Network Working Group   M. Rose
Request for Comments: 1227   Performance Systems International, Inc.
                                                              May 1991

                     SNMP MUX Protocol and MIB

Status of this Memo

   This memo suggests a mechanism by which a user process may associate
   itself with the local SNMP agent on a host, in order to implement
   portions of the MIB. This mechanism would be local to the host.

   This is an Experimental Protocol for the Internet community.
   Discussion and suggestions for improvement are requested. Please
   refer to the current edition of the "IAB Official Protocol Standards"
   for the standardization state and status of this protocol.
   Distribution of this memo is unlimited.

Table of Contents

1. Introduction

   On typical kernel/user systems, an agent speaking the SNMP [1] is
   often implemented as a user-process, that reads kernel variables in
   order to realize the Internet-standard MIB [2]. This approach works
   fine as long as all of the information needed by the SNMP agent
   resides in either the kernel or in stable storage (i.e., files).
   However, when other user-processes are employed to implement other

Rose   [Page 1]

network services, such as routing protocols, communication between
the SNMP agent and other processes is problematic.

In order to solve this problem, a new protocol, the SNMP multiplexing
(SMUX) protocol is introduced. When a user-process, termed a SMUX
peer, wishes to export a MIB module, it initiates a SMUX association
to the local SNMP agent, registers itself, and (later) fields
management operations for objects in the MIB module.

Carrying this approach to its fullest, it is possible to generalize
the SNMP agent so that it knows about only the SNMP group of the
Internet-standard MIB. All other portions of the Internet-standard
MIB can be implemented by another process. This is quite useful, for
example, when a computer manufacturer wishes to provide SNMP access
for its operating system in binary form.

In addition to defining the SMUX protocol, this document defines a
MIB for the SMUX. Obviously, this MIB module must also be
implemented in the local SNMP agent.

2. Architecture

There are two approaches that can be taken when trying to integrate
arbitrary MIB modules with the SNMP agent: request-response and
cache-ahead.

The request-response model simply propagates the SNMP requests
received by the SNMP agent to the user process which exported the MIB
module. The SMUX peer then performs the operation and returns a
response. In turn, the SNMP agent propagates this response back to
the network management station. The request-response model is said
to be agent-driven since, after registration, the SNMP agent
initiates all transactions.

The cache-ahead model requires that the SMUX peer, after
registration, periodically updates the SNMP agent with the subtree
for the MIB module which has been registered. The SNMP agent, upon
receiving an SNMP request for information retrieval, locally performs
the operation, and returns a response to the network management
station. (SNMP set requests are given immediately to the SMUX peer.)
The cache-ahead model is said to be peer-driven since, after
registration, the SMUX peer initiates all transactions.

There are advantages and disadvantages to both approaches. As such,
the architecture envisioned supports both models in the following
fashion: the protocol between the SNMP agent and the SMUX peer is
based on the request-response model. However, the SMUX peer, may
itself be a user-process which employs the cache-ahead model with

other user-processes.

Obviously, the SMUX peer which employs the cache-ahead model acts as
a "firewall" for those user-processes which actually implement the
managed objects in the given MIB module.

Note that this document describes only the SMUX protocol, for the
request-response model. Each SMUX peer is free to define a cache-
ahead protocol specific for the application at hand.

3. Protocol

The SMUX protocol is simple: the SNMP agent listens for incoming
connections. Upon establishing a connection, the SMUX peer issues an
OpenPDU to initialize the SMUX association. If the SNMP agent
declines the association, it issues a closePDU and closes the
connection. If the SNMP agent accepts the association, no response
is issued by the SNMP agent.

For each subtree defined in a MIB module that the SMUX peer wishes to
register (or unregister), the SMUX peer issues a RReqPDU. When the
SNMP agent receives such a PDU, it issues a corresponding RRspPDU.
The SNMP agent returns RRspPDUs in the same order as the RReqPDUs
were received.

When the SMUX peer wishes to issue a trap, it issues an SNMP Trap-
PDU. When the SNMP agent receives such a PDU, it propagates this to
the network management stations that it is configured to send traps
to.

When the SNMP agent receives an SNMP GetRequest-PDU, GetNextRequest-
PDU, or SetRequest-PDU which includes one or more variable-bindings
within a subtree registered by a SMUX peer, the SNMP agent sends an
equivalent SNMP PDU containing only those variables within the
subtree registered by that SMUX peer. When the SMUX peer receives
such a PDU, it applies the indicated operation and issues a
corresponding SNMP GetResponse-PDU. The SNMP agent then correlates
this result and propagates the resulting GetResponse-PDU to the
network management station.

