Проектируем SNMP управление.

Ред. 1.05
от 08.12.2012г.

Оглавление

Общие положения

Стандарт SNMP (**S**imple **N**etwork **M**anagment **P**rotocol) создан для диагностики, контроля, управления и мониторинга **любых достижимых** в сети устройств и программных компонент (это подчёркивает **N**etwork в названии стандарта и протокола). К поддерживающим SNMP устройствам относятся любые сетевые устройства: маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки... Нас будет интересовать только утилитарное применение SNMP в своих разработках, поэтому относительно общих характеристик протокола ограничимся только самым поверхностным описанием («на пальцах») тех его черт, которые нужны в этих целях.

Нужны некоторые уточнения относительно «достижимых в сети». SNMP работает над транспортным протоколом UDP/IP. SNMP будет успешно работать с теми сетевыми хостами, которые доступны транспортному протоколу, например:

- хосты **собственной** локальной сети (LAN) будут доступны;
- если для доступа во внешнюю сеть (WAN) администратор LAN или провайдер связи в WAN используют NAT-трансляцию IP (например с помощью `iptables`), то WAN будет доступна;
- если администратор LAN настроил файервол так, что для UDP портов 161 и 162 он непрозрачен, то WAN становится недоступной;
- если хосты некоторой **другой** LAN с локальными IP (10.X.X.X, 172.X.X.X, 192.168.X.X) выходят в WAN с помощью NAT-трансляции IP, то хосты такой LAN будут недоступны через WAN.

В модели управления протоколом SNMP у нас всегда будет представлено три компонента:

- Управляемое **устройство** (аппаратное или программное);
- **Агент** (то, что в других сетевых системах называется **сервер**) SNMP — программный модуль сетевого управления, располагающийся на управляемом устройстве, обладающий локальным знанием управляющей информации устройства, и переводящий эту информацию в специфичную для SNMP форму или из неё (медиация данных). ;
- **Менеджер** (то, что в других сетевых системах называется **клиент**) — программное обеспечение, запрашивающее информацию от агента, или через агента управляющее устройством.

Агент SNMP получает запросы по UDP-порту, по умолчанию это порт 161, но разработчик может изменить его в своей системе. Менеджер может посылать запросы с любого доступного порта источника на порт агента. Ответ агента будет отправлен назад на порт источника на менеджере. Менеджер получает уведомления от агента (Traps и InformRequests) на порту 162 (по умолчанию).

Сам по себе SNMP не определяет, какая информация и переменные должны быть предложены управляемой системой. Вместо этого SNMP использует расширяемую разработку, в которой доступная информация определяется базами управляющей информации (Managment Information Base, MIB). Базы MIB описывают структуру управляемых данных, они используют иерархическое пространство имен, содержащее идентификаторы объектов (Object ID, OID). Каждый OID определяет переменную, которая может быть считана либо установлена с помощью SNMP. В устройстве может быть определено много **объектов** SNMP. Объекты **конкретного проекта** описываются в так называемых MIB-файлах, строго определённого синтаксиса, становящиеся составной частью общей базы MIB. Объектами могут быть **переменные** и **уведомления**. Переменные могут быть двух типов: **скалярные** (одиночные) переменные и **таблицы**. Значения переменных могут считываться или записываться **агентом по запросам менеджера**.

Скалярная переменная характеризуется своим отдельным OID (можно грубо считать это как имя переменной), **типом** и, естественно, имеет **значение**. Например: `.1.3.6.1.4.1.9876.11.5` — это может быть OID скалярной (отдельной) переменной. Типом переменной (определяющим набор допустимых значений для переменной) могут быть (их весьма много), например: `INTEGER`, `Integer32`, `DisplayString` , `IpAddress`, `...` Могут быть определены переменные ограниченного типа: `INTEGER (1..30)`. Может быть описаны скалярные переменные структурного типа (`SEQUENCE`).

Таблица — это набор (массив) **однотипных** скалярных переменных, имеющая единый общий OID. Таблицы имеют динамическую размерность. Каждому элементу таблицы сопоставляется свой OID, образующийся из OID таблицы. Например, `.1.3.6.1.4.1.9876.11.15` — это может быть OID табличной переменной, а входящие в таблицу скалярные переменные будут иметь OID: `.1.3.6.1.4.1.9876.11.15.1`, `.1.3.6.1.4.1.9876.11.15.2`, `.1.3.6.1.4.1.9876.11.15.3`, таким образом это таблица из 3-х элементов.

Уведомления посылаются агентом менеджеру асинхронно, то есть здесь инициатива исходит не от менеджера (считать или записать), а от агента. Уведомления могут быть 2-х типов: Traps и InformRequests. Поскольку это UDP, агент отсылая Traps не знает (и не имеет возможности получить информацию): ожидает ли кто его уведомление и будет ли оно получено. Уведомление InformRequests было введено позже и предусматривает возврат подтверждения (получатель отправляет подтверждение, повторяющее всю информацию из InfromRequest).

Понятно, что всё множество OID-ов представляет собой единое **дерево** — например, `.3.2.7...` :

```
1   2   3   4  ...
      ↓
      1  2  3  ...
               ↓
         1   2   3   4   5  6  7 ...
                              ↓
                              ...
```

В такой схеме каждый OID (в мире!) совершенно уникален и занимает своё собственное место в иерархии базы MIB. Программы агентов SNMP работают **исключительно** с такой числовой формой представления OID (поэтому и Simple в названии протокола). Но человеку крайне неудобно работать с такой формой информации. Поэтому уже упоминавшиеся MIB-файлы вводят символьные синонимические имена для OID, в которых их будут представлять человеку утилиты SNMP. Например, `.1.3.6.1.4.1.9876.11.5` может соответствовать синоним `.iso.org.dod.internet.private.enterprises.olej.management.currentValue` (но такой символьное синонимическое представление возможно только тогда, когда соответствующие MIB-файлы присутствуют в системе, и доступны утилитам SNMP по путям поиска).

