-combustible_restante)/10;
        break;
   }

case 6:
case 5:
   if(combustible_restante<=10){
     factor_relleno=10;
     break;}
   else{
        factor_relleno=(110-combustible_restante)/10;
        break;
   }

case 4:
case 3:
  if(combustible_restante>=90){
     factor_relleno=1;
     break;
   }
```

```
            else{
                factor_relleno=(100-combustible_restante)/10;
                break;
            }

        case 2:
        case 1:
            if(combustible_restante>=80){
                factor_relleno=1;
                break;
            }
            else{
                factor_relleno=(90-combustible_restante)/10;
                break;
            }
    }
    f_frelleno = fopen("frelleno_demon_Trap/factor_relleno.txt","w+");
    fprintf(f_frelleno, "%d", factor_relleno);
    printf("Factor de relleno: %d\n\n", factor_relleno);
    fclose(f_frelleno);
    }
```

Figura 5.10. Código del algoritmo del factor de relleno.

Se puede observar que el algoritmo se ha codificado utilizando una función *switch* para separar los tramos anteriormente descritos. Al final del fragmento, el valor obtenido se escribe en el archivo *buffer factor_relleno.txt*.

Una vez calculado, si el valor del factor de relleno es igual o mayor que 6 (ver figura 5.8), el agente debe enviar una *trap* para indicar la urgencia de relleno del depósito. En el siguiente apartado (5.2.2.5.) se describe la función encargada de hacerlo.

5.2.2.5. Aplicaciones trapcombustible y trappotencia

Se trata de dos programas que soportan los objetos de tipo *notify trapcombustible* y *trappotencia*. Como se puede observar en la figura 5.4, el valor del objeto *factor_relleno* está asociado al *trap* soportado por *trapcombustible*, de forma que cuando este valor supera cierto valor, *trapcombustible* envía un *trap*. Se ha considerado que cuando el factor de relleno alcanza el valor 6, es buen momento para comenzar a mandar avisos para que se rellene el depósito. Por otra parte, el valor del objeto *potencia_total* está asociado a *trappotencia*, de forma que cuando la potencia consumida por los servidores se acerca a la potencia máxima que puede entregar el generador, se envía una notificación. Se ha considerado que este *trap* se envíe cada vez que la potencia total consumida por los servidores supere los 9500W, siendo 10KW la potencia máxima entregada por el generador. Estas condiciones de envío se implementan en el código de *trapcombustible_demon.c* y *trappotencia_demon.c* respectivamente, obtenido utilizando los comandos *mib2c -c mib2c.notify.conf GENSERV-MIB::frelleno* y *mib2c -c mib2c.notify.conf GENSERV-MIB::trappotencia*.

Esta implementación de las condiciones se muestra en la figura 5.11.

Fragmento del código de *trapcombustible_demon.c*

```
fp = fopen ("factor_relleno.txt", "r");
if (fscanf(fp, "%d", &fr1)==1){
        printf("%d",fr1);
        }else{
        printf("Error al leer el factor de relleno.\n");
}
/*Envía un trap SNMP v2c cuando el valor del factor de relleno es superior a 6*/
if(fr1>6){
        send_trapcombustible_trap(); /*Envía un trap cada 5 segundos*/
        sleep(5);
}
```

Fragmento del código de *trappotencia_demon.c*

```
fp = fopen ("potencia_total.txt", "r");
if (fscanf(fp, "%d", &pot1)==1){
        printf("%d",pot1);
        }else{
        printf("Error al leer la potencia total.\n");
}
/*Envía un trap SNMP v2c cuando el valor de potencia total es superior a 9000W*/
if(pot1>9500){
     send_trappotencia_trap(); /*Envía un trap cada 5 segundos*/
     sleep(5);
}
```

Figura 5.11. Código de condiciones de envío de *traps.*

En la figura 5.11 se puede observar que ambas funciones son análogas y obtienen su valor de los archivos correspondientes, enviando las *traps* según las condiciones anteriormente explicadas.

Para obtener los programas *trapcombustible_demon* y *trappotencia_demon* se utilizan los comandos *sudo make trapcombustible_demon* y *sudo make trappotencia_demon*.

5.3. Puesta en marcha del prototipo

El prototipo requiere que todos los programas anteriores se ejecuten en paralelo, de forma que se comuniquen entre ellos a través del *buffer*, como se observa en la figura 5.4, para soportar las funciones necesarias. Se deben iniciar en primer lugar los programas que realizan el soporte de los objetos gestionados por el subagente AgentX, de forma que se inicien las variables y la relación de las mismas con los buffers de datos. Posteriormente, se inicia el programa *control_generador*, que se encarga de simular el dispositivo (generador diésel) en funcionamiento.

Es aconsejable esperar unos segundos entre el inicio de cada proceso. Esto es debido a que, a causa de las limitaciones de hardware, se producen problemas al lanzarse todos al mismo tiempo. Además, es aconsejable introducir la espera si se tiene en cuenta que se utilizan *buffers* comunes y que algunos de los *demons* de AgentX comparten variables (*ptotal_demon* con *trappotencia_demon* y *frelleno_demon* con *trapcombustible_demon*).

5.4. Resultados

En este apartado se comprueba el funcionamiento del prototipo según los requerimientos prefijados. El primer paso es lanzar los *demons* necesarios, tal y como se muestra en la figura 5.11. La limitación de procesamiento impuesta por el hardware de la Raspberry Pi hace que sea aconsejable esperar al menos dos minutos desde que se lanza el script. Una vez cumplido el tiempo, se comprueba que el inicio no haya tenido errores.

En la figura 5.12 se muestran todos los procesos del prototipo en ejecución junto con el *demon* de Net-SNMP.

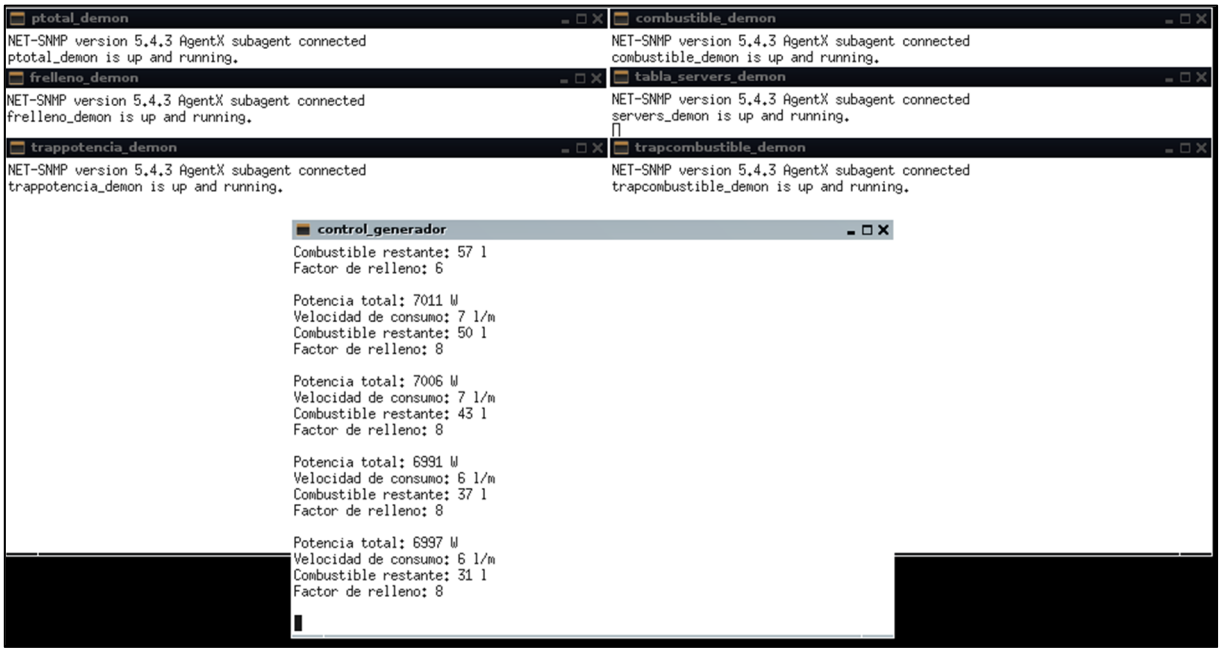


Figura 5.12. Procesos del prototipo en ejecución.

Se puede observar que todos los programas desarrollados en el apartado 5.2.2 están en funcionamiento.

El primer paso consiste en comprobar que los objetos son accesibles desde un gestor remoto. Para ello utilizaremos la suite de software *MG-SOFT MIB Browser* (ver apartado 3.1.4), que permite la gestión de objetos SNMP desde un interfaz gráfico. Para observar los objetos de la MIB correctamente es necesario cargar y compilar la propia *MIB GENSERV-MIB*; para ello se utiliza el programa *MIB Compiler*. En la figura 5.13 se muestra el aspecto de este programa.

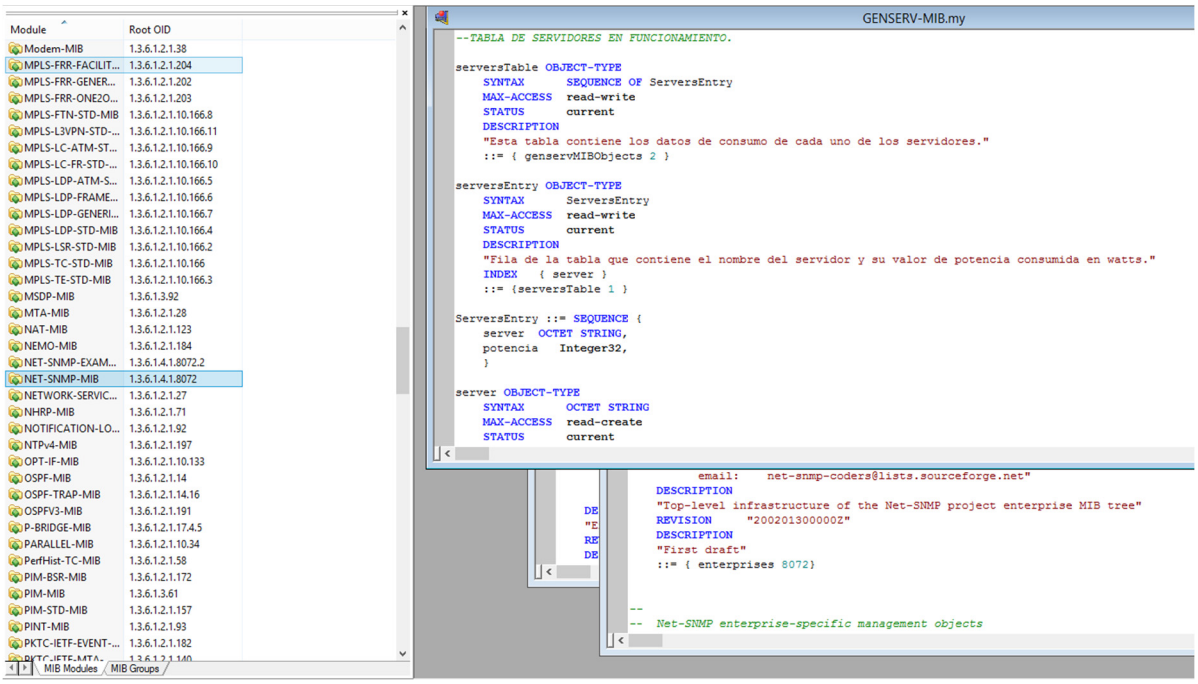


Figura 5.13. Aspecto de *MIB Compiler*

La lista de MIBs que se encuentran ya compiladas está situada en la columna de la izquierda. Se debe compilar la MIB específica del prototipo, *GENSERV-MIB*, y también sus dependencias si fuera necesario; dicho de otra forma, compilar la(s) MIB(s) de la(s) que *cuelga*.

La dependencia primaria es la MIB que precede a *GENSERV-MIB*; es decir, la MIB de la cual está *colgada* ésta en la estructura de árbol. En la figura 5.14 se incluye el fragmento de código de *GENSERV-MIB* que indica la dependencia de la MIB precedente (*NET-SNMP-EXAMPLES-MIB*).

