UNIVERSIDADE DA CORUÑA

Departamento de Tecnoloxías da Información
e as Comunicacións

LABORATORIO DE GESTIÓN DE REDES: CREACIÓN DE MÓDULOS EN NET-SNMP

ÍNDICE DE CONTENIDOS

1 ¿QUÉ ES NET-SNMP? 5

2 COMANDOS SNMP BÁSICOS 5

3 AGENTE NET-SNMP 6

3.1 DESCRIPCIÓN DE AGENTE SNMP 6
3.2 EXTENDIENDO EL AGENTE 6
3.3 CONTENIDOS DE NET-SNMP PARA DESARROLLADORES 6
3.3.1 Localizaciones de Ficheros para Desarrolladores 7
3.4 LIBRERÍA DE LA API NET-SNMP 7
3.5 MÓDULOS EJEMPLO 8
3.6 SOPORTE TÉCNICO PARA DESARROLLADORES 8

4 CREACIÓN DE MÓDULOS 9

4.1 MÓDULOS 9
4.2 CREACIÓN DE MÓDULOS 9
4.3 DEFINICIÓN DE UNA MIB 10
4.3.1 Ficheros MIB 10
4.4 VARIABLES DE ENTORNO MIB 11
4.5 GENERACIÓN DE PLANTILLAS DE CÓDIGO 12
4.6 MODIFICACIÓN DE PLANTILLAS DE CÓDIGO 13
4.7 CONFIGURACIÓN DEL MÓDULO 13

5 MODELADO DE DATOS 15

5.1 RUTINA INIT_MODULE 15
5.2 OBJETOS ESCALARES 16
5.2.1 Objetos Escalares en demo_module_1 16
5.2.2 Modificaciones para la Obtención de Objetos Escalares 17
5.3 TABLAS SIMPLES 18
5.3.1 Tablas Simples en demo_module_2 19
5.3.2 Modificaciones para la Obtención de Datos de Tablas Simples 19
5.3.3 Procesado de Múltiples SET en demo_module_2 20

6 ALMACENAMIENTO DE DATOS DE UN MÓDULO 23

6.1 ACERCA DEL ALMACENAMIENTO DE DATOS EN UN MÓDULO 23
6.1.1 Ficheros de configuración 23
6.1.2 Definición de Tokens de Configuración 23
6.2 DATOS PERSISTENTES EN UN MÓDULO 24
6.2.1 Almacenamiento de Datos Persistentes 24
6.2.2 Lectura de Datos Persistentes 25
6.3 ALMACENAMIENTO DE DATOS PERSISTENTES EN DEMO_MODULE_4 25
6.3.1 Almacenamiento de Datos Persistentes en demo_module_4 25
6.3.2 Lectura de Datos Persistentes en demo_module_4 26
6.3.3 Utilización de SNMP_CALLBACK_POST_READ_CONFIG en demo_module_4 27

7 IMPLEMENTACIÓN DE ALARMAS 28

7.1 INVERVALOS DE REFRESCO 28
7.2 NOTIFICACIÓN DE TRAPS ASÍNCRONAS 28
7.3 UMBRALES PARA EL ENVÍO DE TRAPS 29
7.3.1 Lectura de Datos de Fichero de Configuración en demo_module_3.conf 30
7.3.2 Utilización de SNMP_CALLBACK_POST_READ_CONFIG en demo_module_3 31
7.3.3 Generación de Traps en demo_module_3 31

8 DESARROLLO DE MÓDULOS 33

8.1 Despliegue de un Módulo 33
8.2 Carga Dinámica de Módulos 33
8.3 Carga Dinámica de un Módulo con Reinicio del Agente 33
8.4 Carga Dinámica de un Módulo sin Reinicio del Agente 34

1 ¿Qué es NET-SNMP?

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

- 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*).
  - 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 dinámicamente, *scripts* externos y comandos, y los protocolos de multiplexación SNMP (*SMUX*) y eXtensibilidad del Agente (*AgentX*).
- Una biblioteca para el desarrollo de nuevas aplicaciones SNMP, con APIs para C y Perl.

NET-SNMP está disponible para muchos sistemas operativos UNIX y similares (Linux, FreeBSD...), e incluso para Microsoft Windows. Puede descargarse de http://net-snmp.sourceforge.net/download.html.

2 Comandos SNMP básicos

Net-SNMP incluye una serie de herramientas de línea de comandos. Cada herramienta tiene una página de ayuda asociada. Las herramientas se encuentran localizadas en el directorio `/usr/local/bin`.

Los comandos *snmp* pueden ser utilizados para realizar peticiones sobre el agente y para verificar el comportamiento de nuevos módulos. Leer las páginas de ayuda para una información detallada de su utilización.

3 Agente Net-SNMP

El agente de gestión Net-SNMP de un sistema es un agente de SNMP (Simple Network Management Protocol).

3.1 Descripción de Agente SNMP

SNMP utiliza el término de gestor para la aplicación cliente que accede a los datos sobre un dispositivo gestionado o sistema. El gestor usualmente se ejecuta en un sistema diferente al del sistema gestionado. El término agente es utilizado para el programa que implementa la pila de protocolos para dar servicio a las peticiones del gestor. El agente SNMP típicamente se ejecuta en el dispositivo gestionado. El agente ofrece servicios en un equipo TCP/IP designado. El puerto por defecto SNMP es 161.

La información sobre el dispositivo destino está contenida en una MIB (Management Information Base). Las MIBs son utilizadas por agentes y gestores de modo que ambos programas tienen conocimiento de los datos disponibles. La MIB indica al gestor sobre las funciones y datos del dispositivo. La MIB también indica al gestor cómo dirigir o acceder a esa información en la forma de objetos gestionados. Para acceder a esa información de gestión, el gestor envía peticiones al agente. Las peticiones contienen identificadores para los objetos de la MIB que son de interés para el gestor. Si la petición puede ser completada con éxito, el agente devuelve una respuesta que contiene los valores de los datos requeridos.

La mayoría de agentes SNMP soportan la pila de protocolos básica SNMP, y algunas MIBs mínimas. Sin embargo, para realizar la gestión de un dispositivo más efectiva, deben soportarse en el dispositivo gestionado MIBs adicionales. Las MIBs adicionales son proporcionadas por vendedores de dispositivos para proporcionar información de gestión sobre características propias del dispositivo gestionado.

Una MIB que se añade al agente SNMP se conoce comúnmente como una extensión porque la nueva MIB extiende las capacidades del agente. De ese modo, un agente que puede aceptar extensiones es *extensible*. El agente Net-SNMP, descrito en este manual, es un agente extensible. Las extensiones al agente se denominan módulos.

3.2 Extendiendo el agente

El agente Net-SNMP puede ser extendido de los diferentes modos. La que utilizaremos para la realización de la práctica será mediante la carga dinámica de un módulo. Ver Apartado 8 para más información sobre cómo desplegar módulos como módulos dinámicos y en subagentes.

3.3 Contenidos de Net-SNMP para Desarrolladores

El agente incluye el siguiente contenido para los desarrolladores:

- Herramientas de desarrollo, y módulos Perl necesitados por las herramientas
- Librerías de API para la utilización de funciones Net-SNMP

- Módulos para la demostración de cómo implementar el modelado de algunos tipos de datos

Adicionalmente, puede ser de utilidad el acceso al código fuente de Net-SNMP.

3.3.1 Localizaciones de Ficheros para Desarrolladores

Los ficheros de desarrollados son instalados en las localizaciones que se muestran en la siguiente tabla.

Directorio	**Contenido**
/usr/local/bin	Herramientas de línea de comandos que son útiles para los desarrolladores.
/usr/local/sbin	Ficheros ejecutables para el demonio agente *snmpd* y el demonio receptor de *traps snmptrapd*, que proporcionan los servicios SNMP.
/usr/local/lib	Las librerías compartidas que contienen las funciones de la API de Net-SNMP.
/usr/local/include/net-snmp	Ficheros cabecera necesarios para las librerías de la API.
/usr/local/share/snmp	Ficheros de configuración que son utilizados por la herramienta *mib2c*.
/usr/local/shared/mibs	Las MIBS soportadas por el agente Net-SNMP.
/usr/share/perl/5.8.7/	Módulos Perl necesarios para la herramienta *mib2c*.