Вопрос: а откуда берутся идентификаторы в старшей (корневой) части дерева? Они описаны в MIB-файлах, находящихся в каталоге `/usr/share/snmp/mibs`. Этот каталог, совместно с `$HOME/.snmp/mibs` (зависящим от имени пользователя от которого выполняется!), считаются каталогами поиска MIB по умолчанию (могут быть добавлены или переопределены опциями утилит SNMP):

```
$ ls -w100 /usr/share/snmp/mibs
AGENTX-MIB.txt                      IPV6-TC.txt               SNMP-NOTIFICATION-MIB.txt
BRIDGE-MIB.txt                      IPV6-UDP-MIB.txt          SNMP-PROXY-MIB.txt
DISMAN-EVENT-MIB.txt                LM-SENSORS-MIB.txt        SNMP-TARGET-MIB.txt
DISMAN-SCHEDULE-MIB.txt             MTA-MIB.txt               SNMP-USER-BASED-SM-MIB.txt
DISMAN-SCRIPT-MIB.txt               NET-SNMP-AGENT-MIB.txt    SNMP-USM-AES-MIB.txt
EtherLike-MIB.txt                   NET-SNMP-EXAMPLES-MIB.txt SNMP-USM-DH-OBJECTS-MIB.txt
HCNUM-TC.txt                        NET-SNMP-EXTEND-MIB.txt   SNMPv2-CONF.txt
HOST-RESOURCES-MIB.txt              NET-SNMP-MIB.txt           SNMPv2-MIB.txt
HOST-RESOURCES-TYPES.txt            NET-SNMP-PASS-MIB.txt      SNMPv2-SMI.txt
IANA-ADDRESS-FAMILY-NUMBERS-MIB.txt NET-SNMP-TC.txt            SNMPv2-TC.txt
IANAifType-MIB.txt                  NET-SNMP-VACM-MIB.txt      SNMPv2-TM.txt
IANA-LANGUAGE-MIB.txt               NETWORK-SERVICES-MIB.txt  SNMP-VIEW-BASED-ACM-MIB.txt
IANA-RTPROTO-MIB.txt                NOTIFICATION-LOG-MIB.txt  TCP-MIB.txt
IF-INVERTED-STACK-MIB.txt           RFC1155-SMI.txt           TRANSPORT-ADDRESS-MIB.txt
IF-MIB.txt                          RFC1213-MIB.txt           TUNNEL-MIB.txt
INET-ADDRESS-MIB.txt                RFC-1215.txt              UCD-DEMO-MIB.txt
IP-FORWARD-MIB.txt                  RMON-MIB.txt              UCD-DISKIO-MIB.txt
IP-MIB.txt                          SCTP-MIB.txt              UCD-DLMOD-MIB.txt
IPV6-FLOW-LABEL-MIB.txt              SMUX-MIB.txt             UCD-IPFWACC-MIB.txt
IPV6-ICMP-MIB.txt                   SNMP-COMMUNITY-MIB.txt    UCD-SNMP-MIB.txt
IPV6-MIB.txt                        SNMP-FRAMEWORK-MIB.txt    UDP-MIB.txt
IPV6-TCP-MIB.txt                    SNMP-MPD-MIB.txt
```

Это определения основных системных OID. Поддержка описанных там объектов компилирована в коде стандартного SNMP агента `snmpd`, ним осуществляется их поддержка.

Ещё большой набор MIB представлен в качестве информации (но не является набором для поиска по умолчанию) в подкаталогах `/usr/share/mibs`:

```
$ tree /usr/share/mibs | head -n10
/usr/share/mibs
|-- iana
|   |-- IANA-ADDRESS-FAMILY-NUMBERS-MIB
|   |-- IANA-CHARSET-MIB
|   |-- IANA-FINISHER-MIB
|   |-- IANA-GMPLS-TC-MIB
|   |-- IANAifType-MIB
|   |-- IANA-IPPM-METRICS-REGISTRY-MIB
|   |-- IANA-ITU-ALARM-TC-MIB
|   |-- IANA-LANGUAGE-MIB
$ ls -l /usr/share/mibs
итого 28
drwxr-xr-x. 2 root root  4096 июля  30 14:46 iana
drwxr-xr-x. 2 root root 12288 июля  30 14:46 ietf
drwxr-xr-x. 2 root root  4096 июля  30 14:46 irtf
drwxr-xr-x. 2 root root  4096 янв.  14  2012 site
drwxr-xr-x. 2 root root  4096 июля  30 14:46 tubs
```

Это большая группа MIB-файлов общесистемных определений,

Стандарт SNMP **надсистемный** (или подсистемный, или внесистемный — как правильно считать?). Это означает, что если разработчик создал для своего устрйства переменную `.1.3.6.1.4.1.9876.11.5`, то она под таким же OID будет (должна!) отображаться в операционной системе Windows, Linux, QNX, а также и в любой встраиваемой реализации оборудования без поддержки вообще со стороны какой либо операционной системы (SNMP код, статически прикомпонованный к проекту). Агенты SNMP могут программно реализоваться любыми средствами, лишь бы они удовлетворяли стандарту SNMP. В операционных системах POSIX/UNIX базовую функциональность SNMP традиционно реализует проект `net-snmp` (сайт проекта: http://www.net-snmp.org/), дополнительно существует ряд сторонних проектов, расширяющих его функциональность по части разработки агентов, и представляющие различные формы менеджеров.

И последнее. SNMP стандарт — это страшная система при бесконтрольном использовании (как это я себе представляю). Представьте себе: разработанная кем-то видеокамера, после многих лет молчания, начинает неожиданно щёлкать по SNMP командам с другого конца мира, и отсылать по запросам видеокадры. Или: ответственное оборудование по командам из сети отказывается функционировать и превращается в груду железа.

Инструменты

Мне повезло в том смысле, что на момент написания этого текста у меня в руках была свежая инсталляция системы Linux, дистрибутив Fedora 17 RFR, в которой по умолчанию не установлена SNMP. Поэтому можно по шагам устанавливать только те инструменты SNMP, которые необходимо, и тем придать этому процессу системность. Убедиться, что за система, и что в ней изначально не установлено ничего, относящегося к SNMP можно следующим образом:

```
$ uname -a
Linux notebook 3.5.2-1.fc17.i686.PAE #1 SMP Wed Aug 15 16:30:14 UTC 2012 i686 i686 i386 GNU/Linux
$ cat /etc/system-release
RFRemix release 17 (Beefy Miracle)
$ ps -a | grep snmp
$ ls -w120 /lib/systemd/system/*snmp*.service
ls: невозможно получить доступ к /lib/systemd/system/*snmp*.service: Нет такого файла или каталога
$ ls /etc/init.d/*snmp*
ls: невозможно получить доступ к /etc/init.d/*snmp*: Нет такого файла или каталога
$ yum list all net-snmp*
...
Установленные пакеты
net-snmp-libs.i686                      1:5.7.1-4.fc17                      @russianfedora/$releasever
Доступные пакеты
net-snmp.i686                           1:5.7.1-5.fc17                      updates
net-snmp-agent-libs.i686                1:5.7.1-5.fc17                      updates
net-snmp-devel.i686                     1:5.7.1-5.fc17                      updates
net-snmp-gui.i686                       1:5.7.1-5.fc17                      updates
net-snmp-libs.i686                      1:5.7.1-5.fc17                      updates
net-snmp-perl.i686                      1:5.7.1-5.fc17                      updates
net-snmp-python.i686                    1:5.7.1-5.fc17                      updates
net-snmp-sysvinit.i686                  1:5.7.1-5.fc17                      updates
net-snmp-utils.i686                     1:5.7.1-5.fc17                      updates
```

Демоны

Теперь мы можем начать установку инструментария SNMP по частям, по мере необходимости:

```
$ sudo yum install net-snmp.i686
...
Установлено:
  net-snmp.i686 1:5.7.1-5.fc17
Установлены зависимости:
  lm_sensors-libs.i686 0:3.3.2-5.fc17                  net-snmp-agent-libs.i686 1:5.7.1-5.fc17
Обновлены зависимости:
  net-snmp-libs.i686 1:5.7.1-5.fc17
Выполнено!
New leaves:
  net-snmp.i686
```

На этом этапе установлена базовая функциональность SNMP — демоны (сервисы) реализующие агентов:

```
$ ls -w120 /lib/systemd/system/*snmp*.service
/lib/systemd/system/snmpd.service  /lib/systemd/system/snmptrapd.service
$ ls /etc/snmp
snmpd.conf  snmptrapd.conf
$ which snmpd
/usr/sbin/snmpd
$ which snmptrapd
/usr/sbin/snmptrapd
$ yum list installed net-snmp*
```

```
...
Установленные пакеты
net-snmp.i686                                          1:5.7.1-5.fc17
@updates
net-snmp-agent-libs.i686                               1:5.7.1-5.fc17
@updates
net-snmp-libs.i686                                     1:5.7.1-5.fc17
@updates
```

У нас появилось в `/lib/systemd/system/` управление сервисом SNMP (агентом `snmpd`) с помощью `systemd` (старую систему управления сервисами `sysinit` я не стану рассматривать, но там нет никаких принципиальных отличий). Появились конфигурационные файлы управления демонами в `/etc` (`snmpd.conf` и `snmptrapd.conf`) — это основные инструменты, с которыми предстоит работать.