```
…
IMPORTS
    netSnmpExamples                          FROM NET-SNMP-EXAMPLES-MIB
    OBJECT-TYPE, Integer32,
    MODULE-IDENTITY                  FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP      FROM SNMPv2-CONF
…
```

Figura 5.14. Dependencias de GENSERV-MIB

Se comprueba que no está compilada, por lo que será necesario hacerlo. Una vez identificada esta dependencia, es necesario averiguar si *NET-SNMP-EXAMPLES-MIB* tiene a su vez dependencia de otras MIBs no compiladas. En su código se identifican estas MIBs, como se observa en la figura 5.15.

```
…
IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE, Integer32,
    NOTIFICATION-TYPE                   FROM SNMPv2-SMI
    SnmpAdminString                     FROM SNMP-FRAMEWORK-MIB
    netSnmp                             FROM NET-SNMP-MIB
    RowStatus, StorageType              FROM SNMPv2-TC
    InetAddressType, InetAddress        FROM INET-ADDRESS-MIB
…
```

Figura 5.15. Dependencias de *NET-SNMP-EXAMPLES*

De nuevo se comprueba la lista de MIBs compiladas en el programa, y se observa que falta *NET-SNMP-MIB*. Debemos cargarla y comprobar que no tiene dependencias sin cubrir. Esto se comprueba de forma análoga a las anteriores MIBs. Esta ya forma parte de las cargadas en el programa, por lo que se llega al fin de las dependencias. Ahora sólo es necesario dar la orden de compilar las MIBs *NET-SNMP-MIB*, *NET-SNMP-EXAMPLES-MIB* y *GENSERV-MIB* para que *MG-SOFT MIB Browser* pueda reconocer completamente las OIDs de los objetos del prototipo. Una vez hecho esto, obtenemos el resultado en forma de árbol reflejado en la figura 5.16.

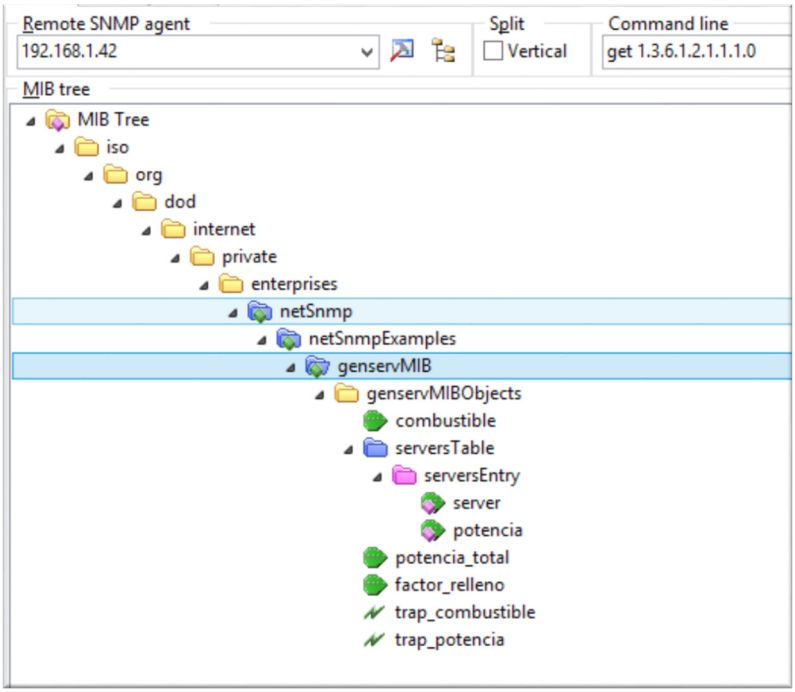


Figura 5.16. *GENSERV-MIB* observada desde un gestor remoto

En esta figura se pueden ver la dependencia de *GENSERV-MIB* de *NET-SNMP-EXAMPLES-MIB* y *NET-SNMP-MIB*, sus precedentes en la estructura de árbol.

Para comprobar el funcionamiento de los objetos gestionados, se realiza un *snmpwalk* partiendo del objeto *genservMIB* del árbol de la figura 5.17. El resultado devuelto por *MG-SOFT MIB Browser* se representa en la figura 5.18.

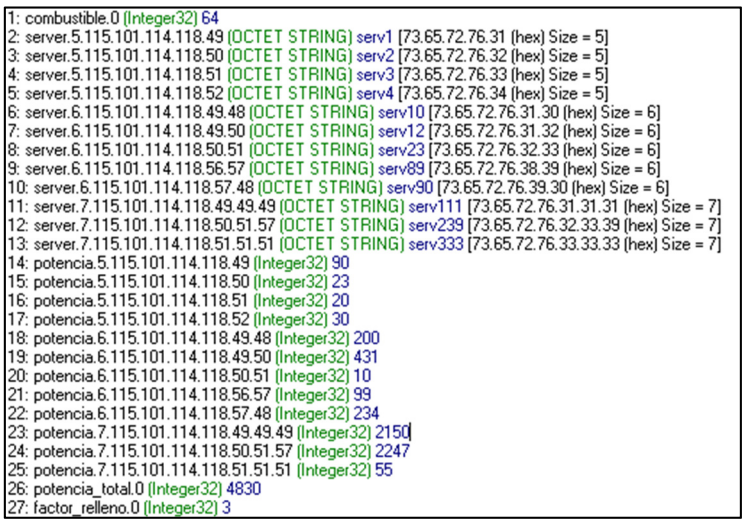


Figura 5.17. Comando *walk* sobre *GENSERV-MIB*

Se puede observar que los elementos de la tabla siguen el orden lexicográfico explicado en el apartado 4.3.1, teniendo en cuenta que el índice en este caso es un objeto de tipo *string*.

A continuación se muestran una serie de gráficas que corresponden a las evoluciones de los valores de los objetos gestionados *potencia_total*, *factor_relleno* y *combustible*. Como se ha explicado en el apartado 5.2.2.4, estos objetos están relacionados entre sí por los algoritmos soportados por las funciones del programa *control_generador*.

Para realizar las capturas se ha utilizado de nuevo el programa *MG-SOFT MIB Browser*. En las figuras 5.19, 5.20, 5.21, 5.22 y 5.23 se muestran los valores de diversos objetos gestionados para algunos valores de potencia máxima de los servidores conectados. De esta manera, se comprueba el funcionamiento del algoritmo que origina *factor_relleno* (ver apartado 5.2.2.4) en los diversos tramos de potencia contemplados, la variación pseudoaleatoria de la potencia total (de nuevo, ver apartado 5.2.2.4) y la velocidad de consumo de combustible en relación a la potencia consumida por los servidores. Se ha introducido un factor de escala 1/100 en el valor de *potencia_total* (línea roja), otro de x10 en el de *factor_relleno* (línea verde); y el de *combustible* (línea azul) se presenta tal cual. Además, se ha acelerado el tiempo de manera que cada 5 segundos se simula un minuto completo; es decir, la aceleración tiene un factor 12 (60/5). Estas manipulaciones se han realizado para facilitar la comparación de los valores y también para poder realizar la simulación en un tiempo razonable. También es importante destacar que se simula el relleno automático del depósito cada vez que alcanza un nivel de combustible menor a 11 litros, de forma que se faciliten las pruebas de simulación. En la figura 5.18 se incluye

el código añadido en el programa *control_generador*.