Tabla 1 - Localizaciones de Ficheros de Net-SNMP

3.4 Librería de la API Net-SNMP

Net-SNMP incluye las siguientes librerías de la API:

- `libnetsnmp`
- `libnetsnmpagent`
- `libnetsnmpmibs`
- `libnetsnmphelpers`

En plataformas x86, sólo están disponibles las librerías Net-SNMP de 32–bit disponibles en el directorio `/usr/local/lib`.

Las funciones contenidas en las librerías Net-SNMP son utilizadas en los nuevos módulos MIB, como en el agente.

3.5 Módulos Ejemplo

El directorio `demos_net_snmp` contiene varios módulos de demostración. Los módulos demo ilustran métodos para la creación de módulos para resolver varios tipos de problemas de obtención de información. En capítulos posteriores de este manual se discuten en detalle los módulos de demostración. La siguiente tabla lista y describe los módulos demo. La tabla también proporciona referencias cruzadas a las secciones que discuten cada una de las demos.

Nombre Módulo	Contenido	Sección
demo_module_1	Modelado de datos para objetos escalares	"5.2 Objetos Escalares" en la página 16
demo_module_2	Modelado de datos de una tabla simple con objetos *read-write*.	"5.3 Tablas Simples" en la página 18
demo_module_3	Implementación de Alarmas	"7.3.3 Generación de Traps en demo_module_3" en la página 31
demo_module_4	Persistencia de datos de un módulo ante el reinicio del agente	"6.3 Almacenamiento de Datos Persistentes en demo_module_4" en la página 25

Tabla 2 - Descripción de Módulos de Demostración

3.6 Soporte Técnico para Desarrolladores

La comunidad *open source* de http://www.net-snmp.org da soporte técnico para desarrolladores de módulos para el agente Net-SNMP. La lista de correo de discusión de desarrolladores es **net-snmp-coders@lists.sourceforge.net**. El archivo para la lista de correo está localizado en http://sourceforge.net/mailarchive/forum.php?forum_id=12455.

4 Creación de Módulos

Este capítulo proporciona una guía básica para la creación de módulos de agentes de un sistema de gestión. Este capítulo incluye un proceso que puede utilizarse para implementar una MIB como un módulo en un agente de un sistema de gestión.

4.1 Módulos

El término módulo tal y como se utiliza en este documento tiene asociados dos significados cercanos entre si. Generalmente un módulo se refiere al "contenedor" de nuevos fragmentos de datos de gestión que el desarrollador necesita que el agente comunique. En este sentido, un módulo es un concepto abstracto.

Sin embargo, un módulo abstracto debe ser representado como un fichero objeto compartido, que corre en un sistema de gestión. El fichero de objeto compartido, o el programa asociado, se refiere a menudo como un módulo. Por tanto, un módulo puede ser definido como un programa C que funciona junto con un agente para gestionar recursos adicionales.

Todos los módulos se comunican a través de funciones de librería de la API. Las funciones de la API son utilizadas bien en módulos que se compilan junto al agente, o cargadas dinámicamente, o corriendo en un subagente separado.

4.2 Creación de Módulos

Se pueden crear módulos para el agente, con el propósito de permitir la gestión de una aplicación específica, dispositivo, sistema o red mediante una aplicación de gestión. El agente incluye y documenta las funciones que requiere un módulo. Las funciones son utilizadas para registrar el módulo en el agente, para gestionar peticiones sobre los datos del módulo y para realizar otras tareas del módulo.

No es necesario codificar un módulo de forma manual, aunque se puede realizar así si se prefiere. El proceso para describir un módulo manualmente se describe en http://www.net-snmp.org . Ese proceso está fuera del ámbito de este documento.

El proceso de alto nivel descrito en este manual para la implementación de un módulo es el siguiente:

1. **Definir la MIB de los objetos gestionados.**

   Para definir una MIB, se debe conocer que datos de gestión están asociados con el sistema o entidad a gestionar. Se deben asignar nombres de variables a cada elemento discreto de gestión. Se debe también determinar los atributos y tipos de datos ASN.1

   La definición de la MIB está fuera del ámbito de este manual. Ver el Apartado "4.3 Definición de una MIB" en la página 10 para más información sobre MIBs.

2. **Generar las plantillas de código para un módulo a partir de la MIB.**

Para generar las plantillas de código, con la herramienta *mib2c* se convierten los nodos de la MIB en ficheros de código fuente C. Las plantillas de código incluyen funciones de la API para el registro de datos, y manejo de peticiones de datos. Ver el Apartado "3.4 Librería de la API Net-SNMP" de la página 7 para más información.

3. **Modificar las plantillas de código para completar la recolección de datos y porciones de gestión del módulo.**

   Para modificar las plantillas de código, se debe determinar como implementar tal funcionalidad en el agente. Ver el Apartado "3.5 Módulos Ejemplo" de la página 8 para más información.

4. **Compilar los ficheros C en un fichero objeto compartido**.

   Se compila un módulo para el agente del mismo modo que se compilaría cualquier fichero de objeto C compartido.

5. **Decidir el método el despliegue y configuración del módulo.**

   Se debe determinar si se configura el módulo como un subagente separado, o se carga de forma dinámica en el agente SNMP. Ver el Apartado "8 Desarrollo de Módulos" de la página 33 para más información sobre el despliegue.

4.3 Definición de una MIB

La definición de la MIB es una de las tareas que consumen más tiempo en la creación de un módulo.

La herramienta *mib2c*, utilizada para la conversión de MIBs a código C, incluye el chequeo de errores en la sintaxis de la MIB. Se puede utilizar *mib2c* para chequear la sintaxis de una MIB.

Es útil utilizar una de las MIBs estándares que se incluyen en Net-SNMP como modelo para la creación de nuevas MIBs. El directorio `/usr/local/share/snmp/mibs` contiene todas las MIBs estándares soportadas por Net-SNMP.

Es importante señalar que el nombre asignado a MODULE-IDENTITY. Este nombre debe coincidir con el nombre del fichero con los guiones eliminados, y separados los grupos por mayúsculas. Por ejemplo, NET-SNMP-SYSTEM-MIB.txt utiliza *netSnmpSystemMIB* para el MODULE-IDENTITY. Un fichero de una MIB que no utiliza este formato puede no funcionar con *mib2c*.

El fichero NET-SNMP-EXAMPLES-MIB.txt se incluye en el directorio de MIBs, y puede ser de ayuda puesto que define variables de diferentes tipos de datos.

4.3.1 Ficheros MIB

Es necesario asegurarse de que se utilizan nombres únicos para los ficheros MIB. Todos las MIBs de usuario están en el mismo espacio de nombres que las MIBs estándares, incluso aunque se tengan las MIBs de usuario en un directorio separado. La mayoría de las MIBs derivadas de RFCs tienen números de RFC en sus nombres para identificarlas con facilidad,

y asegurar de ese modo la utilización de nombres únicos. Otras MIBs siguen convenciones de nombres, que disminuyen las posibilidades de duplicación de nombres.

Las MIBs usualmente se nombran siguiendo las siguientes convenciones:

- Utiliza letras en mayúscula, y utilizar el carácter "_" para separar los diferentes segmentos que compongan el nombre del fichero.
- Comenzar el nombre de la MIB con el nombre de la compañía. Por ejemplo, si la MIB es para la compañía denominada *Acme*, el primer segmento del nombre de la MIB debería ser ACME.
- Indicar el tipo de objetos en medio del nombre. Por ejemplo, si la MIB es para un *router*, se podría utilizar ROUTER como parte del nombre.
- Incluir MIB como el último segmento del nombre.
- Añadir la extensión de fichero ".txt".

4.4 Variables de Entorno MIB

Se deben establecer las variables de entorno `$MIBS` y `$MIBDIRS` para asegurar que las herramientas que utilizan MIBs pueden localizar y cargar nuevos ficheros MIB. Las herramientas que utilizan MIBs incluyen *mib2c* y todos los comandos SNMP como *snmpget*, *snmpwalk* y *snmpset*.

Se establece la variable de entorno para incluir el fichero MIB que se desee utilizar. Por ejemplo, para añadir una MIB denominada MYTESTMIB.txt a la lista de MIBs, se utiliza el siguiente comando (en las *shells sh* o *bash*):

```
$ export MIBS=+MYTESTMIB
```

El anterior comando añade el fichero MIB `MYTESTMIB.txt` a la lista de módulos MIB por defecto que soporta el agente.

