NISTIR 4467

SNMPLIB: A SIMPLE NETWORK MANAGEMENT PROTOCOL FUNCTION LIBRARY FOR IBM PC COMPATIBLE COMPUTERS

Robert Crosson

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards
and Technology
Center for Computing and
Applied Mathematics
Gaithersburg, MD 20899

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

NIST

NISTIR 4467

SNMPLIB: A SIMPLE NETWORK MANAGEMENT PROTOCOL FUNCTION LIBRARY FOR IBM PC COMPATIBLE COMPUTERS

Robert Crosson

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards
and Technology
Center for Computing and
Applied Mathematics,
Gaithersburg, MD 20899

November 1990

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
John W. Lyons, Director

SNMPLIB:
A Simple Network Management Protocol Function Library for IBM PC Compatible Computers

by

Robert Crosson
Advanced Distributed Computing Systems Group
Scientific Computing Environments Division
National Institute of Standards and Technology

Abstract

Many Simple Network Management Protocol applications exist, but few general purpose packages run on a personal computer. SNMPLIB is a library of function calls written in Microsoft C version 5.0 for IBM PCs and compatibles. Its data structures and functions are described. With SNMPLIB a user can write a program to dynamically monitor operational variables of compatible networked devices and take action when those variables cross threshhold values. The network input/output routines work with any Ethernet interface for a PC or a compatible for which a packet driver has been installed, making the application somewhat independent of the Ethernet interface used in the computer. A sample application program, SNMPMON, is supplied which monitors user specified variables in user specified devices and writes their values to the PC's display. SNMPLIB and SNMPMON have been compiled on a Sun Microsystems workstation and successfully used there with a similar, but Sun Ethernet interface specific, set of input/output functions.

1.0 Introduction

The Simple Network Management Protocol[1] (SNMP) is becoming widely supported by vendors of Internet Protocol (IP) network interconnection devices as a vehicle for managing the operation of those devices. The management tools provided by the vendors, however, are usually designed to manage only the vendor's equipment. Because they are designed to be very visual and elegant, they frequently run on only specific computers, such as expensive workstations, and their Ethernet interfaces. A few vendor independent management packages are available, but these also require workstations for execution. This project was undertaken to generalize and reduce the cost of managing a network using SNMP.

Note: Mention of brand names does not constitute endorsement of these products in any way.

Two of the project's goals were to develop a simple means of accessing operational parameters of network devices, and to have the application run on a computer which was both inexpensive, when compared to a workstation, and readily available. The PC/MS-DOS based IBM type Personal Computer was chosen as the execution engine because it is so common. And, since Ethernet type networks are the most popular networks linking personal computers, that type of network interface was supported.

Another goal was that the resulting product would be flexible enough to accommodate differing network configurations so users would be able to develop customized network management applications. A sample application would be included both as an example and as a useful tool, should the user not be able to create his own. Being as independent as possible of the brand of Ethernet interface in the computer was an additonal goal.

2.0 SNMPLIB and SNMPMON

SNMPLIB is a library of functions written in Microsoft C (small model), version 5.0, designed to meet the goals of the project. The functions allow the user to create a management tool specific to his network's configuration, while maintaining flexibility about what parameters are accessed. Values of operational parameters of specified network devices can be retrieved and evaluated, and, based on user defined criteria, the retrieved information can be archived or displayed, or alarms can be sounded.

SNMPLIB isolates the user from the mechanics of communicating with the devices. The library functions read the user's database and properly format and transmit the information request messages. SNMPLIB uses the User Datagram Protocol (UDP) over IP to communicate with devices being accessed. The functions also provide methods of accessing the database of returned values so the application can evaluate the current values to determine whether or not some action should be taken.

A generalized application, SNMPMON, is supplied as an example which uses SNMPLIB functions and which can replace a customized program. SNMPMON commands the functions to generate information request messages based on a user created database file of devices and variables. When replies are received, SNMPMON displays on the computer's console the returned values of the variables.

An attempt at Ethernet interface independence was made by writing the network input/output functions so they could communicate with a packet driver. A packet driver is a routine which isolates the interface peculiarities from the application, providing a common set of function calls for every interface, which are accessed with commands in a standard format. A set of packet drivers for various PC Ethernet interfaces is in the public domain (see Appendix C).

3.0 Types of SNMP Data

Network devices which respond to SNMP information requests are called SNMP agents and are grouped into SNMP communities. All of the agents may be in the same community but are not required to be. When SNMP is using the IP protocol suite as the transport protocol, agents within a community are identified by their Internet addresses.

Computers running SNMP applications which generate information requests are called SNMP management entities. Programs which run on management entities and access SNMP agents are management applications.

Agents have operational variables associated with them called a Management Information Base[2] (MIB) (see Appendix B). The variable values are constantly updated as the agent performs its functions. The values of the variables are the data acquired by management applications.

A MIB variable is identified by a name, as defined in reference [2], and, if necessary, by an index. Indices differentiate among variables of the same name in an agent.

'ifOperStatus' (Interface Operational Status) is a MIB variable which requires an index. Most agents have at least two interfaces. Each interface has a variable 'ifOperStatus' that records whether the interface is up, down, or being tested so each interface is given a unique number, or index. The operational status of the first interface is identified by the notation ifOperStatus.1, the second by ifOperStatus.2.

4.0 SNMP Database File Format

To know which variables are associated with which agents, SNMPLIB must be given a database file of agent names and variable names. To identify which variables are to be accessed in which agent, the database information is separated into groups of variables associated with an agent, called an variable list. Groups of variable lists associated with an SNMP community are called a community list. To differentiate between community names, agent names, variable names, and indices, the database file must be generated in a specific format.

4.1 Community and Variable Lists

A community list begins with a community name, which is followed by the one or more variable lists. Community lists are terminated by the end of the database file, or by encountering another community name. Variable lists within a community list are separated by blank lines. Lines containing comments (see 4.6) are not considered blank lines.

Each variable list must be part of a community list. Therefore, the first non-comment and non-blank line must contain a community name. After a community has been specified, that name is used as the default community name for following variable lists until a new community name is encountered. When a new community name is encountered, it becomes the default community name. Any variable lists not immediately following a community name are given the default community name. Community names have unique properties (see 4.2) so that community lists are separated from each other.

A variable list consists of the agent's name followed by the variables and associated indices to be accessed in that agent. The index for a variable must be on the same line as the variable and separated from it only by one or more space or tab characters. No blank lines may appear in a variable list.

An example SNMPLIB database file is in Appendix A.

4.2 Community names

A community name must start at the beginning of a line, and must not begin with the '#' character. Should it do so, it will be considered a comment, not a community name (see "Comments", 4.6). No other names in the SNMP database file start at the beginning of a line. A community name is terminated by a space, tab, or end-of-line character unless the name is enclosed in double quote characters (e.g., "community name"). When a community name begins with a double quote character, i.e., the double quote character is the first character on the line, the community name is terminated by another double quote and so may include space or tab characters. It may not, however, include another double quote character besides the beginning and ending ones. Otherwise, any number of double quote characters may be in a name as long as the first character of the name is not a double quote. Then the name is terminated by a space, tab, or new-line character. The double quote characters enclosing a name are not considered part of the name when SNMP messages are generated.

4.3 Agent names

Since agent names must not start in the first character position on a line, they must be preceded by one or more space or tab characters. If the agent name immediately follows a community name, the agent name may be on the same line as the community name, but separated from it by one or more spaces or tabs. If a community name is not supplied immediately before an agent name, SNMPLIB will assign the default community name to that agent. An agent name is terminated by a space, tab, or end-of-line character.

Because SNMPLIB currently has no agent name resolution capability, agent names can be only the agent's Internet address in standard Internet "dot" notation, such as 192.167.34.13. Since no spaces or tabs are allowed in an agent name, it need not be enclosed in double quotes.

4.4 Variable names

Like agent names, variable names may not begin with the first character on a line, and are terminated by a space, tab, or end-of-line character. Valid variable names are defined in reference [2], and must be entered exactly in the form presented there, including spelling and capitalization (see also Appendix B). Variable names need not be enclosed in double quote characters because no spaces are allowed in them.

Variable names must not be separated from each other or the agent name they are associate with by blank lines. Blank lines separate variable lists from each other.

4.5 Indices

An index must be on the same line as the variable it modifies. It must be separated from the variable name by one or more spaces or tabs. If a variable name has no index on the same line, SNMPLIB assumes no index is to be associated with that variable.

An index is specific to its associated variable. Some variables don't require indices; others require multiple indices (see examples of variables with and without indices in Appendix A). When multiple indices are required, they may be separated by either a single space or comma. When they are separated by a space, the entire set of indices must be enclosed by a pair of double quote characters. When double quotes are not used, the index name is terminated by the first space, tab, or end-of-line character encountered.

4.6 Comments

A comment -- some text in the database which is disregarded by SNMPLIB -- is initiated by a '#' character either at the beginning of a line in the database, or elsewhere in a line preceded by a space or tab character. A comment is terminated by the end of the line on which it was started. A '#' character not at the beginning of a line and not preceded by a space or tab character will not be considered as the beginning of a comment, but will be included in the field currently being scanned.

4.7 Configuration information

The only network interface configuration information currently required to run SNMPLIB is that for a packet driver. Three variables must be provided: the local interface's IP address, the IP

network mask, and the local gateway's IP address. All three values are assigned by the manager of the network on which the host running this application resides.

Configuration information is entered in the database file in a format similar to that of a community list. The community name for the configuration information must be '<Configuration>' ("<>" brackets included). The agent field must be 'pktdvr'. The variables are 'locipaddr=x.x.x.x' for the local IP address, 'ipmask=x.x.x.x' for the IP network mask, and 'gw=x.x.x.x' for the local gateway's IP address. In the examples each 'x' is an integer between 0 and 255, inclusive. The single quote characters (') are neither required nor allowed.

5.0 SNMPLIB Data Structures

5.1 The SNMP Database in Memory

When SNMPLIB reads the user's database file (as described in 6.1), a new database is created in memory. The data from the file are arranged in memory in agent variable lists, which are linked together in the same order in which they appeared in the file. Each agent's name has associated with it a block of data containing the agent's name (i.e., its Internet address) as a null-terminated string of ASCII characters, its community in a similar ASCII string, and a pointer to a string of variable information blocks.

Variable information blocks are allocated, one to each variable, for an agent. A variable information block contains the following information: the variable name as a string of ASCII characters terminated by a null character, the variable's index if any, a pointer to the variable's current value (the storage for the value is allocated separately and is the correct size for that variable), a flag indicating if the variable's value has changed since the value was last accessed by the user's application, and a time stamp indicating the last time the value was set.

The flag is incremented every time the value changes, so it may be different by more than one if the value changed multiple times since it was last accessed by the user's application. If the value hasn't changed since it was last read by the user, the value will not have changed. (Because the flag is a 16-bit signed integer, it would be very unlikely that it had overflowed and been incremented to its previously read value.) Usually the application sets the change flag to zero rather than remembering the value when it was last accessed.