```
…
/*RELLENO DE COMBUSTIBLE PARA PRUEBAS DE SIMULACIÓN*/
        if (combustible_restante<11){
            combustible_restante=99;
            printf("Un encargado ha rellenado el depósito\n");
        }
…
```

Figura 5.18. Código de rellenado automático de combustible.

Como se puede observar, es un código sencillo que actualiza el valor del combustible restante y lanza un aviso de texto de que se ha producido el relleno.

En la primera gráfica (figura 5.19) se considera un valor de potencia total (en rojo) cercano al máximo, por lo que el consumo de combustible es alto y la velocidad a la que se vacía el depósito la más grande considerada (10 litros de combustible cada 5 segundos en el tiempo manipulado de la simulación). Se puede observar que el factor de relleno (en verde) es inversamente proporcional al combustible restante (en azul) cuando la potencia consumida se mantiene aproximadamente estable. En cierto momento (aproximadamente corresponde al tiempo 18:45:01), el valor de la potencia total cae por debajo de los 7.000W, de forma que el algoritmo de control que calcula el factor de relleno hace que éste aumente más lentamente (en las figuras 5.8 y 5.9 se puede comprobar el algoritmo y su correspondencia con los resultados obtenidos)

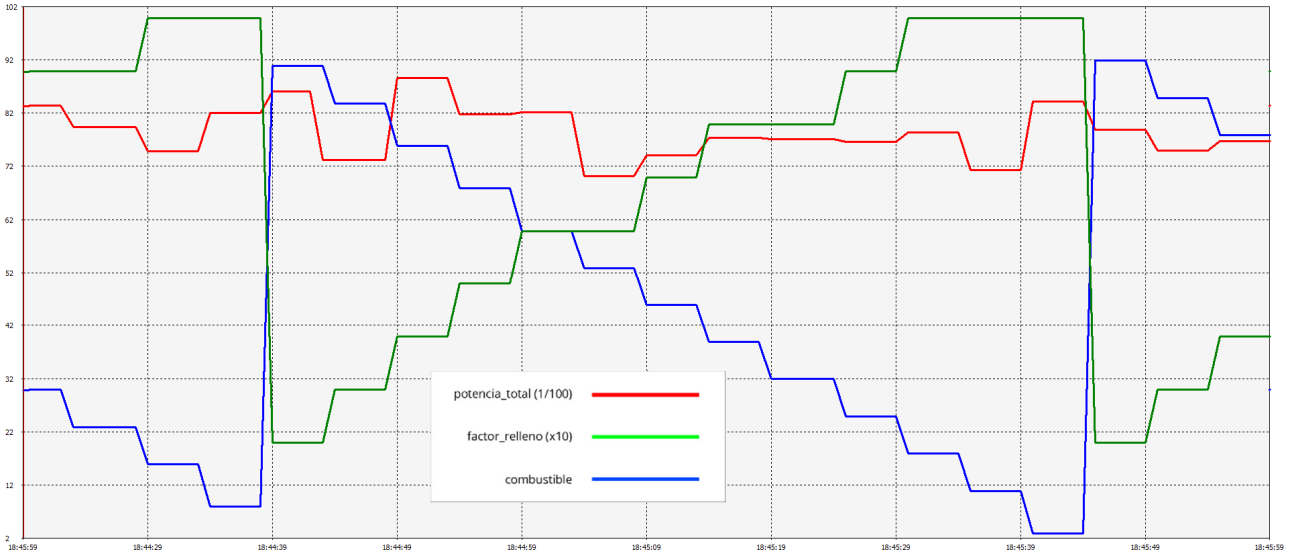


Figura 5.19. Gráfica temporal *potencia_total-factor_relleno-combustible* con 10KW de potencia máxima

En la siguiente figura (5.20), el valor de la potencia total (en rojo) es menor, de forma que se sitúa entre los

tramos 2 y 3 del algoritmo representado en las figuras 5.8 y 5.9.

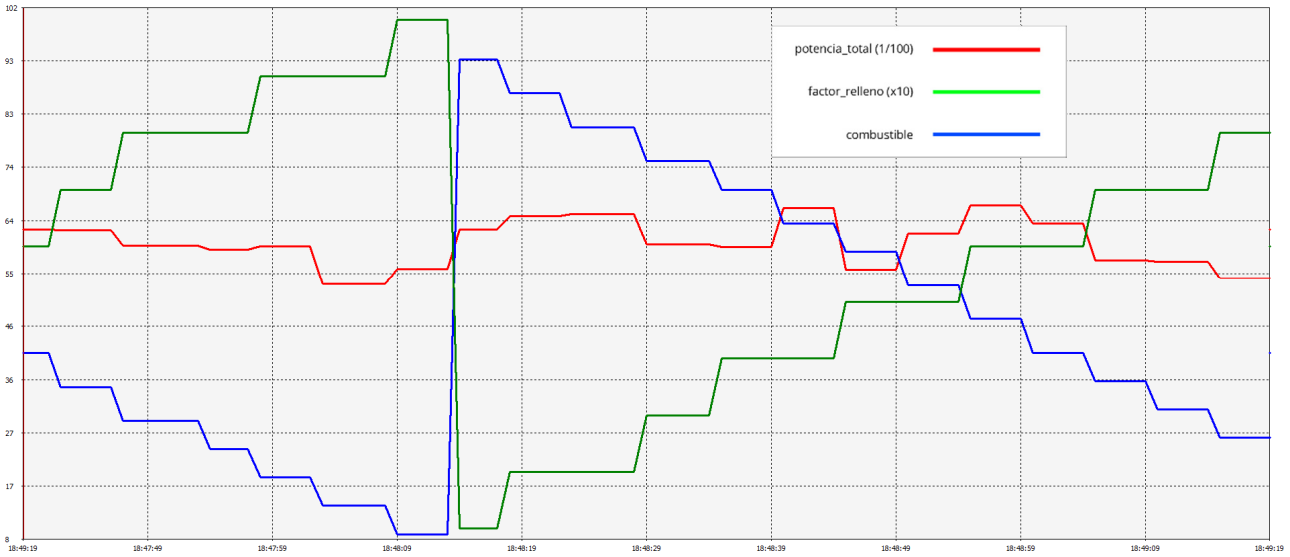


Figura 5.20. Gráfica temporal *potencia_total-factor_relleno-combustible* con 7.5KW de potencia máxima

Se puede observar como el aumento del valor del factor de relleno (en verde) se produce de forma más lenta que en la figura 5.19, debido a que la velocidad de consumo es menor (ya que la potencia consumida es también menor).

En la siguiente gráfica (figura 5.21) se muestra más claramente esta relación inversa de la velocidad de aumento del factor de relleno (verde) con la potencia total consumida (rojo). Además, también hay que tener en cuenta que la velocidad de consumo es menor, por lo que la ralentización del factor de relleno es mayor aún ya que la potencia se sitúa entre los tramos 3 y 4 del algoritmo de control(de nuevo, ver figuras 5.8 y 5.9).

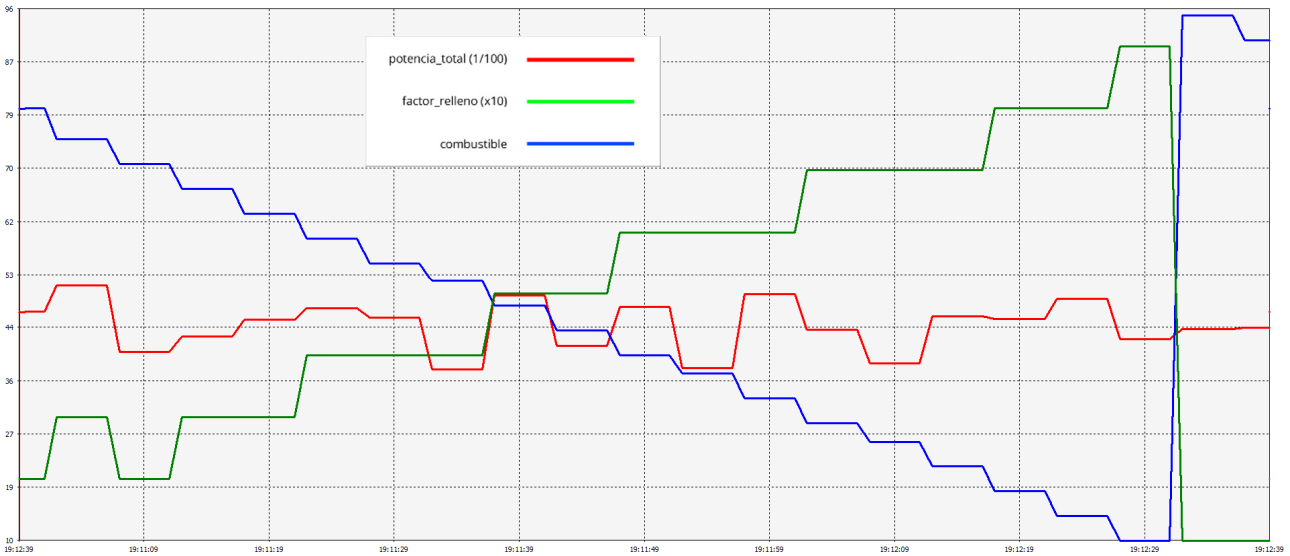


Figura 5.21. Gráfica temporal *potencia_total-factor_relleno-combustible* con 5.5KW de potencia máxima

En la figura 5.22, la potencia máxima que está conectada es de 3.5KW (rojo), por lo que se sitúa entre los tramos 4 y 5 del algoritmo de control (ver figuras 5.8 y 5.9). Esto hace que el consumo de combustible sea más

lento (en azul, el combustible disponible) y el aumento del factor de relleno (en verde) se ralentice aún más respecto a las gráficas anteriores (figuras 5.19, 5.20, 5.21). Estos valores coinciden con los previstos si se analiza el algoritmo de control.

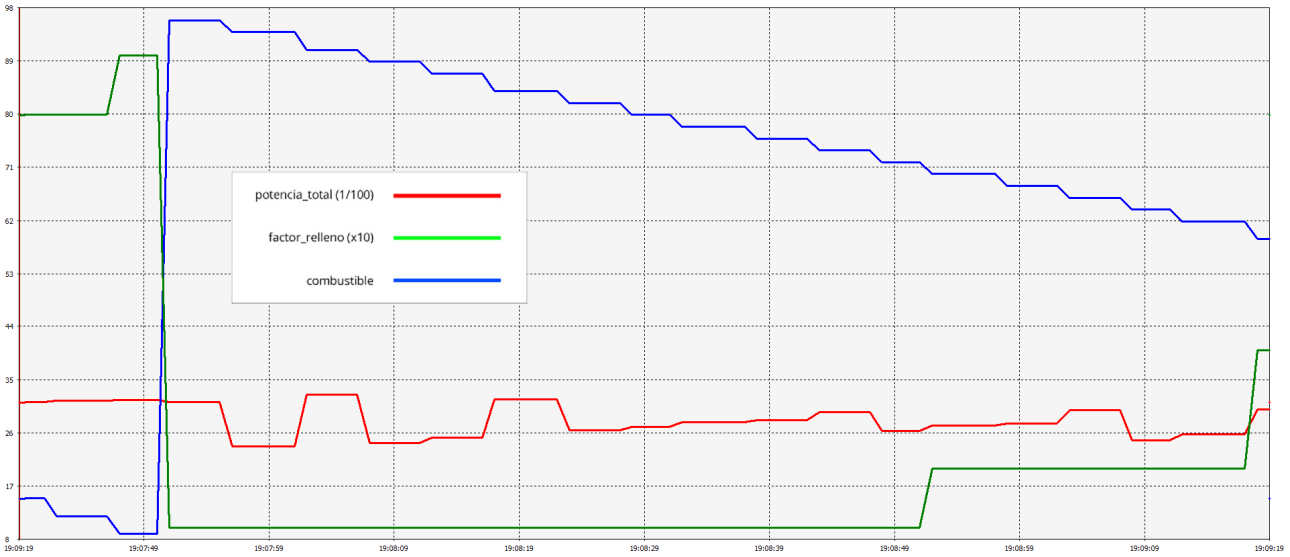


Figura 5.22. Gráfica temporal *potencia_total-factor_relleno-combustible* con 3.5KW de potencia máxima

En la figura 5.23 se muestran los resultados correspondientes a una potencia consumida baja (de 1.500W como máximo, en rojo), que se sitúa en el tramo 5 del algoritmo de control (ver figura 5.8). La ralentización del consumo (azul) y del aumento del factor de relleno (verde) son evidentes.

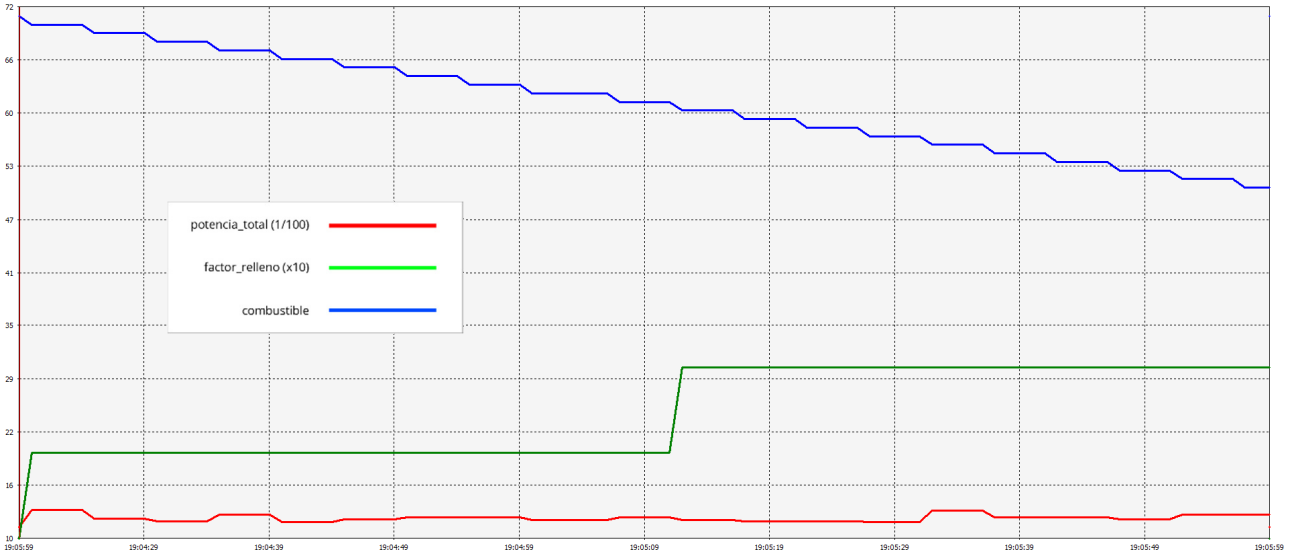


Figura 5.23. Gráfica temporal *potencia_total-factor_relleno-combustible* con 1.5KW de potencia máxima

En la figura 5.24 se muestra un cambio brusco de potencia al simular la conexión de un nuevo servidor. Se puede observar como el valor del factor de relleno (verde) reacciona instantáneamente a este aumento de potencia requerido.

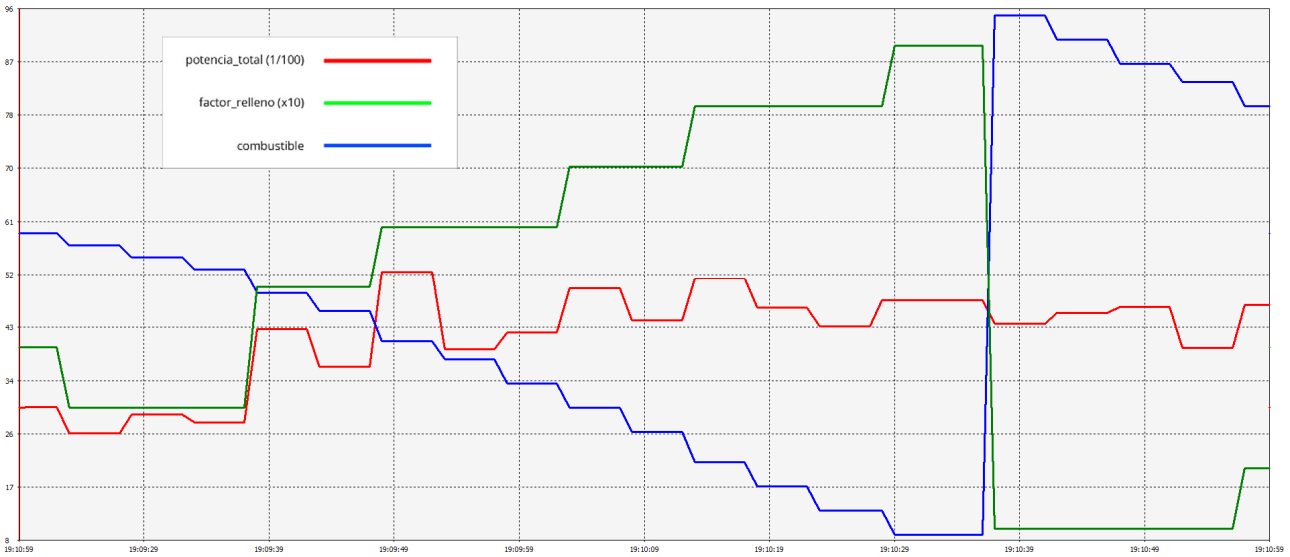


Figura 5.24. Simulación de aumento brusco de potencia

En detalle, se observa como en cierto momento (el correspondiente a las 19:09:39) se produce un aumento de la potencia total (rojo) debido a la simulación de conexión de un servidor; esto se ha realizado añadiendo un servidor a la tabla *servers_table.txt*, del que el objeto *serversTable* toma su valor instantáneamente. La reacción del factor de relleno (verde) es instantánea, y su valor aumenta dos unidades (en vez de una, que es lo que ocurre cuando no hay cambio de tramo en el algoritmo de control –figura 5.8-).

Finalmente, se comprueba el funcionamiento de las *traps*. Como ya se ha explicado, en este prototipo se incluyen dos objetos de tipo *trap* que están asociados al valor de dos objetos escalares. En concreto, *trappotencia*, asociado a *potencia_total* y *trapcombustible*, asociado a *factor_relleno*.

Se utiliza de nuevo el programa MG-SOFT MIB Browser, que recibe *traps* de forma automática. En la figura 5.25 se muestra una serie de *traps* de tipo *trapcombustible*, recibidos cuando el valor de *factor_relleno* es superior a 6 (ver apartado 5.2.2.5).

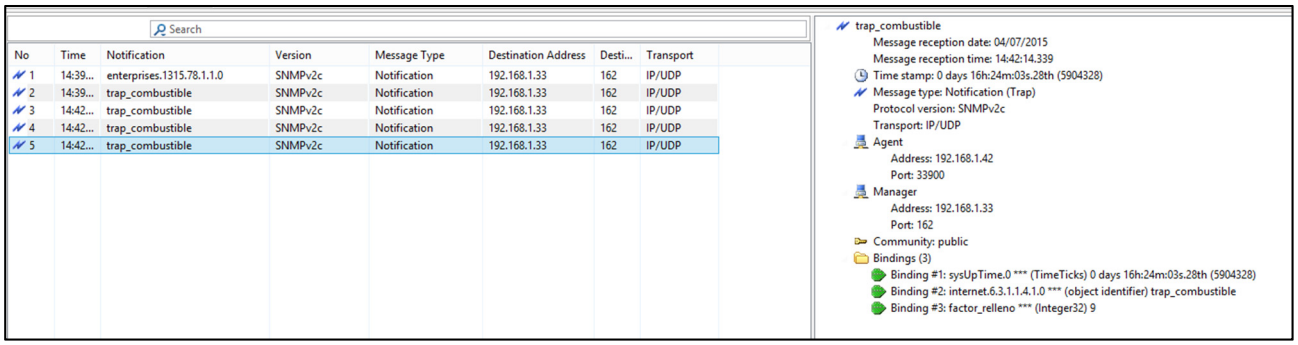


Figura 5.25. Captura de la recepción de *traps* (*trapcombustible*)

En la captura se observan varios datos: el tipo de *trap*, la versión de SNMP de dicho *trap* (v2c), las IPs del agente y del gestor, el puerto de envío de *traps* SNMP (162, ver apartado 2.1.3.1), y las variables asociadas (*bindings*: *sysUpTime*, OID de *trap_combustible* y valor de *factor_relleno*). La variable asociada al valor del factor de relleno, *factor_relleno*, tiene un valor de 9 en el caso de este *trap*.

En la figura 5.26 se muestran *traps* de tipo *trappotencia*, que, como ya se ha explicado, se envían cuando el valor de *potencia_total* es mayor que 9500, el límite superior fijado en el programa *trappotencia* (de nuevo, ver apartado 5.2.2.5).

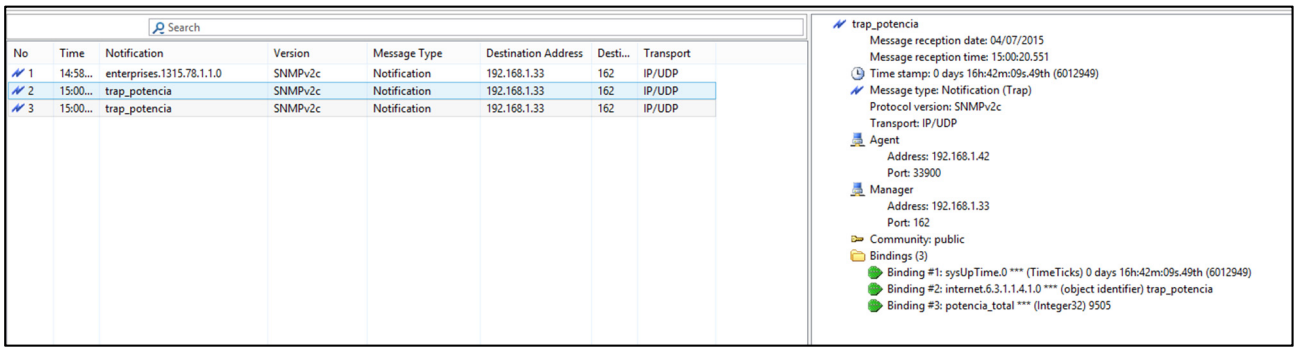


Figura 5.26. Captura de la recepción de *traps* (*trappotencia*)

De nuevo se observan los datos asociados a la *trap* (*bindings*); se destaca el valor del que depende su envío, *potencia_total*. El envío se produce al ser su valor, 9505, mayor de 9500.

Con esto finaliza la comprobación del funcionamiento del prototipo. Se han confirmado los resultados esperados.

5.5. Rendimiento

En este apartado se realiza un breve análisis de rendimiento del dispositivo.

Para comenzar, se obtiene una lista de procesos activos en la Raspberry Pi, ordenada según el consumo de memoria y CPU, con el comando *ps au --sort –rss*, cuyo resultado se muestra en la figura 5.27.