When either the SNMP agent or the SMUX peer wishes to release the
SMUX association, the ClosePDU is issued, the connection is closed,
and all subtree registrations for that association are released.

3.1. Tricky Things

Although straight-forward, there are a few nuances.

3.1.1. Registration

   Associated with each registration is an integer priority, from 0 to
   (2^31)-1. The lower the value, the higher the priority.

   Multiple SMUX peers may register the same subtree. However, they
   must do so at different priorities (i.e., a subtree and a priority
   uniquely identifies a SMUX peer). As such, if a SMUX peer wishes to
   register a subtree at a priority which is already taken, the SNMP
   agent repeatedly increments the integer value until an unused
   priority is found.

   When registering a subtree, the special priority -1 may be used,
   which selects the highest available priority.

   Of course, the SNMP agent may select an arbitrarily worse priority
   for a SMUX peer, based on local (configuration) information.

   Note that when a subtree is registered, the SMUX peer with the
   highest registration priority is exclusively consulted for all
   operations on that subtree. Further note that SNMP agents must
   enforce the "subtree mounting effect", which hides the registrations
   by other SMUX peers of objects within the subtree. For example, if a
   SMUX peer registered "sysDescr", and later another SMUX peer
   registered "system" (which scopes "sysDescr"), then all requests for
   "sysDescr" would be given to the latter SMUX peer.

   An SNMP agent should disallow any attempt to register above, at, or
   below, the SNMP and SMUX subtrees of the MIB. Other subtrees may be
   disallowed as an implementation-specific option.

3.1.2. Removing Registration

   A SMUX peer may remove registrations for only those subtrees which it
   has registered. If the priority given in the RReqPDU is -1, then the
   registration of highest priority is selected for deletion.
   Otherwise, only that registration with the precise priority is
   selected.

3.1.3. Atomic Sets

   A simple two-phase commit protocol is used between the SNMP agent and
   the SMUX peers. When an SNMP SetRequest-PDU is received, the SNMP
   agent determines which SMUX peers will participate in the
   transaction. For each of these peers, at least one SNMP SetRequest-
   PDU is sent, with only those variables of interest to that peer.

   Each SMUX peer must either accept or refuse the set operation,

without actually performing the operation. If the peer opts to
accept, it sends back an SNMP GetResponse-PDU with an error-status of
noError(0); otherwise, if the peer refuses to perform the operation,
then an SNMP GetResponse-PDU is returned with a non-zero error-
status.

The SNMP agent examines all of the responses. If at least one SMUX
peer refused the operation, then a SMUX SOutPDU is sent to each SMUX
peer, with value rollback, telling the SMUX peer to discard any
knowledge of the requested operation.

Otherwise if all SMUX peers accepted the operation, then a SMUX
SOutPDU is sent to each SMUX peer, with value commit, telling the
SMUX peer to perform the operation.

In either case, the SMUX peer does not generate a response to the
SMUX SOutPDU.

3.1.4. Variables in Requests

When constructing an SNMP GetRequest-PDU or GetNextRequest-PDU for a
SMUX peer, the SNMP agent may send one, or more than one variable in
a single request. In all cases, the SNMP agent should process each
variable sequentially, and block accordingly when a SMUX peer is
contacted.

3.1.5. Request-ID

When the SNMP agent constructs an SNMP GetRequest-PDU,
GetNextRequest-PDU, or SetRequest-PDU, for a SMUX peer, the
request_id field of the SNMP takes a special meaning: if an SNMP
agent generates multiple PDUs for a SMUX peer, upon receipt of a
single PDU from the network management station, then the request_id
field of the PDUs sent to the SMUX peer must take the same value
(which need bear no relationship to the value of the request_id field
of the PDU originally received by the SNMP agent.)

3.1.6. The powerful get-next operator

Each SMUX peer acts as though it contains the entire MIB when
processing a SNMP GetNext-PDU from the SNMP agent. This means that
the SNMP agent must check each variable returned in the SNMP
GetResponse-PDU generated by the SMUX peer to ensure that each
variable is still within the same registered subtree as caused the
SNMP GetNext-PDU to be sent to that peer. For each variable which is
not, the SNMP agent must include it in a SNMP GetNext-PDU to the peer
for the succeeding registered subtree, until responses are available
for all variables within their expected registered subtree.