The time stamp is a count of the number of seconds since some reference time. The reference time may change among machines. In MS/PC-DOS machines the reference time is January 1, 1970.

5.2 The Management Information Base

Another important SNMPLIB data structure is the object information structure. It contains a pointer to a character string, a general pointer which may point to various objects or be null, a flag telling what type of object the general pointer is indicating, an flag for the type of value (integer, counter, string, etc.) the variable has associated with it, and a pointer to a function which returns an integer. When SNMPLIB generates its working version of the MIB, each variable, whether it has a value associated with it (ifOperStatus) or it is made up of other variables (ifEntry), has an object information structure.

5.2.1 MIB Structure Elements

In the MIB, the object name pointer points to the null-terminated ASCII string containing the variable's name. The flag variable describes the type of object pointed to by the general pointer. If the flag is zero, the defined variable has a value and the general pointer may be either null or a pointer to an array of pointers to null-terminated ASCII strings. The strings are ASCII representions of the numerical values of the variable. If the value is a one, the pointer in the array with an index of one points to the proper ASCII string. This is similarly true for all positive values of the variable.

If the flag is a positive one, the variable has no value, but represents an array of other MIB variables in an array of object information structures, the first member of which is indicated by the general pointer. If the flag is negative, the general pointer points to an object information structure of a different type than if the flag is a positive one (see 5.2.2).

The type field is used to indicate, when applicable, what types of values (integer, string, counter, etc.) can be taken by the MIB variable defined by the structure. The function pointer, initially null, may be used to point to a routine which performs special processing of the specific MIB variable's value.

5.2.2 The MIB array of structures

The MIB as implemented in this application is simply a mechanism for translating from MIB variables to object identifiers in an SNMP message, and vice versa. It consists of arrays of object information structures linked together in the form of an inverted tree.

At each level in the tree is an array of structures. There is a structure at the appropriate level in the MIB for each variable. The general pointer of the first structure in each array indicates the structure at the next higher level which points to this array. A structure with null pointers and a zero flag variable is the last structure in each array. At the higher levels in the tree the

general pointers in intervening structures in an array point to first structures in lower level arrays of structures. At the lowest level in the tree, the general pointers of intervening structures are null except when the values of a variable can be represented by null-terminated ASCII character strings (see 5.2.1).

Since the flag variable indicates whether or not the general pointer in the structure points to a lower level array of structures, the MIB tree can be traversed downward until the bottom is reached. At that point a MIB variable has been defined which must have a value. Since the first structure in any array of structures points upward in the tree, the tree can be traversed upward, also. These two properties are used to encode an object identifier in an SNMP message -- a series of numbers identifying the variable -- from the name of the variable, or to decode an object identifier to determine the name of the variable represented.

5.2.3 Using the MIB

An SNMP object identifier is a series of integers separated by periods. Each variable in the MIB has a unique SNMP identifier. For all MIB variables, the first five integers in an SNMP identifier are 43, 6, 1, 2, and 1. The MIB variable 'ifOperStatus' has the identifier 43.6.1.2.1.2.2.1.8. From the top of the MIB, one must descend through the MIB variables 'interfaces', 'ifTable', and 'ifEntry' to get to 'ifOperStatus'. 'interfaces' is the second entry in the MIB's top level, so it produces the 2 after the sequence 43.6.1.2.1. 'ifTable' is the second entry in 'interfaces', and generates the following 2. 'ifEntry' is the first entry in 'ifTable' to account for the next 1. 'ifOperStatus' is the eighth entry in 'ifEntry', for the final 8. This produces the SNMP object identifier for 'ifOperStatus' of 43.6.1.2.1.2.2.1.8.

5.2.3.1 Decoding identifiers

For use in SNMPLIB, the MIB tree therefore is arranged so that in each array of structures at any level, the array element index for the variable of interest at that level corresponds to its equivalent digit in the SNMP identifier. Given that the MIB is identified by 43.6.1.2.1, the 'interfaces' structure's index is 2 to correspond to the next integer in the identifier, the 'ifTable' structure's index is 2 in its array of structures, the 'ifEntry' structure's index is 1, and the 'ifOperStatus' structure's index is 8. Thus, 43.6.1.2.1.2.2.1.8 eventually leads to the 'ifOperStatus' variable.

5.2.3.2 Encoding identifiers

Because SNMP identifier encoding rules prevent any integer in an identifier from being zero, the zeroeth element in each structure array is not used to decode an identifier. As described in 5.2.2, SNMPLIB uses it for encoding one. In the MIB, the general pointer in the structure for the variable 'ifEntry' points to the array of

structures containing the one for 'ifOperStatus'. However, it really points to the first element -- the one with an index of zero -- in the array of structures. Since the general pointer in that structure will never point to an object because its index is zero, it can be used to point "backwards" to the structure for 'ifEntry'.

The same is true for the first structure in the array containing 'ifEntry'; it points to the structure for 'ifTable'. By counting the number of structures in an array between the one for the variable of interest and the first structure, the index of the interesting structure can be determined. For 'ifEntry' no structures are between it and the first, so its index is one. 'ifOperStatus' has seven structures between it and the first structure, so its index is eight. By finding a structure's index at each level starting from the bottom of the tree, any variable's SNMP identifier can be constructed in reverse order.
When the general pointer in the first structure in an array of structures is a null pointer and its flag is a negative sixteen, the top of the MIB tree has been reached. The MIB prefix of five integers can then be added to the existing list of indices to produce a complete SNMP identifier.

To ease finding a specific variable's structure in the MIB, a set of pointers are provided. One pointer is available for each variable so its structure can be found without traversing the set of MIB structures.

The object information structure pointer 'mibroot' points to the highest level MIB array of structures. Its member structures are for the variables 'system', 'interfaces', 'at', 'ip', 'icmp', 'tcp', 'udp', and 'egp'. In the current form, the branches of the MIB tree for 'icmp', 'tcp', 'udp', and 'egp' are not implemented

5.3 SNMP Trap Message Structures

When SNMP agents generate unsolicited information, or trap, messages, the type of message is indicated by an integer in the message. This integer is used as the index in an array of object information structures. The function pointer in the selected structure points to the function which will perform a task appropriate to the type of trap message received.

If the variable specified in the trap message is one being monitored by the users program, that variable's value will be modified to match that in the trap message. The change flag and timestamp will also be updated. If the variable in the trap message is not being monitored, a message is displayed on the console indicating the content of the message.

5.4 The Socket Data Structure

To store information related to the types of active network connections while the application is running, a socket data structure is created for each connection. The concept of a socket is used so that multiple connections from the computer running the application to other devices may be maintained simultaneously. A received packet contains a remote socket number which associates it with a program running on the remote device, and a local socket number which links it with the application running on the local computer.

The elements of the socket data structure are 1) a pointer to another socket data structure (so structures can be linked together), 2) a pointer to an agent information structure as previously described in the section 5.1, 3) a socket identifier (also called a network descriptor), 4) a request identifier number used for linking a reply message to a request message, 5) a network interface driver option variable, 6) the remote computer's Internet address, 7) the remote computer application's socket number, 8) the local socket number, and 9) a network protocol indicator (indicates UDP for SNMPLIB).

These structures are linked together using the socket data structure pointers, with the variable 'socketroot' pointing to the beginning of the chain.

5.5 Packet Driver Data Structures

The part of the program which communicates with the packet driver for the network interface also uses a number of structures and queues. An Address Resolution Protocol (ARP) table of Internet address and Ethernet address equivalences is maintained through a chain of 'arpentry' structures. Packets received by the packet driver are stored on a packet queue, from which they may be removed and added to either the UDP packet queue or the Transmission Control Protocol queue. Internet Control Message Protocol packets and ARP packets are processed immediately, so they require no queues.

6.0 SNMPMON Operation

SNMPMON is an application program using the SNMPLIB functions to access a set of specific variables within some SNMP agents. It is an example of how a user can create a custom network management tool. Its operation is explained below. Source code for both SNMPMON and SNMPLIB is available from the author.

SNMPMON requires as a command-line parameter the name of a database file containing the names of the communities, agents, and variables to be accessed. When SNMPMON runs using a packet driver as the software interface to the Ethernet interface, the database file must also contain interface configuration information (see 4.7).

6.1 Beginning Execution

When SNMPMON starts, it checks for the database file name. If one is supplied, the name is passed to the SNMPLIB initialization routine 'initsnmp()'. This routine initializes the SNMP data structures and creates the memory-resident copy of the SNMP agent and variable database. Upon the successful return from 'initsnmp()', the routine 'pollagents()' is called to interrogate the specified agents for their values of the indicated variables. If the call to 'pollagents()' completes successfully, the memory-resident database of agent variables may be accessed and the values tested for any abnormalities. SNMPMON prints the values of the variables on the console using 'showall()'.

6.2 Showinfo

The routine 'showall()' is a special invocation of 'showinfo()'. 'showinfo()' demonstrates how a user can access the memory-resident SNMP agent and variable database. First, 'showinfo()' asks for a pointer to the first agent information block by calling 'getnxtroutp()' with a null argument. It then passes to 'getnxtvarp()' the pointer from the call to 'getnxtroutp()' and a null variable information block pointer to get a pointer to the first variable information block for the specified agent.

The operational flag passed to 'showinfo()' indicates whether all variable values are to be displayed. If not all variables are being shown, only those which have changed in value will be displayed. If the value should not be displayed, the 'getnxtvarp()' is called to get a pointer to the next variable information block.

If the value should be shown, and if the agent's address hasn't yet been displayed, the agent's name from the agent information block is retrieved using 'getroutaddr()' and is displayed. Then the variable's name and index are retrieved using 'getvarstr()' and 'getinxstr()', and are displayed. Finally the variable's value is to be accessed. First, though, it should be determined whether the value can be represented as a character string, such as "up" or "down", instead of its numerical value. A routine, 'xlatevalue()', is available to determine if a string representation of the value exists.

'xlatevalue()', when passed a pointer to a variable information block, will try to return a pointer to a null-terminated ASCII string representing the variable's value. If the value can't be translated, a null pointer will be returned. 'showinfo()' calls 'xlatevalue()' to attempt to retrieve an ASCII string for the variable information block of interest. If it is successful, the string is shown on the console. If it is not successful, the variable's actual value must be displayed. Retrieving the value and determining how to display it requires more work.

First, the type of variable is determined by calling 'getvartype()'. This returns a pointer to a null-terminated ASCII string of characters which indicate the variable type. Four types of variables are currently supported, unsigned long integers, integers, hexadecimal strings of characters, and ASCII strings of characters. 'getvartype()' returns pointers to the strings "unsigned long", "int", "hex string", or "ASCII string", respectively. From the content of the indicated string, 'showinfo()' determines what type of memory storage must be allocated to retrieve the variable's value.