```
pi@raspberrypi ~ $ sudo ps au --sort -rss
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2682  0.4  1.8  17400  8468 pts/6    S    12:36   0:08 ./trappotencia_demon
root      2721  9.4  1.7  16840  7808 pts/4    S    13:07   0:08 ./trapcombustible_demon
root      2583  0.4  1.7  16792  7752 pts/1    S    12:34   0:09 ./combustible_demon
root      2631  0.4  1.7  16792  7752 pts/3    S    12:35   0:08 ./frelleno_demon
root      2680  0.4  1.7  16792  7752 pts/5    S    12:36   0:08 ./ptotal_demon
root      2610  0.4  1.7  16788  7748 pts/2    S    12:35   0:08 ./tabla_servers_demon
root      2029  0.1  1.7  25936  7736 tty7     Ss+  12:32   0:03 /usr/bin/X :0 -auth /var/run/lightdm/root/:0 -noli
root      2570  0.0  0.9  12644  4464 pts/0    S+   12:34   0:01 snmpd -f -Lo -C --rwcommunity=public --master=agen
pi        2636  0.0  0.6   7620  2936 pts/4    Ss+  12:35   0:00 /bin/bash
pi        2574  0.0  0.6   7620  2920 pts/1    Ss+  12:34   0:00 /bin/bash
pi        2597  0.0  0.6   7620  2920 pts/2    Ss+  12:34   0:00 /bin/bash
pi        2622  0.0  0.6   7620  2920 pts/3    Ss+  12:35   0:00 /bin/bash
pi        2660  0.0  0.6   7620  2920 pts/5    Ss+  12:35   0:00 /bin/bash
pi        2671  0.0  0.6   7620  2920 pts/6    Ss+  12:35   0:00 /bin/bash
pi        2560  0.0  0.6   7608  2908 pts/0    Ss   12:34   0:00 /bin/bash
pi        2690  0.0  0.6   7608  2908 pts/7    Ss   12:36   0:00 /bin/bash
pi        2724  1.0  0.6   7608  2908 pts/8    Ss   13:08   0:00 /bin/bash
root      2569  0.0  0.3   6816  1604 pts/0    S+   12:34   0:00 sudo snmpd -f -Lo -C --rwcommunity=public --master
```

Figura 5.27. Procesos del prototipo y su consumo de recursos

Se puede ver que la ocupación de memoria de cada uno de los subagentes es de entre el 1.8% y el 1.7%, mientras que la de CPU es menos uniforme, de forma que *trapcombustible* está consumiendo el 9.4% de recursos mientras que la suma del resto es de un 2.1%. Se ha comprobado que la variación del consumo de

CPU con el tiempo también es mayor; como muestra, en la figura 5.28 se puede ver una captura realizada poco tiempo después de la anterior en las mismas condiciones.