Теперь у нас уже есть возможность запустить агенты SNMP и взаимодействовать с ними из сети (UDP порты 161 и 162). Но у нас нет пока средств даже посмотреть как работает установленный по умолчанию. У нас нет **утилит** SNMP:

```
$ which snmpwalk
which: no snmpwalk in ...
...
$ net-snmp-config --base-cflags
bash: net-snmp-config: команда не найдена
```

(Показаны примеры утилит из разных установочных пакетов).

Пакеты и программы

Прежде чем устанавливать всё оптом, хорошо бы разобраться что откуда происходит и где лежит!

```
$ yum info net-snmp*
...
Установленные пакеты
Название: net-snmp
Архитектура: i686
Период: 1
Версия: 5.7.1
Выпуск: 5.fc17
Объем: 859 k
Источник: installed
Из источника: updates
Аннотация: A collection of SNMP protocol tools and libraries
Ссылка: http://net-snmp.sourceforge.net/
Лицензия: BSD
Описание: SNMP (Simple Network Management Protocol) is a protocol used for
        : network management. The NET-SNMP project includes various SNMP tools:
        : an extensible agent, an SNMP library, tools for requesting or setting
        : information from SNMP agents, tools for generating and handling SNMP
        : traps, a version of the netstat command which uses SNMP, and a Tk/Perl
        : mib browser. This package contains the snmpd and snmptrapd daemons,
        : documentation, etc.
        :
        : You will probably also want to install the net-snmp-utils package,
        : which contains NET-SNMP utilities.
...
Доступные пакеты
...
Название: net-snmp-utils
Архитектура: i686
Период: 1
Версия: 5.7.1
```

```
Выпуск: 5.fc17
Объем: 174 k
Источник: updates
Аннотация: Network management utilities using SNMP, from the NET-SNMP project
Ссылка: http://net-snmp.sourceforge.net/
Лицензия: BSD
Описание: The net-snmp-utils package contains various utilities for use with the
        : NET-SNMP network management project.
        :
        : Install this package if you need utilities for managing your network
        : using the SNMP protocol. You will also need to install the net-snmp
        : package.
...
```

Обратите внимание: лицензия везде BSD, а не GPL (это ещё одна причина почему в Linux не очень сильно любят рассуждать про SNMP). Из полученного таким образом списка, например:

- `net-snmp-utils` — это именно тот набор утилит, который нам сейчас и нужен;
- `net-snmp-devel` и некоторые другие — понадобятся очень вскоре;
- `net-snmp-gui` — это графические утилиты для работы с базой данных MIB (наверное, это очень приятно в работе, но я не знаю что это такое);
- `net-snmp-sysvinit` — это инструменты для старой системы инициализации сервисов, может быть очень полезным в других дистрибутивах (например Debian);

Но особенно плодотворным будет поиск по именам требуемых нам программ пакетов, их содержащих:

```
$ yum provides mib2c
...
1:net-snmp-perl-5.7.1-5.fc17.i686 : The perl NET-SNMP module and the mib2c tool
Источник: updates
Совпадения с:
Имя файла   : /usr/bin/mib2c
$ yum provides snmpwalk
...
1:net-snmp-utils-5.7.1-5.fc17.i686 : Network management utilities using SNMP, from the NET-SNMP
project
Источник: updates
Совпадения с:
Имя файла   : /usr/bin/snmpwalk
$ yum provides snmptranslate
...
1:net-snmp-utils-5.7.1-5.fc17.i686 : Network management utilities using SNMP, from the NET-SNMP
project
Источник: updates
Совпадения с:
Имя файла   : /usr/bin/snmptranslate
$ yum provides net-snmp-config
...
1:net-snmp-devel-5.7.1-5.fc17.i686 : The development environment for the NET-SNMP project
Источник: updates
Совпадения с:
Имя файла   : /usr/bin/net-snmp-config
$ yum provides smilint
...
libsmi-0.4.8-9.fc17.i686 : A library to access SMI MIB information
Источник: updates
Совпадения с:
Имя файла   : /usr/bin/smilint
```

Утилиты

Вот теперь мы полностью определились с пакетами, подлежащими установке, и теперь можно вернуться к установке. Устанавливаем утилиты SNMP:

```
$ sudo yum install net-snmp-utils.i686
...
Установлено:
  net-snmp-utils.i686 1:5.7.1-5.fc17
Выполнено!
New leaves:
  net-snmp-utils.i686
$ ls -w100 /usr/bin/snmp*
/usr/bin/snmpbulkget   /usr/bin/snmpget      /usr/bin/snmpstatus     /usr/bin/snmptrap
/usr/bin/snmpbulkwalk  /usr/bin/snmpgetnext  /usr/bin/snmptable      /usr/bin/snmpusm
/usr/bin/snmpconf      /usr/bin/snmpinform   /usr/bin/snmptest        /usr/bin/snmpvacm
/usr/bin/snmpdelta     /usr/bin/snmpnetstat  /usr/bin/snmptls         /usr/bin/snmpwalk
/usr/bin/snmpdf        /usr/bin/snmpset      /usr/bin/snmptranslate
```

Из этих утилит большинство — это утилиты, выполняющие запросы к SNMP агенту (`snmpd`), то есть это консольные **менеджеры**: `snmpget`, `snmpgetnext`, `snmpwalk`, ... Это утилиты runtime. Но есть и утилиты другого назначения, например мы будем активно использовать `snmptranslate` — эта утилита не требует запуска `snmpd`, а в статике транслирует OID-ы, определённые в MIB-файлах, в разные формы: числовую, символьную, древесную иерархию... Теперь мы можем опробовать установленные утилиты, но для этого нам нужно будет запустить ранее инсталлированный агент (демон, сервер, службу) `snmpd`. Обычно запускается как сервис Linux, и далее мы и будем так делать, но на начальных этапах будем запускать его в отладочном режиме, когда он не переходит в режим демона, а выводит отладочную информацию на терминал. **Многочисленные** опции запуска `snmpd` смотрим, как обычно:

```
$ snmpd --help
...
# snmpd -f -Le -d
mibII/mta_sendmail.c:open_sendmailst: could not guess version of statistics file
"/var/log/mail/statistics"
Created directory: /var/lib/net-snmp/cert_indexes
Created directory: /var/lib/net-snmp/mib_indexes
NET-SNMP version 5.7.1
...
$ ps -A | grep snmp
 8037 pts/6    00:00:00 snmpd
```

Запуск `snmpd` производен от имени `root`.

