Programmer's Guide to the SNP Facility

A Facility for Monitoring TCP/IP-Based Communications

Nilesh R. Gohel

Mallinckrodt Institute of Radiology
Electronic Radiology Laboratory
510 South Kingshighway Boulevard
St. Louis, Missouri 63110
314/362-6965 (Voice)
314/362-6971 (FAX)

Version 2.10.0

August 3, 1998

This document describes usage of a software facility based on the Solaris 2.3 (SunOS 5.3) operating system for monitoring TCP/IP communications on shared media networks. It may be used to monitor DICOM communications based on TCP/IP.

---

1 Introduction

The SNP facility provides a means to monitor data communications on shared media networks (e.g. Ethernet) at the TCP level. The monitoring is performed in real-time. The facility was specifically developed for the Sun Solaris 2.3 (Sun OS 5.3) operating system. Only one association (network connection) may be monitored at a time. Parameters of the communication that need to be specified are initiator IP host name/address, acceptor IP host name/address, and acceptor's TCP port number. Additional parameters include the buffersize, device filename and number, and timeouts. Callback functions also need to be specified to retrieve the parsed TCP data and state information from the facility. In order to monitor communications, the following set of steps / function calls should be implemented in the given order:

SNP_Init

SNP_RegisterCallback (One for data in each direction, and one for state info.)

SNP_Start

SNP_Stop

SNP_Terminate

A coding example entitled "Generic usage of SNP facility" is provided.

For a more complete description of the software architecture, refer to the manual entitled "DICOM Test Tools: A Guide to Programs for Testing DICOM Functionality."

2 Data Structures

There are instances in which an application using the SNP facility stores the parsed TCP data stream to data files. In such cases, the data is stored in two files (one for data from the initiator, and the other for data from the acceptor) in which case it is necessary to store a header with each of buffer of data. Such a header would be used to signify the ends of associations, sequence the buffers of data between the files, and also provide information about the length of the buffer. The following data structure, found in the header file for the facility (snp.h), describes an example header format for the buffers.

```
typedef struct {
u_long type;
u_long seq;
u_long len;
} TCP_BUF_HEAD;
```

type represents the type of header and / or its direction. Here are the types and their definitions as per snp.h:

0 ITOA Data from initiator to acceptor
1 ATOI Data from acceptor to initiator
2 SNP_EOA Signifies the End of Association

seq is sequence number of the buffer and may be used to check for the sequence between the files for data from the initiator to acceptor, and acceptor to initiator. len provides the length of the data buffer to follow.

3 Include Files

To use the SNP facility, applications need to include these files in the order given below:

```
#include “dicom.h”
#include “lst.h”
#include “condition.h”
#include “snp.h”
```

4 States

The following are the integer state constants as defined in snp.h used to report the status of the network monitoring. The meaning of each is provided.

NORMAL All is well

END_ASSOC End of association

DATA_OVERFLOW Data overflow error

GETMSG_FAIL getmsg() failure

RESET_ASSOC_INI Association aborted by initiator

RESET_ASSOC_ACC Association aborted by acceptor

NONCONTIGDATA Non contiguous data passed to application

WRITECALLBACKFAIL Failure in callback to write data

LSTINSFAIL Failure using LST facility Insert()

DROPPEDPACKETS Kernel processing has dropped packets

BAD_END_ASSOC Bad end of association - was not able to capture all data

CON_TIMEOUT Connection timed out (with segments still to be ack'ed)

STRGETMSG_TIMEOUT strgetmsg() timed out (in STREAM setup)

5 Return Values

The SNP facility uses the COND facility to form and report conditions. The COND facility is documented in the Programmer's Guide to the COND facility.

SNP routines return a condition value. These are condition values the SNP facility may return:

SNP_NORMAL	Normal return from SNP routine
SNP_MALLOCERROR	Error in performing memory allocation (malloc)
SNP_CLOSEERROR	Error in closing file
SNP_OPENERROR	Error in opening file
SNP_SIGSETERROR	Error setting up interrupt (signal)
SNP_STREAMSETUP	Error setting up kernel level streams processing
SNP_LSTCREATFAIL	Error creating LST list
SNP_CALLBACKSMISSING	All callbacks not registered
SNP_CALLBACKFAIL	Error using callback function
SNP_ARGERROR	Problem with function argument
SNP_IOCTLFAIL	ioctl failure
SNP_UNIMPLEMENTED	Error - unimplemented function
SNP_PUTMSGFAIL	putmsg failure
SNP_DLPIFAIL	Failure in DLPI routine
SNP_DLPIEXPECT	DLPI function strgetmsg received unexpected message
SNP_ALARMSET	Failure of alarm setting function
SNP_GETMSGFAIL	getmsg failure in function

6 SNP Routines

This section provides detailed documentation for each SNP facility routine.

SNP_Init

Name

SNP_Init –initializes snooping on a TCP/IP association .

Synopsis

CONDITION SNP_Init()

Description

The routine is called to set up the SNP facility for snooping on a TCP/IP association. It should be called before any other SNP function is called.

Return Values

SNP_NORMAL

SNP_Terminate

Name

SNP_Terminate –terninate snooping on a TCP/IP association .

Synopsis

CONDITION SNP_Terminate()

Description

The routine is called to tear down the SNP facility for snooping on a TCP/IP association.