Si se desea incluir todas las MIBS localizadas en el sistema se debe asignar a la variable de entorno `$MIBS` el valor `ALL`, tal y como se realiza a continuación:

```
$ export MIBS=ALL
```

De ese modo, las herramientas que utilizan MIBS incluirán todas las MIBS que se encuentran situadas en que se encuentren las rutas de búsqueda de ficheros MIB.

La ruta de búsqueda de ficheros MIB por defecto es `/usr/local/share/snmp/mibs`. Se puede modificar la ruta de búsqueda estableciendo la variable `$MIBDIRS`. Por ejemplo, para añadir la ruta `/home/user/mydir/mibs` a las rutas de búsqueda de MIBS e incluir todas las MIBS definidas en dichas localizaciones, se ejecutan los siguientes comandos (en la *shell sh* o *bash*):

```
$ export MIBDIRS=$HOME/mydir/mibs:/usr/local/share/snmp/mibs
$ export MIBS=ALL
```

Otra localización en la que buscan las herramientas que utilizan MIBS es en `$HOME/.snmp/mibs`. Por tanto, si los ficheros relativos a las MIBS se localizan en el directorio `$HOME/.snmp/mibs` no es necesario configurar la variable `$MIBDIRS`.

Tanto los ficheros MIB a cargar como las localizaciones de búsqueda de MIBs se pueden configurar también en el fichero *snmp.conf* (ver *man snmp.conf* para más información).

4.5 Generación de Plantillas de Código

La herramienta *mib2c* se utiliza para generar ficheros C de cabecera e implementación a partir de una MIB. Se puede utilizar los ficheros generados como plantillas para el código del módulo. Se pueden modificar las plantillas de forma apropiada, y posteriormente utilizar las plantillas para desarrollar el módulo. Antes de comenzar la generación del módulo, *mib2c* valida la sintaxis del fichero. Cualquier error se muestra por salida estándar. Se deben corregir los errores de sintaxis antes de generar el código. Este chequeo de errores permite la utilización de *mib2c* durante la creación de una MIB para validar la sintaxis de la MIB.

Es importante establecer de forma correcta las variables de entorno de la MIB como se describe en el apartado anterior antes de utilizar la herramienta *mib2c*.

El comando *mib2c* debe ejecutarse contra nodos de la MIB, no sobre la MB entera una única vez. Se necesita especificar el nombre de la MIB, pero el fichero MIB debe estar localizado en algunas de las rutas de búsqueda de MIBs. Al invocar en línea de comandos a *mib2c*, se debe especificar un fichero de configuración y el nombre de una o más nodos de la MIB. El fichero de configuración debe encajar con el tipo de dato en el nodo de la MIB. El comando debe ser utilizado según el siguiente formato:

```
$ mib2c -c <configfile> <MIBnode> [<MIBnode2> <MIBnode3> ...]
```

Por ejemplo, si se tiene un nodo en una MIB denominado `scalarGroup`, se debe utilizar el siguiente comando para generar las plantillas de código:

```
$ mib2c -c mib2c.scalar.conf scalarGroup
```

Como resultado, se generan los ficheros `scalarGroup.h` y `scalarGroup.c`.

Si la MIB contiene datos de tipo escalar y tablas de datos, se debería ejecutar *mib2c* de forma separada por los nodos de cada tipo de dato. Es necesario especificar el fichero de configuración apropiado por cada tipo de dato.

La siguiente tabla lista los ficheros de configuración de *mib2c*. La tabla describe el propósito de cada fichero de configuración, para ayudar a decidir con el fichero de configuración a utilizar por cada dato de gestión.

Fichero configuración	**Propósito**
mib2c.scalar.conf	Para datos escalares, incluyendo enteros y no-enteros. Este fichero de configuración provoca que mib2c genere manejadores para los objetos escalares situados en el nodo

	MIB especificado. Se ignoran las definiciones de objetos de la MIB correspondientes a nodos no terminales, objetos contenidos en tablas y traps/notificaciones.
mib2c.iterate.conf	Para tablas de datos que no se mantienen en la memoria del agente. Las tablas son localizadas externamente, y se necesita recorrer la tabla para localizar la fila correcta. Cuando se utiliza este fichero de configuración, *mib2c* genera un par de rutinas que permiten iterar a través de la tabla. Las rutinas pueden ser utilizadas para seleccionar la fila apropiada para una petición dada. La fila se pasa a la rutina del manejador de la tabla. Esta rutina maneja el resto del procesado para todos los objetos columnares, para las peticiones GET y SET.

Tabla 3 - Ficheros de Configuración para su Utilización con la Herramienta *mib2c*

Ver ayuda en línea de *mib2c* para obtener más detalles sobre la utilización de la herramienta *mib2c*. En el capítulo "5 Modelado de Datos" de la página 15 se describen ejemplos de utilización de *mib2c*.

4.6 Modificación de Plantillas de código

Las plantillas de código generadas por mib2c incluyen código que registra los OIDs de los datos de la MIB y registran los manejadores de peticiones de datos. La rutina `init_module` en la plantilla `mibnode.c` proporciona el código básico para la obtención de datos. Se debe modificar la plantilla para proporcionar la obtención y gestión de datos. Ver "5.1 Rutina init_module" para obtener información sobre la inicialización de una rutina.

La siguiente tabla muestra dónde encontrar más información sobre cómo realizar la recogida de información de diferentes tipos de datos.

Tipo de dato	**Referencia**
Objetos Escalares	Apartado "5.2 Objetos Escalares"
Tablas Simples	Apartado "5.3 Tablas Simples"

Tabla 4 - Documentación de Recogida de Datos

4.7 Configuración del Módulo

La configuración del módulo depende fundamentalmente del módulo. Se puede proporcionar una configuración automática como parte del proceso de instalación del módulo.

Alternativamente, se pueden proporcionar los pasos y sugerencias como parte de la documentación del usuario final. Si se desea que los usuarios puedan establecer parámetros de configuración para el módulo, se pueden almacenar los parámetros de configuración en

un fichero de configuración. De ese modo, cuando el módulo se inicia, obtiene los parámetros de un fichero de configuración. Ver Apartado “6 Almacenamiento de Datos de un Módulo” para más información.

Para cualquier módulo, se debe decidir si la ejecución del módulo se realiza como un subagente o como un módulo cargado dinámicamente (ver Apartado “8 Desarrollo de Módulos” para más información).

5 Modelado de Datos

Este apartado proporciona información sobre cómo modificar la rutina *init_module()* de un módulo para manejar varios tipos de datos. El apartado discute los siguientes ejemplos de código:

demo_module_1	Ejemplo de datos escalares.
demo_module_2	Ejemplo de una tabla simple.

5.1 Rutina init_module

Cuando se carga un módulo en el agente, el agente llama a la rutina `init_module()` del módulo. La rutina `init_module()` registra los OIDs para los objetos que maneja el módulo. Después de realizarse ese registro, el agente asocia el nombre del módulo con los OIDs registrados. Todos los módulos deben tener esta rutina `init_module()`.

La utilidad *mib2c* genera la rutina `init_module()`. La rutina proporciona el código básico para la obtención de datos, que se debe modificar de forma apropiada en función del tipo de dato a gestionar.

Si en una MIB existen varios nodos MIB, la utilidad *mib2c* crea varios ficheros ".c".

Cada fichero generado contiene una rutina `init_mibnode()`. Un módulo debe tener sólo una rutina de inicialización, que debe ser conforme con la convención de `init_module()`.

De ese modo, cuando exista más de un nodo MIB representado en un módulo, se deben combinar los contenidos de inicialización de todos los ficheros ".c" generados en un solo fichero para asegurar que la rutina de inicialización de cada nodo MIB es llamada por el `init_module()`.

Se pueden combinar ficheros para construir un módulo de varios modos diferentes:

- Crear un fichero del módulo para invocar a todas las rutinas de inicialización.

  Con esta aproximación, la rutina `init_myMib()` en *myMib.c* sería similar al siguiente pseudo código:

  ```
  #include "scalarGroup.h"
  #include "tableGroup.h"
  ...
  init_myMib() {
     init_scalarGroup();
     init_tableGroup();
  }
  ```

  donde `init_scalarGroup()` y `init_tableGroup()` están en ficheros diferentes.