Теперь можно проверить работу утилит SNMP относительно стандартных системных OID.

```
$ snmpwalk -v1 localhost -c public system
SNMPv2-MIB::sysDescr.0 = STRING: Linux notebook 3.5.2-1.fc17.i686.PAE #1 SMP Wed Aug 15 16:30:14 UTC
2012 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (48754) 0:08:07.54
SNMPv2-MIB::sysContact.0 = STRING: Root <root@localhost> (configure /etc/snmp/snmp.local.conf)
SNMPv2-MIB::sysName.0 = STRING: notebook
SNMPv2-MIB::sysLocation.0 = STRING: Unknown (edit /etc/snmp/snmpd.conf)
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORID.1 = OID: SNMP-MPD-MIB::snmpMPDMIBObjects.3.1.1
SNMPv2-MIB::sysORID.2 = OID: SNMP-USER-BASED-SM-MIB::usmMIBCompliance
SNMPv2-MIB::sysORID.3 = OID: SNMP-FRAMEWORK-MIB::snmpFrameworkMIBCompliance
SNMPv2-MIB::sysORID.4 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.5 = OID: TCP-MIB::tcpMIB
SNMPv2-MIB::sysORID.6 = OID: IP-MIB::ip
SNMPv2-MIB::sysORID.7 = OID: UDP-MIB::udpMIB
SNMPv2-MIB::sysORID.8 = OID: SNMP-VIEW-BASED-ACM-MIB::vacmBasicGroup
```

```
SNMPv2-MIB::sysORID.9 = OID: SNMP-NOTIFICATION-MIB::snmpNotifyFullCompliance
SNMPv2-MIB::sysORID.10 = OID: NOTIFICATION-LOG-MIB::notificationLogMIB
SNMPv2-MIB::sysORDescr.1 = STRING: The MIB for Message Processing and Dispatching.
SNMPv2-MIB::sysORDescr.2 = STRING: The MIB for Message Processing and Dispatching.
SNMPv2-MIB::sysORDescr.3 = STRING: The SNMP Management Architecture MIB.
SNMPv2-MIB::sysORDescr.4 = STRING: The MIB module for SNMPv2 entities
SNMPv2-MIB::sysORDescr.5 = STRING: The MIB module for managing TCP implementations
SNMPv2-MIB::sysORDescr.6 = STRING: The MIB module for managing IP and ICMP implementations
SNMPv2-MIB::sysORDescr.7 = STRING: The MIB module for managing UDP implementations
SNMPv2-MIB::sysORDescr.8 = STRING: View-based Access Control Model for SNMP.
SNMPv2-MIB::sysORDescr.9 = STRING: The MIB modules for managing SNMP Notification, plus filtering.
SNMPv2-MIB::sysORDescr.10 = STRING: The MIB module for logging SNMP Notifications.
SNMPv2-MIB::sysORUpTime.1 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.2 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.3 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.4 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.5 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.6 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.7 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.8 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.9 = Timeticks: (8) 0:00:00.08
SNMPv2-MIB::sysORUpTime.10 = Timeticks: (8) 0:00:00.08
$ snmpget -v1 localhost -c public SNMPv2-MIB::sysName.0
SNMPv2-MIB::sysName.0 = STRING: notebook
$ snmpget 192.168.1.5 -v1 -c public system.sysDescr.0
SNMPv2-MIB::sysDescr.0 = STRING: Linux notebook 3.5.2-1.fc17.i686.PAE #1 SMP Wed Aug 15 16:30:14 UTC
2012 i686
```

Это были **выполняемые** запросы к `snmpd`. А вот как выглядит статическая диагностика одного из поддеревьев OID-ов:

```
$ snmptranslate -Tp -OS SNMPv2-MIB::sysOREntry
+--sysOREntry(1)
   |  Index: sysORIndex
   |
   +-- ---- INTEGER   sysORIndex(1)
   |        Range: 1..2147483647
   +-- -R-- ObjID     sysORID(2)
   +-- -R-- String    sysORDescr(3)
   |        Textual Convention: DisplayString
   |        Size: 0..255
   +-- -R-- TimeTicks sysORUpTime(4)
            Textual Convention: TimeStamp
```

Численное представление OID того же узда поддерева:

```
$ snmptranslate -On SNMPv2-MIB::sysOREntry
```

– версия 2 (SNMPv2c) на основе сообществ (community) — опция `-c` в утилитах `snmp*` указывает имя **сообщества**, по принадлежности к которому в конфигурационном файле `snmpd.conf` разграничиваются полномочия доступа;

```
.1.3.6.1.2.1.1.9.1
```

И полная спецификация (тип, способ доступа, ...) всё того же узла:

```
bash-4.2$ snmptranslate -Td -On SNMPv2-MIB::sysOREntry
.1.3.6.1.2.1.1.9.1
sysOREntry OBJECT-TYPE
  -- FROM      SNMPv2-MIB
  MAX-ACCESS   not-accessible
```

```
  STATUS       current
  INDEX        { sysORIndex }
  DESCRIPTION  "An entry (conceptual row) in the sysORTable."
::= { iso(1) org(3) dod(6) internet(1) mgmt(2) mib-2(1) system(1) sysORTable(9) 1 }
```

Версии протокола SNMP

За время эволюции протокол SNMP претерпел несколько изменений версии протокола. На сегодня используемыми версиями являются:

- версия 1 (SNMPv1) — изначальная версия, появившаяся в 1988г. (Linux тогда ещё не существует);
- версия 3 (SNMPv3) не привносит никаких существенных изменений в протокол помимо добавления криптографической защиты.

Протоколы не совместимы между собой, но SNMP агенты умеют обрабатывать все из них, но менеджеры обязаны указывать версию используемого в запросе протокола (опция `-v` в утилитах `snmp*`, или в программных вызовах).

Обработка MIB-файлов

Для определения набора OID для собственного проекта пишут MIB-файлы. MIB-файлы определяют: а).поддерево иерархии собственных OID в базе данных MIB (в дереве), б).структуру поддерева, в).OID узлов поддерева, г). типы данных объектов в поддереве. В принципе, писать MIB-файлы не обязательно, как уже отмечалось, символьные имена узлов дерева нужны для восприятия только человеком, и если вы знаете полную числовую запись OID интересующего вас параметра, то можете написать код агента, обслуживающий такой OID. Но на практике так не делают. А делают в следующей последовательности:

1. Составляется один или несколько связанных MIB-файлов по очень строгим синтаксическим правилам (MIB-файл — это текстовый файл).
2. Делается тщательная автоматическая синтаксическая выверка и правка MIB-файлов. Для этого используются утилиты: `snmptranslate` и `smilint` (ранее мы определили, что `smilint` находится в составе пакета `libsmi.i686`, но ещё не установили его).
3. Помещаем MIB-файлы в один из каталогов, где они будут доступны подсистеме SNMP для поиска по путям по умолчанию.
4. После этого применяются **автоматические генераторы кода**, которые по описанным в MIB-файлах OID делают генерацию шаблонов кода обслуживания. Существует достаточно много сторонних пакетов генерации шаблонов кода на языках C, C++, Java, ... Один из таких генераторов в код C `mib2c` входит в состав пакета `net-snmp-perl` (мы определили это выше).
5. Полученные шаблоны кода включаются в код своего проекта субагента.