When appropriate memory storage has been obtained, 'getvarval()' is called to duplicate the value in the storage location. From there it can be displayed correctly. Finally, the variable's time stamp is retrieved using 'getvartime()', is interpreted in the local manner, and is displayed.

After the time stamp has been displayed, the variable's change flag is set to zero and 'getnxtvarp()' is called again to find the next variable to process. Failing to find another variable causes 'getnxtroutp()' to be called to find the next agent information block. When the last variable in the last agent information block has been processed, 'showinfo()' terminates and control is returned to 'main()'.

6.3 Back to main

When 'main()' resumes execution, it checks to see if any unsolicited packets, such as SNMP trap messages, have been received by calling the 'rcvpkts()' routine. If there were no execution errors and no packets received, as determined by the value returned by 'rcvpkts()', the section of the program is entered which counts the number of times the current loop has been executed. If the count is not a multiple of sixty, a character is displayed on the console to show that the program is still executing properly, and control is passed to the keyboard servicing routine.

If the count is a multiple of sixty, 'pollagents()' is called again to obtain new values for the database variables. If 'pollagents()' has been called ten times without calling 'showall()', 'showall()' is also executed. Otherwise, 'showchgs()' is called to show on the console only those variable values that have changed since the last 'pollagents()' invocation.

Next, keyboard activity is queried. If an 's' (show) has been entered, 'showall()' is called. If a 't' (time) is returned, the current time is displayed. When an 'a' (acknowledge) is entered, any alarms that might be sounding (see next paragraph) are disabled. If a 'q' (quit) has been hit, the variable 'endflag' is incremented to notify the program to exit gracefully. If no key had been pressed, the routine to check for alarm situations is entered.

An alarm situation is defined in SNMPMON as when the operational status of an agent interface has been retrieved and it was not "up". If such a situation is discovered, an alarm is sounded once a second for the first five seconds, and once every minute thereafter until an 'a' is entered on the keyboard or the alarm condition goes away. Alarm conditions are checked by calling the routine 'ckopstat()'.

6.4 Ckopstat

In a manner similar to that used in 'showinfo()', 'ckopstat()' uses the routines 'getnxtroutp()' and 'getnxtvarp()' to get pointers to the agent and variable information blocks in the memory-resident SNMP agent and variable database. The name of each variable is retrieved by invoking 'getvarstr()' and is compared to 'ifOperStatus', the MIB variable for an interface's operational status. If the comparison is exact, the variable's value is checked. If the value is not "up", 'ckopstat()' returns a minus one; otherwise, it goes to the next variable. If no variables are 'ifOperStatus', or all 'ifOperStatus' variables are "up", a zero is returned.

6.5 Main again

If 'ckopstat()' returns a non-zero value, variables are set to sound the alarm as previously described in 6.3. Then 'main()' calls 'wait()' to wait for 1000 milliseconds. When this completes the loop variable 'endflag' is checked. If it is zero, execution returns to the beginning of the loop. If it is not zero, the loop is exited and 'endsnmp()' is called. This gracefully terminates independent SNMPLIB functions, such as the packet driver, so that the main program can exit without leaving the packet driver in a state which might interfere with the operating system. Then SNMPMON exits successfully.

6.6 SNMPMON Source Code

Two comments should be made about the source code for SNMPMON. Commented out debugging statements exist that should not confuse the reader. Also, conditional compilation statements have been inserted so that the same code could be compiled on a Sun Microsystems workstation with the Gnu C compiler. Compiling the code without defining the 'BSD' variable produces a program configured to execute on a computer running the MS/PC-DOS operating system.

7.0 Packet Driver Source Code

Writing this program to use a packet driver required a small amount of assembly language programming. This is because packet drivers do not generate their own buffers for storing incoming packets, nor do they maintain a queue of filled packet buffers. Both of these functions must be performed by the application program. The two SNMPLIB routines for doing these functions, when written in C,

produced run-time errors from stack pointer checking. Since stack checking could not be disabled, the routines were written in assembly language instead.

The routine supplying empty packet buffers was written to eliminate any outside subroutine calls, especially to the operating system. It was feared that the asynchronous nature of packet arrivals and subsequent packet buffer requests might confuse or corrupt the operating system. Thus, the number of packet buffers are defined at the time of assembly and cannot be increased or decreased dynamically.

8.0 SNMPLIB Subroutine Descriptions

Brief descriptions of the SNMPLIB functions are in Appendix D

Appendix A

A Sample SNMPLIB Database File

```
# an snmp database file
<Configuration>         # configuration information
        pktdvr          # packet driver configuration information
                gw=223.14.1.254 # local gateway's IP address
                locipaddr=223.14.1.132  # host's IP address
                ipmask=255.255.255.0    # local net's IP mask

# snmp database
# The following parameters were tested and worked correctly unless
#       noted in a following comment.  Note that limits set by the
#       protocol on the size of SNMP packets permit only about four
#       variables per agent variable list.  However, multiple agent
#       variable lists for the same agent (with the agent name
#       reproduced for each list), can be entered in this file.
# Note: The agent throws away the request and doesn't answer if it
#       doesn't understand the question or it doesn't know the
#       answer.
#
#
TestComm                # first agent's community
        223.14.1.126    # first agent's IP address
                sysDescr        # ex.,
#                               # value = "Portable Gateway s/n 1234"
                sysObjectID     # ex., value = "43.6.1.4.1.1.1.1.42"
                sysUpTime       # ex., value = 52274315
#               ifType          1       # ex.,
#                                       #   value = "ethernet-csmacd"
#               ifType          2       # works
#               ifType          3       # works
#               ifOperStatus    1       # ex., value = "up", "down",
#                                       #       "testing"
#               ifOperStatus    2       # works
#               ifOperStatus    3       # works
#               ifInErrors      1       # ex., value = 27
#               ifOutErrors     1       # ex., value = 19
#               ifInErrors      2       # works
#               ifOutErrors     2       # works
#               ifInErrors      3       # works
#               ifOutErrors     3       # works
#               ifMtu           1       # ex., value = 1514
#               ifSpeed         1       # ex., value = 10000000
#               ifMtu           2       # works
#               ifSpeed         2       # works
#               ifMtu           3       # works
#               ifSpeed         3       # works
#               ifPhysAddress   1       # ex.,
#                                       #   value = 00-00-c0-16-39-fe
#               ifAdminStatus   1       # ex., value = "up", "down",
#                                       #       "testing"
```

```
#               ifPhysAddress   2       # works
#               ifAdminStatus   2       # works
#               ifPhysAddress   3       # works
#               ifAdminStatus   3       # works
#               ifInOctets      1       # ex., value = 15485129
#               ifInUcastPkts   1       # ex., value = 2890
#               ifInOctets      2       # works
#               ifInUcastPkts   2       # works
#               ifInOctets      3       # works
#               ifInUcastPkts   3       # works
#               ifInNUcastPkts  1       # ex., value = 1936542
#               ifInDiscards    1       # ex., value = 276
#               ifInNUcastPkts  2       # works
#               ifInDiscards    2       # works
#               ifInNUcastPkts  3       # works
#               ifInDiscards    3       # works
#               ifInErrors      1       # ex., value = 17
#               ifInUnknownProtos       1       # ex., value = 3
#               ifInErrors      2       # works
#               ifInUnknownProtos       2       # works
#               ifInErrors      3       # works
#               ifInUnknownProtos       3       # works
#               ifOutOctets     1       # ex., value = 19277695
#               ifOutUcastPkts  1       # ex., value = 10
#               ifOutOctets     2       # works
#               ifOutUcastPkts  2       # works
#               ifOutOctets     3       # works
#               ifOutUcastPkts  3       # works
#               ifOutNUcastPkts 1       # ex., value = 1643890
#               ifOutDiscards   1       # ex., value = 9
#               ifOutNUcastPkts 2       # works
#               ifOutDiscards   2       # works
#               ifOutNUcastPkts 3       # works
#               ifOutDiscards   3       # works
#               ifOutErrors     1       # ex., value = 65
#               ifOutQLen       1       # ex., value = 3
#               ifOutErrors     2       # works
#               ifOutQLen       2       # works
#               ifOutErrors     3       # works
#               ifOutQLen       3       # works
#               ifLastChange    1       # agent returns wrong type
#                                       # returns INT instead of
#                                       # TimeTicks, doesn't work
#               atIfIndex       "2 223.14.1.33" # ex., value = 2
#               atIfIndex       "3,223.14.4.2"  # works
#               atNetAddress    "2 08:00:20:00:0e:c8"   # cannot
#                                       #   poll, doesn't work
#               atPhysAddress   "2 223.14.1.33" # on interface 2
#                                       #   return the physical address
#                                       #   for the given Internet
#                                       #   address, ex.,
#                                       # value = 00-00-c0-f9-16-02
```

```
#               atPhysAddress   "3,223.14.2.33" # works, used comma
#                                       instead of space
#               ipForwarding    # ex., value = 1 (forwarding), 2 (not)
#               ipRouteDest     223.14.1.0      # ex.,
#                                       #   value = 192.18.3.1
#               ipRouteDest     223.14.2.0      # works
#               ipRouteDest     223.14.3.0      # works
#               ipRouteDest     0.0.0.0         # agent says
#                                       #   the request had bad format,
#                                       #   doesn't work
#               ipRouteDest     223.14.3.1      # cannot poll
#               ipRouteAge      223.14.4.0      # works
#               ipDefaultTTL    # ex., value = 255
#               ipInReceives    # ex., value = 2054
#               ipInHdrErrors   # ex., value = 10
#               ipInAddrErrors  # ex., value = 0
#               ipForwDatagrams # ex., value = 192598
#               ipInUnknownProtos       # ex., value = 0
#               ipInDiscards    # ex., value = 25
#               ipInDelivers    # ex., value = 95041
#               ipOutRequests   # ex., value = 10548
#               ipOutDiscards   # ex., value = 3
#               ipOutNoRoutes   # ex., value = 27
#               ipReasmTimeout  # agent returns wrong variable
#                               #   type, doesn't work
#               ipReasmReqds    # ex., value = 99
#               ipReasmOKs      # ex., value = 105
#               ipReasmFails    # ex., value = 4
#               ipFragOKs       # ex., value = 125
#               ipFragFails     # ex., value = 1
#               ipFragCreates   # ex., value = 15
#               ipAdEntAddr     223.14.1.126    # ex.,
#                                       #   value = 225.6.119.54
#               ipAdEntIfIndex  223.14.1.126    # ex., value = 2
#               ipAdEntNetMask  223.14.1.126    # ex.,
#                                       #   value = 255.255.255.0
#               ipAdEntBcastAddr        223.14.1.126
#                                       # ex.,
#                                       #   value = 255.255.255.255
#               ipRouteDest     223.14.1.0      # ex.,
#                                       #   value = 192.3.26.250
#               ipRouteIfIndex  223.14.1.0      # ex., value = 1
#               ipRouteMetric1  223.14.1.0      # ex., value = 5
#               ipRouteMetric2  223.14.1.0      # works
#               ipRouteMetric3  223.14.1.0      # works
#               ipRouteMetric4  223.14.1.0      # works
#               ipRouteNextHop  223.14.1.0      # ex.,
#                                       #   value = 224.16.193.1
#               ipRouteType     223.14.1.0      # ex.,
#                                       #   value = 1 ("other"),
#                               #  2 ("invalid"), 3 ("direct"),
#                               #  4 ("remote")
```

```
#               ipRouteProto    223.14.1.0      # ex.,
#                                       #   value = 1 ("other"),
#                                       2 ("local"), 3 ("netmgmt"), 4 ("icmp"),
#                                       5 ("egp"), etc.
#               ipRouteAge      223.14.1.0      # ex., value = 250

comm2                                           # another community
        223.14.2.1                              # an agent name
                ifType          2               # a variable list
                ifOperStatus    2
#               ifInErrors      2               # commented out
#               ifOutErrors     2               # commented out

        223.14.3.253                            # a new agent name,
#                                               same community as before
                ifType          3               # a variable list
                ifOperStatus    3
#               ifInErrors      3               # commented out
#               ifOutErrors     3               # commented out

comm2                                           # restating the community,
#                                               not really necessary, but
#                                               permitted
        223.14.1.126
                ifIndex         1
                ifDescr         1
                ifType          1
```