- Combinar el código de las rutinas de inicialización en una rutina de inicialización.

Si se utiliza esta aproximación, la rutina `init_myMib()` podría ser similar al siguiente pseudo código:

```
init_myMib() {
    <init code - scalarGroup> /* found in scalarGroup.c */
    <init code - tableGroup>  /* found in tableGroup.c */
}
```

En ambos casos, el resto del código en *myMib.c* puede ser similar al siguiente pseudo código:

```
/* manejadores get/set para el nodo scalarGroup –
localizados en  scalarGroup.c */
/* manejadores get_first/get_next para el nodo tableGroup
- localizados en tableGroup.c */
```

Las siguientes secciones discuten cómo debe modificarse el código de obtención de datos para los diferentes tipos de datos.

5.2 Objetos Escalares

Los objetos escalares son utilizados para variables correspondientes a nodos hoja del árbol de la MIB que no son parte de una tabla. Si una MIB contiene objetos escalares, se debe ejecutar *mib2c* con un fichero de configuración específico para los nodos de la MIB que contienen los escalares. Se debería utilizar el comando *mib2c*, dónde *mibnode1* y *mibnode2* son objetos de la MIB que contienen nodos de datos escalares para los que se desea generar código, del siguiente modo:

```
$ mib2c -c mib2c.scalar.conf mibnode1 mibnode2
```

Se pueden especificar tantos nodos de datos escalares como se deseen. Este comando genera dos ficheros de código C denominados *mibnode.c* y *mibnode.h* para cada nodo MIB que ha sido especificado en línea de comandos. Se deben modificar los ficheros *mibnode1.c* y *mibnode2.c* para posibilitar al agente obtener datos de objetos escalares. Ver la ayuda de *mib2c* para más información sobre la utilización del comando.

5.2.1 Objetos Escalares en demo_module_1

El código de ejemplo `demo_module_1` se proporciona para ayudar a entender cómo modificar el código generado por el comando *mib2c* para realizar una obtención de datos escalares.

El fichero `README_demo_module_1` contiene instrucciones de cómo realizar las siguientes tareas:

- Generar las plantillas de código desde una MIB que contiene objetos escalares.
- Compilar los ficheros fuente para generar un objeto librería compartido que implementa un módulo.
- Configurar el agente para cargar dinámicamente el módulo.
- Comprobar el módulo con los comandos SNMP (snmpget, snmpgetnext, snmpset, etc.) para mostrar que el módulo funciona tal y como se esperaba.

El módulo `demo_module_1` genera las plantillas de código `me1LoadGroup.c` y `me1LoadGroup.h`. Se puede comparar los ficheros generados con los ficheros `demo_module_1.c` y `demo_module_1.h`. La utilidad *mib2c* genera `me1LoadGroup.c`, que contiene la función `init_me1LoadGroup()`.

Al comparar la función generada con la función `init_demo_module_1()` del fichero `demo_module_1.c` se pueden apreciar los cambios realizados sobre la plantilla.

Los ficheros `demo_module_1.c` y `demo_module_1.h` han sido modificados apropiadamente para obtener los datos escalares. Se pueden utilizar esos ficheros como modelo para aprender cómo trabajar con escalares.

5.2.2 Modificaciones para la Obtención de Objetos Escalares

El código de ejemplo `demo_module_1`, `demo_module_1.c`, proporciona la carga media del sistema para 1, 5 y 15 minutos, respectivamente.

La función `init_module()`define los OIDs para los siguientes objetos escalares:

- `me1SystemLoadAvg1min`
- `me1SystemLoadAvg5min`
- `me1SystemLoadAvg15min`

Esos OIDs se establecen en el fichero fuente `demo_module_1.c`, para reflejar que están en la MIB SDK-DEMO1-MIB.txt. Los OIDs se definen como se indica a continuación:

```
static oid me1SystemLoadAvg1min_oid[] =
{ 1,3,6,1,4,1,4242,1,1,1 };
static oid me1SystemLoadAvg5min_oid[] =
{ 1,3,6,1,4,1,4242,1,1,2 };
static oid me1SystemLoadAvg15min_oid[] =
{ 1,3,6,1,4,1,4242,1,1,3 };
```

El comando *mib2c* utiliza la función `netsnmp_register_scalar()` para registrar las siguientes funciones manejadoras:

- `get_me1SystemLoadAvg1min()`
- `get_me1SystemLoadAvg5min()`
- `get_me1SystemLoadAvg15min()`

De ese modo, cuando se recibe una petición GET o GET_NEXT, se invoca al manejador correspondiente.

Por ejemplo, para la carga media de 1 minuto, se puede registrar de forma manual la función manejadora `get_me1SystemLoadAvg15min()`. El manejador obtiene los datos en el escalar `me1SystemLoadAvg15min`. El manejador se debe situar en la función `netsnmp_register_read_only_instance()` tal y como se muestra a continuación:

```
netsnmp_register_scalar(
     netsnmp_create_handler_registration("me1SystemLoadAvg1min",
          get_me1SystemLoadAvg1min,
          me1SystemLoadAvg1min_oid,
          OID_LENGTH(me1SystemLoadAvg1min_oid),
          HANDLER_CAN_RONLY
```

```
        )
);
```

Alternativamente, se puede utilizar el comando *mib2c* para generar de forma automática los cuerpos de cada una de la funciones manejadoras. Se debe sustituir /* XXX... en el código generado, con el código que recupera el valor del dato para devolverlo como resultado de la petición. Por ejemplo, el siguiente código debe ser modificado:

```
case MODE_GET:
    snmp_set_var_typed_value(requests->requestvb, ASN_OCTET_STR,
         (u_char*) /* XXX: a pointer to the scalar's data */,
         /* XXX: the length of the data in bytes */);
break;
```

Este código debe ser modificado para incluir la propia estructura de dato para devolver datos a las peticiones. En `demo_module_1` se puede observar el siguiente código, resultado de realizar la modificación del código de la plantilla:

```
case MODE_GET:
     data = getLoadAvg(LOADAVG_1MIN);
     snmp_set_var_typed_value(requests->requestvb, ASN_OCTET_STR,
         (u_char*) data , strlen(data));
     free(data);
break;
```

Nótese que el fichero MIB contiene la especificación de una tabla y de escalares.

Cuando se ejecuta *mib2c -c mib2c.scalar.conf scalar-node* se genera sólo el código de plantilla para los nodos escalares de la MIB.

5.3 Tablas Simples

Una tabla simple tiene cuatro características:

1. La tabla se indexa por un único valor entero.
2. Tales índices van desde 1 a un máximo determinado.
3. Todos los índices contenidos en el rango son válidos.
4. Los datos para un índice en particular pueden ser obtenidos directamente, por ejemplo, indexando en una estructura de datos subyacente.

Si alguna de esas características no se verifica, la tabla no se trata de una tabla simple sino de una general.

Las técnicas descritas aquí son aplicables sólo para tablas simples.

Si una MIB contiene tablas simples, se debe ejecutar *mib2c* con un fichero de configuración que maneja la generación de código de tablas simples. Se debería utilizar el siguiente comando, dónde *mibnode1* y *mibnode2* son nodos de datos que contienen tablas sobre las que se desea generar el código:

```
$ mib2c -c mib2c.iterate.conf mibnode1 mibnode2
```

Se pueden especificar tantos nodos de tablas simples como se desee. Este comando genera dos ficheros de código C denominados `mibnode.c` y `mibnode.h` por cada nodo MIB que se especifica en línea de comandos. Se debe modificar los ficheros `mibnode1.c` y `mibnode2.c` para posibilitar al agente obtener datos de tablas simples. Ver la ayuda en línea de comandos de *mib2c* para más información sobre la utilización de la herramienta *mib2c*.

El código de ejemplo `demo_module_2` que se describe a continuación muestra cómo generar plantillas de código para tablas simples.

5.3.1 Tablas Simples en demo_module_2

Se proporciona el código de ejemplo demo_module_2 para ayudar a entender cómo modificar el código generado por el comando mib2c para realizar la obtención de datos de tablas simples.

