Escuela Técnica Superior de Ingenieros
Industriales y de Telecomunicación
UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Aplicación de la arquitectura SNMPv3 en la implementación de comunicaciones seguras para entornos IoT

(Application of the SNMPv3 architecture in the implementation of secure communications for IoT environments)

Para acceder al Título de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Iván González Takmovtseva

Julio - 2018

Resumen

En este proyecto se realiza un entorno de gestión seguro basado en IoT. Este tipo de redes hace uso de equipos con capacidades de cómputo limitadas y con bajo consumo energético. Debido a que protocolos como HTTPS o similares requieren capacidades de cómputo elevadas, se implementa el protocolo SNMP en su tercera versión. Esta, proporciona una comunicación segura extremo a extremo gracias a que garantiza autenticación, cifrado e integridad de la información intercambiada. Por lo tanto, en el proyecto se implementa un arquitectura de agente y sistema gestor. Dicho agente se implementa sobre una Raspberry Pi, la cual hace las funciones de proxy entre la red IoT y el gestor. La comunicación entre agente y gestor se establece a través del protocolo SNMP. Finalmente, todo esto se muestra con un prototipo. Este, consiste en la monitorización de un vehículo mediante esta tecnología. El intercambio de contenido entre el agente y el vehículo se realiza utilizando las propiedades del protocolo OBD-II. Este, establece una serie de comandos que son capaces de acceder a un determinado valor del coche.

Abstract

In this proyect, a secure managment system which is based on IoT networks is going to be created. This kind of networks use low-budget computers with low potency and low energy consumption. Protocols such as HTTPS or similar need high potency, so they cannot be used. Therefore, the third version of SNMP protocol is going to be implemented. This version is used because it allows to create a secure end to end communication using ciphering, integrity of the messages and autentication. Hence, an architecture that uses an agent and a managment system is implemented in the proyect. That agent is developed in a Raspberry Pi whose main function is to carry out the role of a proxy between the car and the manager. The exchange of information between the manager and the agent is established using the SNMP protocol. Finally, a prototype of all that has been metioned before is going to be shown. This one consits of a vehicle which is monitored using the tecnology mentioned before. The exchange of content between the agent and the vehicle is realized using the properties of the OBD-II protocol. This one indicates a bunch of commands which can be used to extract a specific value of the car.

Índice

1. Introducción

En este capítulo inicial se tratan las cuestiones previas relativas al proyecto. Motivaciones para la realización del proyecto, objetivos planteados y estructura de la memoria.

Desde el origen de TCP/IP(1969) se intentaron diseñar diferentes técnicas para la gestión de redes y equipos dentro de estas. Al princio, se utilizaron herramientas basadas en el protocolo ICMP (Internet-Control Message Protocol). En este, la principal herramienta es PING (Packet INternet Groper), que permite comprobar la comunicación entre dos máquinas, calcular tiempos medios de respuesta y pérdidas de paquetes.

Con el crecimiento exponencial de Internet a partir de los años 80,surge la necesidad de herramientas estándar de gestión más potentes. Algunas de las que surgieron son las siguientes:

- **HEMS** (High level entity management system), que es una generalización del que fue quizás el primer protocolo de gestión usado en Internet (HMP).
- **SNMP** (Simple network management protocol), que es una versión mejorada de SGMP (Simple gateway-monitoring protocol).
- **CMOT** (CMIP over TCP/IP), que intenta incorporar, hasta donde sea posible, el protocolo (common management information protocol),servicios y estructura de base de dato que se están estandarizando por ISO.

A principios de 1988, el IAB recibe las propuestas y decide el desarrollo de SNMP como solución a corto plazo y CMOT a largo plazo (ya que se suponía que con el tiempo las redes TCP/IP evolucionarán a redes OSI). HEMS es más potente que SNMP, pero como se supone que se va a producir la transición a OSI, se elige SNMP por ser más simple y necesitar menos esfuerzo para desarrollarse (ya que se supone que va a desaparecer con el tiempo).

Actualmente, SNMP es un estándar utilizado universalmente para la monitorización y gestión de los equipos de una red.

Dado que, las personas tienen un tiempo, una atención y una precisión limitados, y no se les da muy bien conseguir información sobre cosas en el mundo

real. Es por estas razones, que la IoT (Internet of Things) es la próxima etapa en la revolución de la información.

El concepto de Internet de las cosas fue propuesto por Kevin Ashton en el Auto-ID Center del MIT en 1999, donde se realizaban investigaciones en el campo de la identificación por radiofrecuencia en red (RFID) y tecnologías de sensores.

Hoy en día, el internet de las cosas tiene una función fundamental en la sociedad. La capacidad de conectar dispositivos embebidos con capacidades limitadas de CPU, memoria y energía significa que IoT puede tener aplicaciones en casi cualquier área. Estos sistemas podrían encargarse de recolectar información en diferentes entornos, desde ecosistemas naturales hasta edificios y fábricas, por lo que podrían utilizarse para monitoreo ambiental y planeamiento urbanístico [1].

El principal problema que tiene el IoT es que no es seguro. Por lo tanto, como alternativa se implementa el protocolo de gestión SNMP en su tercera versión, la cual proporciona una comunicación segura de extremo a extremo gracias al uso de cifrado, autenticiación e integridad de los mensajes.

A lo largo de este proyecto se utilizan las propiedades del los sistemas del internet de las cosas en colaboración con el protocolo SNMP para establecer una gestión centralizada y segura de un entorno IoT.

1.1. Motivación

SNMP nos proporciona la opción de la gestión de red para el control distribuido en cualquier sistema de comunicaciones en red. Gracias a este protocolo, se ofrece la gestión remota de los dispositivos conectados en una red de comunicaciones.

Debido a la evolución tecnológica, cada vez es mas común tener más dispositivos conectados a la red. Esto es lo que se denomina el Internet of Things (IoT). Esto permite que dichos dispositivos sean gestionados de forma sencilla y centralizada con el uso de protocolos como SNMP. En busca de esto, se ha planteado como proyecto implementar un sistema de gestión SNMP en vehículos, que nos permita obtener información asociada a estos. De esta forma, cualguier gestor autorizado es capaz de conocer en todo momento el

estado del vehículo.

Para lograr esto, se hace uso de una Raspberry Pi, la cual se conecta a través de un puerto serie al vehículo, y el protocolo de gestión SNMP. Esta, se encarga de obtener la información correspondiente del vehículo a través del puerto serie y enviarla mediante SNMP a un sistema gestor.

1.2. Objetivos

En el planteamiento del proyecto se proponen los siguientes objetivos:

- Gestionar los vehículos usando las propiedades de la tecnología del Internet de las cosas (IoT).
- Creación de un agente extensible SNMP e implementación sobre la Raspberry Pi.
- Establecimiento de una comunicación segura, autenticación y cifrado, entre agente y gestor SNMP.
- Definir e interpretar el intercambio de información entre la Raspberry Pi y el vehículo.
- Implementación del dispositivo de forma autónoma. Una vez inicializado, funciona de forma automática.
- Itercambio de notificaciones y avisos entre equipo gestionado y gestor.

2. Conceptos teóricos

Dado que la tendencia natural de una red cualquiera es crecer conforme se añaden nuevas aplicaciones y más y más usuarios hacen uso de la misma, los sistemas de gestión empleados han de ser lo suficientemente flexibles para poder soportar los nuevos elementos que se añaden, sin necesidad de realizar cambios drásticos en la misma.

Por ello, la gestión de red integrada, como conjunto de actividades dedicadas al control y vigilancia de recursos bajo el mismo sistema de gestión, se ha convertido en un aspecto de enorme importancia en el mundo de las telecomunicaciones. La gestión de red se suele centralizar en un centro de gestión, donde se controla y vigila el correcto funcionamiento de todos los equipos integrados en las distintas redes de la empresa en cuestión. Para ello, el centro de gestión consta de una serie de métodos de gestión, de recursos humanos y de herramientas de apoyo.

Hoy en día la herramienta más utilizada para la gestión de redes es el protocolo SNMP. Este, es usado por gestores de redes y sistemas para recopilar stadisticas de uso (tráfico de la red, uso de la cpu o espacio disponible en el disco duro), así como parámetros de configuración del sistema.

2.1. Internet of Things

La Internet de las cosas es un tema emergente de importancia técnica, social y económica. En este momento se están combinando productos de consumo, bienes duraderos, automóviles y camiones, componentes industriales y de servicios públicos, sensores y otros objetos de uso cotidiano con conectividad a Internet y potentes capacidades de análisis de datos que prometen transformar el modo en que se vive.

Por lo general, el término Internet de las Cosas se refiere a escenarios en los que la conectividad de red y la capacidad de cómputo se extienden a objetos, sensores y artículos de uso diario que habitualmente no se consideran computadoras, permitiendo que estos dispositivos generen, intercambien y consuman datos con una mínima intervención humana. Sin embargo, no existe ninguna definición única y universal.

El concepto de combinar computadoras, sensores y redes para monitorear y controlar diferentes dispositivos ha existido durante décadas. Sin embargo, la reciente confluencia de diferentes tendencias del mercado tecnológico está

permitiendo que la Internet de las Cosas esté cada vez más cerca de ser una realidad generalizada. Estas tendencias incluyen la conectividad omnipresente, la adopción generalizada de redes basadas en el protocolo IP, la economía en la capacidad de cómputo, la miniaturización, los avances en el análisis de datos y el surgimiento de la computación en la nube.

Actualmente, el término Internet de las cosas se usa con una denotación de conexión avanzada de dispositivos, sistemas y servicios que va más allá del tradicional M2M (máquina a máquina) y cubre una amplia variedad de protocolos, dominios y aplicaciones.

Existen múltiples aplicaciones hoy en dia que se aprovechan de esta tecnología. A continuación se pueden observar algunos ejemplos [3]:

- Seguimiento de activos e inventario. Los propietarios de valiosas piezas de equipo, tales como excavadoras, cargadores frontales, motoniveladoras y tractores, pueden estar al tanto de su ubicación y su condición.
- La red eléctrica (smart grid). Utiliza computadoras y las redes de comunicación para dar un mejor seguimiento y control de la red eléctrica.
- Aplicaciones del IoT para el uso doméstico. Control de elementos del hogar como luces, temperatura, etc.
- Agricultura y Ganadería. Hacer seguimiento de magnitudes como la temperatura, humedad, luminosidad y demás factores que pueden influir en la producción.
- Automatización industrial. Esto también ya se está dando en las fábricas para el control de procesos. Actualmente, los procesos de fabricación se supervisan y controlan internamente. Con la capacidad de conexión a internet se suma el potencial para el reporte y control en forma remota.
- Medicina y deportes. Los médicos pueden monitorizar las condiciones de un paciente y diagnosticar problemas o recibir alertas de eventos potencialmente letales. En los deportes, el uso de sensores para la recopilación de datos de salud y rendimiento de los atletas.
- Gestión del tráfico. Uso de datos anónimos de los usuarios para enviar la posición y velocidad a través del móvil y poder mostrar la información de las carreteras.

El uso de IoT no solo corresponde a una ventaja tecnológica, sino que también es un arma de doble filo en términos de seguridad. La naturaleza interconectada de los dispositivos de la IoT significa que cada dispositivo mal asegurado conectado a Internet podría afectar la seguridad y la resistencia de Internet a nivel global.

La seguridad es realmente importante en diferentes aspectos cuando se usa este tipo de tecnología. Por ejemplo, que un vehículo que este en marcha sea atacado y ponga en peligro a sus ocupantes [2].

Dado que, protocolos como HTTPS o similares son difíciles de implementar debido a su alta necesidad de computo, como alternativa, para solventar los problemas de seguridad, se utiliza el protocolo SNMP en su tercera versión.

2.2. Evolución histórica del protocolo SNMP

El protocolo de gestión de red simple o SNMP (Simple Network Management Protocol), es un protocolo de la capa de aplicación que facilita el intercambio de información de gestión entre dispositivos de red [4]. La primera versión de este protocolo (SNMPv1) fue diseñado a mediados de los 80 por Case, McCloghrie, Rose, and Waldbusser, como una solución a los problemas de comunicación entre diferentes tipos de redes.

SNMP se sitúa en el tope de la capa de transporte de la pila OSI, o por encima de la capa UDP de la pila de protocolos TCP/IP. Siempre en la capa de transporte como se puede observar en la siguiente figura

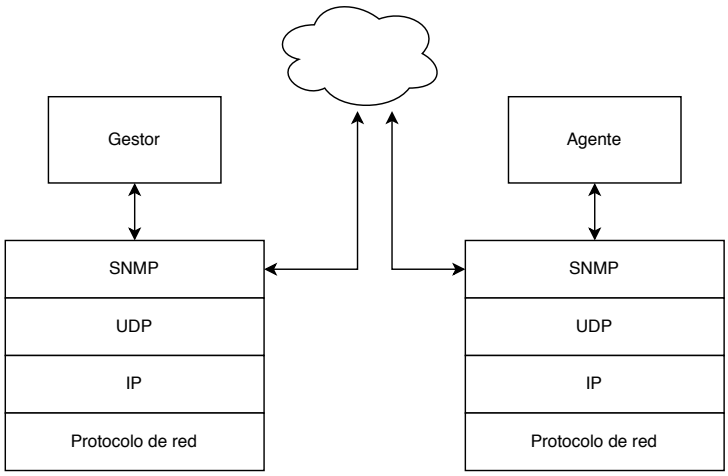


Figura 1: Arquitectura TCP/IP

Existen tres versiones de SNMP: SNMP versión 1 (SNMPv1), SNMP versión 2 (SNMPv2) y SNMP versión 3 (SNMPv3). SNMPv1 constituye la primera definición e implementación del protocolo SNMP, estando descrito en las RFC 1155, 1157 y 1212 del IETF (Internet Engineering Task Force). El vertiginoso crecimiento de SNMP desde su aparición en 1988, puso pronto en evidencia sus debilidades, principalmente su imposibilidad de especificar de una forma sencilla la transferencia de grandes bloques de datos y la ausencia de mecanismos de seguridad; debilidades que tratarían de ser subsanadas en las posteriores definiciones del protocolo.

2.3. Modelos

Una Arquitectura de gestión describe un marco general de un modelo de gestión integrado en un entorno heterogéneo.

Existen cuatro modelos: Modelo de la información, modelo organizativo, modelo de comunicación y modelo funcional.

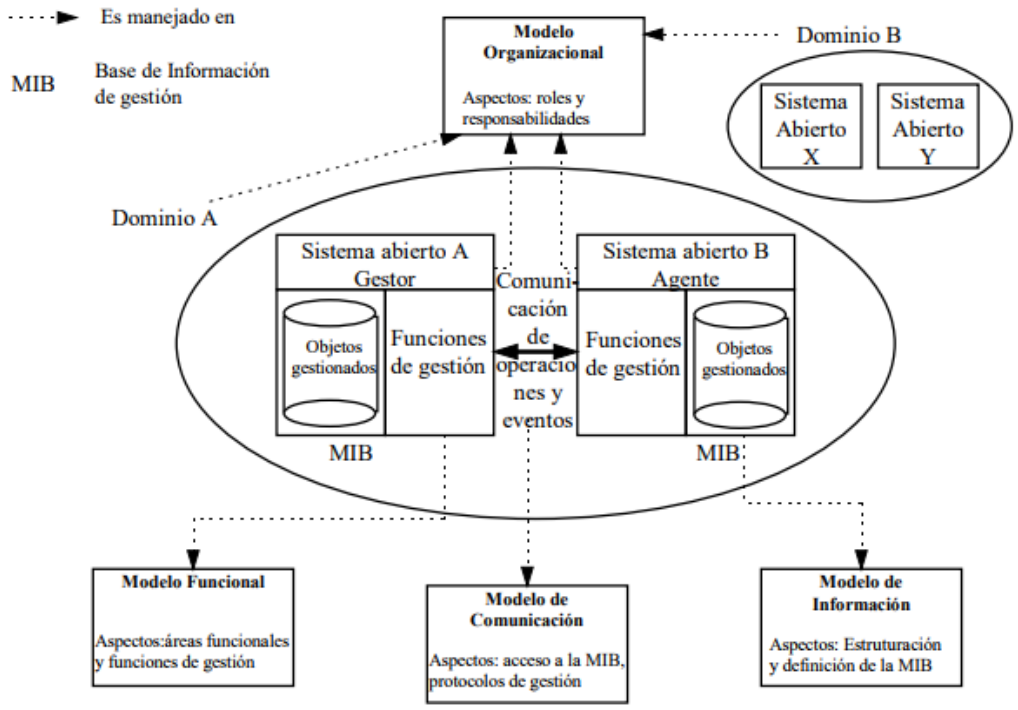


Figura 2: Modelo de gestión en OSI

2.3.1. Modelo de la información

El papel de la información de gestión en el entorno de administración, considera las reglas para definir la información de administración. Estas reglas están descritas a través del uso de la estructura de la información de administración (SMI) que precisa las reglas para definir la información de administración independientemente de los detalles de implementación, utilizando para ello el lenguaje de Notación Sintáctica Abstracta ASN.1.

SMI define la base de datos que almacena una colección de objetos administrados pertenecientes a la red, esta es llamada la base de información de administración (MIB). Los agentes mantendrán una MIB local, atenderán solicitudes de la estación de gestión y podrán enviar de manera asíncrona informes de eventos importantes. La MIB local de cada agente mantiene información sobre objetos del recurso que gestiona almacenada en forma de pares atributo-valor. Los objetos están estandarizados para recursos del mismo tipo (p.e., todos los concentradores tendrán los mismos objetos).

2.3.2. Modelo administrativo

Este modelo que se ocupa de establecer los diferentes papeles o funciones dentro del sistema de gestión y su distribución espacial.

Se basa en el concepto de perfiles de comunidad y políticas administrativas entre agentes y gestores.

Conceptos: En el entorno de gestión, una entidad SNMP es una "entidad lógica" en nombre de la cual un agente o una aplicación de gestión están procesando un mensaje. El entorno de gestión es responsable de proporcionar los siguientes atributos.

- Autentificación: Como las entidades SNMP identifican sus mensajes.
- Privacidad: Como las entidades SNMP protegen sus mensajes.
- Autorización: Como una entidad agente SNMP determina los objetos que son accesibles a una entidad de aplicación de gestión dada, y las operaciones que se pueden realizar en estos objetos.

SNMP v2 está diseñado para soportar múltiples entornos de administración. El entorno que se observará se denomina entorno de gestión basado en comunidades, debido a que usa el concepto de comunidad empleado en el

SNMP original.

SNMP define una comunidad como una relación entre entidades SNMP. Una comunidad SNMP se escribe como una cadena de octetos sin interpretación. Esta cadena se llama nombre de comunidad. Cada octeto toma un valor entre 0 y 255.

Cuando se intercambian mensajes SNMP, contienen dos partes, una cadena de comunidad que es mandada en texto sencillo y datos, que contienen una operación SNMP y los operandos asociados.

En cada mensaje SNMP V3 los parámetros de seguridad están codificados como una cadena de octetos. El significado de estos parámetros de seguridad depende del modelo de seguridad que se utiliza. SNMPv3 proporciona características de seguridad importantes:

- Confidencialidad: El cifrado de paquetes para impedir la escucha por una fuente no autorizada.
- Integridad: Integridad de los mensajes para asegurar que un paquete no ha sido alterado durante el tránsito que incluye un mecanismo opcional por repetición de paquetes.
- Autenticación: Para comprobar que el mensaje es de una fuente válida.

2.3.3. Modelo de la comunicación

Define los esquemas para el intercambio de información entre los diferentes componentes del sistema de gestión. En nuestro caso se basa en el intercambio de información entre agente y gestor mediante SNMP.