Return Values

SNP_NORMAL

SNP_RegisterCallback

Name

SNP_RegisterCallback –to register callback functions to pass TCP parsed data and SNP facility state information to higher software layers

Synopsis

CONDITION SNP_RegisterCallback(CONDITION(*callback) (), int callbackType, void *ctx)

callback	the name of the function to be used to pass parsed TCP data or SNP facility state information back to the application
callbackType	specifies that the callback function being register should be used for data in a particular direction or on state information. The forms of the callback functions are provided:

SNP_CALLBACK_ITOA on data from Initiator to Acceptor with callback function of the form:

CONDITION callback_func_name(char *buffer, int buffer_size, void *ctx)

SNP_CALLBACK_ATOI on data from Acceptor to Initiator with callback function of the form:

CONDITION callback_func_name(char *buffer, int buffer_size, void *ctx)

SNP_CALLBACK_STATE on state information with callback function of the form:

CONDITION callback_func_name(int state, void *ctx)

ctx	context pointers used by application that are passed back in callbacks

Description

This routine registers callback functions for the passing of parsed TCP data and state information to higher software layers. Use of the SNP facility for snooping requires that all three callback functions be registered. While snooping, the functions may be re-registered thus providing greater freedom to change callbacks during operation.

Return Values

SNP_NORMAL

Name

SNP_Start - sets up and starts snooping on all TCP/IP associations with the same parameters (includes initiator name or IP address, acceptor name or IP address, acceptor TCP port)

Synopsis

CONDITION SNP_Start(char *device, int ppa, char *initiator, char *acceptor, int port, int timeOutCon, int timeOutBuf, int bufferSpace)

device	shared media network device driver file name on which to be snooping e.g. Ethernet interface: "/dev/le"
ppa	Physical Point of Access (PPA) - corresponds to the number of the above device e.g. 0 for /dev/le0 which is the first network device of type /dev/le
initiator	host name or IP address of communication initiator
acceptor	host name or IP address of communication acceptor
port	port number on acceptor that will be used
timeOutCon	number of seconds for timeout on connection for which there is no traffic and there are outstanding acknowledgements
timeOutBuf	number of seconds for timeout by STREAMS buffer module in the kernel space
bufferSpace	number of bytes of space used for chunks by STREAMS kernel buffer module

Description

SNP_Start starts the snooping for associations after setting up the STREAMS chain in the kernel for filtering and buffering of the TCP stream to be monitored, and interfacing with the network device driver. Upon return of this function, the set up for the monitoring is complete.

As the SNP facility uses asynchronous I/O, the snooping operation is then interrupt-driven using callbacks to pass data and state information. By examining the state information, the calling software is able to determine the end of associations. A coding example follows the function definitions to illustrate usage of the facility.

Return Values

```
SNP_NORMAL
SNP_CALLBACKSMISSING
SNP_ARGERROR
SNP_OPENERROR
SNP_MALLOCERROR
SNP_STREAMSETUP
SNP_SIGSETERROR
SNP_LSTCREATFAIL
```

SNP_Stop

Name

SNP_Stop - To stop the snooping activities.

Synopsis

CONDITION SNP_Stop()

Description

SNP_Stop stops the snooping and performs most of the tear-down activities.

Return Values

```
SNP_NORMAL
SNP_CLOSEERROR
```

SNP_StateMsg

Name

SNP_StateMsg - To get the textual interpretation of a SNP facility state number.

Synopsis

char* SNP_StateMsg(int state)

state Number of state to be interpreted

Description

SNP_StateMsg returns the textual representation of state number provided

Return Values

Pointer to character string interpreting SNP facility state number.

SNP_Debug

Name

SNP_Debug - Turns on/off debugging messages of SNP facility.

Synopsis

void SNP_Debug(BOOLEAN flag)

flag TRUE to turn on debugging, FALSE to turn off debugging.

Description

SNP_Debug turns on/off debugging messages of SNP facility

Return Values

None

7 Code Examples

7.1 Generic usage of SNP facility

The following is an example of how the facility may used be to monitor some associations. Although, callback functions are registered in the code, they are not specified. The sequence of events for set up and tear down is important to note.

```
/* Place SNP facility in debug mode - turn off for now
*/

SNP_Debug(FALSE);

/* Initialize SNP facilities
*/

cond = SNP_Init();
if (cond != SNP_NORMAL) {
    COND_DumpConditions();
    exit(1);
}

/* Register callback functions
*/

cond = SNP_RegisterCallback(callbackState, SNP_CALLBACK_STATE, NULL);
if (cond != SNP_NORMAL) {
    COND_DumpConditions();
    exit(1);
}

cond = SNP_RegisterCallback(callbackITOA, SNP_CALLBACK_ITOA, NULL);
if (cond != SNP_NORMAL) {
    COND_DumpConditions();
    exit(1);
}

cond = SNP_RegisterCallback(callbackATOI, SNP_CALLBACK_ATOI, NULL);
if (cond != SNP_NORMAL) {
    COND_DumpConditions();
    exit(1);
}

/* Commence the snooping with given arguments
*/

cond = SNP_Start(“/dev/le”, 0, “dicom1”, “dicom2”, 104, 20, 5, 32768);
if (cond != SNP_NORMAL) {
    COND_DumpConditions();
    exit(1);
```

```
}


printf("\nInitialization complete .... ready to monitor communica-
tions\n");

/*  Until the correct number of associations have been
    monitored or until something goes wrong keep snooping
    - update user with number of associations to go (Note:
    Global varible "assoc" is the number of associations
    remaining, decrement by callback for state information
    on receiving each END_OF_ASSOC message)
*/

while (assoc > 0) {
    sleep(1);
}

/* If finished in a bad state .... something went wrong
*/

if (current_state != NORMAL) {
    printf("\nError: %s\n",SNP_StateMsg(current_state));
    exit(1);
} else
    printf("\nCompleted monitoring associations normally\n");

/* Discontinue snooping operations
*/

cond = SNP_Stop();
if (cond != SNP_NORMAL)
    COND_DumpConditions();

/* Terminate activities with SNP facility
*/

cond = SNP_Terminate();
if (cond != SNP_NORMAL)
    COND_DumpConditions();
```