Appendix B

Management Information Base Variables Implemented in SNMPLIB

Top level variables

- system
- interfaces
- at
- ip
- icmp (not implemented)
- tcp (not implemented)
- udp (not implemented)
- egp (not implemented)

System variables

- sysDescr
- sysObjectID
- sysUpTime

Interface variables

- ifNumber
- ifTable

ifTable variables

- ifEntry

ifEntry variables

- ifIndex
- ifDescr
- ifType
- ifMtu
- ifSpeed
- ifPhysAddress
- ifAdminStatus
- ifOperStatus
- ifLastChange
- ifInOctets
- ifInUcastPkts
- ifInNUcastPkts
- ifInDiscards
- ifInErrors
- ifInUnknownProtos
- ifOutOctets
- ifOutUcastPkts
- ifOutNUcastPkts
- ifOutDiscards
- ifOutErrors
- ifOutQLen

at variables

atTable

atTable variables

atEntry

atEntry variables

atIfIndex
atPhysAddress
atNetAddress

ip variables

ipForwarding
ipDefaultTTL
ipInReceives
ipInHdrErrors
ipInAddrErrors
ipForwDatagrams
ipInUnknownProtos
ipInDiscards
ipInDelivers
ipOutRequests
ipOutDiscards
ipOutNoRoutes
ipReasmTimeout
ipReasmReqds
ipReasmOKs
ipReasmFails
ipFragOKs
ipFragFails
ipFragCreates
ipAddrTable
ipRoutingTable

ipAddrTable variables

ipAdEntry

ipAdEntry variables

ipAdEntAddr
ipAdEntIfIndex
ipAdEntNetMask
ipAdEntBcastAddr

ipRoutingTable variables

ipRouteEntry

ipRouteEntry variables

ipRouteDest
ipRouteIfIndex
ipRouteMetric1
ipRouteMetric2
ipRouteMetric3
ipRouteMetric4
ipRouteNextHop
ipRouteType
ipRouteProto
ipRouteAge

Appendix C

The Clarkson packet driver collection

Availability

The Clarkson collection of packet drivers is available by FTP, by archive-server, Fido file request, and by modem. It comes in two flavors -- executables only (drivers.arc), and source+executables (driverss.arc). All of the following instructions apply to both drivers.arc and driverss.arc.

FTP:

sun.soe.clarkson.edu:/pub/ka9q/drivers.arc
grape.ecs.clarkson.edu:/e/tcpip/drivers.arc

Archive-server:

Send mail to archive-server@sun.soe.clarkson.edu and put the following command as the body of your message:

```
help
```

This will send you a help message. Reading this help message will tell you how to fetch the packet drivers.

Modem:

Call the Clarkson Heath User's Group's BBS: (315) 268-6667, 8N1, 1200/2400 Baud, 24 hours. Change to file area 24 and download drivers.arc.

Opus:

260/360 in the Nodelist. Drivers.arc is file requestable.

Appendix D

Descriptions of SNMPLIB functions.

SNMPLIB user interface routines.

Function: int clrvarchfl(struct varinfo *)
Parameter: a pointer to an SNMP varible information structure.

Description:
Clear the value of the change flag in the indicated variable information block.

Returned value: zero on success; negative values indicate errors.

Function: int endrcv(void)
Parameters: none

Description:
Clean up before stopping the receipt of SNMP packets by releasing all UDP network descriptors. 'endsnmp()' is preferred.

Returned value: always zero.

Function: int endsnmp(void)
Parameters: none

Description:
Clean up all loose ends and turn off the UDP packet driver.

Returned value: always zero.

Function: int getinxstr(struct varinfo *, char *, int)
Parameters: 1. a pointer to an SNMP variable information structure.
2. a pointer to an empty variable index string buffer.
3. the length in bytes of the index buffer.

Description:
Given a pointer to a variable information block, a pointer to a character buffer, and its length, put the index of the variable information block into the buffer in the form of a null-terminated ASCII string.

Returned value: the number of characters put into the buffer, not including the '\0', on success; a negative number on error.

Function: struct varinfo *getnxtchange(struct routinfo *,
struct varinfo *)
Parameters: 1. a pointer to an agent information structure.
2. a pointer to a variable information structure.

Description:
Given a pointer to an agent information block and a pointer to a variable information block associated with that agent, return a pointer to the next variable block associated with the same agent in which the change flag is set. If the given pointer to a variable block is NULL, return the first variable block.

Returned value: a pointer to a variable information structure, or a null pointer on error or no block left.

Function: struct routinfo *getnxtroutp(struct routinfo *)
Parameter: a pointer to an agent information structure.

Description:
Find the next agent information structure in the chain after the given agent information structure. If the given agent information structure pointer is a null pointer, find the first agent information structure in the chain.

Returned value: a the pointer to the next agent information structure on success, or a null pointer on error or no structures left.

Function: struct varinfo *getnxtvarp(struct routinfo *,
struct varinfo *)
Parameters: 1. a pointer to an agent information structure.
2. a pointer to a variable information structure.

Description:
Given a pointer to an agent information block and a pointer to one of its variable information blocks, return a pointer to the next variable information block for that agent. If the given variable information block pointer is a null pointer, return the first variable information block for the given agent.

Returned value: a pointer to a variable information structure on success, or a null pointer on error or no blocks left.

Function: int getroutaddr(struct routinfo *, char *, int)
Parameters: 1. a pointer to an agent information structure.
2. a pointer to an empty router address buffer.
3. the length of the router address buffer in bytes.

Description:
Put the address of the agent in the agent information structure in the empty buffer as a null-terminated string of ASCII characters in "dot" notation.

Returned value: the number of characters, not including the '\0', put into the buffer on success, or a negative number on error.

Function: struct routinfo *getrouterp(char *, struct routinfo *)
Parameters: 1. a pointer to an IP address in ASCII string form.
2. a pointer to an agent information structure.

Description:
Return a pointer to the agent information block which not only follows the given agent information block, but which also has an address which matches the agent address in the null-terminated ASCII string. If the agent information block pointer is a null pointer, return a pointer the the first agent information block with an address that matches the address in the ASCII string.

Returned value: a pointer to an agent information block on success; a null pointer on errors.

Function: int getvarchfl(struct varinfo *)
Parameter: a pointer to a variable information structure.

Description:
Return the value of the change flag of the indicated variable information block. The value of the flag is not changed. Also see 'clrvarchfl()'.

Returned value: the current value of the change flag (a positive number if the variable has changed since it was last accessed); a zero if no change has occurred; a negative number on errors.

Function: struct varinfo *getvarp(char *, char *, char *)
Parameters: 1. a pointer to an IP address in null-terminated ASCII string form.
2. a pointer to an SNMP variable name in null-terminated ASCII string form.
3. a pointer to an SNMP variable index in null-terminated ASCII string form.

Description:
Given pointers to null-terminated ASCII strings containing an agent address, a variable name, and an index, return a pointer to a matching variable information block.

Returned value: a pointer to a variable information structure on success, or a null pointer on errors.

Function: int getvarstr(struct varinfo *, char *, int)
Parameters: 1. a pointer to a variable information structure.
2. a pointer to an empty variable name buffer.
3. the length of the name buffer in bytes.

Description:
Given a pointer to a variable information block, a pointer to a character buffer, and its length, put the name of the variable in the information block into the buffer as a null-terminated ASCII string.

Returned value: the number of characters put into the buffer, not including the '\0', on success, or a negative number on errors.

Function: unsigned long getvartime(struct varinfo *)
Parameter: a pointer to a variable information structure.

Description:
Return the time stamp of the variable associated with the variable information block.

Returned value: the time stamp of the indicated variable as an unsigned long integer on success, or a negative number on errors.

Function: char *getvartype(struct varinfo *)
Parameter: a pointer to a variable information structure.

Description:
Return a pointer to a null-terminated ASCII character string which indicates the type of the variable in the variable information block.

Returned value: a pointer to a null-termianted ASCII string containing the type of variable on success, or a null pointer on errors.

Function: int getvarval(struct varinfo *, void *, int)
Parameters: 1. a pointer to a variable information structure.
2. a pointer to a variable value storage area.
3. the size of the variable value storage area in bytes.

Description:
Copy the value of the variable in the variable information structure into the indicated storage location.

Returned value: the number of bytes used for the value, or a negative number on error.

Function: int initrcv(void)
Parameters: none

Description:
Enable the SNMP monitoring code to receive SNMP trap packets.

Returned value: zero on success; a negative number on errors.

Function: int initsnmp(char *)
Parameter: a pointer to an ASCII string containing the name of the SNMP data base file.

Description:
Given a pointer to a null-terminated ASCII string containing the name of the SNMP database file, pass the file name to the routine which creates the memory-resident SNMP database and initialize the MIB database and the UDP driver.

Returned value: zero on success; a negative number on errors.

Function: int pollagents(void)
Parameters: none

Description:
Interrogate the agents in the SNMP database file by sending SNMP request packets and receiving the replies, thereby updating the memory-resident SNMP database of variable values.

Returned value: zero on success; a negative number on errors.

Function: int prhexstr(unsigned char *, int, char *, int)
Parameters: 1. a pointer to a string of binary bytes.
2. the number of binary bytes to convert.
3. a pointer to an empty ASCII character buffer.
4. the size of the character buffer in bytes.

Description:
Given a pointer to a string of bytes, the number of bytes to convert, the buffer to put them in, and the size of the buffer, interpret the string as hexadecimal data to be converted to a null-terminated ASCII string in the given buffer.