```
pi@raspberrypi ~ $ sudo ps au --sort -rss
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      2682  0.3  1.9  17688  8716 pts/6    S    12:36   0:08 ./trappotencia_demon
root      2721  1.0  1.8  17208  8244 pts/4    S    13:07   0:08 ./trapcombustible_demon
root      2583  0.3  1.7  16792  7752 pts/1    S    12:34   0:09 ./combustible_demon
root      2631  0.3  1.7  16792  7752 pts/3    S    12:35   0:08 ./frelleno_demon
root      2680  0.3  1.7  16792  7752 pts/5    S    12:36   0:08 ./ptotal_demon
root      2610  0.3  1.7  16788  7748 pts/2    S    12:35   0:08 ./tabla_servers_demon
root      2029  0.1  1.7  25936  7736 tty7     Ss+  12:32   0:04 /usr/bin/X :0 -auth /var/run/lightdm/root/:0
root      2570  0.0  0.9  12644  4464 pts/0    S+   12:34   0:01 snmpd -f -Lo -C --rwcommunity=public --master
```

Figura 5.28. Procesos del prototipo y su consumo de recursos (II)

En este caso, el consumo de memoria es muy similar al anterior, pero el de CPU disminuye notablemente.

Por razón de esta variabilidad, se considera que hacer una predicción del rendimiento del prototipo con más carga de trabajo no es aconsejable sin un estudio más profundo en este sentido.

En la figura 5.29 se muestra el resultado del comando *top*, que proporciona el estado de los recursos del sistema en tiempo real.