Подобное рассмотрение MIB-файлов отложим до рассмотрения реализации примера. А установку недостающих пакетов завершим сейчас:

```
$ sudo yum install libsmi-*
...
Установлено:
  libsmi-devel.i686 0:0.4.8-9.fc17
Обновлено:
  libsmi.i686 0:0.4.8-9.fc17
Выполнено!
New leaves:
  libsmi-devel.i686
$ which smilint
/usr/bin/smilint
$ sudo yum install net-snmp-perl
...
Установлено:
```

```
  net-snmp-perl.i686 1:5.7.1-5.fc17
$ which mib2c
/usr/bin/mib2c
$ sudo yum install net-snmp-devel
...
Установлено:
  net-snmp-devel.i686 1:5.7.1-5.fc17
Установлены зависимости:
  elfutils-devel.i686 0:0.154-2.fc17                  elfutils-libelf-devel.i686 0:0.154-2.fc17
lm_sensors-devel.i686 0:3.3.2-5.fc17              popt-devel.i686 0:1.13-10.fc17
rpm-devel.i686 0:4.9.1.3-7.fc17
$ which net-snmp-config
/usr/bin/net-snmp-config
```

Вот теперь мы имеем весь необходимый инструментарий для создания SNMP управления в собственном проекте.

Разработка

Для прохождения всего процесса проектирования нужна тестовый проект. Единственное требование к такому проекту — его простота. Чтобы его детали не заслоняли процесса проектирования. Но он должен быть и не тривиальный, чтобы можно было наблюдать созданный результат в действии.

Тестовый проект

В качестве тестового проекта выбрана простейшая задача авторегулирования:

1. Есть некоторое регулируемое значение (это одна переменная);
2. Значение этой управляемой переменной считывается, и в зависимости от разбалансировки вырабатывается управляющее воздействие (разница значений);
3. Из управляющего воздействия формируется шаг корректировки (умножением на коэффициент петлевого усиления), поэтому можно имитировать недорегулирование, перерегулирование...
4. Шаг корректировки записывается во вторую переменную;
5. Устройство корректирует регулируемое значение на полученный шаг корректировки;
6. Если в результате записи п.5 разбалансировка всё ещё не нулевая — возвращаемся на п.1.

В проекте предстоит создать: а). программу субагента к `snmpd`, б). настройки `snmpd` для работы с субагентом, в). менеджер (клиент) запросов к переменным субагента. Не вникая в детали (это всё можно найти на сайте `net-snmp`), техника субагента в SNMP предполагает использование специального протокола AgentX, по которому главный агент (`snmpd`) может передать на обработку другому локальному процессу запросы к OID, которые он не может обработать сам.

Локальный эквивалент менеджера

Подобный проект хорош ещё и тем, что для него очень легко собрать локальный эквивалент SNMP проекта, в котором переменные доступны не через механизм SNMP, а доступ к ним (под теми же именами функциональных вызовов) реализован локально, и всё это просто монолитно скомпоновано на этапе компиляции. Тогда единой операцией сборки будут собираться 2 подобных в поведении проекта: тестовый SNMP и локальный. Вот `Makefile` — сценарий сборки, в той части, которая ответственна за сборку менеджеров (клиентов):

```
PROGLIST = cli_locl cli_snmp
all: prog
prog:  $(PROGLIST)
cli_locl: cli.c locl.c common.h
        $(CC) $(COPT) -lm cli.c locl.c -o $@
cli_snmp: cli.c snmp.c common.h
        $(CC) $(COPT) -lm cli.c snmp.c -o $@
clean:
        @rm -f *.o $(PROGLIST)
```

Как легко видеть, вся общая часть проекта (файл `cli.c`) — единая для обоих клиентских приложений, и только реализация интерфейса доступа к переменным подменяется: `locl.c` в случае локального размещения переменных, и `snmp.c` для SNMP доступа. Такой локальный эквивалент позволяет изучить **поведение** проекта в его локальной реализации, прежде, чем приступать к отработке реализации SNMP. Например:

```
$ ./cli_locl -v
команда (h-подсказка): h
реализованные команды: * ? = h q
команда (h-подсказка): ?
текущее значение = 0
команда (h-подсказка): * 1.3
```

```
усиление = 1.3
команда (h-подсказка): = 200
новое значение = 200
0+260  260-78  182+23  205-7  198+3  201-1
команда (h-подсказка): ?
текущее значение = 200
команда (h-подсказка): q
```

Показан пример перехода системы из состояния с значением управляемой переменной 0 в состояние с значением 200, с коэффициентом усиления (перерегулированием) 1.3, стационарное состояние управляемой переменной устанавливается за 6 последовательных циклов чтения-коррекции.

MIB-файлы и OID

Под любой конкретный проект, использующий SNMP управление, полезно сформировать для определения набора OID описательные MIB-файлы. Формат файлов текстовый. Синтаксис MIB-файлов очень жёстко регламентирован.

Содержимое

MIB-файлы определяют: а).поддерево иерархии собственных OID в базе данных MIB (в дереве), б).структуру поддерева, в).OID узлов поддерева, г). типы данных объектов в поддереве. Может составляться один MIB-файл, но обычно несколько связанных MIB-файлов. Почему несколько? Потому, что поддерево MIB проекта — это иерархия, на верхнем уровне которой, например OID соответствующий **всем проектам** фирмы, компании разработчика за многие годы. Этот MIB-файл будет многократно использоваться, на него будут ссылаться нижележащие MIB-файлы, скажем, подразделений компании, ещё ниже — MIB-файлы отдельных проектов. Для нашего тестового проекта я формирую MIB-фай верхнего уровня:

```
$ cat OLEJ-MIB.txt
OLEJ-MIB DEFINITIONS ::= BEGIN
-- Top-level infrastructure of the OLEJ-SNMP projects enterprise MIB tree
-- Title: Olej TOP LEVEL MIB
-- Version : 1.0
-- Revision History:
-- ********************************************************************
-- 04/12/2012 - v1.0 Create base functionality

IMPORTS
    MODULE-IDENTITY, enterprises FROM SNMPv2-SMI;

olej MODULE-IDENTITY
    LAST-UPDATED    "201212040000Z"
    ORGANIZATION    "no organization"
    CONTACT-INFO
         "email:    olej@front.ru"
    DESCRIPTION     "Top-level MIB .1.3.6.1.4.1.9876"
    REVISION        "201212040000Z"
    DESCRIPTION     "First draft"
::= { enterprises 9876 }

END
```

Имя файла не имеет принципиального значения, принципиальное значение имеет имя MIB-определения (`OLEJ-MIB`) с которого должен начинаться файл (и заканчиваться `END`). Смысл его понятен из дефиниции `IMPORT`:

- импортируется имя OID (`enterprises`) из системных MID-определений верхнего уровня (`SNMPv2-SMI`);
- это та точка (`.1.3.6.1.4.1`) всемирной иерархии в базе MIB, к которой прикрепляется OID (`9876`)

вершины моего поддерева (`olej`), от которой будут прикрепляться все мои иерархии «вниз»;

- в принципе, этот корневой OID организации (`9876`), должен быть единоразово получен в международных комитетах, отвечающих за протоколы, и OID должен быть уникален для всех организаций мира;
- как вы понимаете, это оргвопросы, и я указал в этой позиции произвольное значение (с малой вероятностью пересечься с используемыми).