SNMP es un protocolo asíncrono de petición-respuesta basado en el modelo de interrupción-sondeo directo, esto significa que una entidad no necesita esperar una respuesta después de enviar un mensaje, por lo que puede enviar otros mensajes o realizar otras actividades.

SNMP tiene pocos requisitos de transporte ya que usa un servicio de transporte no orientado a conexión, y que normalmente es UDP. Aunque esto ratifica el Axioma Fundamental, hay una razón más importante: Las funciones de administración de red se realizan cuando hay problemas, de modo que la aplicación de administración es la que decide qué restricciones realiza para el tráfico de administración. La elección de un servicio de transporte no

orientado a conexión permite a la estación determinar el nivel de retransmisión necesario para complacer a las redes congestionadas.

Mensajes SNMP

SNMP permite el intercambio de información a través de la red entre la estación de gestión y el agente en forma de mensajes SNMP. Cada mensaje incluye un número de versión que indica la versión de SNMP, un nombre de comunidad utilizado en el intercambio, y uno de los cinco tipos de PDU's definidos: GetRequest, GetNextRequest, SetRequest, GetResponse y Trap.

Las tramas tienen el siguiente formato [5]

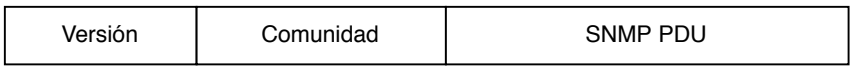


Figura 3: Mensaje SNMP

Donde:

- Versión: Número de versión de protocolo que se está utilizando (0 para SNMPv1, 1 para SNMPv2c, 2 para SNMPv2p y SNMPv2u, 3 para SNMPv3, ...)
- Comunidad: La función de este campo es enviar junto con la tarea que se pretende llevar a cabo una identificación básica del usuario. Su fin es controlar el acceso no autorizado a un dispositivo SNMP.
- SNMP PDU: Contenido de la Unidad de Datos de Protocolo, depende de la operación que se ejecute.

Tipos de operaciones/mensajes (PDUs)

Existen tres tipos de de operaciones que se pueden realizar sobre una entidad SNMP:

- **Lectura**: un gestor recupera instancias de objetos gestionados de un agente (Get, GetNext, Get-Bulk). Permiten al gestor obtener el estado del dispositivo gestionado, mediante el mecanismo de sondeo o polling.
- **Escritura**: un gestor modifica o crea nuevas instancias de objetos en un agente (SET). Permiten al gestor actuar sobre el dispositivo gestionado.

- **Notificaciones**: un agente notifica a un gestor de la ocurrencia de una situación anómala (TRAP, SNMPv2-TRAP, INFORM). El gestor es informado inmediatamente, evitando los retardos debidos al sondeo

A continuación, se describen los campos del tipo de dato PDU. En la figura que se muestra a continuación se puede obervar su estructura [6].

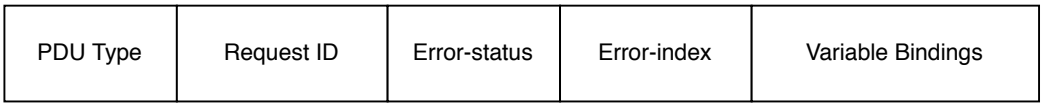


Figura 4: Formato de PDU

Donde:

- PDU type: indica el tipo de PDU.
- Request Id: se utiliza para diferenciar las distintas peticiones, añadiendo a cada una de ellas un único identificador.
- Error-status: se utiliza para indicar que a ocurrido una excepción durante el procesamiento de una petición. Sus valores posibles son: noError(0), tooBig(1), noSuchName(2), badValue(3), readOnly(4), genErr(5).
- error-index: si no es cero indica qué variable de la petición es errónea.
- Variable Bindings: es una lista de nombres de variables y sus correspondientes valores. En algunos casos (GetRequest), el valor de las mismas es NULL. En el caso de las Traps, proporcionan información adicional relativa a la Trap, dependiendo el significado de este campo de cada implementación en particular.

Peticiones de Lectura

Para examinar eficientemente la información de administración, el entorno de administración usa un método inteligente para identificar las instancias: Es usado un OBJECT IDENTIFIER, formado por la concatenación del nombre del tipo objeto con un sufijo.

- **El operador Get**

Si la aplicación de administración conoce las instancias que necesita, realiza una get-request. Sólo se pueden devolver los errores tooBig y genErr. Por otro lado, para cada variable de la petición, la instancia se recupera de la vista del

MIB con esta operación: Si el agente no implementa el tipo objeto asociado con la variable, se devuelve la - excepción noSuchObject. Si la instancia no existe o no la soporta el MIB, se devuelve la excepción noSuchInstance.

- **El operador Get-Next**

Si la aplicación no conoce exactamente la instancia de la información que necesita, realiza una getnext-request. Sólo se pueden devolver los errores tooBig y genErr.

Por otro lado, para cada variable de la petición, se devuelve la primera instancia que siga a la variable que esté en la vista del MIB con esta operación:

- Si no hay una próxima instancia para la variable en el MIB, se devuelve una excepción endOfMibView.
- Si se reconoce la siguiente instancia de la variable en el MIB, se devuelve el valor asociado. El operador get-next puede usarse para comprobar si un objeto es soportado por un agente.

Debido al almacenamiento que se hace en el MIB, las tablas se examinan en un orden columna-fila; así, se examina cada instancia de la primera columna, cada instancia de la segunda y así seguidamente hasta el final de la tabla. Entonces, se devuelve la instancia del siguiente objeto, y sólo se devuelve una excepción si el operando de get-next es mayor, lexicográficamente hablando, que la mayor instancia del MIB.

Como el operador get-next soporta múltiples operandos, se puede examinar eficientemente la tabla entera. La aplicación de administración conoce que llega al final de la tabla cuando se devuelve un nombre cuyo prefijo no coincide con el del objeto deseado.

- **El operador Get-Bulk**

Su función es minimizar las interacciones en la red, permitiendo al agente devolver paquetes grandes. Este esquema tiene que funcionar con un servicio de transporte no orientado a conexión de modo que la aplicación sea la responsable de controlar las interacciones. Las aplicaciones de administración también deben controlar los tiempos de las interacciones.

Cuando un agente recibe una get-bulk, calcula el mínimo del tamaño máximo del mensaje del emisor y del tamaño de la generación de mensajes del propio agente.De aquí resta la suma de dos cantidades, el tamaño de las capas de

privacidad/autentificación de la generación de la respuesta y del tamaño de una respuesta sin variables instanciadas.

Dicha diferencia indica la cantidad máxima de espacio posible para las instancias de las variables en la respuesta. Si la diferencia es menor de cero, la respuesta se pierde, si no, se genera la respuesta, que tendrá cero o más variables instanciadas.

El agente examina las variables de la petición usando el operador get-next y añadiendo cada nueva instancia con su valor a la respuesta y disminuyendo la cantidad de espacio libre. Si no hay suficiente espacio, se envía la respuesta antes de colapsar el espacio libre.

Finalmente, el espacio libre se ocupa, o se realiza el máximo número de repeticiones. Es importante tener en cuenta que el agente puede terminar una repetición en cualquier momento.

Peticiones de Escritura

El operador set es una aplicación de administración que conoce exactamente las instancias que necesita y genera un set-request.

La semántica de la operación set es tal que la operación tiene éxito si y sólo si todas las variables se actualizan.

Cada instancia se examina y se hace una comprobación para verificar que la variable es soportada por la vista del MIB para esa operación y si la variable existe, el agente puede modificar las instancias del tipo objeto correspondiente y el valor proporcionado es correcto sintácticamente.

Si la variable no existe, el agente devuelve el error apropiado y se liberan los recursos.

Notificaciones / Avisos

Cuando un agente detecta un evento extraordinario, se genera una invocación snmpV2-trap, que se envía a una o más aplicaciones de administración.

En este caso su estructura es la siguiente:

PDU Type	Enterprise	Agent-addr	Generic-trap	Specific-trap	TimeStamp	VB

Figura 5: PDU de tipo trap

Donde:

- Enterprise: Identifica el subsistema de gestión de red que ha emitido el Trap.
- Agent-addr: Dirección IP del agente que generó el Trap.
- Generic-trap: Tipo de Trap genérico predefinido.
- Specific-trap: código de Trap específico e indica de una forma más específica la naturaleza del Trap.
- Timestamp: Tiempo transcurrido entre la última reinicialización de la entidad de red y la generación del trap.

Direcciones y Dominios de transporte

La entidad que envía transforma y envía el mensaje como un único datagrama UDP a la dirección de transporte de la entidad receptora. Por convención, los agentes SNMP escuchan en el puerto UDP 161 y envían las notificaciones al puerto UDP 162, aunque una entidad debería configurarse para usar cualquier selector de transporte aceptable.

En la siguiente figura se puede observar el esquema de comunicación entre un sistema agente y un sistema gestor.

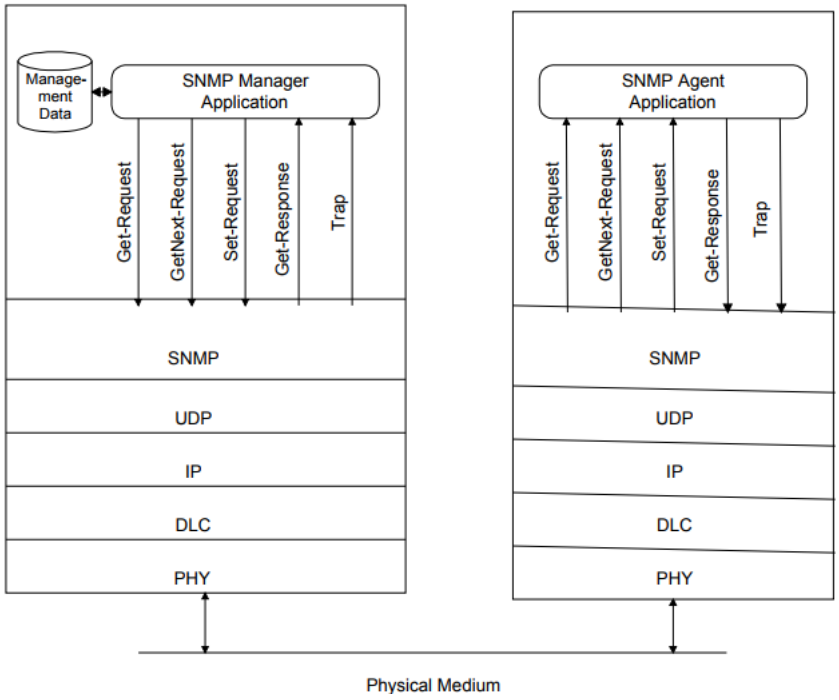


Figura 6: Arquitectura SNMP

2.4. MIB

La Base de Información Gestionada (Management Information Base o MIB) es un tipo de base de datos que contiene información jerárquica, estructurada en forma de árbol, de todos los parámetros gestionables en cada dispositivo gestionado de una red de comunicaciones. Define las variables usadas por el protocolo SNMP para supervisar y controlar los componentes de una red.

Está compuesta por una serie de objetos que representan los dispositivos en la red. Cada objeto manejado en un MIB tiene un identificador de objeto único e incluye el tipo de objeto, el nivel de acceso, restricciones de tamaño, y la información del rango del objeto. Los objetos de una MIB se definen usando un subconjunto del ASN.1, la versión 2 de la estructura de la información de gestión (Structure of Management Information Version 2 o SMIv2) definido en el RFC 2578 [7].

La MIB-II es la base de datos común para la gestión de equipos en Internet. Se apoya en el modelo de información estructurada definido en el RFC 1155,

que establece las bases para definir la MIB, indica los tipos de objetos que se pueden usar y define el uso de ASN.1. Esta cuelga del nodo 1.3.6.1.2.1 del árbol de registro.

Dentro de la MIB II existen dos tipos de nodos: estructurales y de información.

- **Nodos estructurales**: Son aquellos que solo tienen descrita su posición en el árbol. "son ramas". Por ejemplo:

```
ip OBJECT IDENTIFIER ::= { 1 3 6 1 2 1 4 }
```

Figura 7: Ejemplo nodo estructural

- **Nodos con información**: Son nodos "hoja". De ellos no cuelga ningún otro nodo. Estos nodos están basados en la macro OBJECT TYPE, por ejemplo:

```
ipInReceives OBJECT TYPE
        SYNTAX   Counter
        ACCESS   read-only
        STATUS   mandatory
        DESCRIPTION   " Texto indicando para que vale "
::={ 3 }
```

Figura 8: Ejemplo nodo información

Este fragmento ASN.1 (Figura 7) nos indica que el objeto "ipInReceives" es un contador de sólo lectura que es obligatorio incorporar si se quiere ser compatible con la MIB-II (aunque luego no se utilice) y que cuelga del nodo ip con valor tres. Previamente se ha mostrado el nodo estructural "ip" con su valor absoluto. En este se puede ver que el identificador del objeto "ipInReceives" es "1.3.6.1.2.1.4.3".

Estructura de la MIB II

Todos los objetos de la MIB de SNMP, se identifican de la siguiente forma: iso identigfied-organization(3) dod(6) internet(1) mgmt(2) mib2(1)... o, de manera alternativa 1 3 6 1 2 1 ....

La MIB II se compone de los siguientes nodos estructurales:

- System: de este nodo cuelgan objetos que proporcionan información genérica del sistema gestionado. Por ejemplo, dónde se encuentra el sistema, quién lo administra...
- Interfaces: En este grupo está la información de las interfaces de red presentes en el sistema. Incorpora estadísticas de los eventos ocurridos en el mismo.
- At (address translation o traducción de direcciones): Este nodo es obsoleto, pero se mantiene para preservar la compatibilidad con la MIB-I. En él se almacenan las direcciones de nivel de enlace correspondientes a una dirección IP.
- Ip: En este grupo se almacena la información relativa a la capa IP, tanto de configuración como de estadísticas.
- Icmp: En este nodo se almacenan contadores de los paquetes ICMP entrantes y salientes.
- Tcp: En este grupo está la información relativa a la configuración, estadísticas y estado actual del protocolo TCP.
- Udp: En este nodo está la información relativa a la configuración, estadísticas del protocolo UDP.
- Egp: Aquí está agrupada la información relativa a la configuración y operación del protocolo Exterior Gateway Protocol.
- Transmission: De este nodo cuelgan grupos referidos a las distintas tecnologías del nivel de enlace implementadas en las interfaces de red del sistema gestionado.
- SNMP: En este nodo se almacena la información relativa a SNMP.

Esta estructura se puede observar en la siguiente figura.

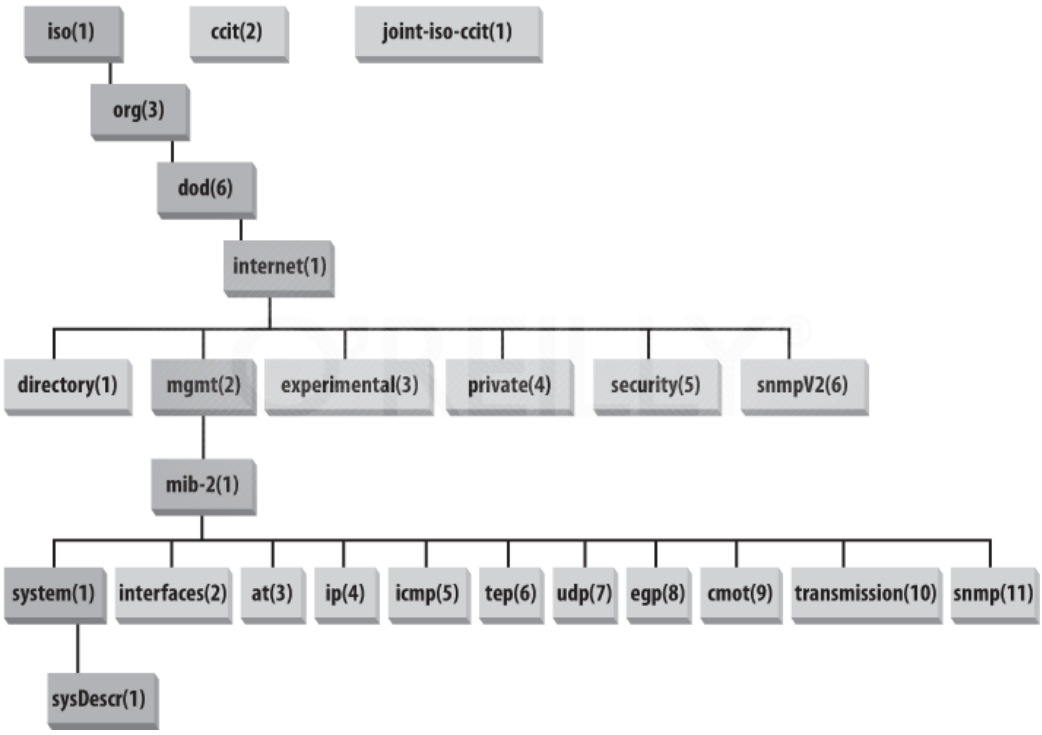


Figura 9: Árbol MIB II

Sintaxis

En el RFC 1155 están definidos los siguientes tipos de objetos:

- Universales
  - **Integer**: para objetos que se representen con un número entero.
  - **Octet String**: para texto.
  - **Null**: cuando el objeto carece de valor.
  - **Object Identifier**: para nodos estructurales.
  - **Sequence y Sequence of**: para arrays.
- Application Type
  - **IpAddress**: para direcciones IP.
  - **Counter**: para contadores.
  - **Gauge**

  - **Timeticks**: para medir tiempos. Cuenta en centésimas de segundos.
  - **Opaque**: para cualquier otra sintaxis ASN.1.

- Definición de tablas

  - Las tablas son un tipo estructurado. se definen usando los tipos "Sequence" y "Sequence of" y la cláusula "index". La tabla consiste en un array ("Secuence") de secuencias de ("Secuence of") filas.

2.5. On Board Diagnostics

OBD (On Board Diagnostics) es un sistema de diagnóstico a bordo en vehículos (coches y camiones). Actualmente se emplean los estándares OBD-II (Estados Unidos), EOBD (Europa) y JOBD (Japón) que aportan un monitoreo y control completo del motor y otros dispositivos del vehículo. Los vehículos pesados poseen una norma diferente, regulada por la SAE, conocida como J1939.

Evolución histórica de OBD

Para reducir la contaminación del aire, la "California Air Resources Board" (CARB) determinó en 1988 que todos los automóviles a gasolina contaran con OBD (On Board Diagnostics), que controlara los límites máximos de emisiones y además un autocontrol, el On Board Diagnostics de componentes relevantes de las emisiones de gas a través de dispositivos de mando electrónicos. Para que el conductor detecte un mal funcionamiento del OBD se impuso la obligación de tener una lámpara que indique fallos (MIL - Malfunction Indicator Lamp).

Medidas más estrictas en los límites de emisiones en 1996 llevó a la creación del OBD II. La interfaz estándar del OBD-II no solamente es utilizada por el fabricante para sus funciones avanzadas de diagnóstico sino también por aquellos que van más allá de lo que la ley exige.

A continuación, se va a mostrar en mayor detalle el protocolo OBD-II. Esto es debido, a que es utilizado a lo largo de la realización del proyecto.

OBD II

OBD II es la segunda generación del sistema de diagnóstico a bordo, sucesor de OBD I. Alerta al conductor cuando el nivel de las emisiones es 1.5 mayor a las diseñadas. A diferencia de OBD I, OBD II detecta fallos eléctricos, químicos y mecánicos que pueden afectar al nivel de emisiones del vehículo. Por ejemplo, con OBD I, el conductor no se daría cuenta de un fallo químico del catalizador. Con OBD II, los dos sensores de oxígeno, uno antes y el otro después del catalizador, garantizan el buen estado químico del mismo.