```
top - 13:29:42 up 57 min,  9 users,  load average: 0,42, 0,26, 0,24
Tasks: 124 total,   1 running, 123 sleeping,   0 stopped,   0 zombie
%Cpu(s): 10,7 us,  7,5 sy,  0,0 ni, 81,4 id,  0,0 wa,  0,0 hi,  0,3 si,  0,0 st
KiB Mem:    448736 total,   252900 used,   195836 free,    17984 buffers
KiB Swap:   102396 total,        0 used,   102396 free,    89372 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 2159 xrdp      20   0 20060 9264  952 S   5,6  2,1   1:25.83 xrdp
 2434 pi        20   0 19220  15m 2672 S   4,9  3,6   1:35.51 Xvnc
 2558 pi        20   0 96572  11m 7888 S   4,6  2,7   1:12.36 lxterminal
 2754 pi        20   0  6364 1408 1028 R   1,6  0,3   0:02.83 top
 2037 ntp       20   0  5508 1988 1560 S   0,3  0,4   0:00.76 ntpd
 2065 snmp      20   0 11556 6048 2788 S   0,3  1,3   0:04.34 snmpd
 2311 pi        20   0 89824  10m 8688 S   0,3  2,4   0:15.11 lxpanel
 2475 pi        20   0 90200  11m 8936 S   0,3  2,6   0:31.26 lxpanel
 2721 root      20   0 17552 8568 1940 S   0,3  1,9   0:08.65 trapcombustible
 2753 root      20   0     0    0    0 S   0,3  0,0   0:00.12 kworker/0:0
    1 root      20   0  2144  728  620 S   0,0  0,2   0:01.78 init
    2 root      20   0     0    0    0 S   0,0  0,0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0,0  0,0   0:00.18 ksoftirqd/0
    5 root       0 -20     0    0    0 S   0,0  0,0   0:00.00 kworker/0:0H
    6 root      20   0     0    0    0 S   0,0  0,0   0:00.46 kworker/u:0
```

Figura 5.29. Salida del comando *top*

En el campo *%Cpu* se muestran diversos valores: *us* corresponde al consumo de recursos de programas de usuario (entre los que se encuentran los del prototipo), *sy* se refiere a programas de sistema e *id* a los recursos libres de la CPU (*idle*). Si se realiza una vigilancia de las variaciones de la tabla durante unos minutos, se observa que los recursos consumidos por el sistema se mantienen prácticamente estables y que el aumento principal de consumo de la CPU lo produce el envío de *traps*, lo que es consistente con los datos de las figuras 5.27 y 5.28, que muestran una gran diferencia en el proceso *trapcombustible*. En el momento que muestra la captura, se puede observar que están disponibles un 43.6% de la memoria principal (195.836 de 448.736KB) y un 81.4% de los recursos de la CPU (el campo *id* corresponde a *Idle CPU Ticks*).

Con los resultados obtenidos, se puede suponer que hay margen suficiente para que el sistema soporte una mayor cantidad de subagentes AgentX. Pero hay que tener en cuenta que el envío de *traps* supone un esfuerzo

extra que causa picos de ocupación en los recursos de la CPU del dispositivo. También hay que considerar que hay procesos como *XRDP* (escritorio remoto, ver apartado 3.1.3), *Xorg* (entorno gráfico de GNU/Linux) y *PCManFM* (gestor de ventanas del escritorio *LXDE*, utilizado por *Raspbian*) que no son necesarios una vez configurado el dispositivo final. Por lo tanto, se puede prescindir de ellos liberando recursos, lo que hace posible ajustar aún más el uso de los mismos. Aun así, parece aconsejable realizar un estudio individualizado para cada caso.

6. Conclusiones y líneas futuras

Para finalizar, se presentan las conclusiones, incluyendo el grado de cumplimiento de los objetivos inicialmente planteados.

En este proyecto se ha tratado de crear un agente SNMP extensible autónomo utilizando para ello recursos de código abierto y componentes de bajo coste. Para ello se eligió un computador de placa reducida, *Raspberry Pi*, y se instalaron, configuraron y desarrollaron las aplicaciones de hardware necesarias para su funcionamiento. Con el prototipo finalizado, se puede concluir que los objetivos que se plantearon han sido satisfechos: se ha desarrollado un agente extendido completamente funcional y autónomo, y todos sus componentes de software son accesibles tanto para obtener información como para ser modificados.

En concreto, se ha implementado la gestión SNMP a través de agentes extendidos AgentX de una serie de objetos genéricos que, como se ha demostrado en el desarrollo del prototipo (apartado 5), pueden ser modificados para gestionar gran cantidad de objetos concretos. Hay que tener en cuenta que, con los cambios necesarios, se puede implementar la gestión SNMP de cualquier dispositivo imaginable.

También se ha comprobado el funcionamiento del dispositivo en el entorno del Laboratorio de Telemática de la ETSIIT de la Universidad de Cantabria, dado que uno de los objetivos planteados era su aplicación para uso educativo, concretamente en las prácticas de la asignatura *Gestión y Operación de Redes*, perteneciente al programa del Grado en Ingeniería de Tecnologías de Telecomunicación.

Respecto a las dificultades encontradas durante este proyecto, principalmente han sido debidas a algunos errores relacionados con *Net-Snmp*, la *suite* de software que soporta todas las funciones de SNMP y AgentX. No hay que olvidarse de que se trata de software de código abierto complejo, en continuo desarrollo, y de carácter no comercial, por lo que está en continua evolución y los errores son relativamente comunes. Aun así, a base de investigación, lectura extensa de mensajes publicados entre los desarrolladores y paciencia, estos errores han podido ser subsanados o sorteados.

Finalmente, las posibilidades de ampliación del prototipo son numerosas y dependen principalmente de qué se desee gestionar y de las limitaciones de la potencia de hardware de la *Raspberry Pi* (ver apartado 5.5). Pero, en esencia, cualquier dispositivo flujo de datos que se conecte al prototipo puede ser gestionado, con el trabajo previo del desarrollador. Por ejemplo, se pueden utilizar las funciones de manejo de tablas dinámicas para gestionar valores de la bolsa, partiendo de las funciones del prototipo implementado, y asociar *traps* a alarmas relacionadas con la compra y venta de acciones. O conectar sensores de cualquier tipo a la Raspberry Pi y dirigir su salida a *buffers* como los utilizados en el desarrollo de este proyecto, para finalmente gestionar aquello que midan.

Referencias

[1] - Alexander Clemm. *Network Management Fundamentals*. Cisco Press, noviembre de 2006.

[2] - J. Case, M. Fedor, M. Schoffstall, J. Davin. *RFC 1157 - A Simple Network Management Protocol (SNMP)*. Internet Engineering Task Force, mayo de 1990.

[3] - J. Case, K. McCloghrie, M. Rose, S. Waldbusser. *RFC 1441 - Introduction to version 2 of the Internet-standard Network Management Framework*. Internet Engineering Task Force, abril de 1993.

[4] - J. Case, K. McCloghrie, M. Rose, S. Waldbusser. *RFC 1452 - Coexistence between version 1 and version 2 of the Internet-standard Network Management Framework*. Internet Engineering Task Force, abril de 1993.

[5] - K. McCloghrie, M. Rose. *RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II*. Internet Engineering Task Force, marzo de 1991.

[6] - M. Rose, K. McCloghrie. *RFC 1155 - Structure and Identification of Management Information for TCP/IP-based Internets*. Internet Engineering Task Force, mayo de 1990.

[7] - Douglas Mauro, Kevin Schmidt. *Essential SNMP (2nd Edition)*. O'Reilly Media Incorporated, 2009.

[8] - William Stallings. *SNMP and SNMPv2: The Infrastructure for Network Management*. IEEE Communications Magazine, marzo de 1998.

[9] - ITU-T Study Group 17 ASN.1 Project, Paul Thorpe. *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*. ITU-T.

[10] - Comunidad de desarrolladores de Net-SNMP. *Tutorial de Net-SNMP (http://www.net-snmp.org/tutorial)*. Proyecto Net-SNMP, 2005.

[11] - M. Rose. *RFC 1227 - SNMP MUX Protocol and MIB*. Internet Engineering Task Force, mayo de 1991.

[12] - M. Daniele, B. Wijnen, M. Ellison, D. Francisco. *RFC 2741 – Agent Extensibility (AgentX) Protocol Version 1*. Internet Engineering Task Force, enero de 2000.

[13] - *Centro de Información Oficial de ARM (http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0301h/index.html)*. ARM Limited, 2004-2009.

[14] - Responsables del proyecto Raspberry Pi. *Información oficial del proyecto Raspberry Pi (https://www.raspberrypi.org/)*. Raspberry Pi Foundation.

[15] – Free Software Foundation. *GNU General Public License version 2.0 (GPLv2)*. Free Software Foundation Incorporated, junio de 1991.

[16] – Comunidad de desarrolladores de Net-SNMP. *Manual de mib2c (http://www.net-snmp.org/docs/man/mib2c.html)*. Proyecto NetSNMP (www.netsnmp.org), 2009.

[17] – Comunidad de desarrolladores de Net-SNMP. *Mailing list del proyecto Net-SNMP (http://sourceforge.net/p/net-snmp/mailman/message/28160638/)*. Proyecto Net-SNMP.

[18] – Responsables del proyecto Net-SNMP. *Página oficial (www.net-snmp.org)*. Proyecto Net-SNMP.

[19] - R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser. *RFC 3416 - Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. Internet Engineering Task Force, diciembre de 2002

[20] – *Página oficial de Raspberry Pi* (www.raspberrypi.org)

[21] – *Página oficial de MG SOFT MIB Browser* (www.mg-soft.si/mgMibBrowserPE.html)

A1. Árbol de SUBAGENT-1-MIB.

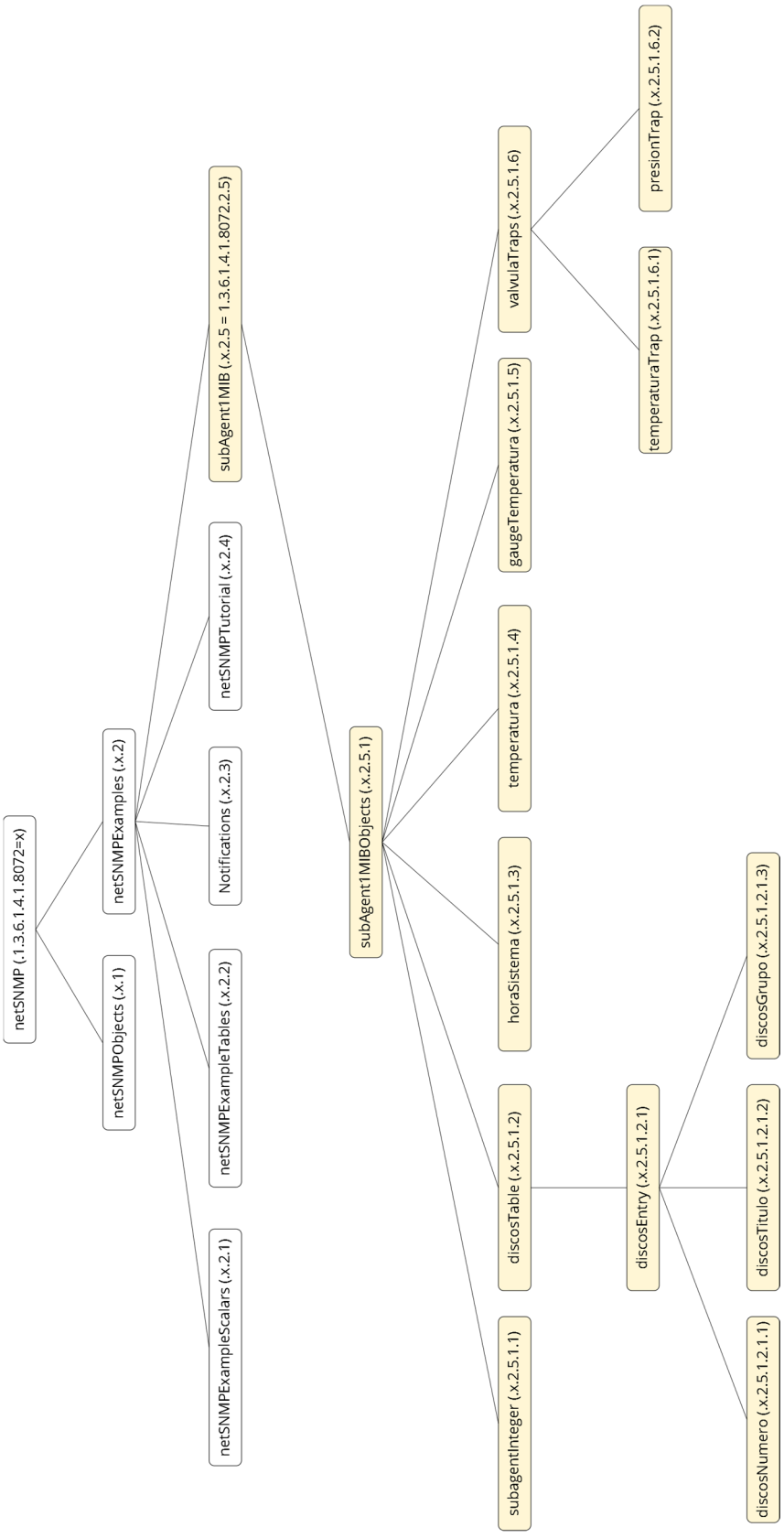


Figura A1.1. Árbol de *SUBAGENT-1- MIB*

A2. SUBAGENT-1-MIB

Se incluye el código de la MIB de ejemplo *SUBAGENT-1-MIB* utilizada en el apartado 4, para usar como referencia.