Строки, начинающиеся с `'--'`в MIB считаются комментариями. Если в развитии вашего проекта планируется использовать MIB-файлы с менеджером SNMP `OpManager` (или другими GUI подобными), то в текстах MIB, в русскоязычных комментариях, необходимо избегать литеры «ь», которую парсер менеджера воспринимает неверно, и которая повергает его «в безумие».

Все нижележащие MIB-файлы описывают иерархию поддерева, исходящую от OID верхнего уровня (`.1.3.6.1.4.1.9876`). Для описываемого простейшего тестового проекта достаточно ещё одного файла:

`$ cat OLEJ-MANAGEMENT-MIB.txt`

```
bash-4.2$ cat OLEJ-MANAGEMENT-MIB.txt
OLEJ-MANAGEMENT-MIB DEFINITIONS ::= BEGIN

IMPORTS
    OBJECT-TYPE, NOTIFICATION-TYPE,
    MODULE-IDENTITY, Integer32,
    Gauge32, IpAddress                FROM SNMPv2-SMI
    DisplayString                     FROM SNMPv2-TC
    olej                              FROM OLEJ-MIB
    ;

management MODULE-IDENTITY
    LAST-UPDATED    "201212040000Z"
    ORGANIZATION    "no organization"
    CONTACT-INFO
         "email:    olej@front.ru"
    DESCRIPTION     "SubAgent level MIB .1.3.6.1.4.1.9876.11"
    REVISION        "201212040000Z"
    DESCRIPTION     "First draft"
::= { olej 11 }

--------------------------------------------------------------------------------

-- IP addr. собственный хоста
hostIpAddress OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS current
    DESCRIPTION "Host IP address."
::= { management 1 }

-- имя хоста
hostName OBJECT-TYPE
    SYNTAX      DisplayString
    MAX-ACCESS  read-only
    STATUS current
    DESCRIPTION "Host name."
::= { management 2 }

-- регулируемая переменная, доступная только по чтению
currentValue OBJECT-TYPE
    SYNTAX      Integer32
```

```
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION "Current Value."
::= { management 5 }

-- шаг коррекции, используется только по записи
nextStep OBJECT-TYPE
    SYNTAX      Integer32
    MAX-ACCESS  read-write
    STATUS      current
    DESCRIPTION "Next Step."
::= { management 7 }

--------------------------------------------------------------------------------

END
```

Две переменные (`hostIpAddress` и `hostName`) добавлены здесь «для красного словца» - чтобы показать широту типизации данных. Они не используются в проекте, это достаточно широко используемая практика в MIB — определять OID, которые на сегодня не задействованы. Имена всех переменных начинаются с **малой** буквы. А вот следующие две переменные — это и есть те переменные, на которых строится тестовый проект:

- `currentValue` — это целочисленная регулируемая переменная;
- `nextStep` — это целочисленная добавка к `currentValue` на очередном шаге установления значения;

Местоположение

Утилиты SNMP не работают с MIB-файлами, размещёнными «где попало», они работают с MIB-файлами, помещёнными в нескольких определённых каталогах (их довольно много), например: `/usr/share/mibs`, `/usr/share/snmp/mibs`, `$HOME/.snmp/mibs` ... (список таких каталогов они называют MIB directory search list, он включает в себя: а). предопределённый список, часть которого перечислена выше, а полный может быть найден в `man`, б). пути определённые в переменной окружения MIBS и в). пути определяемые в `snmp.conf` — но указать явно утилитам путь к MIB **нельзя**). Список предопределённых путей смотрим:

```
$ net-snmp-config --default-mibdirs
/home/olej/.snmp/mibs:/usr/share/snmp/mibs
```

Помещаем созданные MIB-файлы в один из каталогов, где они будут доступны подсистеме SNMP для поиска по путям по умолчанию. Вполне удачное место в файловой системе, где можно **начинать** отработку MIB-файлов, это `$HOME/.snmp/mibs`. Создадим:

```
$ mkdir -vp ~/.snmp/mibs
mkdir: создан каталог «/home/olej/.snmp»
mkdir: создан каталог «/home/olej/.snmp/mibs»
```

Поместим туда созданные нами MIB-файлы:

```
$ tree ~/.snmp
/home/olej/.snmp
`-- mibs
    |-- OLEJ-MANAGEMENT-MIB.txt
    `-- OLEJ-MIB.txt
1 directory, 2 files
$ ls -l ~/.snmp/mibs
итого 8
-rw-r--r-- 1 olej olej 1553 дек.   4 22:23 OLEJ-MANAGEMENT-MIB.txt
-rw-r--r-- 1 olej olej  656 дек.   5 01:48 OLEJ-MIB.txt
```

Выверка

Теперь с определёнными новыми OID проекта можно поработать. Это хороший этап и плотно поэкспериментировать с утилитами SNMP. Начинаем использование утилит с синтаксической выверки созданных MIB-файлов. Поверьте мне на слово, что синтаксические требования к записи MIB-файла жёстче, чем у любого языка программирования, и это занятие переведёт вам ещё немало крови. Поэтому используется **автоматическая** синтаксическая выверка MIB-файлов. Для этого используются утилиты: `snmptranslate` и `smilint`:

```
$ snmptranslate -On -m +OLEJ-MIB -IR olej
Expected DESCRIPTION (:): At line 19 in /home/olej/.snmp/mibs/OLEJ-MIB.txt
.1.3.6.1.4.1.9876
```

Вот и первая ошибка, хотя и уровня предупреждения (команда отработала): пропущено (было) определение `DESCRIPTION` (2-е, после `REVISION`), 1-е определение `DESCRIPTION` относилось ко всему OID, но каждый следующий `REVISION` (а их может быть со временем много) хотел бы свой `DESCRIPTION`. И такой уровень придирок будет по случаю любого определения MIB. Исправляем:

```
$ snmptranslate -On -m +OLEJ-MIB -IR olej
.1.3.6.1.4.1.9876
$ snmptranslate -On -m +OLEJ-MANAGEMENT-MIB -IR currentValue
.1.3.6.1.4.1.9876.11.5
$ snmptranslate -On -m +OLEJ-MANAGEMENT-MIB -IR nextStep
.1.3.6.1.4.1.9876.11.7
```

Вот те же OID в символьном изображении:

```
$ snmptranslate -Of OLEJ-MANAGEMENT-MIB:currentValue
.iso.org.dod.internet.private.enterprises.olej.management.currentValue
$ snmptranslate -Of OLEJ-MANAGEMENT-MIB::nextStep
.iso.org.dod.internet.private.enterprises.olej.management.nextStep
```

Очень тщательный и придирчивый синтаксический контроль осуществляет утилита `smilint` (мы её инсталлировали выше). На достаточно сложных иерархиях MIB вычистить синтаксис без её помощи очень затруднительно:

```
$ smilint -l3  -s -p ./OLEJ-MIB.txt ./OLEJ-MANAGEMENT-MIB.txt
```