Para ofrecer la máxima información posible para el mecánico, guarda un registro del fallo y las condiciones en que ocurrió. Cada fallo tiene un código asignado. El mecánico puede leer los registros con un dispositivo que envía comandos al sistema OBD II llamados **PID (Parameter ID)**.

Hoy en día OBD II es el método más utilizado para extraer información de un vehículo y así detectar posibles averías. Existen multitud de programas que hacen uso de esta tecnología. Estas herramientas, se encargan de preguntar e interpretar la información que proviene del vehículo, y además, se encargan mostrarsela al usuario de forma gráfica.

En la actualidad se pueden encontrar aplicaciones para móviles como **DashCommand**, la cual permite monitorizar en tiempo real el estado del vehículo. Para ello hace uso de una interfaz gráfica que muestra la información de manera intuitiva. En la Figura 10 se puede observar dicha interfaz. Además de esta app, existen muchas otras para el mismo cometido, como Torque Pro, HobDrive, OBD Car Doctor Pro, etc [8].

Además de aplicaciones para móviles, también existen diferentes programas para ordenador que permiten realizar la misma tarea. Un ejemplo de estas herramientas es Scantool. A través de esta aplicación, es posible observar la información del vehículo en un ordenador. En concreto, esta solo se es posible usarla en sistemas operativos en base linux, como por ejemplo Ubuntu.

Figura 10: Interfaz principal de DashCommand

Más adelante se muestra con más detalle esta tecnología.

3. Aspectos prácticos

En este apartado se muestran los aspectos prácticos del proyecto desarrollado. Se detallan los componentes hardware y las aplicaciones software que son utilizadas durante la realización del trabajo.

3.1. Raspberry Pi

La Raspberry Pi Foundation es una organización sin ánimo de lucro que dio sus primeros pasos como fundación en 2008, pero que en realidad llevaba gestándose desde mucho tiempo atrás.

En 2011 desarrolló la Raspberry Pi, un ordenador de placa reducida o (placa única), que revoluciona el mundo de los SBC (Single Board Computer) por su pequeño tamaño y bajo costo, desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

En la siguiente figura se puede observar el hardware que usa la raspberry 3.

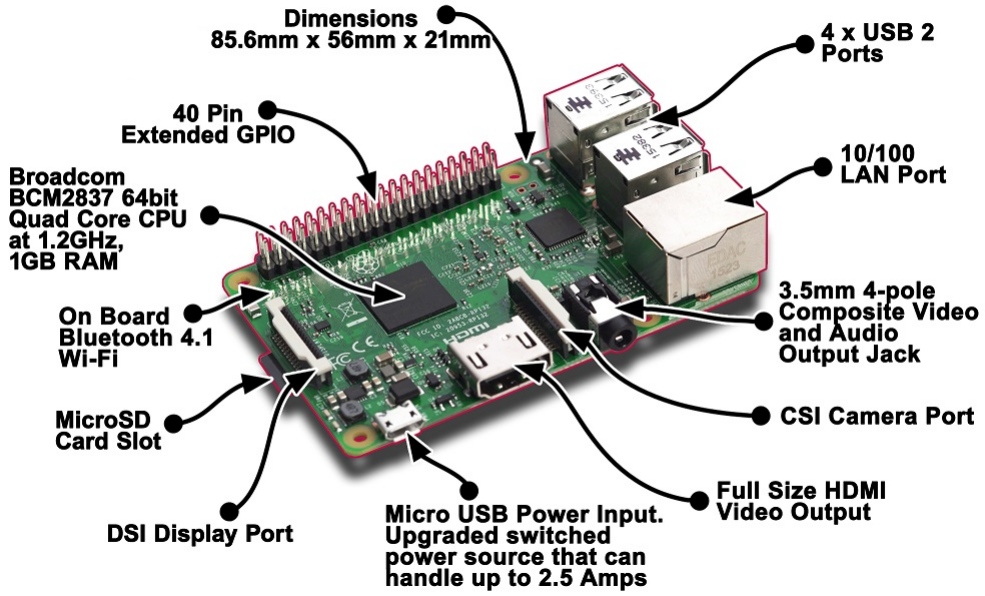


Figura 11: Raspberry Pi 3

Normalmente, el uso de la Raspberry Pi está destinados a proyectos de carácter educativo. A lo largo de los años de vida de este mini-ordenador los

desarrolladores han hecho con él de todo, desde convertirlo en un completo ordenador portátil con Linux hasta tenerlo como una retro-consola, configurar un termostato o una unidad de control de una casa inteligente.

Algunos usos que han sido dados a este ordenador son la conversión de este dispositivo en un centro multimedia de manera que se pueda conectarlo a una televisión y poder acceder a todo el contenido multimedia, por ejemplo, a través de Kodi [9].

Otros uso para el que se ha usado este equipo es la conversión de este dispositivo en un servidor casero. De esta manera se puede tener, por ejemplo, un servidor VPN siempre disponible (incluso un nodo Tor), así como un servidor de archivos para compartir dentro de la red local, un cliente de descargas torrent e incluso montar una nube propia gracias a plataformas como Nextcloud.

En el caso del trabajo, se hace uso de la Raspberry Pi como un miniordenador capaz de comunicarse con un vehículo y convertir esta información en paquetes SNMP para que un gestor sea capaz de obtenerla.

3.2. OBD II

En los siguientes apartados se resumen el interfaz físico OBDII, los diversos protocolos de capa de enlace usados en OBDII y los modos estandarizados de acceso a la información del vehículo.

Interfaz OBD II

El conector OBDII, estandarizado en la norma SAE J1962, es un conector hembra de 16 pines, dispuestos en dos filas, que se localiza en el habitáculo del vehículo, cercano al tablero de mandos en la plaza del conductor. En la Figura 11 se puede observar un esquema del conector OBDII similar al que se puede encontrar en cualquier vehículo actual.

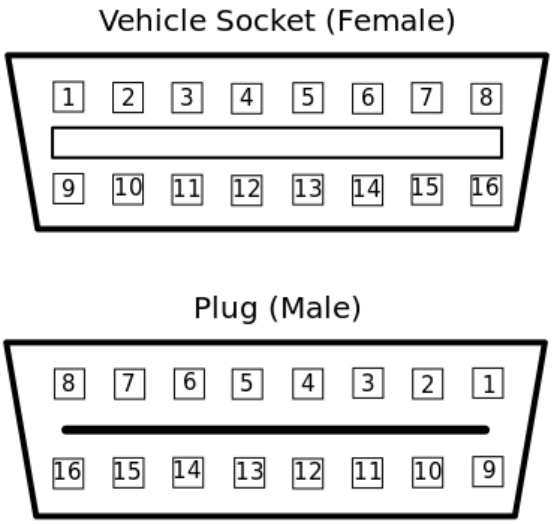


Figura 12: Pines OBD2

Aunque los fabricantes pueden hacer uso de los pines no estandarizados si lo desean, dentro de este proyecto sólo se han utilizado aquellos que se recogen en el estándar. El significado de cada uno de los 16 pines recogidos en la norma SAE J1962 se muestra en la siguiente tabla:

Pin	Descripción	Pin	Descripción
1	No estandarizado	9	No estandarizado
2	J1850 PWM + y J1850 VPW +	10	J1850 PWM-
3	No estandarizado	11	No estandarizado
4	GND (señal)	12	No estandarizado
5	CAN high	13	No estandarizado
6	K-Line para ISO 14230-4 e ISO 9141-2	14	CAN low
7	No estandarizado	15	L-Line para ISO 9141-2
8	No estandarizado	16	+12V

Figura 13: ines OBD II

Protocolos de capa de enlace OBD II

Dentro de la norma SAE J1979 se recogen cinco protocolos de señales distintos utilizados en el interfaz OBDII, siendo cada fabricante el que decide

cuál de ellos utilizar en cada modelo de vehículo. Estos protocolos describen los principales requisitos en el enlace de datos y la comunicación física entre el vehículo y el equipo externo.

1. SAE J1850 PWM: Se trata de un protocolo de bus para las comunicaciones con herramientas de diagnóstico dentro de un vehículo que es utilizado principalmente por Ford. Hace uso de una modulación del ancho de pulso para codificar la información.

2. SAE J1850 VPWM: Es un protocolo muy similar al anterior con la diferencia que utiliza una modulación de ancho de pulso variable para codificar la información.

3. ISO 9141-2: Se considera el estándar promovido por los organismos europeos para satisfacer los requisitos de OBDII en vehículos con una tensión de alimentación de 12 voltios.

4. ISO 14230 (KWP2000): Se trata de una versión renovada del protocolo ISO 9141-2, con el que comparte una capa física idéntica.

5. ISO 15765 (CAN): Define el estándar para el diagnóstico de vehículos de carretera sobre el bus CAN (Controller Area Network). Actualmente es utilizado por fabricantes como Ford, Volvo y Mazda, aunque progresivamente se está convirtiendo en el bus interno más utilizado en el automóvil, dadas sus altas prestaciones en velocidad y fiabilidad. Por ello, se intenta utilizar de igual forma el bus CAN para efectuar la comunicación entre el procesador central de abordo y los equipos de diagnóstico externos.

Modos estandarizados de acceso a OBD II

OBDII proporciona gran cantidad de datos desde las ECUs (Unidades Electrónicas de Control) del vehículo y ofrece una fuente muy rica de información cuando existe un problema dentro del mismo. Las ECUs se pueden considerar como el cerebro de cualquier sistema electrónico de control del vehículo; suelen estar compuestas por varios circuitos integrados y componentes que se montan sobre una placa de circuito impreso y se empaquetan de forma robusta, a fin de soportar las condiciones adversas bajo las que suelen trabajar.

Para el acceso a la información que se puede obtener a través de OBD-II, la norma SAE J1979 define varios modos de operación, en concreto 9. Cada uno de ellos permite acceder a distinto tipo de información referente al vehículo,

si bien existen modos con ciertas similitudes. En la Figura 15, se realiza una breve descripción de los modos de medición estandarizados.

Los bytes de respuesta se representan con las letras A, B, C, etc. A es el byte más significativo. Los bits de cada byte se representan del más significativo al menos con los números del 7 al 0:

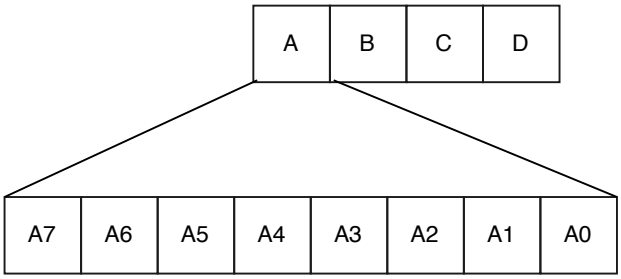


Figura 14: Formato bytes respuesta

A continuación se muestra como es un intercambio de información mediante OBD-II.

En primer lugar el agente que desea interrogar al vehículo inicia la comunicación con un comando estandarizado por OBD-II. Por ejemplo, si el agente desease obtener las revoluciones por minuto del motor en tiempo real se enviaría la siguiente petición.

```
01 0C
```

En esta petición, `01` hace referencia se trata de una petición del modo 1. Por otro lado, `0C` indica que se desea obtener el valor del pid 12, que es las revoluciones por minuto del motor.

Como resultado a la petición se podría obtener la siguiente respuesta:

```
41 0C 1A F8
```

En este caso la respuesta está codificada en dos bytes, `0x1A` y `0xF8`. El valor retornado (1A F8) es en realidad un valor entero hexadecimal de dos bytes. Convertido a decimal es 6904. De acuerdo al estándar, el valor retornado está en cuartos de vuelta, por lo que al dividir 6904/4 las rpm del motor son 1726. Es de notar que esta comunicación que se acaba de realizar fue completamente independiente del tipo de vehículo y protocolo utilizado.

Modo	Descripción
1	Proporciona el acceso a datos en tiempo real de valores analógicos o digitales de salidas y entradas de las ECU del vehículo, así como información sobre el estado de los sensores
2	Las ECUs toman una muestra de todos los valores relacionados con las emisiones, en el momento exacto de ocurrir un fallo
3	Este modo permite extraer de la memoria de las ECUs del vehículo todos los códigos de error (DTCs) almacenados en cada una de ellas
4	Borra los datos almacenados, incluyendo los códigos de error (DTC)
5	Este modo devuelve los resultados de las pruebas realizadas a los sensores de oxígeno del vehículo para determinar el funcionamiento de los mismos y, basándose en ellos, analizar la eficiencia del convertidor catalítico
6	Es una solicitud que permite obtener los resultados de todas las pruebas de abordo, para sistemas monitorizados de forma continua o no continua
7	Este modo permite leer de la memoria de las ECUs todos los DTCs detectados con una antigüedad máxima de un único ciclo de conducción, es decir, que aparecieron en el último o en el ciclo actual
8	Este modo permite poder llevar a cabo ciertas operaciones sobre un sistema de abordo mediante un equipo de testeo externo
9	Es utilizado para recuperar información específica del vehículo

Figura 15: Tabla que muestro los modos OBD-II

3.3. Herramientas Software

En este apartado se detalla el software utilizado para el desarrollo del proyecto.

En el proyecto el sistema operativo que se ha decidido utilizar es Ubuntu Mate, que es un sistema operativo basado en GNU/Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity.

Está orientado al usuario novel y promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia de usuario. Está compuesto de

múltiple software normalmente distribuido bajo una licencia libre o de código abierto.

Al usar este sistema operativo, para la implementación de SNMP se ha usado el software NET-SNMP explicado más adelante.

Para la instalación de dicha distribución, se ha usado una tarjeta micro SD vacía. Dicho SO se ha descargado desde la propia página de Raspberry Pi. El proceso de instalación está completamente automizado por lo que solo hay que seguir los pasos que se indican.

Para el desarrollo del proyecto se han usado las siguientes herramientas:

- Editores de texto, por ejemplo Gedit. Usados para escribir el programa y editar contenido de los archivos de configuración.
- SNMPD, el cual es un agente SNMP que escucha un determinado puerto y espera paquetes SNMP. Cuando llega una petición, este recolecta la información de dicha PDU y devuelve otra PDU con el resultado a la fuente.
- Herramientas como Net-SNMP, OBDSim y Socat, las cuales se detallan mas adelante.

Para facilitar el desarrollo del proyecto sobre la Raspberry Pi, se ha instalado un software adicional que permite el uso de esta en modo escritorio remoto. De tal forma que el contenido de la Raspberry Pi se visualiza en el entorno del desarrollador, sin necesidad de tener que conectarle salidas de vídeo, ni teclados ni ratones.

Para realizar esto, se ha instalado sobre la Raspberry Pi el software XRDP. Esto se ha hecho mediante el siguiente comando.

```
$ sudo apt-get install xrdp
```

En el entorno del desarrollador, se ha utilizado el software escritorio remoto , el cual es propio de Windows, para realizar la conexión. Para poder realizar este enlace es necesario conocer la dirección ip de la Raspberry. Esto se realiza aplicando el comando `ifconfig` en esta.

3.3.1. Entorno de gestión NET-SNMP

NET-SNMP es un conjunto de aplicaciones usado para implementar el protocolo SNMP usando IPv4 e IPv6. Incluye [10] :

- Aplicaciones de línea de comandos para:
  - Tomar información de dispositivos capaces de manejar el protocolo SNMP, ya sea usando peticiones simples (snmpget, snmpgetnext) o múltiples (snmpwalk, snmptable, snmpdelta).
  - Manipular información sobre la configuración de dispositivos capaces de manejar SNMP (snmpset).
  - Conseguir un conjunto de informaciones de un dispositivo con SNMP (snmpdf, snmpnetstat, snmpstatus).
  - Traducir entre OIDs numéricos y textuales de los objetos de la MIB, y mostrar el contenido y estructura de la MIB (snmptranslate).
- Un navegador gráfico de la MIB (tkmib), usando Tk/perl.
- Un demonio para recibir notificaciones SNMP (snmptrapd). Las notificaciones seleccionadas pueden guardarse en un log (como syslog o un archivo de texto plano), ser reenviadas a otro sistema de gestión de SNMP, o ser pasadas a una aplicación externa.
- Un agente configurable para responder a peticiones SNMP para información de gestión (snmpd). Incluye soporte para un amplio rango de módulos de información de la MIB, y puede ser extendido usando módulos cargados dinamicamente, scripts externos y comandos.
- Una biblioteca para el desarrollo de nuevas aplicaciones SNMP, con APIs para C y Perl.

3.3.2. OBDSim

OBDsim es un programa que permite simular un dispositivo ELM327. Usando este software, se puede simular el comportamiento que tiene un vehículo usando OBD-II. A continuación se muestran algunas características del programa [11]:

- Uso de comando AT ELM 327.
- Soporte para múltiples ECUs.
- Soporte para diferentes protocolos de enlace.

En la siguiente figura se muestra la interfaz gráfica del programa.

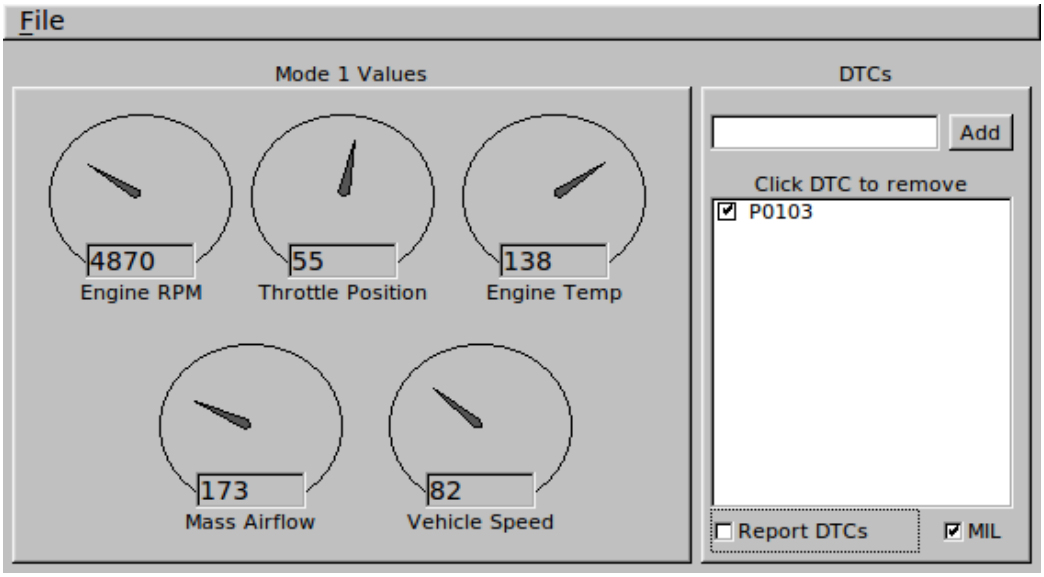


Figura 16: Interfaz gráfica de OBDSim

3.3.3. Socat

Socat es un utilidad basada en line de comandos que establece una comunicación bidireccional para la transferencia de datos. Debido a que los flujos de datos pueden ser de muchos tipos, Socat se puede usar para múltiples propósitos.

El ciclo de vida de un ejemplo que usa Socat típicamente consta de cuatro fases [12]:

- Una fase inicial, donde se inicializan las opciones de Socat.
- La fase de apertura, en la cual Socat abre la primera dirección y luego la segunda. En muchos casos, estas fases están normalmente bloqueados, especialmente en casos con direcciones complejas como socks, en donde se necesitan peticiones de conexión o diálogos de autenticación antes de empezar la siguiente fase.
- Una fase de transferencia, en la cual Socat vigila los dos flujos de datos los relacionas con las aplicaciones que los utilizan mediante `select()`. Socat observa si hay datos disponibles para enviar en un lado y si estos pueden ser escritos en el otro. En caso afirmativo, Socat lee esos datos

y realiza conversión de caracteres, en caso de que sea necesario, escribe los datos y se los entrega al "puerto"que se encarga del otro flujo.