Returned value: the number of bytes used in the buffer, or a negative number on error.

Function: int rcvpkts(void)
Parameters: none

Description:
Check to see if any packets have been received. If there are packets in the received queue, process them (i.e., respond to ARP or ICMP packets immediately, queue UDP and TCP packets). If there are replies to transmitted packets expected which haven't been received yet, loop for 5 seconds waiting for the replies. If the replies aren't received eventually, stop listening for them and go back to listening for trap packets.

Returned value: a positive number indicates packets were processed; zero indicates there were no packets to process; a negative number indicates errors.

Function: char *xlatevalue(struct varinfo *)
Parameter: a pointer to a variable information structure.

Description:
Translate the numeric value of the variable in the variable information block into a null-terminated ASCII character string suitable for sending to a console.

Returned value: a pointer to the character string on success; a null pointer on not being able to translate the value or on an error.

Function: int initbkrefs(void)
Parameters: none.

Description:
Initialize the forward references in the MIB tree structure that couldn't be handled by the compiler.

Returned value: zero.

Function: int initmib(char *)
Parameter: a pointer to a null-terminated ASCII string containing the file name of the SNMP database.

Description:
Initialize the Management Information Base (MIB) by calling initbkrefs() and setting the pointer for 'ifIndex' to 'ifIndex_f'. Forward the name of the SNMP database file passed to it to the routine that creates the in-memory SNMP database.

Returned value: zero on success; a negative number on error.

Function: long atonetaddr(char *)
Parameter: a pointer to a null-terminated ASCII string of characters containing an Internet address.

Description:
Convert a null-terminated ASCII string of characters representing an Internet address in 'dot' notation to a 32-bit binary number in network order containing the Internet address of the characters passed.

Returned value: a long integer containing the binary network address, or a long -1 (which is an illegal Internet address) if the given ASCII address was too long.

Function: int endudp(void)
Parameters: none.

Description:
Call 'udpreleaseall()' to release all UDP network descriptors and associated sockets.

Returned value: always zero.

Function: unsigned char *getcmunp(int)
Parameter: a network descriptor.

Description:
Return the pointer to the null-terminated ASCII character string containing the SNMP community name associated with the descriptor.

Returned value: a pointer to a null-terminated ASCII string, or null pointer on error.

Function: int getconninfo(int, struct sockdata *)
Parameters: 1. a network descriptor.
2. a pointer to a socket data structure.

Description:
Fill in the socket data structure with the information about the connection associated with the descriptor.

Returned value: zero on success, or a negative number if the network descriptor is not allocated.

Function: int getnxtsendnd(int)
Parameter: a network descriptor.

Description:
Given an integer network descriptor for sending packets, find the next network descriptor for sending packets in the socket information table. If the given network descriptor is -1, find the first such network descriptor in the table.

Returned value: the next send network descriptor, or a negative number if none can be found.

Function: int getnxttrapnd(int)
Parameter: a network descriptor.

Description:
Given a network descriptor for receiving trap packets, return the next trap network descriptor in the socket information table. If the network descriptor passed to this routine is -1, return the first trap network descriptor.

Returned value: the next trap newtork descriptor, or a negative number if none can be found.

Function: long getconnreqid(int)
Parameter: a network descriptor.

Description:
Return as a long integer the value of the request identifier in the connection information structure of the outstanding SNMP request message associated with the network descriptor.

Returned value: the positive long integer request identifier value, or a negative number on error.

Function: int getudpnd(long, unsigned int, unsigned int, unsigned long)
Parameters: 1. remote host binary network address.
2. remote host socket number.
3. local host socket number.
4. interface driver option (not used).

Description:
Given a remote host network address, the socket of the application on the remote host, and a socket on a local host, return a network descriptor to send and receive UDP packets on. The option variable is not used in this implementation.

Returned value: a positive network descriptor, or a negative number on errors.

Function: int getudppkt(int, unsigned char *, unsigned int, unsigned int)

Parameters: 1. network descriptor.
2. pointer to the packet buffer.
3. length of the packet buffer in bytes.
4. interface driver flags (not used).

Description:
If there is a packet available from the remote host and the remote socket associated with the given network descriptor and sent to the appropriate local socket, move the data from the oldest such UDP packet into the indicated buffer. The flag variable is not used in this implementation.

Returned value: the number of bytes transferred to the buffer on success, or a negative number on failure.

Function: int initudp(void)
Parameters: none

Description:
Initialize the UDP subroutines. If there is configuration information in the SNMP database, remove it.

Returned value: zero.

Function: int setrinfop(int, struct routinfo *)
Parameters: 1. a network descriptor.
2. a pointer to an agent information structure.

Description:
In the UDP connection information associated with the given network descriptor, set the value of the pointer to the agent information structure to be the same as the given agent information pointer's value.

Returned value: zero on success; a negative number on failure.

Function: int setconnreqid(int, long)
Parameters: 1. a network descriptor.
2. a request identifier.

Description:
Find the socket data structure associated with the network descriptor and set its request identifier to be the same value as the supplied request identifier.

Returned value: zero on success; a negative number on error.

Function: int sendudppkt(int, unsigned char *, unsigned int, unsigned int)

Parameters: 1. a network descriptor.
2. a pointer to a buffer contain UDP data.
3. the length ot the buffer.
4. any network interface flags (not used).

Description:
Send the number of bytes in the data buffer in a UDP packet to the host associated with the network descriptor. The flags variable is not used in this implementation.

Returned value: the number of bytes sent on success; a negative number on failure.

Function: int udplisten(long, unsigned int, unsigned int, unsigned long)

Parameters: 1. a binary Internet address for a remote host.
2. a socket on the remote host.
3. a socket on the local host.
4. any network interface options (not used).

Description:
Given given a remote host binary Internet address and a socket number on that host, a socket on the local host, and a network option, prepare to listen for UDP packets from the remote host to the local one. The remote host number may be zero, in which case all UDP packets from any remote host are accepted.

Returned value: a positive network descriptor upon successfully initiating listening, or a negative number on error.

Function: int udprelease(int)
Parameter: a network descriptor.

Description:
Release the given network descriptor.

Returned value: zero.

Function: int udpreleaseall(void)
Parameters: none

Description:
Release all network descriptors.

Returned value: zero.

Packet driver specific functions.

Calls to the packet driver.

Function: int getlocaddr(int, unsigned char *, int)
Parameters: 1. a packet driver handle.
2. a pointer to a buffer to contain the address.
3. the size of the buffer in bytes.

Description:
Get the local network address associated with the indicated handle from the packet driver and put it in the given buffer.

Returned value: zero on success; a negative number on error.

Function: int getpdint(void)
Parameters: none

Description:
Find the software interrupt between 0x60 and 0x80, inclusive, used by the packet driver.

Returned value: the software interrupt number, or a negative number on errors.

Function: int getdvrinfo(struct dvrinfo *)
Parameter: a pointer to a driver information structure.

Description:
Get all the information about itself that the packet driver can supply and put it in the structure indicated.

Returned value: zero on success, or a negative number on failure.

Function: int initdvr(struct dvrinfo *, unsigned char *, int, char far *)

Parameters:
1. a pointer to a driver information structure.
2. a pointer to a buffer containing a binary 'type' value.
3. the length of the 'type' buffer in bytes.
4. a pointer to a function which will supply the packet driver with data buffers for received packets.

Description:
Given a pointer to a driver information structure for the packet driver to be accessed, send the driver the 'type' (i.e., the value of the Ethernet packet type field) of packets to receive, the length of the 'type' field specification, and the address of the routine which manages memory buffers for the packet driver.

Returned value: zero on success; a negative number on failure.

Function: int initinfo(void)
Parameters: none

Description:
Move the configuration information in the SNMP database to the local Ethernet interface's configuration information structure.

Returned value: zero on success; a negative number on error.

Function: char *prdvrerr(int)
Parameter: a packet driver error number.

Description:
Return a pointer to the appropriate packet driver error message given a packet driver error number. The message is in the form of a null-terminated ASCII character string.

Returned value: a pointer to an ASCII string, or a null pointer on error.

Function: int relhandle(int)
Parameter: a packet driver handle.

Description:
Release the packet driver from receiving packets associated with the given handle.

Returned value: zero on success; a negative number on failure.

Function: int resetintf(int)
Parameter: a packet driver handle.

Description:
Tell the packet driver to initialize the interface associated with the handle.

Returned value: zero on success; a negative number on failure.

Function: int sendpkt(unsigned char *, int)
Parameters: 1. a pointer to an Ethernet packet.
2. the length of the packet in bytes.

Description:
Tell the packet driver to send the packet in the buffer.

Returned value: zero on success; a negative number on failure.

Function: void showchain(void)
Parameters: none

Description:
Show information about the chain of Ethernet packet buffers that have not yet been processed. This routine is normally used for debugging purposes only.

Returned value: zero on success; a negative number on errors.

Function: int terminate(int)
Parameter: a packet driver handle.

Description:
Tell the packet driver to stop receiving the type of packet associated with the given handle, to cease functioning, and to release the memory used by the driver if possible.

Returned value: zero on success; a negative number on failure.

****
>>>> The following routines are used for debugging purposes only. <<<<
****

Function: int enchainbuf(int, int, unsigned char far *)
Parameters: 1. a packet driver handle.
2. the length of the packet buffer.
3. a pointer to the packet buffer.

Description:
Note: This routine is useful only with the dummy packet driver and simulates what a packet driver would do with a full buffer. Add the buffer identified by the handle, the buffer pointer, and the buffer length to the chain of unprocessed packet buffers.

Returned value: zero on success; a negative number on failure.

Function: unsigned char far *getbuf(int, unsigned int)
Parameters: 1. a packet driver handle.
2. the length of the desired packet buffer.

Description:
Note: This routine is useful only with the dummy packet driver. Get a buffer of a specific size from the buffer management routine in the same manner a packet driver would request one.

Returned value: a pointer to the buffer on success; a null pointer on failure.

Function: int installpd(void)
Parameters: none

Description:
Note: This routine is useful only with the dummy packet driver. Install the packet driver at the first unused software interrupt between 060h and 080h, inclusive.

Returned value: the number of the interrupt used by the newly installed packet driver on success; a negative number on failure.

Function: int relbuffer(struct bufferblk far *)
Parameter: a far pointer to a buffer block

Description:
Note: This routine is obsolete. Release the buffer block and its associated packet buffer.

Returned value: zero on success; a negative number on failure.

Function: int rmpd(int)
Parameter: the packet driver interrupt vector number

Description:
Note: This routine is useful only with the dummy packet driver.
Remove the packet driver installed at the given software interrupt
vector by setting the software interrupt vector to 0:0.

Returned value: zero on success; a negative number on failure.

****
>>>> End of troubleshooting functions <<<<
****

Internet Protocol related routines.

Function: int addbuf2q(struct bufferblk far *,
struct bufferblk far **)
Parameters: 1. a far pointer to a packet buffer block.
2. a pointer to a buffer block queue (a linked list of buffer block far pointers).