Опция `-l` определяет максимальную степень грубости обнаруженных ошибок, при уровне `-l4` (warning) она всегда что-то найдёт:

```
$ smilint -l4  -s -p ./OLEJ-MIB.txt ./OLEJ-MANAGEMENT-MIB.txt
./OLEJ-MANAGEMENT-MIB.txt:24: [4] warning: node `hostIpAddress' must be contained in at least one
conformance group
./OLEJ-MANAGEMENT-MIB.txt:32: [4] warning: node `hostName' must be contained in at least one
conformance group
./OLEJ-MANAGEMENT-MIB.txt:39: [4] warning: node `currentValue' must be contained in at least one
conformance group
./OLEJ-MANAGEMENT-MIB.txt:46: [4] warning: node `nextStep' must be contained in at least one
conformance group
```

Внимание: квалификаторы вида `OLEJ-MIB` в записи команды во всех утилитах `net-snmp` (`snmptranslate`) — это не **имена файлов** описаний (и никак не соотносятся с именами файлов), а **имена описаний модулей**, так, например, файл OLEJ-MANAGEMENT-MIB.txt начинается со строки описания:

```
OLEJ-MANAGEMENT-MIB DEFINITIONS ::= BEGIN
```

Имена файлов описаний и имена описаний модулей часто очень похожи, или могут совпадать — это обычная практика в `net-snmp`, но их нужно отчётливо **различать**. В команде `smilint`, напротив, записаны имена файлов (подлежащих проверке).

После всех этих действий системе SNMP стали известны специфические OID тестового проекта, и только теперь можно переходить к реализации.

Разработка субагента

Техника написания субагента полно и примерами описана на сайте проекта `net-snmp`: http://www.net-snmp.org/. Кроме этого, можно рекомендовать такую публикацию: Konrad Rzeszutek, «NetSNMP subagent development manual», http://openhpi.sourceforge.net/subagent-manual/book1.html .

Мы не будем писать код C для обслуживания OID проекта (особенно для крупных проектов), а воспользуемся Perl скриптом `mib2c`, для генерации шаблона кода обработки:

```
$ mib2c -h
/usr/bin/mib2c [-h] [-c configfile] [-f prefix] mibNode
 -h            This message.
  -c configfile       Specifies the configuration file to use
               that dictates what the output of mib2c will look like.
 -I PATH       Specifies a path to look for configuration files in
 -f prefix     Specifies the output prefix to use.  All code
               will be put into prefix.c and prefix.h
 -d            debugging output (don't do it.  trust me.)
 -S VAR=VAL    Set $VAR variable to $VAL
 -i            Don't run indent on the resulting code
 -s            Don't look for mibNode.sed and run sed on the resulting code
 mibNode       The name of the top level mib node you want to
               generate code for.  By default, the code will be stored in
               mibNode.c and mibNode.h (use the -f flag to change this)
```

Генерируем шаблон кода для переменной `currentValue`.

```
$ env MIBS="+OLEJ-MANAGEMENT-MIB" mib2c -c mib2c.scalar.conf currentValue
writing to currentValue.h
writing to currentValue.c
running indent on currentValue.c
running indent on currentValue.h
```

Конфигурационный файл `mib2c.scalar.conf` для `mib2c`, указывающий правила генерации для **скалярного** OID, можно бы и не указывать, но тогда детали генерация придётся уточнять в диалоге. Заготовлено довольно много конфигураций для разных **типов** OID, главным образом для разных представлений табличных OID:

```
$ cd /usr/share/snmp
$ ls -w100 mib2c*.conf
mib2c.access_functions.conf    mib2c.container.conf       mib2c.notify.conf
mib2c.array-user.conf          mib2c.create-dataset.conf  mib2c.old-api.conf
mib2c.check_values.conf        mib2c.emulation.conf       mib2c.perl.conf
mib2c.check_values_local.conf  mib2c.genhtml.conf         mib2c.raw-table.conf
mib2c.column_defines.conf      mib2c.int_watch.conf       mib2c.row.conf
mib2c.column_enums.conf        mib2c.iterate_access.conf  mib2c.scalar.conf
mib2c.column_storage.conf      mib2c.iterate.conf         mib2c.table_data.conf
mib2c.conf                     mib2c.mfd.conf
```

Аналогично для второй переменной:

```
$ env MIBS="+OLEJ-MANAGEMENT-MIB" mib2c -c mib2c.scalar.conf nextStep
writing to nextStep.h
writing to nextStep.c
running indent on nextStep.c
running indent on nextStep.h
```

Теперь в рабочем каталоге у нас появились файлы кода, отвечающего за обработку OID проекта:

```
$ ls *.h
common.h  currentValue.h  nextStep.h
```

```
$ ls *.c
cli.c  currentValue.c  locl.c  nextStep.c  snmp.c
```

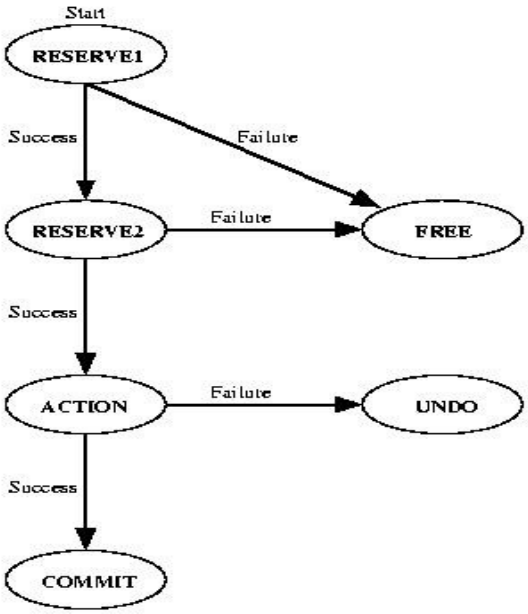
Остаётся **наполнить** обработчики конкретным, определяемым содержанием целевой задачи. Обратите внимание, что для переменной `currentValue` определена операция только чтения, а для `nextStep` — записи (всё в соответствии с описаниями в MIB-файле). Это потребует некоторых осмысленных (под поставленные в задаче цели) коррекций кода шаблонов, обычно незначительных.

Реализация операций

Реализация операции возврата значения на запрос менеджера (наиболее часто необходимая) реализуется крайне просто (то, что нам нужно дописать в генерированный шаблон):

```
int iCurrentValue = 0;
...
        case MODE_GET:
            snmp_set_var_typed_value( requests->requestvb, ASN_INTEGER,
                                      &iCurrentValue , sizeof( iCurrentValue )
                                    );
            break;
```

Мы добавляем здесь к шаблону только адрес возвращаемого значения и длину этого значения.



Чуть сложнее реализуется в шаблоне операции записи новых значений в OID. Она делается как последовательность нескольких шагов, которые позволяют выполнить откат (`MODE_SET_UNDO`) к прежнему значению, если операция записи окажется невозможной. В документации пакета `net-snmp` приводится схема выполнения операции модификации значения, которая здесь воспроизводится. На первом шаге этой последовательности (`MODE_SET_RESERVE1`), обычно, проделывается проверка типа переменной на соответствие того, как она описана в MIB. Если эта проверка не проходит, то операция отвергается.