– Por último, en caso de que uno de los flujos envíe un EOF comienza la cuarta fase, cierre. Esta fase, consiste en cerrar ambos canales y terminar la aplicación.

En el caso del proyecto, esta herramienta se usa para simular una conexión serie entre el simulador de ECUs OBDSim, que se ha explicado con anterioridad, y el programa encargado de resolver las peticiones OBD-II.

4. Desarrollo del prototipo

4.1. Introducción

En este capitulo, se detalla el desarrollo del prototipo. Se explicarán cada uno de los pasos que se han seguido para completar su realización.

Anteriormente, se ha comentado el objetivo del proyecto. En este, se implementa una aplicación sobre la Raspberry Pi que permite comunicarse, mediante el puerto serie, con un vehículo. Una vez establecida la comunicación, esta es capaz de procesar, interpretar y enviar lo parámetros obtenidos del vehículo a un gestor usando SNMP. Todo el proceso de comunicación SNMP, se realizará usando la seguridad que proporciona este protocolo en su versión tres.

Se empezará explicando la instalación de todo el software involucrado en el trabajo y se continuará con el desarrollo de una aplicación que permita la comunicación con el vehículo a través del puerto serie. Además, este programa se encargará de interpretar dicha información y enviársela a un agente maestro, para que este se la envíe a un sistema gestor mediante SNMPv3.

4.2. Descripción del prototipo

Durante este apartado se puede observar de forma gráfica la estructura del proyecto.

En la figura Figura 17 se muestra el esquema principal del proyecto. En este, se puede observar como la Raspberry Pi hace el papel de agente. Esta se conecta al vehículo mediante un puerto serie y al gestor a través de la red. El intercambio de información entre el coche y la Raspberry Pi se realiza mediante el protocolo OBD-II.

Al otro lado de la red, se encuentra el sistema gestor. Este, usa el protocolo SNMP para extraer los datos asociados al vehículo. Además, para dotar de seguridad a este entorno IoT se aplica la versión tres del protocolo SNMP. Gracias a esta, es posible el intercambio de información de manera confidencial y segura.

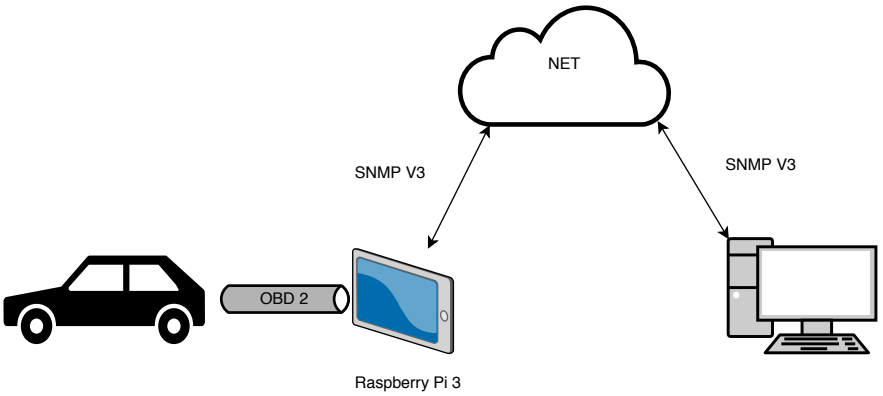


Figura 17: Esquema del proyecto

En la figura Figura 18 se puede ver con mas detalle el esquema de funcionamiento que va dentro de la Raspberry Pi. En este esquema, se observa el uso de un programa denominado main. Este es el encargado de comunicarse con el vehículo mediante OBD2, interpretar las respuestas de este y transformarlas en paquetes SNMP para enviárselas al agente maestro, el cual es el software SNMPD. El agente maestro es el encargado de cargar los valores de los objetos en la MIB. Por otro lado, el programa main se comporta como un agente extensible (Agente X).

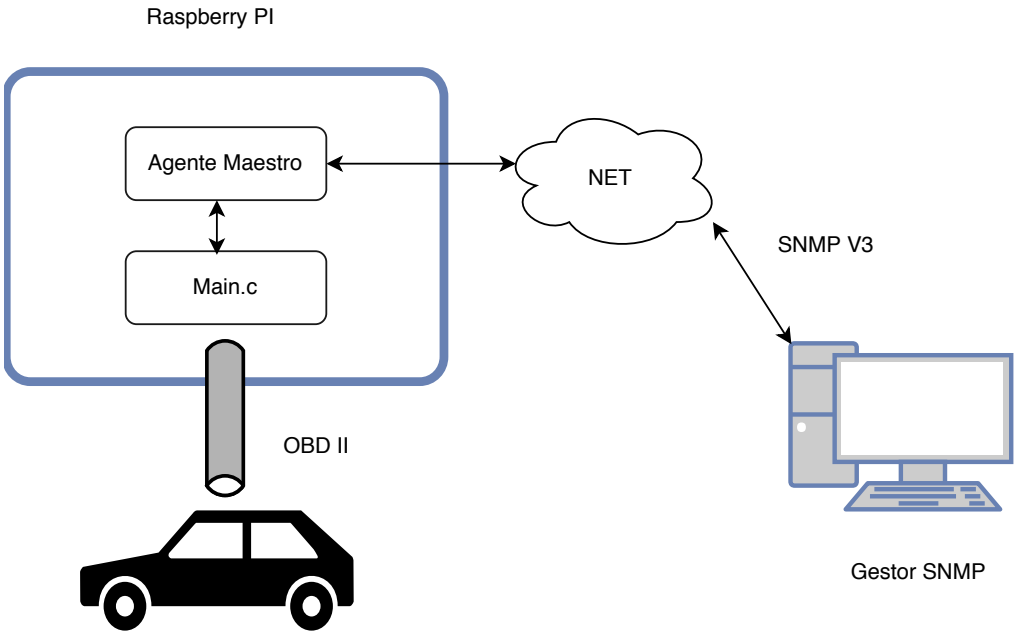


Figura 18: Esquema del proyecto en la Raspberry Pi

4.3. Preparación del entorno de desarrollo

A continuación, en este subapartado se muestra como se han instalado y configurado las distintas aplicaciones que han sido utilizadas en el desarrollo del proyecto.

Instalación de las aplicaciones relacionadas con SNMP

En primer lugar, se instala la aplicación net-snmp, la cual es usada para poder implementar el protocolo SNMP. Para ello, en primer lugar se procede a descargarla mediante el siguiente comando:

```
$ wget http://sourceforge.net/projects/net-snmp/files/net-snmp
/5.7.3/net-snmp-5.7.3.tar.gz
```

A continuación se procede a instalar las siguientes librerias Perl. Esto es debido a que el programa net-snmp depende de estas librerias para el correcto funcionamiento. La instalación de estas librerias se realizan mediante el siguiente comando

```
$ sudo apt-get install libperl-dev
```

Una vez instaladas, se procede a descomprimir el programa net-snmp que se ha descargado anteriormente:

```
$ tar -xvzf net-snmp-5.7.3.tar.gz
```

Después de que el programa esté descomprimido, se procede a configurar net-snmp a través de los siguientes comandos

```
$ cd net-snmp-5.7.3/
$ sudo ./configure --libdir=/usr/lib/.
```

Se utilizan las flags `--libdir=/usr/lib/.` para especificar la carpeta en donde se instalan las librerías propias de SNMP. Se ha especificado la carpeta de instalación debido a un conflicto entre la ruta de las librerías por defecto que pone net-snmp y la ruta que especifica snmpd.

Durante el proceso de configuración se dejó todo con la configuración por defecto que se indica.

Una vez configurado, se procede a la instalación. Para ello se escriben los siguientes comandos:

```
$ sudo make
$ sudo make install
```

Después de que el programa net-snmp ya esté instalado, se procede a instalar los ficheros MIBs a través de la siguiente orden:

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

A continuación, se procede a la instalación del agente maestro (snmpd) mediante el siguiente comando:

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

Finalmente, para activar el agente maestro se utiliza la siguiente instrucción:

```
$ sudo service snmpd start
```

Comprobación de la instalación del agente maestro SNMP

Una vez que se ha instalado tanto la herramienta NET-SNMP como el agente maestro snmpd, se comprueba que se han instalado de forma correcta y que no existe ningún conflicto con otros programas.

Para comprobar el funcionamiento de snmpd se introduce el siguiente comando en la terminal.

```
$ sudo service snmpd status
```

Una vez introducido, la respuesta que se obtiene es algo similar a lo que aparece en la Figura 19. En esta, se puede observar cómo el servicio se encuentra activo.

A continuación, se prueba el funcionamiento del agente realizando un petición SNMPv2 de tipo `get` y comunidad pública a la dirección de loopback. Esto se realiza mediante el siguiente comando.

```
$ sudo snmpget -v 2c -c public localhost SNMPv2-MIB::sysServices.0
```

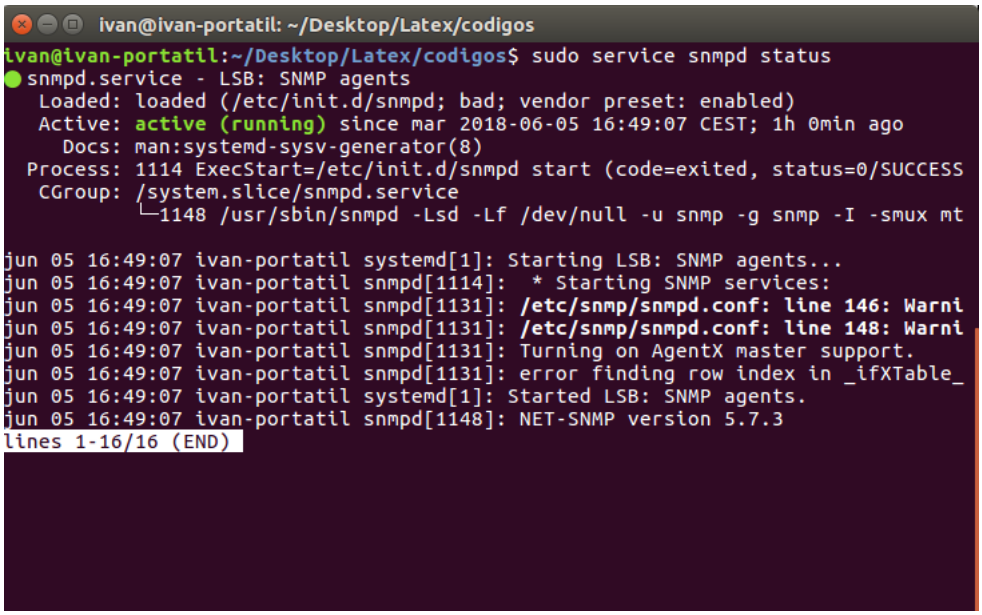


Figura 19: Respuesta al comando service status snmpd

El resultado que se obtiene es el valor del objeto sysService. Dicho nodo se encuentra localizado en la MIB-II. Este, indica el número de servicios primarios que puede ofrecer el ordenador.

El resultado del comando es el siguiente. Este, nos devuelve un número entero cuyo valor es setenta y dos.

```
SNMPv2-MIB::sysServices.0 = INTEGER: 72
```

Instalación del programa OBDSim

A continuación, se detalla el proceso de descarga e instalación del programa OBDSim, así como extensiones relacionadas a este. Este programa se utiliza para simular el comportamiento de un vehículo. Este, se encarga de generar respuestas OBD II a peticiones del cliente, en este caso la Raspberry Pi.

En primer lugar, se procede a descargar OBDSim. Para ello se usa el comando que aparece a continuación. la herramienta `wget` nos permite descargar el programa desde la url `icculus.org`.

```
$ wget http://icculus.org/obdgpslogger/
downloads/obdgpslogger-0.16.tar.gz
```

Una vez descargado, el siguiente paso es descomprimir la aplicación mediante el software `tar` y los argumentos `-zxvf`. Estos últimos, se utilizan para descomprimir los ficheros de extensión `tar.gz`. Una vez descomprimido, se crea la carpeta *build*, la cual se utiliza para realizar la instalación. Esto se realiza mediante la instrucción `mkdir`.

```
$ tar -zxvf obdgpslogger-0.16.tar.gz
$ cd obdgpslogger-0.16
$ mkdir build
$ cd build
```

A continuación mediante el comando `apt-get install` se instalan librerías adicionales para OBDSim. Una vez instaladas, se utiliza el software `cmake`. Esta herramienta permite generar Makefiles, los cuales son usados para compilar el programa. Para esto último, se utiliza el comando `make`.

```
$ sudo apt-get install libbluetooth-dev
libfltk1.1-dev libfltk1.1 fltk1.1-doc fluid
fftw3-dev libgps dev libftdi-dev
$ cmake ..
$ make obdsim
$ cd ../bin/
```

Para hacer funcionar el programa con la interfaz gráfica, se realiza mediante el siguiente comando:

```
$ sudo ./obdsim -g gui_fltk -t /dev/ttyS6
```

El argumento `-g` se utiliza para iniciar el programa con su interfaz gráfica. Mediante la interfaz gráfica se controlan diversos parámetros como las revoluciones del motor, temperatura del motor, posición del acelerador... Por otro lado, el argumento `-t` se utiliza para indicar el puerto serie sobre el que se envía la información. Esta interfaz se puede observar en la Figura 16.

4.4. Definición de la MIB

La base de la información de gestión del proyecto es la MIB, en nuestro caso esta se llama OBD2MIB. A continuación se explican algunos objetos gestionados en la MIB.

- *name*: Se trata de un objeto de tipo string. Este es utilizado para identificar de forma sencilla el vehículo al que se está interrogando.

- *enginerpm*: Se trata de un objeto escalar simple de solo lectura. En este objeto se carga el valor de las revoluciones por minuto del vehículo.
- *throttleposition*: Se trata de un objeto escalar simple de solo lectura. En este objeto se carga el valor del porcentaje de uso del acelerador, es decir, si el acelerador no esta pulsado vale uno, mientras, que si el acelerador está pulsado al máximo vale cero.
- *enginetemp*: Se trata de un objeto escalar simple de solo lectura. En este objeto se carga el valor de la temperatura del motor del vehículo.

Se han decidido implementar estos objetos en la MIB debido a que los valores de estos objetos se pueden controlar de manera sencilla mediante la interfaz gráfica de OBDSim, como se puede ver en la Figura 16, la cual se ha mostrado anteriormente.

En la Figura 20 se puede observar como se define uno de los objetos que se han comentado dentro de la MIB. En este caso es el objeto *enginetemp*. Este, cuelga de la rama mode1 de la MIB.

```
enginetemp OBJECT-TYPE
  SYNTAX          Integer32 (-40..215)
  MAX-ACCESS      read-only
  STATUS          current
  DESCRIPTION     " Engine coolant temperature "
::= { mode1-2 3 }
```

Figura 20: Objeto enginetemp dentro de la MIB

En esta figura se puede observar como los nodos se definen dentro de una estructura de tipo `OBJECT-TYPE`. Dentro de esta se define que el objeto es un entero de treinta y dos bits (`Integer32`), cuyo valor se encuentra en un rango entre -40 y 215. Se puede observar como el acceso a este es de solo lectura (`readonly`) y está activo (`current`). Además, en cada objeto descrito en la MIB se incluye una breve descripción de su utilidad.

El resto de los objetos tienen una estructura similar a la mostrada en el caso de la figura anterior.

Además de objetos enteros con valores dentro de un determinado rango, se define un nodo de tipo string. Como recordar asociaciones entre direcciones

IP o OIDs con automoviles no es algo sencillo, se ha decidido implementar este valor para simplificar las cosas. En el caso del proyecto solo se necesita controlar un único coche. Pero, si por el contrario, fuera necesario gestionar más de un autocar, el gestor puede observar de manera sencilla y visual cual de los automóviles es el que está monitorizando.

En la siguiente figura se puede observar como se define este objeto en la MIB:

```
name  OBJECT–TYPE
          SYNTAX              OCTET STRING
          MAX–ACCESS          read-only
          STATUS              current
          DESCRIPTION         "Object identification name."
          ::= { mode1-2 1 }
```

Figura 21: Objeto name dentro de la MIB

Como se puede observar en la figura anterior. El objeto es de tipo `OCTECT STRING`. Esto significa que el valor asociado a este es una cadena de caracteres. Al igual que en los nodos anteriores el acceso es de tipo `read-only` y el estado es `current`. Estos dos últimos parámetros son así, ya que al igual que ocurre con el resto de objetos no tiene sentido que los valores puedan ser modificados por el sistema gestor.

A la hora de la definición de la MIB, se ha decido que dentro de esta, cuelguen dos ramas. La primera de las ramas, la rama x, hace referencia a los dos primeros modos asociados a los PIDS. Esta es nombrada de esta forma, ya que a pesar de que en la MIB se definan dos ramas iguales, a la hora de mostrarlo se muestra así por simpleza. Por otro lado, de la segunda rama solo cuelga un objeto. Este, está asociado a un DTC (Data Trouble Code), el cual es algo que está definido en el estándar OBD-II. En el caso de que el coche tenga alguna notificación de error para el gestor, se puede utilizar este objeto para enviar los elementos que han causado el error.

La escritura de los distintos nodos que conforman la MIB se realizan en un fichero de texto con extension `.txt`

Instalación de la MIB

A continuación, se detallan los procedimientos seguidos para poder utilizar la MIB que se ha creado (OBD2MIB), sobre NET-SNMP y snmpd.

Para que el agente maestro (snmpd) sea capaz de operar con la MIB que se ha creado, es necesario conocer la carpeta donde obtiene la información. A través del comando `net-snmp-config` y los atributos que se observan a continuación se obtienen la ruta donde debe ser almacenada nuestra MIB.

```
$ net-snmp-config --default-mibdirs
```

Una vez introducido la orden, se observa que la MIB debe de ser almacenada en la carpeta `/usr/local/share/snmp/mibs` o en la carpeta `/home/ivan/.snmp/mibs`. En el caso de este proyecto se decide utilizar la primera opción.

A continuación, es necesario modificar el fichero de configuración de snmpd (*snmpd.conf*). Para ellos se usa el editor de textos gedit.

```
$ sudo gedit /etc/snmp/snmpd.conf
```

En el fichero de configuración se añade la siguiente linea en el apartado *Access Control*.

```
view   systemonly  included   .1.3.6.1.4.1.8072.2.333
```

Mediante esta linea se indica a snmpd que tenga en cuenta todo lo que cuelge del OID `.1.3.6.1.4.1.8072.2.333`, el cual es el identficador de la MIB que se ha creado. Esta, cuelga dentro de la rama `netSnmpExamples`, pertenciente a la MIB `NET-SNMP-EXAMPLES-MIB`.

A ravés de la herramienta `snmptranslate` se comprueba si la MIB esta escrita de forma correcta y si esta bien definida dentro de snmpd. En el Código 12, que aparece en el anexo, se muestra el uso de esta herramienta. Los argumentos `-Tp` y `-IR` se utilizan para observar el resultado en forma de árbol.

4.5. Implementación del agente X

Una vez implementada la MIB, es necesario el desarrollo de un agente X que permita la interacción con los objetos de la MIB. Para ello se desarrolla un pequeño programa, escrito en C, que pueda rellenar con información los objetos de la MIB. Además, este programa también se encarga de comunicarse con el vehículo para obtener los valores adecuados.

Dentro del programa se pueden distinguir tres partes claramente diferenciadas.

La primera parte consiste en implementar los nodos de la MIB a través de un demonio. Este, es el responsable de inicializar los objetos de la MIB en el handler del agente maestro snmpd, para que este pueda acceder a ellos, y de rellenar esos objetos con la información adecuada. La estructura del demonio se realiza mediante la aplicación mib2c, la cual está incluida dentro del paquete NET-SNMP. Mediante este programa se genera un esqueleto genérico en código C sobre el que se realizan modificaciones para ajustarlo al caso del proyecto.