El fichero `README_demo_module_2` contiene instrucciones que describen cómo realizar las siguientes tareas:

1. Generar plantillas de código para una MIB que contiene una tabla simple.
2. Compilar ficheros fuente para generar un objeto librería compartido que implementa un módulo.
3. Establecer cómo cargar el módulo.
4. Comprobar el módulo con comandos *snmp* para mostrar que el módulo funciona tal como se esperaba

La ejecución de *mib2c* en `demo_module_2` genera como plantillas de código los ficheros `me2FileTable.c` y `me2FileTable.h`. Se puede comparar los ficheros generados con los ficheros `demo_module_2.c` y `demo_module_2.h`.

La utilidad *mib2c* genera el fichero `me2FileTable.c`, que contiene la función `init_me2FileTable()`. Se puede comparar esta función con la función `init_demo_module_2()` del fichero `demo_module_2.c` (por ejemplo, mediante la herramienta *meld*).

5.3.2 Modificaciones para la Obtención de Datos de Tablas Simples

En `demo_module_2.c`, la rutina `init_demo_module_2` invoca la función `initialize_table_me2FileTable()`. La función `initialize_table_me2FileTable()` registra los OIDs para la tabla gestionada por la función. La función también invoca algunas funciones Net-SNMP para inicializar las tablas.

Se deberían proporcionar los datos de tablas en esta función `initialize_table_me2FileTable()` si es necesario. La función `initialize_table_me2FileTable()` realiza las siguientes tareas:

Inicialización — La función `initialize_table_me2FileTable()` realiza la inicialización real de la tabla, realizando tareas tales como el

establecimiento del número máximo de columnas.

Definición del OID de la Tabla

La función `initialize_table_me2FileTable()` define el OID de la tabla:

```
static oid me2FileTable_oid[] = {1,3,6,1,4,1,4242,2,1,1};
```

Definición de la Tabla

La función `initialize_table_me2FileTable()` realiza la definición de la tabla. Esta función especifica otra función a invocar, `me2FileTable_get_first_data_point()`, para procesar la primera fila de datos de la tabla. La función `me2FileTable_get_next_data_point()` es invocada para procesar las restantes filas de la tabla.

```
netsnmp_table_helper_add_indexes(table_info,
     ASN_UNSIGNED, /* index: me2FileIndex */ 0);
table_info->min_column = 1;
table_info->max_column = 4;
/* iterator access routines */
iinfo->get_first_data_point =
     me2FileTable_get_first_data_point;
iinfo->get_next_data_point =
     me2FileTable_get_next_data_point;
iinfo->table_reginfo = table_info;
```

`iinfo` es un puntero a una estructura `netsnmp_iterator_info`.

Registro de la Tabla

La función `initialize_table_me2FileTable()` registra la tabla en el agente:

```
netsnmp_register_table_iterator(my_handler, iinfo);
```

El iterador de la tabla es una función de ayuda que el módulo puede utilizar para indexar filas en la tabla. Funcionalmente, el iterador de la tabla es una versión especializada de un manejador de tablas más genérico. El iterador de una tabla facilita la carga del procesamiento de GETNEXT. El iterador de una tabla recorre todos los índices de datos obtenidos a través de las funciones que proporciona el módulo.

Nótese que los ficheros de entrada MIB contienen las especificación de la tabla y de datos escalares. Sin embargo, cuando se ejecuta *mib2c* con mib2c.iterate.conf y se especifica el nombre del nodo, sólo se genera código para la tabla simple.

5.3.3 Procesado de Múltiples SET en demo_module_2

El código de ejemplo `demo_module_2` muestra como realizar una operación de modificación sobre múltiples OIDs. Es este caso, se proporciona un nombre de fichero y una fila de estado.

Cuando el agente procesa una petición SET, se realizan una serie de llamadas al módulo de la MIB. Esas llamadas aseguran que todas peticiones de SET en el paquete entrante pueden ser procesadas de forma satisfactoria. Este procesado permite a los módulos terminar anticipadamente la secuencia de una transacción. Si un módulo aborta una transacción anticipadamente, no se completan ninguna de las transacciones del *set*, para mantener la atomicidad. Sin embargo, este comportamiento incrementa la complejidad del código de procesado de peticiones SET. El siguiente diagrama es un simple diagrama de estados que muestra los pasos del procesado de una operación SET en el agente.

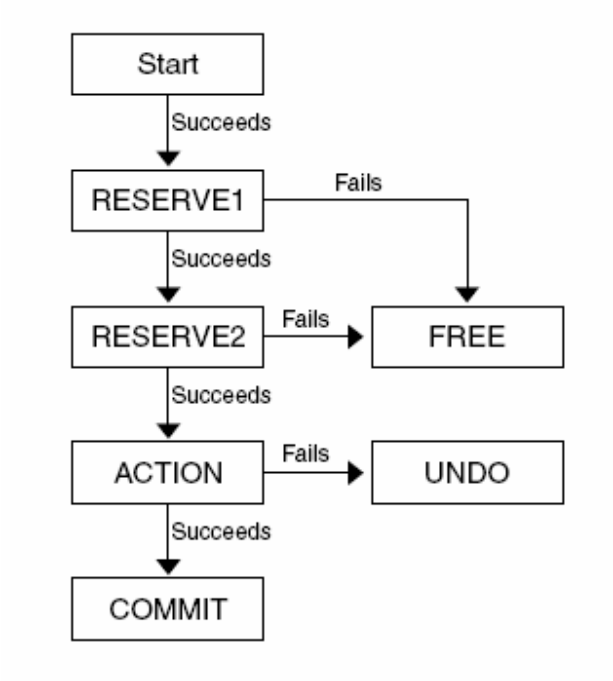

Figura 1 Diagrama de estados del Procesamiento de un SET

En la figura anterior, una operación sin fallo vendría representada por el camino vertical de la izquierda del diagrama. Si cualquiera de los módulos MIB que están involucrados devuelve un error, el agente se ramifica en uno de los estados de fallo. Los estados de fallo se corresponden con el lado derecho de la figura. Esos estados de fallo cuando sea necesario, requieren el limpiado para deshacer las acciones que se hayan realizado dentro del módulo en pasos anteriores.

En el código de ejemplo `demo_module_2` la función `me2FileTable_handler()` muestra como realizar peticiones SET en diferentes estados. La siguiente lista describe cada uno de los estados:

`MODE_SET_RESERVE1` Chequea que el valor a establecer es aceptable.

`MODE_SET_RESERVE2` Reserva cualquier recurso necesario. Por ejemplo, en el ejemplo se llama a la función *malloc()*.

`MODE_SET_FREE` Libera los recursos cuando uno de los valores a establecer

falla por alguna razón.

`MODE_SET_ACTION` Establece la variable tal y como se ha solicitado y almacena la información que pueda ser necesaria para invertir la operación de SET con posterioridad.

`MODE_SET_COMMIT` La operación es exitosa. Se descarta la información almacenada y se realiza el cambio de forma permanente. Por ejemplo, se escribe en el fichero *snmpd.conf* y se libera cualquier recurso reservado.

`MODE_SET_UNDO` Se ha producido un fallo, de ese modo se resetea la variable a su anterior valor. Se libera cualquier recurso almacenado.

Se puede realizar la operación de ser utilizando alguno de los siguientes comandos cuando se utiliza el ejemplo `demo_module_2`:

```
$ snmpset -v1 -c private localhost:1161 me2FileTable.1.2.3 s "test"
$ snmpset -v1 -c private localhost:1161
.1.3.6.1.4.1.4242.2.1.1.1.2.3 s "test"
```

Estos comandos cambian el fichero a monitorizar especificado como valor de la segunda columna en la fila de la tabla `me2FileTable` que tiene como índice el valor `3`.

6 Almacenamiento de Datos de un Módulo

Este capítulo describe cómo un módulo puede almacenar datos que persisten cuando se reinicializa el agente.

6.1 Acerca del Almacenamiento de Datos en un Módulo

Puede ser necesario que un módulo almacene datos de forma persistente. Los datos persistentes es información que almacena el módulo en un fichero y lee de ese fichero mediante propiedades de configuración.

Los datos son preservados ante reinicios del agente.

Los módulos pueden almacenar *tokens* con valores asignados en ficheros de configuración específicos de un módulo.

Un fichero de configuración se crea de forma manual. Un módulo puede escribir los tokens a un fichero y/o leerlos de un fichero. El módulo registra manejadores que se asocian con los *tokens* de configuración de un módulo específico.