Некоторых отдельных комментариев заслуживает то как значения, заданные в команде `snmpset` извлекаются в коде операции модификации значения (`MODE_SET_ACTION`). Заданные в команде `snmpset` параметры (тип и значение для OID) поступают в обработчик в составе последнего параметра типа `netsnmp_request_info*` виде списка значений — один вызов `snmpset` может указывать сразу на несколько последовательных операций. Значения должны извлекаться поочерёдно из всех элементов этого списка. Но в примере проекта это не делается для упрощения, значение переменной выбирается из первого элемента списка:

```
        case MODE_SET_ACTION:
            { netsnmp_variable_list *var = requests->requestvb;
              netsnmp_vardata val = var->val;
              iNextStep = *val.integer;
              iCurrentValue += iNextStep;
            }
```

Совсем особую часть составляют шаблоны и возможности обработки **табличных** OID. В этой части предоставляется на выбор несколько различных моделей генерации обработчиков таких OID (определяется выбором конфигурационного для `mib2c`, о чём уже было коротко рассказано). Эта техника намного более громоздкая, но вполне обозримая с помощью документации `net-snmp`.

Главная функция

Кроме этого нам предстоит ещё написать код главной функции (`main`) приложения субагента, но он **практически одинаков для всех проектов**, а прототип его приводится на сайте проекта `net-snmp` (http://www.net-snmp.org/wiki/index.php/Tutorials), или, как вариант, может быть взят здесь: http://openhpi.sourceforge.net/subagent-manual/c167.html#AEN178. Поместим этот код в произвольный файл, назвав его, положим, `subagent.c`.

Сборка

Собираем субагент, ниже показана часть сценария сборки `Makefile`, ответственная за эту часть:

```
CC = gcc
CFLAGS = `net-snmp-config --cflags`
BUILDAGENTLIBS = `net-snmp-config --agent-libs`
OBJS2 = subagent.o currentValue.o nextStep.o
PROGLIST = cli_locl cli_snmp myagn

all: prog
prog:  $(PROGLIST)

myagn: $(OBJS2)
        $(CC) $(OBJS2) $(BUILDAGENTLIBS) -o $@
        rm -f *.o
subagent.o: subagent.c
        $(CC) $(CFLAGS) -c $< -o $@
currentValue.o: currentValue.c
        $(CC) $(CFLAGS) -c $< -o $@
nextStep.o: nextStep.c
        $(CC) $(CFLAGS) -c $< -o $@
```

В документации пакета `net-snmp` упоминается (предлагается) и другой способ сборки программы субагента, а именно в `Manefile` он сохранён как сборка приложения `myagn2`:

```
myagn2: currentValue.c nextStep.c
        net-snmp-config --compile-subagent myagn2 currentValue.c nextStep.c
```

В таком варианте используется совсем уж типовой вид головной вызывающей программы, который уже никак нельзя изменить (и который собирается только как субагент протокола AgentX). Вряд ли такая ограниченность может считаться достоинством метода, даже не смотря на его простоту.

Тестирование

Такой субагент (изменением одного оператора в коде) может запускаться либо как субагент протокола AgentX, либо как автономный агент SNMP. Поэтому мы начинаем его проверить именно как автономный самостоятельный агент:

```
# ./myagn -v
myagn is up and running.
MODE_GET
MODE_SET_RESERVE1
MODE_SET_RESERVE2
MODE_SET_ACTION
MODE_SET_COMMIT
^C
--------- got signal: 2 -------------
myagn was finished.
```

```
$ snmpget -v2c -c private 127.0.0.1 OLEJ-MANAGEMENT-MIB::currentValue.0
OLEJ-MANAGEMENT-MIB::currentValue.0 = INTEGER: 0
$ snmpget -v2c -c private 127.0.0.1 OLEJ-MANAGEMENT-MIB::nextStep.0
OLEJ-MANAGEMENT-MIB::nextStep.0 = INTEGER: 0
$ snmpset -v2c -c private 127.0.0.1 OLEJ-MANAGEMENT-MIB::nextStep.0 i 13
OLEJ-MANAGEMENT-MIB::nextStep.0 = INTEGER: 13
$ snmpget -v2c -c private 127.0.0.1 OLEJ-MANAGEMENT-MIB::currentValue.0
OLEJ-MANAGEMENT-MIB::currentValue.0 = INTEGER: 13
```

Такой результат достигается **только** при выполнении всех конфигурационных требований для SNMP (которые, вообще то говоря, многочисленные и капризные):

- составление конфигурационного файла `myagn.conf` и помещение его в одно из мест, где он будет доступен программе `myagn` при старте, например `/etc/snmp` (начальным прототипом для `myagn.conf` может служить `/etc/snmp/snmpd.conf`);
- прописывание в `myagn.conf` разрешений для доступа определённых сообществ (опция `-c`) к требуемым поддеревьям OID;
- помещение MIB-файлов в место, где они будут доступны для `root` (не только утилитам SNMP, запускаемым от имени пользователя, но и агентам, запускаемым от `root`), например, `/root/.snmp/mibs`;

Сравнительное тестирование

Ранее уже объяснялось как в рамках единой модели тестового проекта собирается два клиентских приложения: локальный эмулятор модели, и менеджер SNMP, который выполняет ту же работу, но делает это на удалённом хосте сети, используя SNMP протокол. Теперь мы можем сравнить поведение двух полученных моделей — они должны быть идентичны.

```
$ ./cli_locl  -v
команда (h-подсказка): =13
новое значение = 13
0 +13 => 13
команда (h-подсказка): =3
новое значение = 3
13 -10 => 3
команда (h-подсказка): *1.4
усиление = 1.4
```

```
команда (h-подсказка): =7
новое значение = 7
3 +6 => 9
9 -3 => 6
6 +1 => 7
команда (h-подсказка): ?
текущее значение = 7
команда (h-подсказка): q
```

Выполнение сетевой модели с SNMP:

```
$ ./cli_snmp localhost -v
host: localhost, currentValue OID=.1.3.6.1.4.1.9876.11.5, nextStep OID=.1.3.6.1.4.1.9876.11.7
команда (h-подсказка): ?
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 13
текущее значение = 13
команда (h-подсказка): =3
новое значение = 3
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 13
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 13
13 -10 => 3
SNMPv2-SMI::enterprises.9876.11.7.0 = INTEGER: -10
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 3
команда (h-подсказка): ?
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 3
текущее значение = 3
команда (h-подсказка): *1.4
усиление = 1.4
команда (h-подсказка): =7
новое значение = 7
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 3
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 3
3 +6 => 9
SNMPv2-SMI::enterprises.9876.11.7.0 = INTEGER: 6
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 9
9 -3 => 6
SNMPv2-SMI::enterprises.9876.11.7.0 = INTEGER: -3
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 6
6 +1 => 7
SNMPv2-SMI::enterprises.9876.11.7.0 = INTEGER: 1
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 7
команда (h-подсказка): ?
SNMPv2-SMI::enterprises.9876.11.5.0 = INTEGER: 7
текущее значение = 7
команда (h-подсказка): q
```

Зачем ещё нужны две сравнительных модели? Затем, что заставить SNMP работать так как хочется — занятие непростое, и очень зависит от громоздких настроек конфигурационных файлов (`/etc/snmpd/snmpd.conf`, `/etc/snmpd/myagn.conf`, возможный вариант конфигурационных файлов включён в состав архива примеров). Даже воспроизвести показанный выше результат будет не столь простым делом. Локальная модель будет в этом процессе подсказкой (эталоном): как должно быть при разных входных данных.