La segunda parte consiste en inicializar el puerto serie sobre el que se desarrolla la comunicación entre el vehículo y el programa. Para la realización de esta parte, se diseña una función C que permita configurar las opciones del puerto para que la comunicación entre ambas partes sea posible.

Por último, la tercera parte en la que se puede dividir el programa, consiste en la comunicación entre ordenador/Raspberry Pi y el vehículo. Para la realización de esta parte, se implementan una serie de métodos que permitan enviar información a través del puerto serie e interpretar las respuestas del vehículo.

Todas estas partes se pueden observar en la Figura 22.

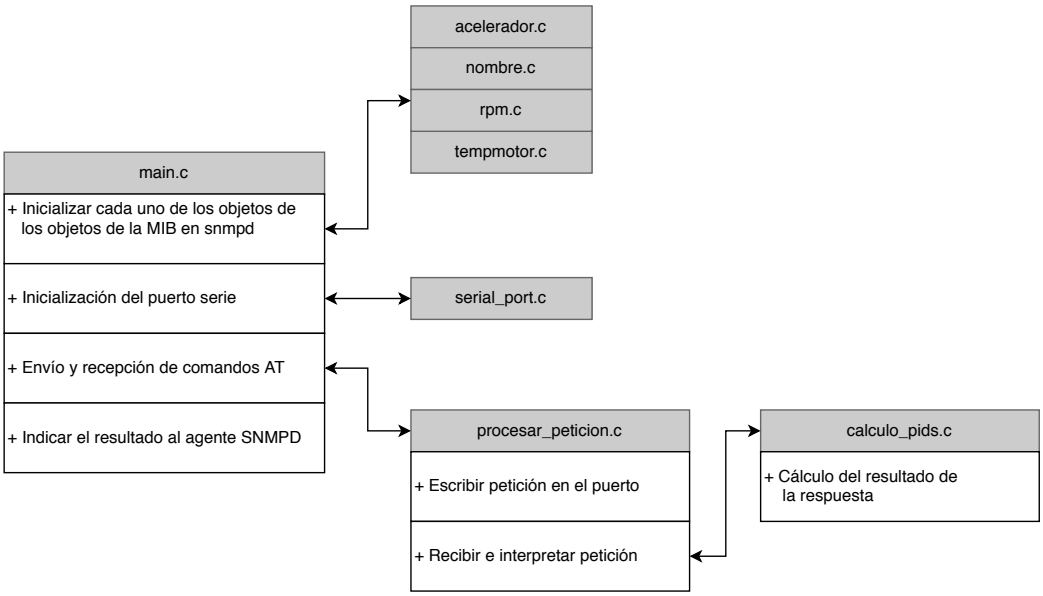


Figura 22: Esquema del programa

Diseño de los objetos

A la hora de diseñar los objetos que forman parte de la MIB, se utiliza la herramienta mib2c. Mediante esta herramienta, es posible la creación de un esqueleto de código C, el cual se modifica en consonancia con lo que se necesite. Para la generación de dicho esqueleto se utiliza el siguiente comando.

```
$ env MIBS="+nombremib" mib2c -c mib2c.int_watch.conf nodomib
```

Donde `nombremib` es el nombre de la MIB a la que pertenece el objeto, en el caso del proyecto es OBD2MIB. El parámetro `-c mib2c.int_watch.conf` indica que el objeto está atado a un número escalar que va variando su valor y `nodomib` es uno de los objetos que se encuentran descritos dentro de la MIB.

Una vez que se ha introducido el comando, mib2c genera un esqueleto de código C. En el código Objeto enginerpm se muestra un ejemplo del resultado generado por mib2c. En este caso, el código pertenece al objeto enginerpm (rpm.c), el cual es el encargado de recoger las revoluciones por minuto del vehículo.

```
1 #include <net-snmp/net-snmp-config.h>
2 #include <net-snmp/net-snmp-includes.h>
3 #include <net-snmp/agent/net-snmp-agent-includes.h>
4 #include "rpm.h"
5
6 long     rpm = 0;
7 void
8 init_rpm(void)
9 {
10    netsnmp_handler_registration *reg;
11
12    const oid rpm_oid[] = { 1,3,6,1,4,1,8072,2,333,1,1,10 };
13    static netsnmp_watcher_info rpm_winfo;
14
15     DEBUGMSGTL(("rpm", "Initializing the rpm module\n"));
16
17     DEBUGMSGTL(("rpm",
18                 "Initializing rpm scalar integer.  Default
                       value = %d\n",
19                 rpm));
20     reg = netsnmp_create_handler_registration(
21               "rpm", NULL,
22                rpm_oid, OID_LENGTH(rpm_oid),
23                HANDLER_CAN_RWRITE);
24
25     netsnmp_init_watcher_info(&rpm_winfo, &rpm, sizeof(long),
26 ASN_INTEGER, WATCHER_FIXED_SIZE);
27
28     if (netsnmp_register_watched_scalar( reg, &rpm_winfo ) < 0
           ) {
29         snmp_log( LOG_ERR, "Failed to register watched rpm" );
30     }
31
32
33     DEBUGMSGTL(("rpm",
34                 "Done initalizing rpm module\n"));
35 }
```

Objeto enginerpm

En el código, se puede observar como las tres primeras lineas incluyen las librerías que se utilizan para crear un handler en el agente maestro. Estas bibliotecas, que son propias de NET-SNMP, permiten el uso de funciones específicas para la implementación de SNMP. Gracias a estas, es posible inicializar un nodo de la MIB en al agente maestro y que este sea capaz de

entender y procesar las peticiones dirigidas al objeto.

A la hora de cargar un valor en el objeto se hace uso de una variable externa, en este caso es de tipo long y se denomina `rpm`. Debido a que esta es cargada en otra parte del programa, se hace uso de una librería personal para pasar su valor de un fichero a otro. Esto se especifica en la siguiente parte del código. En este caso, se definió que la variable, por defecto, tuviese valor cero.

```
4 #include "rpm.h"
5
6 long     rpm = 0;
```

La parte principal del código es la función `init_rpm`. A través de esta función, se crea el handler del objeto en el agente maestro. Este, es el encargado procesar las peticiones que vayan al OID definido en la linea doce, en este ejemplo `1.3.6.1.4.1.8072.2.333.1.1.10`. Esto se realiza mediante la siguiente parte.

```
20     reg = netsnmp_create_handler_registration(
21               "rpm", NULL,
22                rpm_oid, OID_LENGTH(rpm_oid),
23                HANDLER_CAN_RWRITE);
24
25     netsnmp_init_watcher_info(&rpm_winfo, &rpm, sizeof(long),
26 ASN_INTEGER, WATCHER_FIXED_SIZE);
```

La función `netsnmp_create_handler_registration` se encarga de crear el handler tal y como se ha comentado previamente.

Por otro lado, la función `netsnmp_init_watcher_info` se encarga de rellenar el handler creado previamente, con los parámetros correspondientes, e indicar al handler donde debe buscar el valor del objeto. En este caso, tiene que buscar el valor de la variable rpm definida en la linea seis.

Como se ha indicado previamente, el valor de la variable rpm no se carga en esta parte del programa, sino que se obtiene en el *main.c*. Para pasar la variable a otro fichero se hace uso de una librería personal.

En el Código 2 se puede observar como es este archivo. En este, se indica que la función `init_rpm` y la variable `rpm` son usadas en otra parte del códi-

go donde requieren ser utilizadas. Para llamarlas en el otro fichero solo es necesario incluir la siguiente linea al comienzo.

```
#include "rpm.h"
```

Una vez incluida, ya es posible utilizar las funciones y variables que se indican en la librería.

```
1 #ifndef RPM_H
2 #define RPM_H
3
4 void init_rpm(void);
5 long rpm;
6
7 #endif
```

Código 2: Librería rpm.h

Creación del programa principal: main.c

Una vez que nuestros objetos han sido creados, es necesario la realización de un agente extensible que se encargue de inicializar los nodos en el agente maestro y de cargar su valor.

Dicho demonio se va encargar de arrancar los objetos para que el agente maestro sea capaz de acceder a ellos, obtener su valor y cargar el resultado en el maestro. Para obtener este valor, el programa se comunica, usando comandos AT, con el vehículo a través de un puerto serie.

El programa principal es capaz de interactuar con distintas funciones para obtener los resultados esperados. Esto, ya se indicó previamente en la Figura 22.

A continuación se definen las fases que se han seguido a la hora de la realización del código principal.

En primer lugar, se deben incluir las diferentes librerías que son usadas en el programa. Se incluyen tanto librerías propias de NET-SNMP como librerías personales. Estás ultimas, que son similares a las mostradas en la figura Librería rpm.h, deben ser incluidas para pasar parámetros entre funciones.

Una vez declaradas las librerías que se utilizan en el programa, el siguiente

paso es declarar las diferentes variables globales que se usan. Estas, se encuentran declaradas en las librerías y son la encargadas de cargar el valor en el agente maestro. En el caso del ejemplo, se declaran las siguientes variables:

```
#define num_objetos 3

extern long rpm;
extern long acelerador;
extern long tempmotor;
```

Una parte importante antes de iniciar el demonio, es abrir el puerto serie que se utiliza para establecer el intercambio de datos. Esto se realiza mediante la función `open_port()`. Este método se encuentra definido en el fichero `serial_port.c` y es utilizado en `main.c` gracias a la librería `serial_port.h`. Gracias a esta función, es posible modificar los atributos necesarios del puerto serie para la conexión. Por ejemplo flags, regimen binario, nombre del puerto, etc.

A continuación, dentro de la función principal del programa, es necesario arrancar nuestro demonio. Para poder realizar esto, es necesario especificar las características de nuestro subagente e inicializar los diferentes objetos que es capaz de soportar. Esto se puede ver en la porción de código Fragmento de inicialización del demonio.

En este fragmento, es posible observar una serie de funciones propias de las librerías de NET-SNMP para arrancar el demonio.

La primera función que aparece, `netsnmp_ds_set_boolean`, permite indicar si nuestro agente es un agente maestro o un agente X. En el caso de este ejemplo, se decidió que nuestro programa trabaje como un subagente, el agente maestro es snmpd. Esto, se hizo poniendo al valor de la variable `agentx_subagent` a `1`.

Las dos siguiente funciones `SOCK_STARTUP` y `init_agent()` permiten arrancar los sockets TCP/IP de nuestra aplicación e inicializa las librerías del agente respectivamente,

```
62      if (agentx_subagent) {
63          netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID,
                NETSNMP_DS_AGENT_ROLE, 1);
64      }
65
66      if (background && netsnmp_daemonize(1, !syslog))
67          exit(1);
68
69      SOCK_STARTUP;
70      init_agent("main");
71
72      /* Inicializo mis agentes para que sean capaces de tomar
            valores*/
73
74      init_rpm();
75      init_acelerador();
76      init_nombre();
77      init_tempmotor();
78
79      if (!agentx_subagent) {
80          init_vacm_vars();
81          init_usmUser();
82      }
83
84      init_snmp("main");
85
86      if (!agentx_subagent)
87          init_master_agent();
```

Código 3: Fragmento de inicialización del demonio

La siguientes funciones que aparecen permiten inicializar los objetos de la MIB de los que se necesita extraer la información. Añadiendo las siguientes lineas es posible utilizar un solo subagente para controlar estos objetos.

```
init_rpm();
init_acelerador();
init_nombre();
init_tempmotor();
```

La dos siguientes funciones que aparecen,`init_vacm_vars()` y `init_usmUser()`, permiten activar las opciones de VACM y USM, usadas para controlar el acceso a los objetos. Esta opción en nuestro ejemplo no se activa debido a que se trabaja como subagente, por lo tanto, se usan las opciones por defecto que

incorpora snmpd.

Por último, las funciones `init_snmp("main")` y `init_master_agent()` son utilizadas para aplicar la configuración snmp y para iniciar el agente maestro. Esta última abre el puerto 161, el cual el puerto usado por defecto en SNMP.

Una vez el agente X ha sido iniciado, el siguiente paso es hacer que obtenga el valor de los objetos de la MIB. En el Código 4 se puede observar las tareas que realiza el agente mientras está activo para obtener la información de los objetos.

En el fragmento se puede observar tres partes claramente diferenciadas.

La primera parte consiste en cargar las peticiones que se envían en el puerto serie. A la hora de intercambiar el comando y las respuestas entre los diferentes métodos se utiliza la siguiente estructura, la cual se encuentra definida en la librería personal `def.h`.

```
typedef struct {
char peticion[32];
int pet_id;
char respuesta_char[255];
int respuesta_int;
char tipo; // '0' = entero,   '1' = string
} nueva_pdu;
```

Los campos de la estructura tienen los siguientes significados.

- `char peticion[32]`: Se trata de una cadena de caracteres en donde se carga el comando que va se envía por el puerto serie.
- `pet_id`: Este campo se trata de un entero que tiene como valor la id de la petición. Mediante este id es posible la asociación entre una petición y una respuesta.
- `char respuesta_char[255]`: Este campo es una cadena de caracteres que contiene la respuesta sin ser procesada, es decir, lo que llega directamente desde el vehículo. En caso de que la respuesta se trate de una cadena de caracteres también se puede usar este campo.
- `int respuesta_int`: Este campo representa el valor de la respuesta ya procesada.

– `char tipo`: Este campo se trata de un booleano que indica si la respuesta se trata de una cadena de caracteres o de un entero.

```
98    while(keep_running) {
99
100       k = 0;
101       // Obtiene la peticion (Se puede simplificar con switch)
102       while(1){
103           pdu.peticion[k] = peticiones[i];
104           if (peticiones[i] == '\n'){
105               pdu.tipo = tipo[j];
106               pdu.pet_id = pet_id[j];
107               i++;
108               j++;
109               break;
110           }
111           k++;
112           i++;
113       }
114
115       // Aqui obtengo cada una de las pdus individuales
116
117
118       procesar_peticion(&pdu, fd);
119       cargar_int[j-1] = pdu.respuesta_int;
120
121       memset(pdu.peticion, 0, sizeof(pdu.peticion));
122       memset(pdu.respuesta_char, 0,
              sizeof(pdu.respuesta_char));
123
124
125       /***        Aqui relleno los objetos de la MIB
              ***/
126
127       rpm = cargar_int[0];
128       acelerador = cargar_int[1];
129       tempmotor = cargar_int[2];
130       strcpy(nombre,"Ford Mondeo id=1");
131
132       if (j == num_objetos) {
133           j = 0;
134           i = 0;
135           agent_check_and_process(0);
136           while (agent_check_and_process(1) == 0);    ///
                  Puesto para que el demonio no estë enviando
                  peticiones constantemente
137       }
```

Código 4: Fragmento tareas del demonio

La segunda parte consiste en enviar la estructura previamente comentada con el valor de la petición al puerto serie y obtener una respuesta. Esto se realiza mediante la función `procesar_peticion`.

Por último, la tercera parte consiste en cargar los valores de los objetos en el agente. Esto último se realiza mediante la función propia de las librerías de SNMP `agent_check_and_process(1)`. El que el valor de la función sea `1` es para indicar que el agente debe de estar parado hasta que reciba una petición.

Intercambio de información y procesado: `procesar_peticion.c`

Una vez que se ha observado en el punto anterior que funciones realiza el subagente mientras que este está activo, es necesario entender como este obtiene la información del vehículo. A continuación se detalla como el demonio es capaz de extraer los valores necesarios.

En el archivo `main.c` aparece la función `procesar_peticion(&pdu, fd)`. Esta función tiene como parametros de entrada los siguientes:

- `pdu`: La estructura utilizada a largo del programa para pasar parámetros entre los diferentes ficheros.
- `fd`: En esta variable se define el identificador de puerto serie sobre el que se realiza la comunicación.

Dicha función, la cual se ejecuta en cada iteración y que permite obtener los resultados de los objetos, está definida en el archivo `procesar_peticion.c`.

En este fichero se pueden observar tres partes, escritura de la petición en el puerto serie, lectura de las respuestas asociadas a la petición y calculo del resultado.

La primera parte se realiza mediante la siguiente porción de código:

```
82      wlen = write(fd, pdu->peticion, strlen(pdu->peticion));
83      if (wlen != strlen(pdu->peticion)) {
84          printf("Error from write: %d, %d\n", wlen, errno);
85      }
86      tcdrain(fd);
```

Código 5: Fragmento escribir procesar_peticion.c

La escritura de la petición se realiza mediante la función `write()`. Esta función tiene como argumentos de entrada el identificador del puerto serie (`fd`), la petición que se desea enviar (`pdu->peticion`) y lo que ocupa dicha petición (`strlen(pdu->peticion)`).

En el fragmento anterior, además de la escritura de la petición, se puede observar el uso de la función `tcdrain(fd)`. Este método se usa para bloquear el uso del puerto hasta que la petición es enviada.

Una vez que la petición ha sido enviada, el siguiente paso es esperar una respuesta. Para procesar la respuesta, lo que se hace es leer caracter a caracter hasta que se detecte el final de la respuesta.

Para poder explicar como se detecta la respuesta del vehículo, es necesario entender como se intercambia información OBD-II. A continuación en la Figura 23 [1] se muestra una captura para poder averiguar las revoluciones del motor.

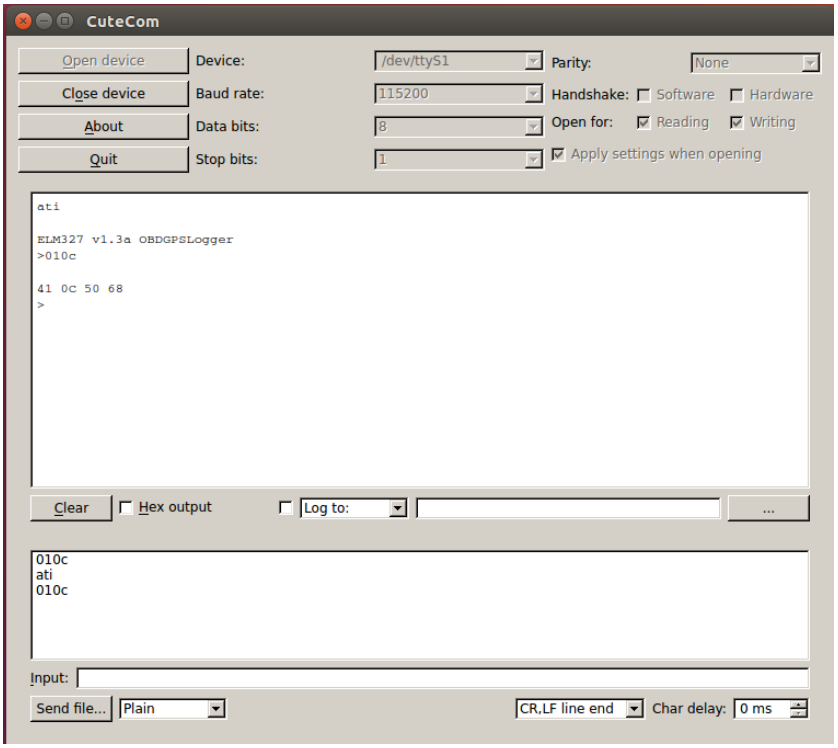


Figura 23: Ejemplo de intercambio de información

[1] El intercambio de información para esta demostración se realizó mediante el sofware Cutecom. Este software nos permite enviar comandos a través de un puerto serie.

En esta imagen se puede observar que para averiguar la información se envía la petición `010c` y un salto del linea. Esto significa que la petición pertenece al modo 1 y pid 12. Como respuesta lo que se obtiene es la siguiente respuesta en ascii:

```
41 0c 50 68
>
```

La respuesta que se obtiene está codificada en ASCII, pero el valor es el hexadecimal de los bytes respuesta. En este caso, la respuesta de es de dos bytes: `0x410c5068`. A partir de estos dos bytes y una operación específica se obtiene el valor en un lenguaje "entendible".