3.2. Protocol Data Units

   The SMUX protocol data units are defined using Abstract Syntax
   Notation One (ASN.1) [3]:

   SMUX DEFINITIONS ::= BEGIN

   IMPORTS
           DisplayString, ObjectName
                   FROM RFC1155-SMI

           PDUs
                   FROM RFC1157-SNMP;

   -- tags for SMUX-specific PDUs are application-wide to
   -- avoid conflict with tags for current (and future)
   -- SNMP-generic PDUs

   SMUX-PDUs ::=
       CHOICE {
           open   -- SMUX peer uses
               OpenPDU,   -- immediately after TCP open

           close   -- either uses immediately before TCP close
               ClosePDU,

           registerRequest -- SMUX peer uses
               RReqPDU,

           registerResponse -- SNMP agent uses
               RRspPDU,

               PDUs,   -- note that roles are reversed:
                           --   SNMP agent does get/get-next/set
                           --   SMUX peer does get-response/trap

           commitOrRollback -- SNMP agent uses
               SOutPDU
       }

   -- open PDU
   -- currently only simple authentication

   OpenPDU ::=
       CHOICE {
           simple

```
            SimpleOpen
        }

SimpleOpen ::=
    [APPLICATION 0] IMPLICIT
        SEQUENCE {
            version     -- of SMUX protocol
                INTEGER {
                    version-1(0)
                },

            identity    -- of SMUX peer, authoritative
                OBJECT IDENTIFIER,

            description -- of SMUX peer, implementation-specific
                DisplayString,

            password    -- zero length indicates no authentication
                OCTET STRING
        }


-- close PDU

ClosePDU ::=
    [APPLICATION 1] IMPLICIT
        INTEGER {
            goingDown(0),
            unsupportedVersion(1),
            packetFormat(2),
            protocolError(3),
            internalError(4),
            authenticationFailure(5)
        }


-- insert PDU

RReqPDU ::=
    [APPLICATION 2] IMPLICIT
        SEQUENCE {
            subtree
                ObjectName,

            priority    -- the lower the better, "-1" means default
                INTEGER (-1..2147483647),

            operation
```

```
                    INTEGER {
                        delete(0),    -- remove registration
                        readOnly(1),  -- add registration, objects are RO
                        readWrite(2)  --   .., objects are RW
                    }
            }

    RRspPDU ::=
        [APPLICATION 3] IMPLICIT
            INTEGER {
                failure(-1)

               -- on success the non-negative priority is returned
            }

    SOutPDU ::=
        [APPLICATION 4] IMPLICIT
            INTEGER {
                commit(0),
                rollback(1)
            }

    END
```

3.3. Mappings on Transport Service

   The SMUX protocol may be mapped onto any CO-mode transport service.
   At present, only one such mapping is defined.

3.3.1. Mapping onto the TCP

   When using the TCP to provide the transport-backing for the SMUX
   protocol, the SNMP agent listens on TCP port 199.

   Each SMUX PDU is serialized using the Basic Encoding Rules [4] and
   sent on the TCP. As ASN.1 objects are self-delimiting when encoding
   using the BER, no packetization protocol is required.

4. MIB for the SMUX

   The MIB objects for the SMUX are implemented by the local SNMP agent:

        SMUX-MIB DEFINITIONS ::= BEGIN

        IMPORTS
                enterprises
                        FROM RFC1155-SMI
                OBJECT-TYPE
                        FROM RFC1212;

        unix   OBJECT IDENTIFIER ::= { enterprises 4 }

        smux   OBJECT IDENTIFIER ::= { unix 4 }

        smuxPeerTable   OBJECT-TYPE
                SYNTAX   SEQUENCE OF SmuxPeerEntry
                ACCESS   not-accessible
                STATUS   mandatory
                DESCRIPTION
                    "The SMUX peer table."
                ::= { smux 1 }

        smuxPeerEntry   OBJECT-TYPE
                SYNTAX   SmuxPeerEntry
                ACCESS   not-accessible
                STATUS   mandatory
                DESCRIPTION
                    "An entry in the SMUX peer table."
                INDEX   { smuxPindex }
                ::= { smuxPeerTable 1}

        SmuxPeerEntry ::=
            SEQUENCE {
                smuxPindex
                    INTEGER,
                smuxPidentity
                    OBJECT IDENTIFIER,
                smuxPdescription
                    DisplayString,
                smuxPstatus
                    INTEGER
            }