Description:
Add the given packet buffer block to the indicated buffer block queue.

Returned value: zero on success; a negative number on failure.

Function: unsigned int calicmpcksm(struct iphdr far *)
Parameter: a far pointer to an IP packet header.

Description:
Calculate the checksum for the ICMP packet in the indicated IP packet.

Returned value: the ICMP checksum as an unsigned integer.

Function: unsigned int calipcksm(struct iphdr far *)
Parameter: a far pointer to an IP packet header.

Description:
Calculate the checksum for the indicated IP packet header.

Returned value: the IP checksum as an unsigned integer.

Function: unsigned int caltcpcksm(struct iphdr far *)
Parameter: a far pointer to an IP packet header.

Description:
Calculate the checksum of the TCP packet in the indicated IP packet.

Returned value: the TCP checksum as an unsigned integer.

Function: unsigned int caludpcksm(struct iphdr far *)
Parameter: a far pointer to an IP packet header.

Description:
Calculate the checksum of the UDP packet in the indicated IP packet.

Returned value: the UDP checksum as an unsigned integer.

Function: int cktcpskt(struct bufferblk far *)
Parameter: a far pointer to a packet buffer block.

Description:
Check the packet associated with the given buffer block to see if it's remote host, remote socket, and local socket are on the open TCP socket list. For those that are kept, set the handle in the buffer block to be the network descriptor in the socket structure and move the packet buffer to the TCP packet queue.

Returned value: zero on success; nonzero on errors.

Function: int ckudpskt(struct bufferblk far *)
Parameter: a far pointer to a packet buffer block.

Description:
Given a pointer to a buffer block which describes a received Ethernet packet, check the packet to see if it's remote host, remote socket, and local socket are on the UDP open socket list. For those that are kept, set the handle in the buffer block to be the network descriptor in the socket structure and move the packet buffer to the UDP packet queue.

Returned value: zero on success; nonzero on errors.

Function: int endudp(void)
Parameters: none

Description:
Turn off all UDP functions by releasing all network descriptors, and disabling packet driver acceptance of IP and ARP packets.

Returned value: zero on success; a negative number on error.

Function: int getudpnd(long, unsigned int, unsigned int, unsigned long)
Parameters: 1. a remote host's Internet address in network notation.
2. the remote host's socket number.
3. the local host's socket number.
4. an option number (not used).

Description:
Given a remote host, remote socket, and local socket, return a network descriptor to send UDP packets on.

Returned value: a network descriptor; a negative number on error.

Function: int getudppkt(int, unsigned char *, unsigned int, unsigned int)

Parameters: 1. a network descriptor.
2. a pointer to an empty packet buffer.
3. the size in bytes of the packet buffer.
4. a flag number (not used).

Description:
Get a packet from the UDP packet queue given the network descriptor of the associated connection, the buffer in which to put the data, the length of the buffer, and any flags.

Returned value: the number of bytes read on success; zero if no packet are available; a negative number on errors.

Function: int icmphdlr(struct bufferblk far *)
Parameter: a far pointer to a packet buffer block.

Description:
Given a far pointer to a packet buffer containing an ICMP packet, respond appropriately to the ICMP message. If the packet is an ICMP destination unreachable, time-to-live exceeded, redirect, or echo message, respond appropriately. Ignore all other ICMP packets.

Returned value: zero on success; a negative number on failure.

Function: int initiphdr(struct iphdr *)
Parameter: a pointer to an IP packet header.

Description:
Initialize the indicated IP packet header by setting the IP version and header length, the type of service, the identification, the fragmentation flags, the time-to-live, and the IP source address.

Returned value: zero.

Function: int initudp(char *)
Parameter: a pointer to the SNMP configuration file name.

Description:
Initialize the UDP subroutines by getting the packet driver software interrupt number and interface class, type, and number, and by moving this data into the packet driver information structure.

Returned value: zero on success; a negative number on failure.

Function: int initudppkt(struct iphdr *, int,
unsigned char *, int)

Parameters: 1. a pointer to an IP packet header.
2. the length in bytes of the IP packet.
3. a pointer to a filled UDP packet data buffer.
4. the length in bytes of the UDP data buffer.

Description:
Given a pointer to an IP packet buffer, its length, a pointer to a filled UDP packet data field, and the length of the data field, create a UDP/IP packet by initializing the UDP and IP headers, and moving the data into the data field. It is assumed that the IP header is complete except for the packet length field, the protocol, and the header checksum. It is also assumed that the source and destination UDP ports are correct.

Returned value: zero.

Function: int iphdlr(struct bufferblk far *)
Parameter: a far pointer to a packet buffer block.

Description:
Process the IP packet taken from the Ethernet packet queue. If it is a TCP, UDP, or ICMP packet, add it to the appropriate packet queue or send it to the appropriate packet handler.

Returned value: zero on success; a negative number on error.

Function: int sendudppkt(int, unsigned char *, unsigned int,
unsigned int)

Parameters: 1. a network descriptor.
2. a pointer to a filled UDP packet data buffer.
3. the length of the UDP packet data buffer.
4. a flags variable (not used).

Description:
Encapsulate in a UDP packet the indicated number of bytes in the filled data buffer and send it to the host associated with network descriptor. The flags variable is not used in this implementation.

Returned value: the number of bytes sent on success; otherwise return a negative number.

Function: int udplisten(long, unsigned int, unsigned int, unsigned long)

Parameters: 1. a remote host IP address in network form.
2. the remote host's socket number.
3. the local host's socket number.
4. an option variable (not used).

Description:
Given a remote host number and a socket number on that host, and a local host socket, listen for UDP packets from that host to this one. If the host number is zero, UDP packets from all hosts will be accepted.

Returned value: a network descriptor upon successfully initiating listening, or a negative number on error.

Function: int udprelease(int)
Parameter: the network descriptor to be released.

Description:
Release any resources associated with the given network descriptor.

Returned value: zero on success; a negative number on error.

Function: int udpreleaseall(void)
Parameters: none

Description:
Release all network descriptors and any associated resources.

Returned value: zero on success; a negative number on error.

Ethernet packet queue management routines.

Function: int addarpent(struct arppkt far *)
Parameter: a far pointer to an ARP packet.

Description:
Given a far pointer to an ARP reply packet, add the necessary data to the ARP table to create an appropriate ARP entry.

Returned value: zero on success; a negative number on failure.

Function: int arphdlr(struct bufferblk far *)
Parameter: a far pointer to a packet buffer block.

Description:
Respond to an ARP packet taken from the packet queue. Send a response to an ARP request, or create an ARP table entry from an ARP reply.

Returned value: zero on success; a negative number on errors.

Function: int cmpstr(unsigned char *, unsigned char *, int)
Parameters: 1. a pointer to an unsigned character string.
2. a pointer to an unsigned character string.
3. the number of characters to be compared.

Description:
Given pointers to two strings of unsigned characters, compare them for the indicated number of bytes.

Returned value: if they match, return zero; otherwise return a negative number.

Function: int fcmpstr(unsigned char far *,
unsigned char far *, int)
Parameters: 1. a far pointer to an unsigned character string.
2. a far pointer to an unsigned character string.
3. the number of characters to be compared.

Description:
Compare the indicated number of bytes of two byte strings pointed to by far pointers.

Returned value: zero if they match; otherwise return a negative number.

Function: int fmovestr(unsigned char far *,
unsigned char far *, int)
Parameters: 1. a far pointer to an unsigned character string.
2. a far pointer to an unsigned character string.
3. the number of characters to be moved.

Description:
Move a string of bytes from one far location to another.

Returned value: the number of bytes moved.

Function: unsigned char *getethaddr(unsigned char *)
Parameter: a pointer to an Internet address in network form.

Description:
Given a pointer to an Internet host address, return a pointer to the appropriate Ethernet address (a string of hexadecimal bytes).

Returned value: if the host is on the local Internet subnet, return the pointer to the host's Ethernet address; if it isn't, return the pointer to the local gateway's Ethernet address.

Function: int getpkts(void)
Parameters: none

Description:
Remove any packets from the packet driver packet queue and dispatch them to the appropriate processing routines.

Returned value: a positive number if packets were processed successfully; zero if no packets were available for processing; a negative number on errors.

Function: int movestr(unsigned char *, unsigned char *, int)
Parameters: 1. a pointer to an unsigned character string.
2. a pointer to an unsigned character string.
3. the number of characters to be moved.

Description:
Given a pointer to a string of bytes, a pointer to the desired location of those bytes, and the number of bytes in the string, move the bytes so that the destination string does not overwrite the source string until the source string has been read.

Returned value: the number of bytes moved on success; a negative number on errors.

Function: int sendarprep(struct arppkt far *)
Parameter: a far pointer to an ARP packet.

Description:
Given a far pointer to an ARP request packet, send an ARP response packet.

Returned value: an integer zero on success; a negative number on failure.

Function: int showarptbl(void)
Parameters: none

Description:
Print the contents of the ARP table on the console.

Returned value: an integer zero on success; a negative number on failure.

Function: unsigned char *srcharptbl(unsigned char *)
Parameter: a pointer to an IP address in network form.

Description:
Given a pointer to a host's IP address as a string of four hexadecimal bytes, look for a corresponding Ethernet address in the ARP tables.

Returned value: a pointer to the correct hexadecimal address (as a string of bytes); otherwise, a null pointer.

Function: unsigned char *getcmunp(int)
Parameter: a network descriptor.

Description:
Return the pointer to the SNMP community name associated with the indicated network descriptor.

Returned value: a pointer to an SNMP community name (as a null-terminated ASCII string), or a null pointer on error.

Function: int getconninfo(int, struct sockdata *)
Parameters: 1. a network descriptor.
2. a pointer to an empty connection information array.

Description:
Fill in the given array of socket information with the information about the connection associated with the indicated network descriptor.

Returned value: zero on success; a negative number on errors.

Function: int getnxtsendnd(int)
Parameter: a network descriptor.

Description:
Find the next network descriptor for sending packets in the socket information table given a current network descriptor. If the current network descriptor is -1, find the first network descriptor for sending packets in the table.

Returned value: a network descriptor, or a negative number on error.

Function: int getnxttrapnd(int)
Parameter: a network descriptor.

Description:
Find the next network descriptor for receiving SNMP trap packets in the socket information table given a current trap network descriptor. If the network descriptor passed to this routine is -1, start with the first trap network descriptor.

Returned value: a network descriptor, or a negative number on error.

Function: long getconnreqid(int)
Parameter: a network descriptor.

Description:
Return the request identifier in the connection information structure associated with the indicated network descriptor.

Returned value: a request identifier as a long integer, or a negative number on error.

Function: int gettstpkt(int, unsigned char *, unsigned int, unsigned int)
Parameters: 1. a network descriptor.
2. a pointer to an empty packet buffer.
3. the length of the packet buffer.
4. a flags variable (not used).