Una vez que se ha entendido como se realiza el intercambio de información, se muestra como se realiza esto dentro del fichero. En el Fragmento leer procesar_peticion.c

```
91      while(1) {
92          int rdlen;
93
94          nbytes = read(fd, &leer, 1);
95          if (leer == '>' || i > 32) break;
96          buf[i] = leer;
97          i++;
98      }
```

Código 6: Fragmento leer procesar_peticion.c

Para poder leer el puerto serie se utiliza la función `read()`. Dicho método tiene como argumentos de entrada el identifcador del puerto serie que se usa (`fd`), lo que se lee (`leer`) y el número de caracteres que tiene que leer (`1`).

A la hora de leer se ha decidido leer caracter a caracter ya que no todas las respuestas tienen el mismo tamaño. La lectura termina cuando se lee el caracter `'>'`.

Al final del ciclo lo que se obtiene es la petición y respuesta cargado en el array `buf`. Finalmente, el caracter final del vector se carga con un valor nulo para conocer el final del array.

La última parte del archivo `procesar_peticion.c` consiste en calcular el valor entendible de la respuesta. En el Fragmento calculo respuesta (procesar_peticion.c) se observa como se realiza esto.

```
102     if (tipo == '0'){
103         response = select_pid(buf,offset, tipo, pdu);
104         calculo_pids(response, pdu);
105     }
```

Código 7: Fragmento calculo respuesta (procesar_peticion.c)

En primer lugar, del fragmento anterior, cabe destacar que solo se realiza el cálculo del valor para los peticiones que requieran valores enteros, debido a que si la respuesta es un string aparece tal cual.

A continuación lo que se realiza es procesar la respuesta para convertirla a los bytes que se necesitan para realizar el cálculo del valor. Esto se realiza mediante el método `select_pid()`.

En el Fragmento procesar respuesta (procesar_peticion.c) se puede observar como se obtiene el formato deseado.

En dicho fragmento, se puede observar que el procesado de la respuesta se realiza en tres pasos.

El primer paso consiste en quitar los espacios que aparecen en la cadena de caracteres de respuesta. Esto se realiza copiando el string respuesta en una variable auxiliar sin copiar los espacios. De esta forma si originalmente se tenía algo similar a `41 0c 50 68` quedaría de la siguiente forma `410c5068`.

El siguiente paso consiste en convertir la cadena ASCII a hexadecimal, de tal forma que si en el ejemplo anterior se tenía `410c5068` se convertiría en `0x410c5068`. Si antes la cadena ocupaba ocho bytes ahora pasa a ocupar 4 bytes. Esta conversión se realiza restando a los números el `0` en ASCII (`0x30`) y restando el valor `0x07` en caso de que sea una letra.

Por ultimo, se carga cada byte de la respuesta en un entero. Si antes se tenía como string `0x410c5068`, en este paso se carga en el entero `response[0]` el valor `0x50`, en `response[1]` se guarda `0x68` y así con el resto (en caso de que hubiese mas). Con esto se obtienen los cuatro bytes respuestas[2] aunque, en este caso, la respuesta solo contiene dos bytes. Por lo tanto, los bytes C y D quedan vacíos.

[2]Recordar que en OBD las respuestas se codifican en cuatro bytes (A, B, C, D) tal y como se pudo ver en la Figura 14.

```
27          // Quito los espacios a la cadena de texto
28
29          k = 0;
30          for (i = (offset+2);i < strlen(pid);i++){
31              if (pid[i] == '\x0d') break; // Detecta el LF (0x0D)
32              if (pid[i] != ' '){
33                  resultado[k] = pid[i];
34                  k++;
35              }
36
37          }
38
39          // Convertir ascii a decimal
40
41          for (i = 0;i < sizeof(resultado);i++){
42              if (resultado[i] <= '\x39') resultado[i] -= '0';
43              else {
44                  resultado[i] -= '7';
45              }
46          }
47
48          // A,B,C,D
49
50
51          response[0] = resultado[4]*16 +resultado[5];
52          response[1] = resultado[6]*16 +resultado[7];
53          response[2] = resultado[8]*16 +resultado[9];
54          response[3] = resultado[10]*16 +resultado[11];
```

Código 8: Fragmento procesar respuesta (procesar_peticion.c)

Una vez que ya se tiene el formato que nos interesa, queda obtener el valor real de la respuesta en un formato entendible. Esto se realiza mediante la función `calculo_pids()`. Esta, se define en el fichero `calculo_pids.c`.

Cálculo del PID: `calculo_pids.c`

En este archivo se incluyen las operaciones que se deben realizar a los bytes respuestas que se encuentran en la cadena `response[]` para obtener el valor entero que realmente marca el vehículo.

A continuación, en el siguiente fragmento de código, se muestra las operaciones realizadas para obtener el valor de las revoluciones por minuto del motor.

```
18      switch(pdu->pet_id){
19          case 12 :
20              resto = (256*response[0]+response[1]) % 4;
21
22              pdu->respuesta_int = (256*response[0]+response[1])
                    / 4;
23              if (resto != 0) pdu->respuesta_int += 1;
```

Código 9: Fragmento procesar respuesta (procesar_peticion.c)

En este caso la operación que se indica se usa para para obtener las revoluciones por minutos del motor. Para ello se hace el siguiente cálculo:

$$\frac{256A + B}{4}$$

En la porción de código anterior se puede observar como cada id tiene una operación asociada. En caso de que se desean realizar más peticiones lo único que hay que realizar es añadir mas casos y sus operaciones pertinentes, de esta forma la escalabilidad es sencilla.

La asociación entre un cálculo y una petición se realiza a través del campo `pet_id` de la estructura pdu. El resultado de dicha operación se carga en el entero `respuesta_int` de la estructura anterior. Este resultado es el que se carga en el agente.

Además de devolver el valor asociado a una petición, también es posible indicar si un determinado valor o no debe generar una trap al gestor. Esta se genera en el siguiente fragmento de código.

```
34              strcpy(comando, "sudo snmptrap -v 2c -c public
                    192.168.1.47 0 OBD2MIB::trapEngineTemp
                    OBD2MIB::engineTemp.0 i -1");
35
36              if (pdu->respuesta_int > 90) {
37                  system(comando);
38              }
```

Código 10: Fragmento generar trap (procesar_peticion.c)

En el fragmento anterior se puede observar como el agente genera una notificación a la dirección ip del gestor. Esto se realiza a través de un comando propio de NET-SNMP. En este caso se genera una trap del objeto `engineTemp`. Además es posible especificar el mensaje que el agente puede envíar como trap. En el ejemplo se ha definido un entero cuyo valor es `-1`.

Compilación del subagente

Una vez que se han escrito todos los códigos relativos al programa, es necesario compilarlos. A la hora de compilarlos se ha decidido utilizar un script de bash y un Makefile para organizar el proceso de compilación.

En la siguiente figura se puede observar como se realiza el proceso de compilación de los ficheros C. Cada código es compilado en el script y este último, es el que se encarga de de llamar al Makefile para generar el programa.

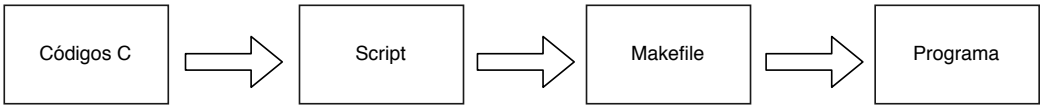


Figura 24: Proceso de compilación

El objetivo principal del script es automatizar el proceso de compilación. Este, se encarga de generar los códigos objetos, que tienen extensión .o, de cada uno de los archivos .c. Además, este se ocupa de iniciar el Makefile.

En el Código 11 se puede observar el script bash. En este, se puede observar que el las objetos maquina se compilan a través de la herramienta `gcc`.

```
1 #!/bin/bash
2
3 gcc -c ./leerobd2/serial_port.c -o ./leerobd2/serial_port.o
4 gcc -c ./leerobd2/procesar_peticion.c -o
    ./leerobd2/procesar_peticion.o
5 gcc -c ./leerobd2/calculo_pids.c -o ./leerobd2/calculo_pids.o
6
7 gcc -c ./rpm/rpm.c -o ./rpm/rpm.o
8
9 gcc -c ./acelerador/acelerador.c -o ./acelerador/acelerador.o
10
11 gcc -c ./tempmotor/tempmotor.c -o ./tempmotor/tempmotor.o
12
13 gcc -c main.c
14
15 sudo make main
```

Código 11: Script com_libraries.sh

Una vez que se han generado los códigos máquina de cada objeto, el siguiente paso es iniciar el Makefile. Gracias a este, es posible organizar la

compilación de manera sencilla. En este, el proceso de compilación del programa se realiza en la siguiente linea:

```
$(CC) -DDISPLAY_STRING -o main $(OBJS1) -lpthread $(BUILDAGENTLIBS)
```

Donde

- La variable `$(CC)` hace referencia a la herramienta gcc.
- El argumento `-DDISPLAY_STRING` hace que los datos recibidos a través del puerto serie sean tratados como cadenas de caracteres ASCII.
- La variable `$(OBJS1)` se utiliza para referenciar a todos los códigos objetos generados por el script previamente comentado.
- El argumento `-lpthread` permite enlazar librerías adicionales al proceso de compilación.
- La variable `$(BUILDAGENTLIBS)` se utiliza para hacer referencia a las librerías específicas del agente.

A través de este comando se compila el subagente (`main`) usando las librerías propias de NET-SNMP.

4.6. Implementación de SNMPv3

Uno de los aspectos de este proyecto es que la comunicación entre el agente y el sistema gestor sea segura. Esto quiere decir que, en el caso de que haya un usuario malicioso escuchando los paquetes intercambiados, este no sea capaz de obtener la información de dichos mensajes. Además, se debe asegurar que dichos datos vayan al unicamente al sistema gestor y no a otro usuario que pueda hacerse pasar por este. Para realizar esto, se debe implementar un mecanismo de autenticación y autorización.

Para realizar esta transferencia de información segura se implementa la tercera versión del protocolo SNMP. Este, incorpora mecanismos de cifrado que proporciona privacidad a los mensajes y añade métodos de autenticación basados en un secreto compartido.

La implementación de esta versión del protocolo se realiza de la siguiente manera.

En primer lugar, se desactiva el agente maestro (snmpd) mediante la siguiente orden:

```
$ sudo service snmpd stop
```

Una vez que el agente maestro ha sido detenido, se procede a editar el fichero de configuración `/var/lib/snmp/snmpd.conf`. En este, al final del documento, se introduce la siguiente linea:

```
createUser USERNAME SHA "AUTH-PASS" AES "ENC-PASS"
```

El comando `createUser` crea un usuario específico de SNMPv3 para poder realizar la autenticación y la encriptación de los mensajes SNMPv3.

En la linea que aparece arriba, se debe remplazar "`AUTH-PASS`" y "`ENC-PASS`" con las claves secretas de autenticación y cifrado respectivamente. En esta situación se puede observar como la autenticación se realiza mediante SHA-1 y el cifrado a través AES.

En el caso de que solo se quiera guardar una única clave para el cifrado y la autenticación, con no escribir el secreto de cifrado basta. Si ocurre esto, snmpd toma como contraseña de cifrado la de autenticación.

Una vez que la orden `createUser` ha sido escrita, el siguiente paso es modificar el fichero de configuración `etc/snmp/snmpd.conf` con la siguiente linea:

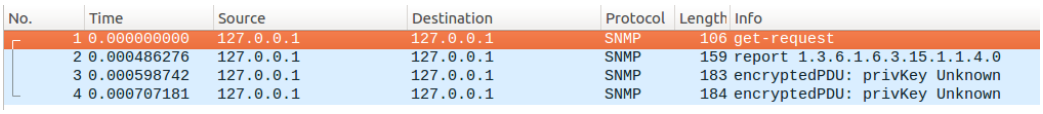
```
rouser USERNAME priv
```

Mediante el comando `rouser` se asegura que el usuario que se ha creado previamente solo tenga privilegios de solo lectura. Además mediante esta linea se indica que el usuario solo puede acceder a través del modo authPriv. Este, es un modo de comunicación que activa la autenticación y la privacidad.

Finalmente el último paso es volver a encender el agente maestro, el cual previamente fue desactivado. Esto se realiza mediante el siguiente comando:

```
$ sudo service snmpd start
```

En la Figura 25 se puede observar una captura Wireshark de un get-request entre un agente y un gestor. Como se puede observar, las PDUs petición-respuesta que llevan los datos están encriptadas. Esta captura pertenece al objeto sysServices de la MIB-2.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	SNMP	106	get-request
2	0.000486276	127.0.0.1	127.0.0.1	SNMP	159	report 1.3.6.1.6.3.15.1.1.4.0
3	0.000598742	127.0.0.1	127.0.0.1	SNMP	183	encryptedPDU: privKey Unknown
4	0.000707181	127.0.0.1	127.0.0.1	SNMP	184	encryptedPDU: privKey Unknown

Figura 25: Ejemplo de SNMPv3

Para la realización de dicha captura, se introduce el siguiente comando en el gestor:

```
$ snmpget -v 3 -l authPriv -a sha -A telematica -x AES
 -X telematica -u obd2 localhost SNMPv2-MIB::sysServices.0
```

En este comando se usa como usuario el valor `obd2` y como contraseña de autenticación y cifrado `telematica`.

La transferencia de información va cifrada, mientras que el gestor obtiene el valor asociado a la petición. En este caso, el gestor obtiene:

```
SNMPv2-MIB::sysServices.0 = INTEGER: 72
```

4.7. Aplicación en el entorno de simulación

Una vez que el agente ha sido compilado correctamente, el siguiente paso consiste en comprobar que el programa funciona de la forma esperada.

En la realidad el agente se comunicaría a través de un puerto serie al dispositivo OBD-II. Sin embargo, para mayor comodidad a la hora de realizar la simulación del prototipo se utiliza una comunicación serie simulada[3] entre el subagente y el programa OBDSim. Este último, responde a los comandos que envía el demonio como si se tratase de un dispositivo OBD-II. Esto, se puede observar en la Figura 26.

En primer lugar, antes de poder realizar nada, es necesario retocar de forma apropiada el fichero de configuración de snmpd.

Para que el agente maestro pueda escuchar cualquier petición que llegue al puerto 161 tanto de IPv6 como IPv4 se descomenta la siguiente linea en el fichero de configuración `etc/snmp/snmpd.conf`.

```
agentAddress udp:161,udp6:[::1]:161
```

Dado que, el intercambio de información entre agente y gestor debe ser seguro se implementa el siguiente usuario en el fichero de configuración `/var/lib/snmp/snmpd.conf`.

```
createUser obd2 SHA telematica AES telematica
```

[3]Para ello se utiliza el software Socat

En este caso se ha decido que el nombre de usuario sea `obd2` y las contraseñas de autenticación y encriptación sean `telematica`.

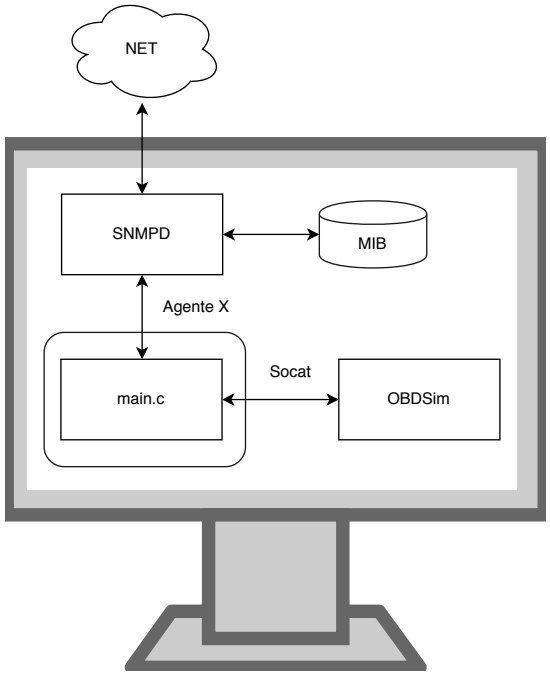


Figura 26: Esquema de simulación

Además, para que el agente maestro requiera autenticación y cifrado, se debe indicar en el documento `etc/snmp/snmpd.conf` que el modo de comunicación sea `authPriv`. Esto se indica añadiendo las siguientes lineas:

```
rocommunity secret  "@IP del gestor"
rouser   obd2 priv
```

En la parte "`@IP del gestor`"se escribe la dirección IP del gestor que se comunica con el agente.

Una vez que el agente maestro está bien configurado, se procede a realizar una conexión serie entre dos puertos simulados. Esto se realiza a través de la herramienta Socat. Para ello se introduce el siguiente comando:

```
$ sudo socat pty,raw,echo=0,link=/dev/ttyS6 pty,raw,echo=0,
  link=/dev/ttyS1
```

Mediante este comando se simula una conexión entre los puertos serie `/dev/ttyS6` y `/dev/ttyS1`. El primer puerto se asigna al programa OBDSim, mientras

que el segundo se asigna al subagente. Esto último se realiza modificando la siguiente linea del fichero `serial_port.c`:

```
char *portname = "/dev/ttyS1";
```

A continuación el siguiente paso es ejecutar el programa OBDSim. Esto se realiza introduciendo el siguientes comando dentro de la carpeta `bin` del programa:

```
$ sudo ./obdsim -g gui_fltk -t /dev/ttyS6
```

El argumento `-g gui_fltk` se utiliza para abrir la interfaz gráfica del programa. Por otro lado, el argumento `-t /dev/ttyS6` se utiliza para que el programa utilice el puerto serie que aparece ahí.

Una vez que el programa OBDSim está en funcionamiento, el siguiente paso es activar el subagente. Para ello se ejucuta el siguiente comando:

```
$ sudo ./main
```

Tras introducir el comando anterior, se establece la comunicación entre el subagente y el vehículo (OBDSim). Por lo tanto, el siguiente paso es definir un sistema gestor que se comunique con el maestro y obtenga información relativa al coche.

Implementación del sistema gestor

Para realizar la simulación se usa, como sistema gestor, un ordenador cuyo sistema operativo es Windows 10. En este entorno, se usa el programa Free SNMP MIB Browser para hacer el rol de gestor. Esta aplicación, desarrollada por ManageEngine, permite el uso de SNMPv3 de forma gratuita, además tiene un handler que permite observar todas las notificaciones que llegan al sistema.

Para que la comuniación entre sistema gestor y agente sea posible, es necesario editar el fichero de configuración del agente maestro. Para ello en el archivo `etc/snmp/snmpd.conf` se añade la siguiente linea:

```
rocommunity secret  192.168.1.42
```

Donde el valor "`192.168.1.42`"se sustituye por la dirección IP del sistema gestor.

Una vez que el agente ha sido configurado, el siguiente paso es configurar el

sistema gestor para que ambos puedan intercambiar paquetes SNMP. Para ello, se configura el programa MIB Browser de la forma apropiada. En la figura Figura 27 se puede observar la interfaz gráfica de la aplicación.

En dicha ventana, se pulsa sobre el icono que tiene forma de engranaje. Una vez que los engranajes han sido pulsados, se abre una ventana como la que aparece en la figura Figura 28. En esta, se selecciona que la versión de SNMP que se utiliza es SNMPv3.

A continuación, se pulsa sobre el botón `add`. Esto se realiza para indicar un usuario al gestor. En concreto, se añade el usuario que se creo en el agente, cuyo nombre de usuario es obd2.