6.1.1 Ficheros de configuración

La ayuda en línea de *snmp_config* discute en general los ficheros de configuración SNMP. Las páginas de ayuda documentan las localizaciones dónde se pueden almacenar dichos ficheros. Esas localizaciones coinciden con las rutas de búsqueda por defecto para los ficheros de configuración SNMP.

La mejor localización para los ficheros de configuración es en el directorio `$HOME/.snmp`, el cual se encuentra como una de la localizaciones de búsqueda por defecto. También se puede establecer la variable de entorno `$SNMPCONFPATH` si se quiere utilizar una ruta que no sea por defecto, para localizar ficheros de configuración.

Cuando se crea un fichero de configuración, se debe nombrar al fichero como *modulo.conf* o *modulo.local.conf*. Se debe situar el fichero en uno de los directorios de los configurados en la ruta de búsqueda de ficheros de configuración.

6.1.2 Definición de *Tokens* de Configuración

Los *tokens* (o nombres) de configuración son utilizados por módulos para obtener datos persistentes durante el tiempo de ejecución.

Cuando un módulo utiliza *tokens* de configuración propios, se deberían crear uno o más ficheros de configuración propios para el módulo. Se podría también crear un fichero de configuración para varios módulos relacionados. Y se pueden definir nuevos *tokens* en el fichero de configuración.

Los *tokens* deben utilizar el mismo formato que las directivas en *snmpd.conf*. Un *token* se define en cada línea del fichero de configuración. Los tokens de configuración son escritos de la forma:

```
Token Value
```

Por ejemplo, un *token* podría ser:

```
my_token 4
```

Los módulos no deberían definir *tokens* nuevos en el fichero SNMP, *snmpd.conf*. Si un módulo almacena *tokens* en ese fichero, pueden producirse colisiones en el espacio de nombres.

6.2 Datos Persistentes en un Módulo

El módulo puede registrar manejadores que están asociados con *tokens* en un fichero de configuración específico con la función *register_config_handler()*. Los manejadores pueden ser utilizados más tarde en un módulo para una tarea específica.

La función `register_config_handler()` se define del siguiente modo:

```
register_config_handler (const char *type_param, const char *token,
      void(*parser)(const char *, char *),
      void(*releaser)(void), const char *help)
```

El primer argumento designa el nombre base del fichero de configuración, el cual debería ser el mismo que el del nombre del módulo. Por ejemplo, si el primer argumento es *my_custom_module*, entonces el agente observa los tokens del fichero de configuración del fichero *my_custom_module.conf*. Nótese que es necesario crear el fichero de configuración manualmente antes de que el módulo pueda hacer uso del fichero.

El segundo argumento designa el *token* de configuración que el módulo utiliza. El tercero, indica el nombre de la función manejadora.

6.2.1 Almacenamiento de Datos Persistentes

El módulo debe utilizar las funciones `read_config_store()` junto con las funciones *callback* para almacenar los datos.

El módulo debe primero registrar una función *callback* con la función `snmp_register_callback()` para que los datos sean escritos al fichero de configuración cuando el agente finaliza. La función `snmp_register_callback()` es como se indica a continuación:

```
int snmp_register_callback(int major, int minor,
        SNMPCallback *new_callback, void *arg);
```

Se debe especificar como *major* el valor `SNMP_CALLBACK_LIBRARY`, como *minor* el valor `SNMP_CALLBACK_STORE_DATA`. Cuando *arg* no está establecido a NULL, *arg* es un puntero a *void* utilizado cuando la función *new_callback* sea ejecutada.

El prototipo para la función *callback*, el puntero *new_callback*, es como se indica a continuación:

```
int (SNMPCallback) (int majorID, int minorID, void *serverarg,
                    void *clientarg);
```

La `función read_config_store()` realiza el almacenamiento cuando finaliza el agente, si se utilizan las *callbacks* registradas.

Nota – Cuando un módulo almacena datos persistentes, los ficheros de configuración se almacenarán en el directorio `/var/net-snmp`. Los pares token-valor modificados son añadidos al fichero, en lugar de sobrescribir los pares token-valor anteriores del fichero. Los últimos valores definidos en el fichero son los valores utilizados.

6.2.2 Lectura de Datos Persistentes

Los datos se leen desde un fichero de configuración del módulo por medio de la utilización de la función `register_config_handler()`. Por ejemplo, se puede invocar a la función del siguiente modo:

```
register_config_handler("my_module", "some_token",
       load_my_tokens, NULL, NULL);
```

Siempre que el token *some_token* sea leído por el agente en el fichero *my_module.conf*, se invocará a la función `load_my_tokens()` con el nombre del token y el valor como argumentos.

6.3 Almacenamiento de Datos Persistentes en demo_module_4

El código de ejemplo `demo_module_4` demuestra la persistencia de datos ante el reinicio del agente. La demo está localizada por defecto en el directorio `demo_module_4`.

Este módulo implementa SDK-DEMO4-MIB.txt. Las plantillas `demo_module_4.c` y `demo_module_4.h` son el resultado de renombrar las plantillas originales `me4FileGroup.c` y `me4FileGroup.h` que fueron generadas por el comando *mib2c*. El nombre de la función de inicialización `init_me4FileGroup` se cambió a `init_demo_module_4`.

Ver el fichero `README_demo_module_4` en el directorio `demo_module_4` para configurar y ejecutar la demo.

6.3.1 Almacenamiento de Datos Persistentes en demo_module_4

Este ejemplo almacena los datos de configuración en el fichero `/var/net-snmp/<login>_demo_module_4.conf` dónde `<login>` es el nombre del usuario que lanza el agente (cuyo valor se especifica en la variable de entorno `$USER`).

En `demo_module_4.c`, las siguientes sentencias registran la función *callback*. La función *callback* se invoca siempre y cuando el agente observa que el módulo de datos necesita ser almacenado, como es el caso de la terminación normal del agente.

```
snmp_register_callback(SNMP_CALLBACK_LIBRARY,
      SNMP_CALLBACK_STORE_DATA, demo4_persist_data, NULL);
```

La función `demo4_persist_data()` utiliza `read_store_config()` para almacenar los datos:

```
int demo4_persist_data(int a, int b, void *c, void *d) {
      char filebuf[300];
      sprintf(filebuf, "%s %s", DEMO4_FILE1, file1);
      read_config_store(DEMO4_CONF_FILE, filebuf);
      sprintf(filebuf, "%s %s", DEMO4_FILE2, file2);
      read_config_store(DEMO4_CONF_FILE, filebuf);
```

```
        sprintf(filebuf, "%s %s", DEMO4_FILE3, file3);
        read_config_store(DEMO4_CONF_FILE, filebuf);
        sprintf(filebuf, "%s %s", DEMO4_FILE4, file4);
        read_config_store(DEMO4_CONF_FILE, filebuf);
}
```

En demo_module_4, puede añadirse un nuevo fichero para la monitorización, utilizando el comando *snmpset*. La fase *commit* de una petición *snmpset* utiliza la función read_config_store () para almacenar la información de fichero:

```
case MODE_SET_COMMIT:
        /*
        * Everything worked, so we can discard any saved information,
        * and make the change permanent (e.g. write to the config
        * file).
        * We also free any allocated resources.
        */
        /* Persist the file information */
        snprintf(&filebuf[0], FILENAME_MAX, "%s%d %s",
              DEMO4_FILE, data->findex, data->fileName);
        read_config_store(DEMO4_CONF_FILE, &filebuf[0]);
        /*
        * The netsnmp_free_list_data should take care of the
        * allocated resources
        */
```

El dato persistente es almacenado en el fichero /var/net-snmp/<login>_demo_module_4.conf.