```
SUBAGENT-1-MIB DEFINITIONS ::= BEGIN


IMPORTS
    netSnmpExamples                         FROM NET-SNMP-EXAMPLES-MIB
    OBJECT-TYPE, Integer32,
    MODULE-IDENTITY                 FROM SNMPv2-SMI
    MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF;


subagent1MIB MODULE-IDENTITY
  LAST-UPDATED "201412170000Z"          -- 17 de diciembre de 2014, 00.00h
  ORGANIZATION "Universidad de Cantabria"
  CONTACT-INFO "Información de contacto: Avenida de Los Castros, etc.
          "
  DESCRIPTION  "MIB que estructura todos los objetos gestionados en este proyecto"
    ::= { netSnmpExamples 5 }   -- Esta línea define la ubicación del primer nodo de esta MIB. En concreto netSnmpExamples.5.
-- Al final de este documento se incluye una versión "gráfica" del árbol de OIDs.


subagent1MIBObjects     OBJECT IDENTIFIER ::= { subagent1MIB 1 }


subagentInteger OBJECT-TYPE
  SYNTAX     Integer32
  MAX-ACCESS  read-write
  STATUS     current
  DESCRIPTION
    "This is an object that simply supports a writable integer
```

```
        when compiled into the agent.  See
        http://www.net-snmp.org/tutorial-5/toolkit/XXX for further
        implementation details."
    DEFVAL { 0 }
    ::= { subagent1MIBObjects 1 }

discosTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF DiscosEntry
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Tabla que representa la lista de discos de una tienda."
    ::= { subagent1MIBObjects 2 }

discosEntry OBJECT-TYPE
    SYNTAX      DiscosEntry
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Una fila de la tabla, que contiene el número de serie, el título y el grupo de cada disco."
    INDEX   { discosNumero } --Define el índice de la tabla.
    ::= {discosTable 1 }

--Definición de la estructura de fila.
DiscosEntry ::= SEQUENCE {
    discosNumero    Integer32,
    discosTitulo  OCTET STRING,
    discosGrupo  OCTET STRING
}

discosNumero OBJECT-TYPE
```

```
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Número de serie del disco."
    ::= { discosEntry 1 }

discosTitulo OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
    "Título del disco."
    ::= { discosEntry 2 }

discosGrupo OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
    "Grupo."
    ::= { discosEntry 3 }

--VARIABLE DEL SISTEMA, FUERA DEL AGENTE. HORA DEL SISTEMA
horaSistema OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
     "Objeto SNMP enlazado con la hora del sistema. Debe ser capaz de monitorizarla en tiempo real. Objeto de
prueba, no incluido en la versión final del proyecto."
```

```
    ::= { subagent1MIBObjects 3 }


--VARIABLES GLOBALES, MODIFICABLES A TRAVES DE INPUTS (ARCHIVOS DE TEXTO, EN ESTE CASO)
temperatura OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
      "Objeto SNMP con una variable relacionada con el valor de un archivo de texto. Net-SNMP la monitoriza en tiempo real."
    ::= { subagent1MIBObjects 4 }


--GAUGE FOR NOTIFICATION
gaugeTemperatura OBJECT-TYPE
    SYNTAX      Gauge32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Objeto SNMP con una variable de tipo gauge relacionada con el valor de un archivo de texto. Net-SNMP la monitoriza en tiempo real. Objeto de prueba no incluido en la versión final del proyecto"
    ::= { subagent1MIBObjects 5 }


--NOTIFICATIONS. Dos notificaciones (traps en v2c) asociados a dos variables
 (virtuales) que representan la temperatura y la presión en una válvula imaginaria.
valvulaTraps      OBJECT IDENTIFIER ::= { subagent1MIBObjects 6}


temperaturaTrap NOTIFICATION-TYPE
    STATUS  current
    OBJECTS {temperatura}
    DESCRIPTION
          "Trap asociado al objeto temperatura que representa la temperatura de la válvula."
```

```
    ::= { valvulaTraps 1 }


presionTrap NOTIFICATION-TYPE
    STATUS  current
    OBJECTS{presion}
    DESCRIPTION
        "Trap asociado a la presión de la válvula. No incluido en la versión final del proyecto, se puede usar como
punto de partida para un sistema de avisos más complejo."
    ::= { valvulaTraps 2 }


-- Todo lo anterior produce esta estructura en forma de árbol al introducir el
-- comando:
-- % snmptranslate -M+. -mSUBAGENT-1-MIB -Tp -IR subagent1MIB
--
--    +--subagent1MIB(5)
--              |
--    +--subagent1MIBObjects(1)
--                        |
--                        +-- -RW- Integer32 subagentInteger(1)
--              +-- discosTable(2)
--              |         |
--                        |         +-- discosEntry(1)
--                        |                   |         Index: discosNumero
--                        |                   |
--                        |                   +-- -RW- Integer32          discosNumero(1)
--                        |                   +-- CR- String     discosTitulo(2)
--                        |                   +-- CR- String     discosGrupo(3)
--              +-- -RW- String horaSistema(3)
--              +-- -RW- Integer32 temperatura(4)
--              +-- -RW- Gauge gaugeTemperatura(5)
--                        |
```

```
--              +-- valvulaTraps (6)
--                      |
--                      +--temperaturaTrap(1)
--                      +--presionTrap(2)
END
```

Figura A2.1. Código de SUBAGENT-1-MIB

A3. Archivo de configuración snmpd.conf

Código completo del archivo de configuración principal de Net-SNMP, mostrado en la figura A4.1.

```
# Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:161

# ACCESS CONTROL
#
view   systemonly  included   .1.3.6.1.2.1.1
view   systemonly  included   .1.3.6.1.2.1.25.1

#MIO. Intento de permitir que TODAS las MIBs sean visibles.
view            all         included .1         80
view            all         excluded            .1.3.6.1.2.1.4.21
view            all         excluded            .1.3.6.1.2.1.4.24
view            system      included .iso.org.dod.internet.mgmt.mib-2.system

# Full access from the local host
rocommunity public  localhost

# Acceso desde local con private, escritura permitida.
rwcommunity private localhost

# Default access to basic system info
#rocommunity public  default    -V systemonly

#Acceso a la community "public" (read only) y a la community "private" (read/write) desde cualquier dispositivo remoto.
rocommunity public default
```

```
rwcommunity private default


#  It's no longer typically necessary to use the full 'com2sec/group/access' configuration
#  r[ou]user and r[ow]community, together with suitable views, should cover most requirements
com2sec readonly default public
com2sec readwrite default private

###############################################################################
#
#  SYSTEM INFORMATION
#

#  Note that setting these values here, results in the corresponding MIB objects being 'read-only'
#  See snmpd.conf(5) for more details
sysLocation    Universidad de Cantabria, Laboratorio de Telemática
sysContact     Me <me@example.org>
                                # Application + End-to-End layers
sysServices    72


#
#  Process Monitoring
#
                        # At least one  'mountd' process
proc  mountd
                        # No more than 4 'ntalkd' processes - 0 is OK
proc  ntalkd    4
                        # At least one 'sendmail' process, but no more than 10
proc  sendmail 10 1


#  Walk the UCD-SNMP-MIB::prTable to see the resulting output
#  Note that this table will be empty if there are no "proc" entries in the snmpd.conf file
```

```
#
#  Disk Monitoring
#
# 10MBs required on root disk, 5% free on /var, 10% free on all other disks
disk       /     10000
disk       /var  5%
includeAllDisks  10%

#  Walk the UCD-SNMP-MIB::dskTable to see the resulting output
#  Note that this table will be empty if there are no "disk" entries in the snmpd.conf file

#
#  System Load
#
                        # Unacceptable 1-, 5-, and 15-minute load averages
load   12 10 5

#  Walk the UCD-SNMP-MIB::laTable to see the resulting output
#  Note that this table *will* be populated, even without a "load" entry in the snmpd.conf file

###############################################################################
#
#  ACTIVE MONITORING
#  En este apartado se incluyen las IPs de los gestores que reciben los traps.
#   send SNMPv2c traps
trap2sink    localhost public
trap2sink    default public
trap2sink    192.168.1.33 public
trap2sink    192.168.4.4 public
trap2sink    192.168.4.5 public
```

```
                        #   send SNMPv2c INFORMs
informsink   localhost public

#
#  Event MIB - automatically generate alerts
#
                        # Remember to activate the 'createUser' lines above
iquerySecName   internalUser
rouser          internalUser
                        # generate traps on UCD error conditions
defaultMonitors         yes
                        # generate traps on linkUp/Down
linkUpDownNotifications  yes

##############################################################################
#
#  EXTENDING THE AGENT
#

#
#  Arbitrary extension commands
#
#  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 A3.1. Código de *snmpd*.conf