Una vez que se pulsa sobre el botón `add` se abre una ventana en donde se indican los parámetros del agente. En esta, aparecen las siguientes opciones:

- `Target Host`: Este campo se rellena con la dirección IP del agente al que se desea interrogar.
- `Target Port`: El puerto que utiliza el agente para escuchar las peticiones SNMP provenientes del gestor. En este caso se deja el valor por defecto, que es el puerto 161.
- `User Name`: Nombre de usuario que accede al agente. Este campo se rellena con el valor obd2.
- `Security Level`: En este campo se selecciona el valor AuthPriv, el cual hace uso de autenticación y cifrado
- `Auth Protocol`: Se selecciona el valor SHA.
- `Auth Password`: Este campo se rellena con la contraseña de autenticación. En el caso del prototipo el valor es telematica.
- `Priv Protocol`: se selecciona el valor AES.
- `Priv Password`: Este campo define la contraseña de cifrado de los paquetes SNMP. En el caso del proyecto se rellena con el valor telematica.

El resto de los campos se dejan en blanco. Una vez que los campos mencionados anteriormente han sido rellenados de la forma adecuada se pulsa el botón `apply` para aplicar los cambios. En la figura Figura 29 se puede observar lo que se ha comentado anteriormente.

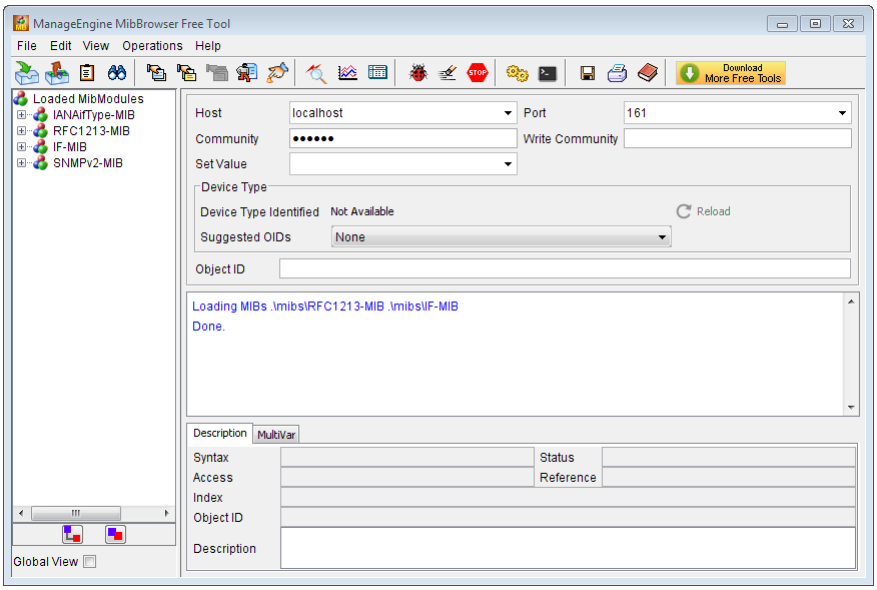


Figura 27: Interfaz gráfica Free SNMP MIB Browser

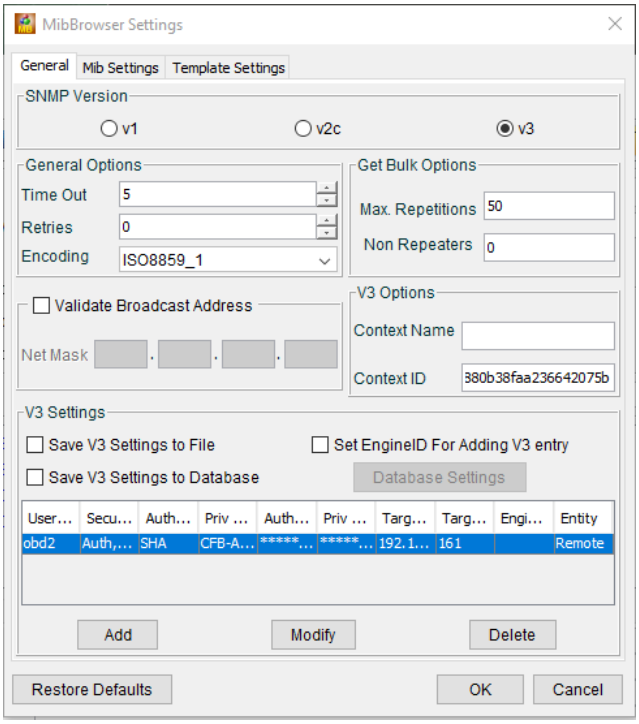


Figura 28: Ventana de configuración del gestor

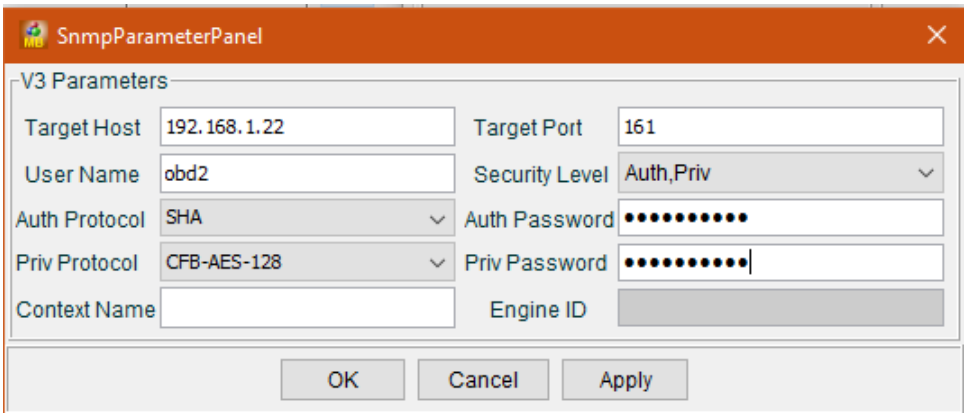


Figura 29: Ventana de creación de usuario

Finalmente, después de haber configurado el gestor de la forma correcta, el último paso es obtener el valor deseado de los objetos definidos en la MIB de OBD-II.

Por ejemplo se obtiene el valor del nombre del vehículo. Este elemento se encuentra en el OID `.1.3.6.1.4.1.8072.2.333.1.1.1`. Para obtener dicho valor se envía una petición de tipo get al agente. Para realizar esto se debe pulsar sobre el icono que es una mano con unas cartas azules.

En la siguiente figura se puede observar el ejemplo mencionado anteriormente.

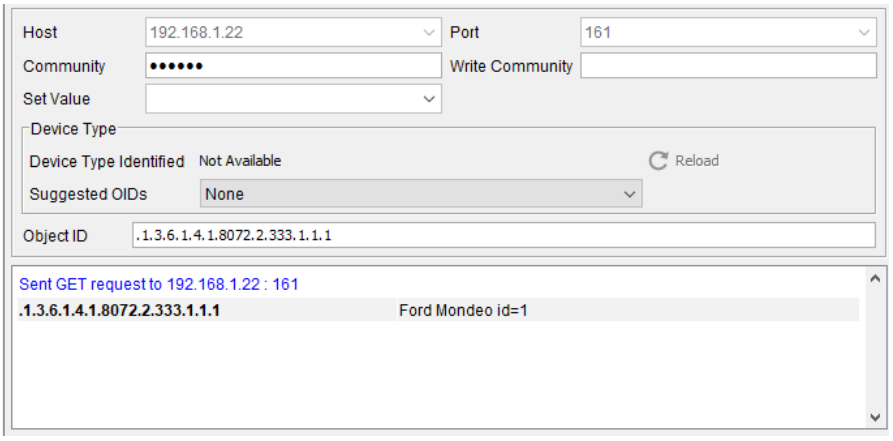


Figura 30: Get al objeto `name` de la MIB

Como se puede observar en la imagen anterior, el valor asociado a ese objeto se muestra en el campo inferior. En este caso, se trata de un string

cuyo resultado es "`Ford Mondeo id=1`"

Si por ejemplo se desea obtener el valor de la temperatura del motor, se realizaría una petición de tipo Get al OID `.1.3.6.1.4.1.8072.2.333.1.1.3.0`.

En la figura Figura 31 se puede observar la obtención del valor relacionado a la temperatura del motor del vehículo.

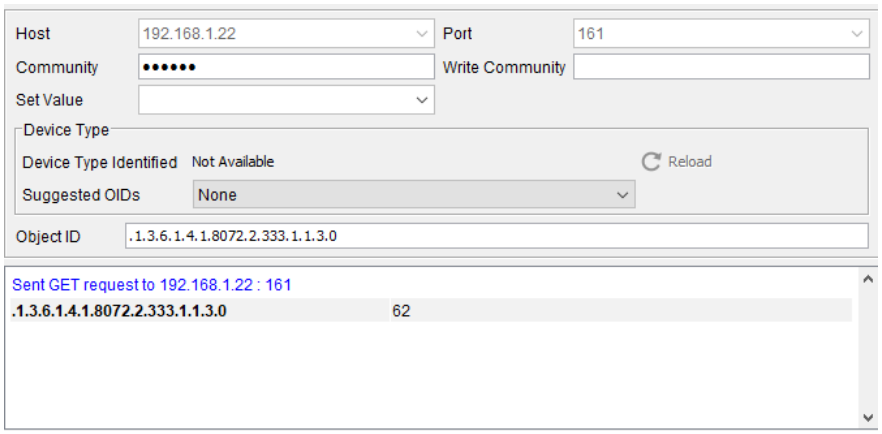


Figura 31: Get al objeto `engineTemp` de la MIB

En este caso, el resultado obtenido es un entero cuyo valor es `62`.

Este programa, además, cuenta con un handler de traps. Este, escucha todas las notificaciones que llegan a la dirección del gestor en el puerto 162. Para abrir este handler se pulsa sobre el icono que tiene una lupa y un rayo.

Tras pulsar dicho botón se abre una nueva ventana, en donde se puede observar todas las notificaciones que llegan al sistema. Cada trap lleva asociados los siguientes campos:

- `Type`: Versión de la trap que llega al gestor.
- `Source`: Dirección IP del sistema que genera la notificación.
- `Date`: Fecha en la que se ha originado la trap.
- `Message`: En este campo aparecen el OID del objeto que genera la notificación, el OID de la notificación y un valor proporcionado por el sistema que genera la trap.

En la figura Figura 32 se puede une ejemplo de notificación recibida por el gestor. En esta, se puede observar todos los parámetros que aparecen.

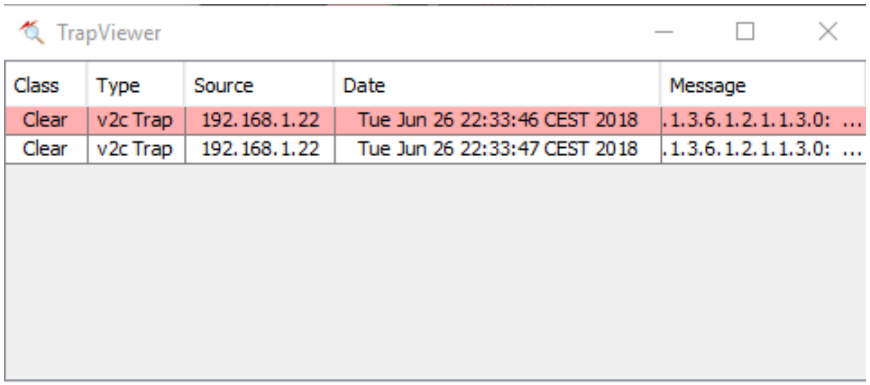


Figura 32: Handler de traps

5. Conclusiones

Hoy en día es frecuente el uso de dispositivos IoT para el desarrollo de redes de sensores. Por lo tanto, sin perder el punto de vista de este tipo de equipos los cuales nos permiten obtener valores asociados a un determinado objeto, se pretendió la realización de una tecnología capaz de obtener información de un vehículo y enviársela a cualquier terminal de la red.

Actualmente, cada vez más vehiculos llevan incorporados puntos de acceso con los que acceder a la red. Pensando en esto, y en conjunción con la tecnología OBD-II, se ha desarrollado un agente SNMP sobre una Raspberry Pi. En este miniordenador, el agente se encargaría de interrogar al coche por los objetos indicados por el sistema gestor. El uso de internet, la cual es una red pública y por lo tanto no confiable, hace que el intercambio de información entre sistema agente y sistema gestor esté sujeto a escuchas y por lo tanto, que cualquier usuario mal-intencionado sea capaz de obtener información personal del vehículo. Para solucionar este problema, simplemente se implemento la versión tres del protocolo SNMP. Esta, al incorporar técnicas de cifrado y autenticación hace que la comunicación entre ambos extremos esté protegida.

A lo largo de la realización de este proyecto se han encontrado ciertas dificultades. Muchos de estos problemas provenían por la instalación y configuración del software principal, NET-SNMP. Estos, se debían a la incompatibilidad con otras aplicaciones o a problemas a la hora de encontrar determinadas librerías. Otros de estos inconvenientes, han sido propiciados por el entendimiento del protocolo SNMP, el cual a pesar de que su nombre incluye la palabra simple, realmente no lo es.

La evolución de este proyecto consistiría en implementar el agente SNMPv3 en dispositivos cada vez más pequeños y por lo tanto más limitado en términos de computación. Por ejemplo dispositivos como arduino. Al fin y al cabo, la Rasoberry Pi se está usando para una única función, que es interrogar al vehículo y enviar valores al sistema gestor. De esta forma, se está malgastando recursos tanto tecnológicos como monetarios.

Finalmente, se puede observar que SNMP puede convertirse en una tecnología eficiente y realmente útil a la hora de monitorizar y gestionar cualquier red de dispositivos IoT. Esto se debe a su capacidad de realizar una gestión centralizada. Por ejemplo, empresas de transporte, domótica, ciudades inteligentes, etc. pueden hacer uso de esta ciencia.

Bibliografía

[1] Artículo Tutorial: *Internet of Things (IoT)* Keysight Technologies, visto el 05/07/2018

[2] Andy Greenberg *Hackers Remotely Kill a Jeep on the Highway—With Me in It* Artículo de WIRED, 21/07/2015.

[3] Alex Infanzon *7 aplicaciones prácticas de IoT que ya son una realidad* Artículo para la revista Forbes México, 19/10/2015.

[4] Douglas R. Mauro & Kevin J. Schmidt. *Essential SNMP (1st ed.).* O'Reilly & Associates. Sebastopol, 2001.

[5] Charles M. Kozierok `http://www.tcpipguide.com/free/t_SNMPVersion1SNMPv1MessageFormat.html`
The TCP/IP Guide, Version 3.0, visto el 05/01/2018

[6] Jose Ángel Irastorza *Tema V: SNMPv1, v2, v3* Apuntes de la Asignatura de Gestión y Operación de Redes, Universidad de Cantabria, 2017

[7] K. Mccloghrie, Cisco Systems, D. Perkins, SNMPinfo, J. Schoenwaelder, Tu Braunchschweig `Structure of Managment Information Version 2` RFC 2578, Network Working Group, Internet Engineering Task Force

[8] *7 Best OBD2 (Android/IOS) apps For Your Car in 2017* `http://www.archer-soft.com/en/blog/7-best-obd2-androidios-apps-your-car-2017`
Archer-Soft, visto el 24/05/2018

[9] Rubén Andrés *15 usos de la Raspberry Pi que no sabías que podías darle* `https://computerhoy.com/noticias/hardware/15-usos-raspberry-pi-que-no-sabias-que-podias-darle-74905`
ComputerHoy, visto el 27/05/2018

[10] `http://www.net-snmp.org/`
NET-SNMP, visto el 20/01/2018

[11] `https://icculus.org/obdgpslogger/obdsim.html`
Icculus, visto el 15/03/2018

[12] Gerhard Rieger. `http://www.dest-unreach.org/socat/doc/socat.html`
Visto el 16/03/2018

Anexos

Árbol de la MIB

```
+--oBD2MIB(333)
    |
    +--oBD2Objects(1)
        |
        +--modex(x)
        |   |
        |   +-- -R-- String      name(1)
        |   +-- -R-- Integer32 engineLoad(2)
        |   |        Range: 0..100
        |   +-- -R-- Integer32 engineTemp(3)
        |   |        Range: -40..215
        |   +-- -R-- Integer32 shortTermFuelTrimBank1(4)
        |   |        Range: -100..100
        |   +-- -R-- Integer32 longTermFuelTrimBank1(5)
        |   |        Range: -100..100
        |   +-- -R-- Integer32 shortTermFuelTrimBank2(6)
        |   |        Range: -100..100
        |   +-- -R-- Integer32 longTermFuelTrimBank2(7)
        |   |        Range: -100..100
        |   +-- -R-- Integer32 fuelPressure(8)
        |   |        Range: 0..765
        |   +-- -R-- Integer32 absolutePressure(9)
        |   |        Range: 0..255
        |   +-- -R-- Integer32 engineRpm(10)
        |   +-- -R-- Integer32 vehicleSpeed(11)
        |   |        Range: 0..255
        |   +-- -R-- Integer32 timingAdvance(12)
        |   |        Range: -64..64
        |   +-- -R-- Integer32 intakeAirtemperature(13)
        |   |        Range: -40..215
        |   +-- -R-- Integer32 mafAirFlowRate(14)
        |   |        Range: 0..655
        |   +-- -R-- Integer32 throttlePosition(15)
        |   |        Range: 0..100
        |   +-- -R-- Integer32 runTimeSinceEngineStart(16)
        |   |        Range: 0..65535
```

```
|   +-- -R-- Integer32 distanceMalunctionLamp(17)
|   |        Range: 0..65535
|   +-- -R-- Integer32 fuelrailpressure(18)
|   |        Range: 0..5177
|   +-- -R-- Integer32 fuelRailGaugePressure(19)
|   |        Range: 0..65535
|   +-- -R-- Integer32 commandedEgr(20)
|   |        Range: 0..100
|   +-- -R-- Integer32 egrError(21)
|   |        Range: -100..99
|   +-- -R-- Integer32 commandedEvaporativePurge(22)
|   |        Range: 0..100
|   +-- -R-- Integer32 fuelTankLevelInput(23)
|   |        Range: 0..100
|   +-- -R-- Integer32 warmsupSinceCodeCleared(24)
|   |        Range: 0..255
|   +-- -R-- Integer32 distanceTravelCleared(25)
|   |        Range: 0..65535
|   +-- -R-- Integer32 evapVaporPressure(26)
|   |        Range: -8192..8191
|   +-- -R-- Integer32 absolutePressure(27)
|   |        Range: 0..255
|   +-- -R-- Integer32 controlModuleVoltage(28)
|   |        Range: 0..65
|   +-- -R-- Integer32 absoluteLoadValue(29)
|   |        Range: 0..25700
|   +-- -R-- Integer32 fuelAirRatio(30)
|   |        Range: 0..2
|   +-- -R-- Integer32 relativeThrottlePos(31)
|   |        Range: 0..100
|   +-- -R-- Integer32 ambientTemp(32)
|   |        Range: -40..215
|   +-- -R-- Integer32 ethanolFuelPer(33)
|   |        Range: 0..100
|   +-- -R-- Integer32 abseVapSysPressure(34)
|   |        Range: 0..327
|   +-- -R-- Integer32 hybridBatteryLife(35)
|   |        Range: 0..100
|   +-- -R-- Integer32 engineOilTemp(36)
|   |        Range: -40..210
|   +-- -R-- Integer32 engineFuelRate(37)
```

```
|   |          Range: 0..3276
|   +-- -R-- Integer32 engineReferenceTorque(38)
|   |          Range: 0..65535
|   +--trapEngineTemp(39)
|
+--mode3(2)
    |
    +-- -R-- Integer32 DTC(1)
```