Description:
Note: This is a debugging routine. Get a UDP packet from the UDP queue.

Returned value: a positive number on success; a negative number on errors.

Function: int set_option(int, int, int, char far *, int)
Parameters: 1. a network descriptor.
2. a protocol indicator.
3. the number of the option to manipulate.
4. a far pointer to the option value.
5. the number of bytes in the option value.

Description:
Enable the option specified for the given network descriptor. For the packet driver no options are accepted. It may be required for non-packet driver implementations.

Returned value: zero.

Function: int net_listen(int, int, struct addr *)
Parameters: 1. a network descriptor.
2. the number of the protocol to use in listening.
3. a pointer to a connection information structure.

Description:
Enable the packet driver for receiving IP packets from the indicated remote host and socket which were sent to the local socket. The current implementation accepts only the IP protocol number.

Returned value: zero on success; a negative number on errors.

Function: int net_readfrom(int, char *, unsigned int,
struct addr *, unsigned int)

Parameters: 1. a network descriptor.
2. a pointer to an empty packet data buffer.
3. the length of the empty packet buffer in bytes.
4. a pointer to a connection information structure.
5. a flags variable (not used).

Description:
Given a network descriptor, a pointer to a buffer, the length of that buffer, a pointer to socket data, and a flags value, check the appropriate packet queue (as identified by the socket protocol) for packets meant for the specified network descriptor. If any are found, copy the data from the the first packet into the buffer, providing the buffer is large enough.

Returned value: zero if no packets are available; the number of bytes transferred into the data buffer if a packet is available; or a negative number on errors.

Function: int net_connect(int, int, struct addr *)

Parameters: 1. a network descriptor.
2. the protocol to use in attempting a connection.
3. a pointer to a connection information structure.

Description:
Given a network descriptor, the protocol to use in sending packets on the descriptor, and a pointer to a structure containing the remote host, remote socket, and local socket to use, establish a connection with the remote host. Only UDP is currently supported as the connection protocol.

Returned value: a network descriptor on success, otherwise a negative number.

Function: int net_writeto(int, char *, unsigned int,
struct addr *, unsigned int)

Parameters: 1. a network descriptor.
2. a pointer to a filled packet data buffer.
3. the length of the empty packet buffer in bytes.
4. a pointer to a connection information structure.
5. a flags variable (not used).

Description:
Given a network descriptor, a pointer to a buffer containing data, the length of the data in the buffer, a pointer to socket information, and any flags, send a packet containing the data in the buffer to the host specified by the network descriptor and the socket information. Only UDP protocol is supported.

Returned value: zero on success, or a negative number on errors.

Function: int net_getdesc(void)
Parameters: none

Description:
Get the next available unused network descriptor.

Returned value: a network descriptor, or a negative number on errors.

Function: int net_releaseall(void)
Parameters: none

Description:
Release all network descriptors currently in use.

Returned value: always zero.

Function: int net_release(int)
Parameter: the network descriptor to be released.

Description:
Release the specified network descriptor.

Returned value: always zero.

Function: void pneterror(char *)
Parameter: a pointer to a string to be printed before the network error message is printed.

Description:
Print an error message on the console based on the errors in the 'neterrno' and 'netsuberrno' variables.

Returned value: none.

Function: int sendtstpkt(void)
Parameters: none

Description:
Note: This is a test routine. Send a UDP packet to the agent at a specific Internet address.

Returned value: the network descriptor of the connection on which the packet was sent, or a negative number on errors.

Function: int setrinfop(int, struct routinfo *)
Parameters: 1. a network descriptor.
2. a pointer to an agent information structure.

Description:
Find the socket information structure associated with the network descriptor. In the socket information structure set the value of the pointer to an agent information structure to be the value of the given agent information structure pointer.

Returned value: zero on success; a negative number on error.

Function: int setconnreqid(int, long)
Parameters: 1. a network descriptor.
2. an SNMP connection request identifier.

Description:
Set the request identifier in the socket information structure associated with the indicated network descriptor to the given value.

Returned value: zero on success; a negative number on error.

SNMP request packet related routines.

Function: struct objinfo *getobjidstrc(unsigned char *)
Parameter: a pointer to an SNMP object.

Description:
Return a pointer to the MIB object information structure associated with the indicated object.

Returned value: a pointer to an object information structure, or a null pointer on error.

Function: long getreq(struct routinfo *, unsigned char *, int *)
Parameters: 1. a pointer to an agent information structure.
2. a pointer to an empty SNMP data packet buffer.
3. the length of the SNMP empty packet buffer in bytes.

Description:
Given a pointer to a structure which contains information about what SNMP variable values should be retrieved from which agent (an agent information structure), and an empty packet buffer of the indicated length, create a packet containing an SNMP Get Request message in the buffer to retrieve the value of the variables listed in the agent information structure.

Returned value: the positive long integer used in the packet as the request identifier value on successfully formating the packet, or a negative long integer on error.

Function: unsigned char *getvardesc(struct objinfo *)
Parameter: a pointer to an object information structure.

Description:
Format an SNMP object for the variable defined by the indicated object information structure.

Returned value: a pointer to the byte string of the object identifier on success; a null pointer on error.

Function: struct objinfo *searcharr(struct objinfo *, char *)
Parameters: 1. a pointer to an object information structure.
2. a pointer to a null-terminated ASCII string containing the name of an MIB object.

Description:
Search the MIB array of object information structures for the structure which corresponds to the variable name pointed to by the name pointer. Start searching the MIB structure at the indicated object structure.

Returned value: a pointer to the appropriate structure, or a null pointer on errors.

SNMP response packet related routines.

Function: int analresp(int, unsigned char *)
Parameters: 1. a network descriptor.
2. a pointer to a buffer contain an SNMP response message.

Description:
Analyze the SNMP response message received on the indicated network descriptor.

Returned value: the network descriptor upon successful analysis, or a negative number on error. Successful analysis depends on the message. It may be updating the agent/variable database, printing a trap message on the console, or nothing at all.

Function: int getreqid(unsigned char *, long *)
Parameters: 1. a pointer to an SNMP request identifier object.
2. a pointer to a request identifier storage area.

Description:
Replace the value of the pointed to request identifier with the value of the request identifier object.

Returned value: zero on success; a negative number on failure.

Function: struct routinfo *getroutinfo(long, struct routinfo *)
Parameters: 1. an IP address in network form.
2. a pointer to an agent information structure.

Description:
Get a pointer to the agent information structure following the indicated one which contains the indicated network address. If the given agent information structure is a null pointer, get a pointer to the first agent information structure associated with the given network address.

Returned value: a pointer to an agent information structure, or a null pointer on error.

Function: struct varinfo *getvarinfo(struct routinfo *,
struct objinfo *, unsigned char *, int)
Parameters: 1. a pointer to an agent information structure.
2. a pointer to an object information structure.
3. a pointer to an encoded SNMP variable index value.
4. the length in bytes of the encoded index value.

Description:
Given a pointer to an agent information structure, a pointer to an object information structure, a pointer to a null-terminated string which contains a encoded index value, and the length of the encoded index string, find a pointer to the variable information structure which is associated with the agent information structure, has the same variable as that in the object information structure, and has the same index as that in the index string.

Returned value: a pointer to a variable information structure, or a null pointer on error.

Function: int ifIndex_f(unsigned char *)
Parameter: a pointer to an SNMP message index number.

Description:
Determine the index number of the interface about which "interesting data" is being reported by an SNMP trap message. (A pointer to this routine is put into the object information structure in the MIB for the 'ifIndex' variable.)

Returned value: the index number of the interface, or a negative number on error.

Function: int ifOperStatus_f(unsigned char *)
Parameter: a pointer to an encoded SNMP interface operational status message.

Description:
Perform the necessary functions when a response from an SNMP agent contains information about an interface's operational status. (Obsolete, but provided for educational purposes. Originally a pointer to this function was put in the MIB for the variable 'ifOperStatus'. The "necessary functions" were to print on the console the value of the agent interface's operational variable.)

Returned value: the index value of the interface indicated in the SNMP message, or a negative number on errors.

Function: int resperr(struct sockdata *, int, int,
unsigned char *)

Parameters: 1. a pointer to a socket data structure.
2. the SNMP response message error status.
3. the SNMP response message error index.
4. a pointer to the SNMP message encoded variable bindings field.

Description:
Analyze an SNMP response error packet given the error status, error index, and a pointer to the variable bindings. An error message is printed on the console that attempts to explain why the request message was in error based on the information in the response message.

Returned value: zero on success; a negative number on failure.

Function: int saveval(struct varinfo *, unsigned char *)

Parameters: 1. a pointer to an SNMP varible information structure.
2. a pointer to an SNMP encoded object.

Description:
Given a pointer to a variable information structure and a pointer to an SNMP object, put the value of the SNMP object in the given information structure.

Returned value: zero on success; a negative number on error.

SNMP object related routines.

Function: unsigned int atohex(char *)
Parameter: a pointer to null-terminated string of hexadecimal digits encoded in ASCII.

Description:
Given a pointer to a null-terminated string of ASCII characters representing a string of hexadecimal digits, convert them to an unsigned integer of the correct value.

Returned value: the unsigned integer value of the hexadecimal string.

Function: unsigned char *checkcmun(unsigned char *, int)
Parameters: 1. a pointer to an SNMP encoded object.
2. a network descriptor.

Description:
Check the SNMP community object pointed to for a match with the community of the network descriptor indicated.

Returned value: on a match, a pointer to the SNMP object following the one pointed to by the object pointer, or a null pointer on a mismatch.

Function: unsigned char *checkpkt(unsigned char *, int, int)
Parameters: 1. a pointer to a filled SNMP packet buffer.
2. the length of the packet buffer in bytes.
3. a network descriptor.

Description:
Check the SNMP format of the packet in the buffer of the indicated length. The first byte of the buffer is the first data byte of the UDP packet and does not include the Ethernet, IP, or UDP packet headers.

Returned value: a pointer to the first SNMP object in the buffer if the format is correct; a null pointer on error.

Function: unsigned char *checkver(unsigned char *)
Parameter: a pointer to an SNMP encoded version object.

Description:
Check the value of the SNMP version object pointed to against the version of SNMP that this software is compatible with.

Returned value: a pointer to the community object in the message on success; a null pointer on error.

Function: long decdsubid(unsigned char **)
Parameter: a pointer to a pointer to an SNMP encoded sub-identifier object.

Description:
Given a pointer to a pointer to an encoded object in an SNMP object identifier, decode the object and adjust the pointer to the object so it points to the first object past the decoded one.

Returned value: the decoded long integer.

Function: int encdsubid(unsigned long, unsigned char *, int)
Parameters: 1. a integer to be encoded.
2. an empty object buffer.
3. the length of the object buffer in bytes.

Description:
Encode the given positive integer to produce an SNMP object in an object identifier. Put the encoded bytes in the designated buffer while checking to see if the buffer is large enough.