6.3.2 Lectura de Datos Persistentes en demo_module_4

En un módulo, los datos se leen de ficheros de configuración por medio del registro de una función *callback* a invocar cuando se localiza un token relevante. Por ejemplo, se puede invocar a la función tal y como se indica a continuación:

```
register_config_handler(DEMO4_CONF_FILE, DEMO4_FILE1,
      demo4_load_tokens, NULL, NULL);
```

Siempre que sea leído por el agente el token demo4_file1 en el fichero de configuración *demo_module_4.conf*, se invocará a la función demo4_load_tokens() con el nombre del token y el valor como argumentos. La función demo4_load_tokens() almacena el valor del token en las variables apropiadas:

```
void demo4_load_tokens(const char *token, char *cptr) {
    if (strcmp(token, DEMO4_FILE1) == 0) {
        strcpy(file1, cptr);
    } else if (strcmp(token, DEMO4_FILE2) == 0) {
        strcpy(file2, cptr);
    } else if (strcmp(token, DEMO4_FILE3) == 0) {
        strcpy(file3, cptr);
    } else if (strcmp(token, DEMO4_FILE4) == 0) {
        strcpy(file4, cptr);
    } else {
        /* Do Nothing */
    }
    return;
```

```
}
```

6.3.3 Utilización de SNMP_CALLBACK_POST_READ_CONFIG en demo_module_4

Entre el arranque del agente y la lectura de todos los tokens de configuración son leídos por el módulos transcurren unos segundos. Durante ese intervalo, el módulo podría realizar ciertas funciones. Por ejemplo, hasta que los nombres de los ficheros persistentes son leídos por el módulo desde `/var/net-snmp/<login>_demo_module_4.conf`, la tabla de ficheros no puede ser publicada. Para manejar esos casos, puede establecerse una función *callback*. Esta función *callback* es llamada cuando se finaliza el proceso de lectura de los ficheros de configuración. Por ejemplo, se podrían invocar a las funciones como se indica a continuación:

```
snmp_register_callback(SNMP_CALLBACK_LIBRARY,
       SNMP_CALLBACK_POST_READ_CONFIG,
       demo_4_post_read_config, NULL);
```

La función `demo_4_post_read_config()` es invocada después de que hayan sido leídos los ficheros de configuración. En este ejemplo, la función `demo_4_post_read_config()` publica la tabla de ficheros, entonces registra de las funciones *callback* para los datos persistentes.

```
int demo4_post_read_config(int a, int b, void *c, void *d) {
    if (!AddItem(file1))
         snmp_log(LOG_ERR, "Failed to add instance in
init_demo_module_4\n");
    if (!AddItem(file2))
         snmp_log(LOG_ERR, "Failed to add instance in
init_demo_module_4\n");
    if (!AddItem(file3))
         snmp_log(LOG_ERR, "Failed to add instance in
init_demo_module_4\n");
    if (!AddItem(file4))
         snmp_log(LOG_ERR, "Failed to add instance in
init_demo_module_4\n");

    snmp_register_callback(SNMP_CALLBACK_LIBRARY,
      SNMP_CALLBACK_STORE_DATA,
      demo4_persist_data, NULL);
}
```

7 Implementación de Alarmas

Este capítulo explica como implementar alarmas en módulos. Se utiliza la `demo_module_3` para ilustrar dichas técnicas.

7.1 Invervalos de Refresco

Los intervalos de refresco, también conocidos como refresco automático, pueden implementarse en un agente. Se utiliza un mecanismo de *callback* que invoca una función específica en intervalos regulares. El refresco de datos puede ser implementado por la función `snmp_alarm_register()`. En `demo_module_3`, la carga de datos se refresca en un intervalo de tiempo configurable, 60 segundos en este ejemplo, utilizando la siguiente *callback*:

```
snmp_alarm_register(60, SA_REPEAT, refreshLoadAvg, NULL);
void refreshLoadAvg(unsigned int clientreg, void *clientarg) {
     // Refresh the load data here
}
```

La función `snmp_alarm_register()` puede ser incluida en la función `init_module()` de modo que el intervalo de refresco se establece durante la inicialización del módulo.

7.2 Notificación de Traps Asíncronas

Generalmente, el chequeo de condiciones de *trap* se realiza mediante la siguiente secuencia:

1. Obtiene los datos actuales para un nodo en particular.

2. Compara los datos con un umbral para chequear si la condición de *trap* se verifica.

3. Envía un *trap* al gestor si la condición se verifica

Los pasos 2 y 3 son implementados en el agente invocando a un algoritmo después de obtener los datos de un nodo. El algoritmo determina si se verifica una condición de alarma. El algoritmo en la mayoría de los casos compara el valor actual de un dato con un umbral. Si el algoritmo indica que se verifica una condición de alarma, se invocan a las funciones de *trap* apropiadas para generar un *trap*. En `demo_module_3`, los pasos 2 y 3 son realizados en la siguiente función:

```
void refreshLoadAvg(unsigned int clientreg, void *clientarg) {
      // Refresh Load data
      // Check if Load data crossed thresholds, send trap if
      // necessary.
      check_loadavg1_state();
      check_loadavg5_state();
      check_loadavg15_state();
}
```

La funciones `check_loadavg_state()` comparan los datos de carga actual con los umbrales. Las funciones también envían los traps si es necesario.

El módulo debe utilizar una función *trap* tal como `send_enterprise_trap_vars()` para enviar un *trap* al gestor. Las notificaciones de *traps* SNMP son definidas en SNMP-NOTIFICATION-MIB.txt. Para `demo_module_3`, las notificaciones de *trap* son definidas en SDK-DEMO3-MIB.txt.

7.3 Umbrales para el Envío de Traps

En el agente, cualquier dato configurable puede ser almacenado en un fichero de configuración específico del módulo. Los datos desde este fichero pueden ser cargados en el módulo durante la inicialización del módulo. En un módulo los datos se leen en los ficheros de configuración por medio del registro de una función *callback* a invocar siempre y cuando se encuentre un *token* de interés.

```
register_config_handler("demo_module_3", "threshold_loadavg1",
      read_load_thresholds, NULL, NULL);
```

En este ejemplo `demo_module_3`, siempre que el agente lee un token `threshold_loadavg1` en el fichero de configuración `demo_module_3.conf`, se invoca la función `read_load_thresholds()`, con el nombre del *token* y el valor como argumento. La función `read_load_thresholds()` almacena el valor del *token* en variables apropiadas y utiliza esos umbrales para determinar condiciones de alarmas.

El código de ejemplo `demo_module_3` se proporciona para ayudar a implementar alarmas. La demo está localizada por defecto en el directorio `demo_module_3`. El fichero `README_demo_module_3` contiene instrucciones que describen cómo realizar las siguientes tareas:

1. Compilar los ficheros fuentes para generar un objeto librería compartido que implementa un módulo.
2. Establecer que el agente cargue dinámicamente el módulo.
3. Verificar el módulo con los comandos snmp para mostrar que el módulo funciona tal y como se esperaba.

El módulo `demo_module_3` implementa SDK-DEMO3-MIB.txt. Los ficheros `me3LoadGroup.c` y `me3LoadGroup.h` fueron generados con la herramienta *mib2c* y posteriormente modificados.

Los datos del módulo son mantenidos en las siguientes variables:

`loadavg1`	Almacena el dato para *me3SystemLoadAvg1min*
`loadavg5`	Almacena el dato para *me3SystemLoadAvg5min*
`loadavg15`	Almacena el dato para *me3SystemLoadAvg15min*

El módulo `demo_module_3` refresca los datos cada 60 segundos. Durando los intervalos de refresco, el módulo también comprueba si las condiciones de *trap* se verifican. Si se verifican las condiciones de *trap*, el módulo genera una trap SNMPv1. La condición de *trap* en este módulo es una simple comparación de los datos actuales con el valor del umbral. Si

se sobrepasa el umbral, se genera un *trap*. El umbral de datos puede ser configurado por medio del fichero `demo_module_3.conf`, situado en `$HOME/.snmp`.

Cuando se recibe una petición *snmpget* sobre esas variables, se invocan las siguientes funciones:

```
int get_me4SystemLoadAvg1min()
int get_me4SystemLoadAvg5min()
int get_me4SystemLoadAvg15min()
```

Estas funciones refrescan los datos relativos a la carga a través de la función `refreshLoadAvg()` que obtienen el valor de carga actual. Sin embargo, la recarga ocurre sólo en respuesta a peticiones GET. La carga de datos también debe realizarse de forma asíncrona sin tener que esperar por la llegada de peticiones GET desde el gestor. El refresco asíncrono permite comprobar la verificación de condiciones de forma continuada para alertar al agente de cualquier problema.

Se puede refrescar los datos de carga sin la existencia de una petición de un gestor, registrando una función *callback* para que se invoque en intervalos regulares. Por ejemplo, se puede invocar a la siguiente función:

```
snmp_alarm_register(60, SA_REPEAT, refreshLoadAvg, NULL)
```

Esta función provoca la invocación de la función `refreshLoadAvg()` cada 60 segundos.