        smuxPindex   OBJECT-TYPE
                SYNTAX   INTEGER
                ACCESS   read-only

```
            STATUS  mandatory
            DESCRIPTION
                "An index which uniquely identifies a SMUX peer."
            ::= { smuxPeerEntry 1 }

    smuxPidentity   OBJECT-TYPE
            SYNTAX  OBJECT IDENTIFIER
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
                "The authoritative designation for a SMUX peer."
            ::= { smuxPeerEntry 2 }

    smuxPdescription OBJECT-TYPE
            SYNTAX  DisplayString (SIZE (0..255))
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
                "A human-readable description of a SMUX peer."
            ::= { smuxPeerEntry 3 }

    smuxPstatus     OBJECT-TYPE
            SYNTAX  INTEGER { valid(1), invalid(2), connecting(3) }
            ACCESS  read-write
            STATUS  mandatory
            DESCRIPTION
                "The type of SMUX peer.

                Setting this object to the value invalid(2) has
                the effect of invaliding the corresponding entry
                in the smuxPeerTable.  It is an implementation-
                specific matter as to whether the agent removes an
                invalidated entry from the table.  Accordingly,
                management stations must be prepared to receive
                tabular information from agents that correspond to
                entries not currently in use.  Proper
                interpretation of such entries requires
                examination of the relative smuxPstatus object."
            ::= { smuxPeerEntry 4 }

    smuxTreeTable   OBJECT-TYPE
            SYNTAX  SEQUENCE OF SmuxTreeEntry
            ACCESS  not-accessible
            STATUS  mandatory
            DESCRIPTION
                "The SMUX tree table."
            ::= { smux 2 }
```

```
        smuxTreeEntry   OBJECT-TYPE
                SYNTAX  SmuxTreeEntry
                ACCESS  not-accessible
                STATUS  mandatory
                DESCRIPTION
                    "An entry in the SMUX tree table."
                INDEX   { smuxTsubtree, smuxTpriority }
                ::= { smuxTreeTable 1}

        SmuxTreeEntry ::=
            SEQUENCE {
                smuxTsubtree
                    OBJECT IDENTIFIER,
                smuxTpriority
                    INTEGER,
                smuxTindex
                    INTEGER,
                smuxTstatus
                    INTEGER
            }

        smuxTsubtree    OBJECT-TYPE
                SYNTAX  OBJECT IDENTIFIER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                    "The MIB subtree being exported by a SMUX peer."
                ::= { smuxTreeEntry 1 }

        smuxTpriority OBJECT-TYPE
                SYNTAX  INTEGER (0..'07ffffff'h)
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                    "The SMUX peer's priority when exporting the MIB
                    subtree."
                ::= { smuxTreeEntry 2 }

        smuxTindex OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                    "The SMUX peer's identity."
                ::= { smuxTreeEntry 3 }

        smuxTstatus     OBJECT-TYPE
                SYNTAX  INTEGER { valid(1), invalid(2) }
```

```
                    ACCESS  read-write
                    STATUS  mandatory
                    DESCRIPTION
                        "The type of SMUX tree.

                        Setting this object to the value invalid(2) has
                        the effect of invaliding the corresponding entry
                        in the smuxTreeTable.  It is an implementation-
                        specific matter as to whether the agent removes an
                        invalidated entry from the table.  Accordingly,
                        management stations must be prepared to receive
                        tabular information from agents that correspond to
                        entries not currently in use.  Proper
                        interpretation of such entries requires
                        examination of the relative smuxTstatus object."
                    ::= { smuxTreeEntry 4 }

            END
```

5. Acknowledgements

    SMUX was designed one afternoon by these people:

```
                Jeffrey S. Case, UTK-CS
                James R. Davin, MIT-LCS
                Mark S. Fedor, PSI
                Jeffrey C. Honig, Cornell
                Louie A. Mamakos, UMD
                Michael G. Petry, UMD
                Yakov Rekhter, IBM
                Marshall T. Rose, PSI
```

6. References

    [1] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple
        Network Management Protocol", RFC 1157, SNMP Research,
        Performance Systems International, Performance Systems
        International, MIT Laboratory for Computer Science, May 1990.

    [2] McCloghrie K., and M. Rose, "Management Information Base for
        Network Management of TCP/IP-based Internets", RFC 1156,
        Performance Systems International and Hughes LAN Systems, May
        1990.

    [3] Information processing systems - Open Systems Interconnection -
        Specification of Abstract Syntax Notation One (ASN.1),
        International Organization for Standardization, International
        Standard 8824, December 1987.

```

   [4] Information processing systems - Open Systems Interconnection -
       Specification of Basic Encoding Rules for Abstract Notation One
       (ASN.1), International Organization for Standardization,
       International Standard 8825, December 1987.

   [5] Rose, M., and K. McCloghrie, "Structure and Identification of
       Management Information for TCP/IP-based Internets", RFC 1155,
       Performance Systems International and Hughes LAN Systems, May
       1990.

7. Security Considerations

   Security issues are not discussed in this memo.

8. Author's Address

   Marshall T. Rose
   Performance Systems International, Inc.
   5201 Great America Parkway
   Suite 3106
   Santa Clara, CA  95054

   Phone: +1 408 562 6222

   EMail: mrose@psi.com
   X.500:  rose, psi, us