A4. GENSERV-MIB

Código completo de la MIB del prototipo descrito en el capítulo 5.

```
GENSERV-MIB DEFINITIONS ::= BEGIN

-- MIB para un prototipo de gestión de combustible para un generador diésel que alimenta una sala de servidores.

-- Escrita por Adrián García Estébanez para PFC de Ingeniería de Telecomunicación, Universidad de Cantabria.


-- IMPORTS: Define el lugar en el que está situada la MIB. En este caso, "cuelga" de netSnmpExamples. En caso de querer cambiarla de lugar, sólo es necesario modificar esta parte.

IMPORTS

    netSnmpExamples                         FROM NET-SNMP-EXAMPLES-MIB

    OBJECT-TYPE, Integer32,

    MODULE-IDENTITY                 FROM SNMPv2-SMI

    MODULE-COMPLIANCE, OBJECT-GROUP     FROM SNMPv2-CONF;


-- Información de la MIB:

genservMIB MODULE-IDENTITY

   LAST-UPDATED "201507030000Z"          -- 3 de julio de 2015, 00.00h

   ORGANIZATION "Universidad de Cantabria"

   CONTACT-INFO "Información de contacto: Avenida de Los Castros, etc. etc.

            "

   DESCRIPTION  "MIB que define un prototipo de gestión de combustible para un generador diesel que alimenta una sala de servidores."

      ::= { netSnmpExamples 6 }    -- Esta línea define la ubicación del primer nodo de esta MIB. En concreto netSnmpExamples.6, OID: 1.3.6.1.4.1.8072.2.6.


-- Se define el nodo principal, bajo el cual estarán situados los objetos gestionados en este prototipo.

genservMIBObjects     OBJECT IDENTIFIER ::= { genservMIB 1 }


-- Definición de objetos:
```

```
--Escalar asociado al volumen de combustible disponible en litros.
combustible OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Objeto SNMP con una variable que corresponde a la cantidad de combustible restante expresada en litros."
    ::= { genservMIBObjects 1 }

--TABLA DE SERVIDORES EN FUNCIONAMIENTO.

serversTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF ServersEntry
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Esta tabla contiene los datos de consumo de cada uno de los servidores."
    ::= { genservMIBObjects 2 }

serversEntry OBJECT-TYPE
    SYNTAX      ServersEntry
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Fila de la tabla que contiene el nombre del servidor y su valor de potencia consumida en watts."
    INDEX   { server }
    ::= {serversTable 1 }

ServersEntry ::= SEQUENCE {
    server  OCTET STRING,
    potencia   Integer32,
```

```
    }


server OBJECT-TYPE
    SYNTAX      OCTET STRING
    MAX-ACCESS  read-create
    STATUS      current
    DESCRIPTION
    "Cada uno de los servidores en los que se mide la potencia."
    ::= { serversEntry 1 }


potencia OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Valor de la potencia en cada uno de los servidores."
    ::= { serversEntry 2 }


-- POTENCIA TOTAL: El siguiente objeto es la suma de potencias de todos los servidores en funcionamiento, cuyo cálculo autómático es realizado por un programa externo contenido en el agente.
potencia_total OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Objeto SNMP con una variable que corresponde a la potencia total instantánea que consumen los servidores."
    ::= { genservMIBObjects 3 }


-- FACTOR_RELLENO: Valor que indica el grado de necesidad de rellenar el depósito.
factor_relleno OBJECT-TYPE
    SYNTAX      Integer32
```

```
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION
    "Objeto SNMP con una variable que representa la necesidad de rellenar el depósito. Se trata de un valor comprendido entre 0 y 10, y resulta de la relación velocidad_de_consumo/combustible_restante. Este cálculo lo realiza una aplicación externa."
    ::= { genservMIBObjects 4 }


-- TRAP COMBUSTIBLE: Trap asociado a la necesidad de rellenar el depósito, es decir, al objeto factor_relleno.
trap_combustible NOTIFICATION-TYPE
    OBJECTS {factor_relleno}
    STATUS  current
    DESCRIPTION
        "Trap que se envía cuando el depósito del generador debe ser rellenado con cierta urgencia. Esta necesidad depende del consumo de los servidores y del nivel de combustible restante en el depósito, y está representada por el objeto factor_relleno."
    ::= { genservMIBObjects 5}


-- TRAP POTENCIA: Trap asociado a la potencia total que puede entregar el generador.
trap_potencia NOTIFICATION-TYPE
    OBJECTS {potencia_total}
    STATUS  current
    DESCRIPTION
        "Trap que se envía cuando los servidores conectados se acercan peligrosamente a la potencia máxima entregada por el generador."
    ::= { genservMIBObjects 6}
END
```

Figura A4.1. Código completo de GENSERV-MIB

A5. Código del programa *control_generador*

```
/*Este programa simula las funciones de un generador diesel ficticio y los datos que se obtienen de su sistema de control.*/

/*FUNCIONES DEL PROGRAMA


SIMULACIÓN DE CONSUMO: Lee el archivo servidores.txt, que contiene la tabla de potencias de los servidores conectados, y obtiene el valor de potencia total consumida de forma periódica.

Escribe el valor de potencia en el archivo potencia_total.txt.

Partiendo de la potencia total consumida, calcula la velocidad con la que se consume el combustible.


CONSUMO DE COMBUSTIBLE: Decrementa el valor de combustible restante periódicamente, de forma proporcional al valor que mide la velocidad de consumo.

Escribe periódicamente el valor de combustible restante en el archivo combustible.txt, el cual actúa como buffer de intercambio de datos.


CÁLCULO DE LA NECESIDAD DE RELLENAR EL DEPÓSITO: Calcula el factor de relleno a partir de los datos anteriores y lo escribe en el archivo factor_relleno.txt periódicamente.

*/

#include <stdio.h>

#include <errno.h>

#include <stdlib.h>

#include <unistd.h>


int main()

{

   FILE *f_servers;

   FILE *f_potencia;

   FILE *f_combustible;

   FILE *f_frelleno;


   char buf[128];

   int potencia_server;
```

```
    int potencia_total=0;
    char nombre_server[32];
    int velocidad_consumo=1;
    int random;
    time_t t;
    int factor_relleno=0;
    int combustible_restante=90;
   //char rompe_bucle;


    while(combustible_restante>0){


        sleep(5);


        /*Lectura del archivo tabla_servers.txt y cálculo de la suma de potencia*/
        f_servers = fopen("tabla_servers_demon/tabla_servers.txt","r");
        potencia_total=0;
        while ( fgets( buf, 128, f_servers )){                                  // Se lee línea a línea hasta fin del archivo.
            potencia_server=0;
            sscanf(buf, "%s %d", nombre_server, &potencia_server);   // Se extraen las columnas una a una.
            random=rand() % 40;
            potencia_server=(6*potencia_server/10)+(potencia_server*random/100);    //La potencia de cada servidor varía de forma pseudoaleatoria entre el 60% y el 100% de su valor máximo.
            potencia_total=potencia_total+potencia_server;                  // Se obtiene el valor de la potencia total máxima de los servidores.
        }
        fclose(f_servers);
        /*Se escribe el valor de la potencia total en potencia_total.txt*/
        f_potencia = fopen("ptotal_demon_Trap/potencia_total.txt","w");
        fprintf(f_potencia, "%d", potencia_total);
        fclose(f_potencia);
        printf("Potencia total: %d W\n", potencia_total);
```

```
/*Cálculo de la velocidad de consumo de combustible.
Nota: Se acelera el consumo real que podría tener un generador por exigencias de la simulación.
Se supone que, a máxima potencia de funcionamiento (10KW), se consumen 10l/minuto, y a mínima potencia (0W), 1l/minuto (debido al ralentí).
*/
if(potencia_total>1000){
   velocidad_consumo=potencia_total/1000;}
   else{
      velocidad_consumo=1;
}
printf("Velocidad de consumo: %d l/m\n", velocidad_consumo);

/*Se simula el consumo de combustible (se decrementa el valor alojado en combustible_restante.txt*/
//Lee archivo, saca valor, le resta los litros correspondientes, sobreescribe archivo. If valor=<0, lo pone a 1 para evitar conflictos matemáticos.
f_combustible = fopen("combustible_demon/combustible_restante.txt","w+");
fscanf(f_combustible, "%d", &combustible_restante);

/*RELLENO DE COMBUSTIBLE PARA PRUEBAS DE SIMULACIÓN*/
if (combustible_restante<11){
   combustible_restante=99;
   printf("Un encargado ha rellenado el depósito\n");
}

if (combustible_restante<=velocidad_consumo){
   combustible_restante=0;}
   else{
      combustible_restante=combustible_restante-(velocidad_consumo);
   }
//rewind(f_combustible);
```

```
        fprintf(f_combustible, "%d", combustible_restante);
        fclose(f_combustible);
        printf("Combustible restante: %d l\n", combustible_restante);

        /*Se actualiza el valor del factor de relleno; sobreescribe el valor en el archivo.*/
        /*El switch separa por tramos la urgencia de rellenado del depósito, dependiendo ésta de la velocidad de consumo de combustible y del combustible restante.*/
        /*Los valores del factor_relleno se encuentran entre 1 y 10, y el umbral para envío de traps se fija en el agente (objeto trapcombustible, umbral por defecto:6)*/
        switch(velocidad_consumo){
           case 10:
           case 9:
              if(combustible_restante<=30){
                 factor_relleno=10;}
              else{
                    factor_relleno=(130-combustible_restante)/10;
                    break;
              }

           case 8:
           case 7:
             if(combustible_restante<=20){
                 factor_relleno=10;
                 break;}
              else{
                    factor_relleno=(120-combustible_restante)/10;
                    break;
              }

           case 6:
           case 5:
```

```
            if(combustible_restante<=10){
                factor_relleno=10;
                break;}
            else{
                    factor_relleno=(110-combustible_restante)/10;
                    break;
            }

        case 4:
        case 3:
          if(combustible_restante>=90){
                factor_relleno=1;
                break;
            }
            else{
                    factor_relleno=(100-combustible_restante)/10;
                    break;
            }

        case 2:
        case 1:
            if(combustible_restante>=80){
                factor_relleno=1;
                break;
            }
            else{
                    factor_relleno=(90-combustible_restante)/10;
                    break;
            }
    }
    f_frelleno = fopen("frelleno_demon_Trap/factor_relleno.txt","w+");
```

```
        fprintf(f_frelleno, "%d", factor_relleno);

        printf("Factor de relleno: %d\n\n", factor_relleno);

        fclose(f_frelleno);

        }


        return 0;

    }
```

Figura A5.1. Código de *control_generador*