Código 12: Resultado del comando snmptranslate

Códigos y librerías asociados a los objetos

```
1 #include <net-snmp/net-snmp-config.h>
2 #include <net-snmp/net-snmp-includes.h>
3 #include <net-snmp/agent/net-snmp-agent-includes.h>
4 #include "rpm.h"
5
6 long     rpm = 0;
7 void
8 init_rpm(void)
9 {
10    netsnmp_handler_registration *reg;
11
12    const oid rpm_oid[] = { 1,3,6,1,4,1,8072,2,333,1,1,10 };
13    static netsnmp_watcher_info rpm_winfo;
14
15    DEBUGMSGTL(("rpm", "Initializing the rpm module\n"));
16
17    DEBUGMSGTL(("rpm",
18                "Initializing rpm scalar integer.  Default
                    value = %d\n",
19                rpm));
20    reg = netsnmp_create_handler_registration(
21              "rpm", NULL,
22               rpm_oid, OID_LENGTH(rpm_oid),
23               HANDLER_CAN_RWRITE);
24
25    netsnmp_init_watcher_info(&rpm_winfo, &rpm, sizeof(long),
26 ASN_INTEGER, WATCHER_FIXED_SIZE);
27
28    if (netsnmp_register_watched_scalar( reg, &rpm_winfo ) < 0
        ) {
29        snmp_log( LOG_ERR, "Failed to register watched rpm" );
30    }
31
```

```
32
33      DEBUGMSGTL(("rpm",
34                  "Done initalizing rpm module\n"));
35  }
```

Código 13: Objeto enginerpm

```
1 #ifndef RPM_H
2 #define RPM_H
3
4 void init_rpm(void);
5 long rpm;
6
7 #endif
```

Código 14: Librería asociada al objeto enginerpm

```
1  #include <net-snmp/net-snmp-config.h>
2  #include <net-snmp/net-snmp-includes.h>
3  #include <net-snmp/agent/net-snmp-agent-includes.h>
4  #include "nombre.h"
5
6  char nombre[255] = "NA";
7
8  void
9  init_nombre(void)
10 {
11      const oid nombre_oid[] = { 1,3,6,1,4,1,8072,2,333,1,1,1   };
12
13
14      netsnmp_handler_registration *reginfo;
15      static netsnmp_watcher_info watcher_info;
16      int watcher_flags;
17
18      reginfo = netsnmp_create_handler_registration("my example
            string", NULL,
19                                                    nombre_oid,
20                                                    OID_LENGTH(nombre_oid),
21                                                    HANDLER_CAN_RWRITE);
22
23      watcher_flags = WATCHER_SIZE_STRLEN;
24
25
26      netsnmp_init_watcher_info6(&watcher_info, nombre,
            strlen(nombre),
```

```
27                                      ASN_OCTET_STR, watcher_flags,
28                                      sizeof(nombre), NULL);
29
30
31      netsnmp_register_watched_instance(reginfo, &watcher_info);
32
33      DEBUGMSGTL(("example_string_instance", "Done initalizing
            example string instance\n"));
34  }
```

Código 15: Objeto name

```
1  #ifndef NOMBRE_h
2  #define NOMBRE_h
3
4  void init_nombre(void);
5  extern char nombre[255];
6
7  #endif
```

Código 16: Librería asociada al objeto name

```
1  #include <net-snmp/net-snmp-config.h>
2  #include <net-snmp/net-snmp-includes.h>
3  #include <net-snmp/agent/net-snmp-agent-includes.h>
4  #include "acelerador.h"
5
6
7  long     acelerador = 0;
8
9  void
10 init_acelerador(void)
11 {
12   netsnmp_handler_registration *reg;
13
14     const oid acelerador_oid[] = {
           1,3,6,1,4,1,8072,2,333,1,1,15 };
15     static netsnmp_watcher_info acelerador_winfo;
16
17     DEBUGMSGTL(("acelerador", "Initializing the acelerador
           module\n"));
18     DEBUGMSGTL(("acelerador",
19                 "Initializing acelerador scalar integer.
                       Default value = %d\n",
```

```
20                  acelerador));
21     reg = netsnmp_create_handler_registration(
22               "acelerador", NULL,
23                acelerador_oid, OID_LENGTH(acelerador_oid),
24                HANDLER_CAN_RWRITE);
25     netsnmp_init_watcher_info(&acelerador_winfo, &acelerador,
           sizeof(long),
26                    ASN_INTEGER, WATCHER_FIXED_SIZE);
27 if (netsnmp_register_watched_scalar( reg, &acelerador_winfo ) <
       0 ) {
28          snmp_log( LOG_ERR, "Failed to register watched
               acelerador" );
29     }
30
31
32   DEBUGMSGTL(("acelerador",
33               "Done initalizing acelerador module\n"));
34 }
```

Código 17: Objeto throttlePosition

```
1 #ifndef ACELERADOR_H
2 #define ACELERADOR_H
3
4 void init_acelerador(void);
5 long acelerador;
6
7 #endif
```

Código 18: Librería asociada al objeto throttlePosition

```
1 #include <net-snmp/net-snmp-config.h>
2 #include <net-snmp/net-snmp-includes.h>
3 #include <net-snmp/agent/net-snmp-agent-includes.h>
4 #include "tempmotor.h"
5
6 long     tempmotor = 0;
7
8 void
9 init_tempmotor(void)
10 {
11   netsnmp_handler_registration *reg;
12
13     const oid tempmotor_oid[] = { 1,3,6,1,4,1,8072,2,333,1,1,3
           };
```

```
14      static netsnmp_watcher_info tempmotor_winfo;
15
16      DEBUGMSGTL(("tempmotor", "Initializing the tempmotor
            module\n"));
17
18      DEBUGMSGTL(("tempmotor",
19                  "Initializing tempmotor scalar integer.
                        Default value = %d\n",
20                  tempmotor));
21      reg = netsnmp_create_handler_registration(
22                "tempmotor", NULL,
23                 tempmotor_oid, OID_LENGTH(tempmotor_oid),
24                 HANDLER_CAN_RWRITE);
25      netsnmp_init_watcher_info(&tempmotor_winfo, &tempmotor,
            sizeof(long),
26                      ASN_INTEGER, WATCHER_FIXED_SIZE);
27  if (netsnmp_register_watched_scalar( reg, &tempmotor_winfo ) <
        0 ) {
28          snmp_log( LOG_ERR, "Failed to register watched
                tempmotor" );
29      }
30
31
32    DEBUGMSGTL(("tempmotor",
33                "Done initalizing tempmotor module\n"));
34  }
```

Código 19: Objeto engineTemp

```
1 #ifndef TEMPMOTOR_H
2 #define TEMPMOTOR_H
3
4 void init_tempmotor(void);
5 long tempmotor;
6
7 #endif
```

Código 20: Librería asociada al objeto engineTemp

Código y librería del puerto serie

```
1 #include <errno.h>
2 #include <fcntl.h>
3 #include <stdio.h>
4 #include <stdlib.h>
```

```
 5 #include <string.h>
 6 #include <termios.h>
 7 #include <unistd.h>
 8 #include <math.h>
 9 #include "../defs.h"
10
11 int set_interface_attribs(int fd, int speed)
12 {
13     struct termios tty;
14
15     if (tcgetattr(fd, &tty) < 0) {
16         printf("Error from tcgetattr: %s\n", strerror(errno));
17         return -1;
18     }
19
20     cfsetospeed(&tty, (speed_t)speed);
21     cfsetispeed(&tty, (speed_t)speed);
22
23     tty.c_cflag |= (CLOCAL | CREAD);    /* ignore modem
           controls */
24     tty.c_cflag &= ~CSIZE;
25     tty.c_cflag |= CS8;         /* 8-bit characters */
26     tty.c_cflag &= ~PARENB;     /* no parity bit */
27     tty.c_cflag &= ~CSTOPB;     /* only need 1 stop bit */
28     tty.c_cflag &= ~CRTSCTS;    /* no hardware flowcontrol */
29
30     /* setup for non-canonical mode */
31     tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR
           | IGNCR | ICRNL | IXON);
32     tty.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
33     tty.c_oflag &= ~OPOST;
34
35     /* fetch bytes as they become available */
36     tty.c_cc[VMIN] = 1;
37     tty.c_cc[VTIME] = 1;
38
39     if (tcsetattr(fd, TCSANOW, &tty) != 0) {
40         printf("Error from tcsetattr: %s\n", strerror(errno));
41         return -1;
42     }
43     return 0;
44 }
45
46 void set_mincount(int fd, int mcount)
47 {
48     struct termios tty;
49
50     if (tcgetattr(fd, &tty) < 0) {
51         printf("Error tcgetattr: %s\n", strerror(errno));
```

```
52          return;
53      }
54
55      tty.c_cc[VMIN] = mcount ? 1 : 0;
56      tty.c_cc[VTIME] = 5;          /* half second timer */
57
58      if (tcsetattr(fd, TCSANOW, &tty) < 0)
59          printf("Error tcsetattr: %s\n", strerror(errno));
60  }
61
62
63  int open_port(){
64      char *portname = "/dev/ttyS0";
65      int fd;
66
67      fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC); //O_SYNC
            guarantees that the call will not return before all data
            has been transferred to the disk
68
69      if (fd < 0) {
70          printf("Error opening %s: %s\n", portname,
                strerror(errno));
71          return -1;
72      }
73
74      /*baudrate 115200, 8 bits, no parity, 1 stop bit */
75
76      set_interface_attribs (fd, B9600);
77      set_mincount(fd, 0);
78
79      return (fd);
80
81  }
```

Código 21: serial_port.c

```
1  #ifndef SERIAL_PORT_h
2  #define SERIAL_PORT_h
3
4
5  int set_interface_attribs(int fd, int speed);
6  void set_mincount(int fd, int mcount);
7  int open_port(void);
8
9  #endif
```

Código 22: serial_port.h

Códigos y librerías asociados al programa principal

```
1 #include <net-snmp/net-snmp-config.h>
2 #include <net-snmp/net-snmp-features.h>
3 #include <net-snmp/net-snmp-includes.h>
4 #include <net-snmp/agent/net-snmp-agent-includes.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <signal.h>
8 #include "./rpm/rpm.h"
9 #include "./acelerador/acelerador.h"
10 #include "./tempmotor/tempmotor.h"
11 #include "defs.h"
12 #include "./leerobd2/serial_port.h"
13 #include "./leerobd2/procesar_peticion.h"
14 #include "./nombre/nombre.h"
15
16
17 #define num_objetos 3 //rpm, acelerador, tempmotor
18
19 /*Valores que van a tomar los elementos de la mib (Variables
    externas)*/
20 extern long rpm;
21 extern long acelerador;
22 extern long tempmotor;
23 char nombre[255];
24
25 /* Estructura de comunicaciön con el puerto serie */
26
27
28
29 static int keep_running;
30
31 RETSIGTYPE
32 stop_server(int a) {
33     keep_running = 0;
34 }
35
36 static nueva_pdu pdu;
37
38
39 int main (int argc, char **argv) {
40
41     int agentx_subagent=1;
42     int background = 0;
43     int syslog = 0;
44
```

```
45      int fd = open_port();
46
47      // Cambiar por lo que se desea buscar
48
49      char peticiones[] = "010c\n0111\n0105\n";
50      char tipo[num_objetos] = {'0','0', '0'};
51      int pet_id[num_objetos] = {12, 17, 5};
52      int cargar_int[num_objetos];
53
54
55      int k, i, j;
56
57      if (syslog)
58          snmp_enable_calllog();
59      else
60          snmp_enable_stderrlog();
61
62      if (agentx_subagent) {
63          netsnmp_ds_set_boolean(NETSNMP_DS_APPLICATION_ID,
                NETSNMP_DS_AGENT_ROLE, 1);
64      }
65
66      if (background && netsnmp_daemonize(1, !syslog))
67          exit(1);
68
69      SOCK_STARTUP;
70      init_agent("main");
71
72      /* Inicializo mis agentes para que sean capaces de tomar
            valores*/
73
74      init_rpm();
75      init_acelerador();
76      init_nombre();
77      init_tempmotor();
78
79      if (!agentx_subagent) {
80          init_vacm_vars();
81          init_usmUser();
82      }
83
84      init_snmp("main");
85
86      if (!agentx_subagent)
87          init_master_agent();
88
89      keep_running = 1;
90      signal(SIGTERM, stop_server);
91      signal(SIGINT, stop_server);
```

```
92
93      snmp_log(LOG_INFO,"nombre_demon is up and running.\n");
94
95
96      i = 0;
97      j = 0;
98      while(keep_running) {
99
100         k = 0;
101         // Obtiene la peticion (Se puede simplificar con switch)
102         while(1){
103             pdu.peticion[k] = peticiones[i];
104             if (peticiones[i] == '\n'){
105                 pdu.tipo = tipo[j];
106                 pdu.pet_id = pet_id[j];
107                 i++;
108                 j++;
109                 break;
110             }
111             k++;
112             i++;
113         }
114
115         // Aqui obtengo cada una de las pdus individuales
116
117
118         procesar_peticion(&pdu, fd);
119         cargar_int[j-1] = pdu.respuesta_int;
120
121         memset(pdu.peticion, 0, sizeof(pdu.peticion));
122         memset(pdu.respuesta_char, 0,
                sizeof(pdu.respuesta_char));
123
124
125         /***        Aqui relleno los objetos de la MIB
                ***/
126
127         rpm = cargar_int[0];
128         acelerador = cargar_int[1];
129         tempmotor = cargar_int[2];
130         strcpy(nombre,"Ford Mondeo id=1");
131
132         if (j == num_objetos) {
133             j = 0;
134             i = 0;
135             agent_check_and_process(0);
136             while (agent_check_and_process(1) == 0);    ///
                    Puesto para que el demonio no estë enviando
                    peticiones constantemente
```

```
137             }
138
139         }
140
141         snmp_shutdown("main");
142         SOCK_CLEANUP;
143
144         return 0;
145 }
```

Código 23: main.c

```
1 #ifndef DEFS_H
2 #define DEFS_H
3
4 typedef struct {
5     char peticion[32];
6     int pet_id;
7     char respuesta_char[255];
8     int respuesta_int;
9     char tipo; // '0' = entero,    '1' = string
10 } nueva_pdu;
11
12 #endif
```

Código 24: defs.h

Comunicación e interpretación de los comandos entre vehículo y agente

```
1 #include <errno.h>
2 #include <fcntl.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <termios.h>
7 #include <unistd.h>
8 #include <math.h>
9 #include "../defs.h"
10 #include "calculo_pids.h"
11
12
13
14 int* select_pid(char *pid, int offset, char tipo, nueva_pdu
      *pdu)
```

```
15 {
16     int i, k;
17     int rpm;
18     unsigned char resultado[] = "000000000000";
19     char *leer; // Lee en caso de que sea un string
20
21     static int response[4];
22     int a,b,c,d;
23
24
25     if ( tipo == '0'){
26
27         // Quito los espacios a la cadena de texto
28
29         k = 0;
30         for (i = (offset+2);i < strlen(pid);i++){
31             if (pid[i] == '\x0d') break; // Detecta el LF (0x0D)
32             if (pid[i] != ' '){
33                 resultado[k] = pid[i];
34                 k++;
35             }
36
37         }
38
39         // Convertir ascii a decimal
40
41         for (i = 0;i < sizeof(resultado);i++){
42             if (resultado[i] <= '\x39') resultado[i] -= '0';
43             else {
44                 resultado[i] -= '7';
45             }
46         }
47
48         // A,B,C,D
49
50
51         response[0] = resultado[4]*16 +resultado[5];
52         response[1] = resultado[6]*16 +resultado[7];
53         response[2] = resultado[8]*16 +resultado[9];
54         response[3] = resultado[10]*16 +resultado[11];
55
56     } else {
57
58         k = 0;
59         for(i = (offset+2) ;i < strlen(pid);i++){
60             pdu->respuesta_char[k] = pid[i];
61             k++;
62         }
63     }
```

```
64
65      return response;
66  }
67
68  int procesar_peticion(nueva_pdu *pdu, int fd){
69
70
71      char *buf;
72      int wlen;
73      int nbytes;
74      char leer;
75      int i;
76      char tipo = pdu->tipo;
77
78      int offset = strlen(pdu->peticion);
79      int *response;
80
81
82      wlen = write(fd, pdu->peticion, strlen(pdu->peticion));
83      if (wlen != strlen(pdu->peticion)) {
84          printf("Error from write: %d, %d\n", wlen, errno);
85      }
86      tcdrain(fd);
87
88      buf = (char *) malloc(255);
89
90      i = 0;
91      while(1) {
92          int rdlen;
93
94          nbytes = read(fd, &leer, 1);
95          if (leer == '>' || i > 32) break;
96          buf[i] = leer;
97          i++;
98      }
99
100     buf[i] = '\0';
101
102     if (tipo == '0'){
103         response = select_pid(buf,offset, tipo, pdu);
104         calculo_pids(response, pdu);
105     }
106
107     free(buf);
108
109     return 0;
110 }
```

Código 25: procesar_peticion.c

```
1 #include <errno.h>
2 #include <fcntl.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <termios.h>
7 #include <unistd.h>
8 #include <math.h>
9 #include "../defs.h"
10
11
12 int calculo_pids(int *response, nueva_pdu *pdu){
13
14     int resto;
15     char comando[500];
16
17
18     switch(pdu->pet_id){
19         case 12 :
20             resto = (256*response[0]+response[1]) % 4;
21
22             pdu->respuesta_int = (256*response[0]+response[1])
                   / 4;
23             if (resto != 0) pdu->respuesta_int += 1;
24
25             break;
26         case 17 :
27             resto = (100*response[0]) %255;
28             pdu->respuesta_int = (100*response[0]) / 255;
29             if (resto != 0) pdu->respuesta_int += 1;
30             break;
31
32         case 5 :
33             pdu->respuesta_int = response[0] - 40;
34             strcpy(comando, "sudo snmptrap -v 2c -c public
                   192.168.1.47 0 OBD2MIB::trapEngineTemp
                   OBD2MIB::engineTemp.0 i -1");
35
36             if (pdu->respuesta_int > 90) {
37                 system(comando);
38             }
39             break;
40
41         default :
42             pdu->respuesta_int = -1;
43     }
```

```
44
45      return 0;
46
47 }
```

Código 26: calculo_pids.c

Script y Makefile para la compilación del programa

```
1 #!/bin/bash
2
3 gcc -c ./leerobd2/serial_port.c -o ./leerobd2/serial_port.o
4 gcc -c ./leerobd2/procesar_peticion.c -o
      ./leerobd2/procesar_peticion.o
5 gcc -c ./leerobd2/calculo_pids.c -o ./leerobd2/calculo_pids.o
6
7 gcc -c ./rpm/rpm.c -o ./rpm/rpm.o
8
9 gcc -c ./acelerador/acelerador.c -o ./acelerador/acelerador.o
10
11 gcc -c ./tempmotor/tempmotor.c -o ./tempmotor/tempmotor.o
12
13 gcc -c main.c
14
15 sudo make main
```

Código 27: com_libraries.sh

```
1 CC=gcc
2
3 OBJS1= main.o ./rpm/rpm.o ./acelerador/acelerador.o
      ./leerobd2/serial_port.o ./leerobd2/procesar_peticion.o
      ./leerobd2/calculo_pids.o ./nombre/nombre.o
      ./tempmotor/tempmotor.o
4 TARGETS= main
5
6 CFLAGS=-I. `net-snmp-config --cflags`
7 BUILDLIBS=`net-snmp-config --libs`
8 BUILDAGENTLIBS=`net-snmp-config --agent-libs`
9
10 # shared library flags (assumes gcc)
11 DLFLAGS=-fPIC -shared
12
13 all: $(TARGETS)
14
```

```
15 main: $(OBJS1)
16 	$(CC) -DDISPLAY_STRING -o main $(OBJS1) -lpthread
       $(BUILDAGENTLIBS)
17
18 clean:
19 	rm $(OBJS2) $(OBJS2) $(TARGETS)
```

Código 28: Makefile