Se puede permitir al gestor configurar este intervalo introduciendo un *token* para representar este valor en el fichero *demo_module_3.conf*.

7.3.1 Lectura de Datos de Fichero de Configuración en demo_module_3.conf

En un módulo, se leen los datos desde ficheros de configuración mediante el registro de funciones *callback*, para que sean invocadas siempre que se localice un *token* apropiado. Por ejemplo, se puede invocar la función del siguiente modo:

```
register_config_handler("demo_module_3", "threshold_loadavg1",
      read_load_thresholds, NULL, NULL);
```

Siempre que el agente lea un token *threshold_loadavg1* en el fichero *demo_module_3*, se invocará a la función `read_load_thresholds()` con el nombre del token y el valor como argumento. La función `read_load_thresholds()` almacena el valor del *token* en las variables apropiadas:

```
void read_load_thresholds(const char *token, char *cptr) {
    if (strcmp(token, "threshold_loadavg1") == 0) {
         threshold_loadavg1=atof(cptr);
    } else if (strcmp(token,"threshold_loadavg5") == 0) {
         threshold_loadavg5=atof(cptr);
    } else if (strcmp(token,"threshold_loadavg15") == 0) {
         threshold_loadavg15=atof(cptr);
    } else {
         /* Do nothing */
    }
    return;
```

```
}
```

7.3.2 Utilización de SNMP_CALLBACK_POST_READ_CONFIG en demo_module_3

Entre el arranque del agente y la lectura en el módulo de todos los *tokens* de configuración transcurren unos segundos. Durante ese intervalo, el módulo podría realizar ciertas funciones. Por ejemplo, hasta que en el módulo no haya leído la configuración de los umbrales, la condición de trap no puede ser chequeada. Para manejar esos casos, puede establecerse una función *callback*. Esta función *callback* es llamada cuando se finaliza el proceso de lectura de los ficheros de configuración. Por ejemplo, se podrían invocar a las funciones como se indica a continuación:

```
snmp_register_callback(SNMP_CALLBACK_LIBRARY,
    SNMP_CALLBACK_POST_READ_CONFIG, demo_3_post_read_config, NULL);
```

La función `demo_3_post_read_config()` es invocada después de que hayan sido leídos los ficheros de configuración. En este ejemplo, la función `demo_3_post_read_config()` registra las funciones *callback* de refresco:

```
int demo_3_post_read_config(int a, int b, void *c, void *d) {
    /* Refresh the load data every 60 seconds */
    snmp_alarm_register(60, SA_REPEAT, refreshLoadAvg, NULL);
    /* Acquire the data first time */
    refreshLoadAvg(0,NULL);
}
```

7.3.3 Generación de Traps en demo_module_3

La función `refreshLoadAvg()` se invoca en intervalos regulares para refrescar los datos.

Inmediatamente después de refrescarse los datos, la función `refreshLoadAvg()` comprueba si se verifican las condiciones de trap invocando a las siguientes funciones:

- `check_loadavg1_state()`
- `check_loadavg5_state()`
- `check_loadavg15_state()`

En `me3LoadGroup.c`, un módulo propiamente podría estar en uno de los siguientes estados: OK o ERROR.

Cuando los valores de los datos actuales sobrepasan el umbral, el estado se establece a ERROR. En ese caso, se genera un trap. Las funciones de chequeo siguen el siguiente algoritmo:

```
check_loadavg1_state() {
      // Step-1: check condition
      if (currentLoad > threshold_loadavg1) {
            new_loadavg1_state = ERROR;
      }
      // Step-2: Generate trap if necessary
      if (new_loadavg1_state != prev_loadavg1_state) {
          /* Send ERROR/OK trap */
          prev_loadavg1_state = new_loadavg1_state;
```

```
            send_trap(trapoid, size, status, description);
        } else if (new_loadavg15_state == prev_loadavg15_state) {
            /* No Change in state .. Do nothing */
        }
    }
```

Cuando la comprobación indica que el umbral ha sido sobrepasado, se utiliza la función `send_enterprise_trap_vars()` para generar un *trap*. El OID del *trap* y los *varbinds* se especifican en el fichero MIB denominado SDK-DEMO3-MIB.txt.

8 Desarrollo de Módulos

Este capítulo discute los modos de desarrollar un módulo. Este capítulo proporciona información para ayudar a decidir si utilizar un subagente o un módulo cargado dinámicamente. Se incluyen módulos de demostración para el despliegue como subagente y como módulos cargados dinámicamente.

8.1 Despliegue de un Módulo

En Net-SNMP, existen varias posibilidades de desplegar un módulo dentro de un agente. Este documento describe el despliegue mediante la carga de un módulo dinámicamente.

Cuando se carga un módulo dinámicamente, el módulo se incluye dentro del agente SNMP sin necesidad de recompilar y *linkar* de nuevo el código binario del agente. Este método es el único modo soportado para cargar un módulo en el agente. No se puede recompilar el agente.

Los detalles de la carga de un módulo se especifican en el fichero de configuración. En tiempo de ejecución, el agente lee el fichero de configuración. El agente localiza los ficheros del módulo que se listan en el fichero de configuración. El agente posteriormente combina los módulos en la imagen del proceso agente.

8.2 Carga Dinámica de Módulos

El modo más simple de cargar módulos dinámicamente es reiniciando el agente después de añadir entradas en el fichero de configuración. La carga dinámica es el mejor método a utilizar mientras se está desarrollando y validando un módulo. La mayoría de los módulos de demostración usan carga dinámica. Se podría utilizar durante la fase de desarrollo y validación, el procedimiento descrito en el Apartado “8.3 Carga Dinámica de un Módulo con Reinicio del Agente” de la página 33.

Cuando se utiliza el módulo en un entorno de producción, ese entorno podría requerir que no se reinicie el agente. Si se quiere cargar módulos sin reiniciar el agente, se puede utilizar el procedimiento descrito en el Apartado “8.4 Carga Dinámica de un Módulo sin Reinicio del Agente” de la página 34.

8.3 Carga Dinámica de un Módulo con Reinicio del Agente

1. Copiar la librería compartida del módulo a un directorio de librerías. Se recomienda mantener los ficheros .so en un directorio con permisos de escritura para usuarios no *root*.

2. Editar el fichero de configuración del agente para posibilitar al agente cargar dinámicamente el módulo. En el fichero *snmpd.conf*, se añade un línea que es similar a lo siguiente, dónde *testmodule* es el nombre del módulo y `/home/username/.snmp/lib/testmodule.so` la ruta absoluta del fichero objeto compartido del módulo.

   ```
   dlmod testmodule /home/username/.snmp/lib/testmodule.so
   ```

3. Reiniciar el agente *snmpd*.

Tras realizar los pasos anteriores, el módulo ya debería estar cargado. Se pueden utilizar los comandos *snmpget* y *snmpset* para acceder a los datos del módulo y de ese modo confirmar que el módulo se ha cargado correctamente. Para que los comandos *snmpget* y *snmpset* localicen la MIB, es necesario configurar por un lado la variable de entorno `MIBS`, y por otro, localizar los ficheros en el directorio `$HOME/.snmp/mibs` o configurar la variable de entorno `$MIBDIRS`, tal y como se describe en el Apartado "4.4 Variables de Entorno MIB" en la página 11.

Para descargar un módulo, se realiza el proceso inverso; se debe eliminar la línea *dlmod* del fichero *snmpd.conf* y reiniciar el agente.

8.4 Carga Dinámica de un Módulo sin Reinicio del Agente

La MIB `UCD-DLMOD-MIB::dlmodTable` proporciona entradas con el nombre, ruta y estado de módulos.

Con el establecimiento de las entradas en esa tabla de la MIB `UCD-DLMOD-MIB`, se puede provocar que el agente cargue o descargue un módulo sin necesidad de reiniciar el agente.

Este procedimiento provoca que el módulo se cargue sólo para la versión actual del agente. Si se quiere que el módulo se cargue cada vez que se reinicie el agente, se debería añadir una línea *dlmod* al fichero *snmpd.conf*. El proceso de añadir una línea se incluye en el procedimiento descrito en el apartado anterior.