Returned value: the number of bytes in the encoding, or a negative number on error.

Function: unsigned char *finddatafld(unsigned char *)
Parameter: a pointer to an SNMP object length field.

Description:
Given a pointer to a length field of an SNMP object, return a pointer to the data field of that object.

Returned value: a pointer to the data field of an SNMP object, or a null pointer on error.

Function: unsigned char *findlenfld(unsigned char *)
Parameter: a pointer to an SNMP object tag field.

Description:
Given a pointer to the tag field of an SNMP object, return a pointer to its length field.

Returned value: a pointer to the length field of an SNMP object, or a null pointer on error.

Function: int getintval(unsigned char *, void *, int)
Parameters: 1. a pointer to an SNMP encoded integer object.
2. a pointer to an integer/long integer variable.
3. the length of the integer/long integer storage area in bytes.

Description:
Get the value of the the encoded SNMP integer object pointed to by the given pointer. Put the integer value in the indicated location, making sure the location is large enough.

Returned value: zero on success or a negative number on error.

Function: unsigned char *getnxtobj(unsigned char *)
Parameter: a pointer to an SNMP object.

Description:
Given a pointer to an SNMP object, return a pointer to the next SNMP object.

Returned value: a pointer to the next SNMP object, or a null pointer on error.

Function: int getobjlen(unsigned char *)
Parameter: a pointer to an SNMP object.

Description:
Get the length in bytes of the data field of an SNMP object from the length field pointed to by the given length field pointer.

Returned value: the length of the indicated data field of an SNMP object, or a negative number on error.

Function: int getobjptrs(unsigned char *, struct objptrs *)
Parameters: 1. a pointer to an SNMP object.
2. a pointer to an empty object pointers structure.

Description:
Given a pointer to an SNMP object and an empty structure of pointers concerning that object, fill in the structure with pointers to the fields of the object.

Returned value: zero on success, or a negative number on error.

Function: int snmpnettoasc(unsigned char *,
unsigned char *, int)
Parameters: 1. a pointer to a MIB ipAddress object.
2. a pointer to an empty ASCII character buffer.
3. the length in bytes of the ASCII buffer.

Description:
Convert an IP address encoded as an SNMP 'IpAddress' object to a null-terminated string of ASCII characters in the familiar dot notation in the buffer indicated. Check for a large enough buffer.

Returned value: zero on success, or a negative number on failure.

Function: int setobjlen(int, unsigned char *)
Parameters: 1. the length of an SNMP encoded object.
2. a pointer to an SNMP encoded object length field.

Description:
Encode the indicated length of an object into an SNMP object length field and put the new value in the given location.

Returned value: the number of bytes used in creating the length field, or a negative number on error.

Function: int wait(unsigned int)
Parameter: the number of milliseconds to wait.

Description:
Wait for at least the number of milliseconds given.

Returned value: always zero.

SNMP database related routines.

Function: struct routinfo *analdb(char *)
Parameter: a pointer to a null-terminated ASCII string containing a file name.

Description:
Given the name of an SNMP database file, read the file and create memory resident agent and variable information structures for use by other functions.

Returned value: a pointer to the structure in the agent and variable information structure linked lists, or a null pointer on errors.

Function: int encdindex(char *, unsigned char *, int, struct objinfo *)
Parameters: 1. a pointer to a null-terminated ASCII string containing an SNMP variable's index value.
2. an empty encoded index buffer.
3. the length in bytes of the index buffer.
4. a pointer to an object information structure.

Description:
Given a pointer to a null-terminated ASCII string of characters containing an index value and a pointer to an object information structure, convert the string into an SNMP encoded index for the type of object in the object information structure. Put it in the indicated buffer.

Returned value: the number of bytes of the encoded index in the buffer, or a negative number on error.

Function: struct routinfo *fillcmunblk(char **)
Parameter: a pointer to a null-terminated ASCII string of community agent lists.

Description:
Given a pointer to a pointer to a null-terminated ASCII string of SNMP agent information, get memory for and fill in with the appropriate data the various structures for one SNMP community. Modify the pointer to the ASCII data to point past the used data.

Returned value: a pointer to the group of filled-in agent information structures on success, or a null pointer on errors.

Function: struct routinfo *fillroutblk(char **, char *)
Parameter: a pointer to a null-termianted ASCII string of agent information lists.

Description:
Given a pointer to a pointer to a null-terminated ASCII string of SNMP agent information, get memory for and fill in with the appropriate data the various structures for one agent. Modify the pointer to the ASCII data to point past the used data.

Returned value: a pointer to a filled-in agent information structure on success, or a null pointer on errors.

Function: int freerblk(struct routinfo *)
Parameter: a pointer to the agent information structure to be freed.

Description:
Free the indicated agent information block, fix the chain of agent information block pointers, and free any variable information blocks associated with the freed agent information block.

Returned value: zero on success; a negative number on error.

Function: int freevblk(struct routinfo *, struct varinfo *)
Parameters: 1. a pointer to an agent information structure.
2. a pointer to the variable information structure to be freed.

Description:
Free the variable information block associated with the indicated agent information block and fix the chain of variable information block pointers.

Returned value: zero on success; a negative number on error.

String utilities.

Function: int cmpstr(unsigned char *, unsigned char *, int)
Parameters: 1. a pointer to a string of unsigned characters.
2. a pointer to a string of unsigned characters.
3. the number of characters to compare.

Description:
Compare two strings a character at a time for the given number of characters.

Returned value: zero on success, non-zero on failure.

Function: unsigned char *findstr(unsigned char *,
unsigned char *)
Parameters: 1. a pointer to a null-terminated string of unsigned characters.
2. a pointer to a null-terminated string of unsigned characters.

Description:
Find the null-terminated first string in the null-terminated second string. The nulls don't have to match.

Returned value: a pointer to the character in the second string where the match begins, or a null pointer on failure.

Function: int getreply(char)
Parameter: the desired character to be received from the console.

Description:
Test to see if the first character received from the console keyboard matches the one passed to this routine. Discard the rest of the input line, including the '\n'.

Returned value: zero on a match, a positive number on a non-match, and a negative number on an error.

Function: int movenstr(unsigned char *, unsigned char *)
Parameters: 1. a pointer to a null-terminated string of unsigned characters.
2. a pointer to a memory location.

Description:
Given a pointer to a null-terminated string and a pointer to a location at which that string is to be replicated, move all of the bytes including the null terminator.

Returned value: the number of bytes moved, not including the null terminator, on success; a negative number on failure.

SNMP trap packet processing routines.

Function: int analtrap(int, unsigned char *)
Parameters: 1. a network descriptor.
2. a pointer to an encoded SNMP trap message.

Description:
Analyze the SNMP trap message for correct format and process any information reported by it. If the variables mentioned in the trap message are contained in the SNMP database of variables being monitored, put the new values in the database. Otherwise, do what is appropriate (usually print a message on the console) with the information contained in the message.

Returned value: zero on success; non-zero on error.

Function: int authenticationFailure_f(long, unsigned char *)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP authentication failure message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'authentication failure' trap message, print on the console the message that the given agent reported an authentication failure.

Returned value: zero.

Function: int cktraps(void)
Parameters: none

Description:
Check if trap packet reception is enabled. If it is not, do so.

Returned value: zero on success; a negative number on errors.

Function: int cleartraps(void)
Parameters: none

Description:
Clear any trap network descriptors that have been allocated so that trap messages will not be accepted.

Returned value: always zero.

Function: int coldStart_f(long, unsigned char)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP cold start message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'cold start' trap message, print on the console the message that the given agent reported undergoing a cold start.

Returned value: always zero.

Function: int egpNeighborLoss_f(long, unsigned char *)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP EGP neighbor loss message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'egpNeighborLoss' trap message, print on the console the message that the given agent reported a External Gateway Protocol neighbor loss condition.

Returned value: always zero.

Function: int enterpriseSpecific_f(long, unsigned char *)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP enterprise specific message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'enterpriseSpecific' trap message, print on the console the message that the given agent reported an enterprise specific condition.

Returned value: always zero.

Function: int linkDown_f(long, unsigned char *)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP link down message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'linkDown' trap message, check the SNMP database for an agent address and an 'ifOperStatus' variable and index matching the one given in the trap message. If one is found, set its value to the equivalent of "down", set the time stamp to the current time, and increment the change flag if the variable's value changed. If no matching variable is found, print a message on the console that the interface for the given agent has gone down.

Returned value: zero on success; a negative number on error.

Function: int linkUp_f(long, unsigned char *)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP link up message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'linkUp' trap message, check the SNMP database for an agent address and an 'ifOperStatus' variable and index matching the one given in the trap message. If one is found, set its value to the equivalent of "up", set the time stamp to the current time, and increment the change flag if the variable's value changed. If no matching variable is found, print a message on the console that the interface for the given agent has come up.

Returned value: zero on success; a negative number on error.

Function: int settraps(void)
Parameters: none

Description:
Start the UDP server listening for SNMP Trap packets.

Returned value: zero on success; a negative number on error.

Function: int warmStart_f(long, unsigned char *)
Parameters: 1. an IP address in network form.
2. a pointer to an encoded SNMP warm start message.

Description:
Given an agent's IP address in the form of a long integer and a pointer to an SNMP 'warmStart' trap message, print on the console the message that the given agent reported undergoing a warm start.

Returned value: always zero.

References:

[1] Case, J., M. Fedor, M. Schoffstall, and C. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1098, Internet Network Working Group, April 1989.

[2] McCloghrie, K., and M. Rose, "Managment Information Base for Network Management of TCP/IP-based Internets", RFC 1066, Internet Network Working Group, August 1988.

[3] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1065, Internet Network Working Group, August 1988.

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4467

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
NOVEMBER 1990

4. TITLE AND SUBTITLE

SNMPLIB: A Simple Network Management Protocol Function Library for IBM PC Compatible Computers

5. AUTHOR(S)

Robert J. Crosson

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

Many Simple Network Management Protocol applications exist, but few general purpose packages run on a personal computer. SNMPLIB is a library of function calls written in Microsoft C version 5.0 for IBM PCs and compatibles. Its data structures and functions are described. With SNMPLIB a user can write a program to dynamically monitor operational variables of compatible networked devices and take action when those variables cross threshold values. The network input/output routines work with any Ethernet interface for a PC or a compatible for which a packet driver has been installed, making the application somewhat independent of the Ethernet interface used in the computer. A sample application program, SNMPMON, is supplied which monitors user specified variables in user specified devices and writes their values to the PC's display. The SNMPLIB and SNMPMON have been compiled on a Sun Microsystems workstation and successfully used there with a similar, but Sun Ethernet interface specific, set of input/output functions.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

computers; data; management; network; personal; SNMP

13. AVAILABILITY

X	UNLIMITED
	FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
	ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
X	ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
71

15. PRICE
A04

ELECTRONIC FORM