NAME

NgMkSockNode, **NgNameNode**, **NgSendMsg**, **NgSendAsciiMsg**, **NgSendReplyMsg**, **NgRecvMsg**, **NgAllocRecvMsg**, **NgRecvAsciiMsg**, **NgAllocRecvAsciiMsg**, **NgSendData**, **NgRecvData**, **NgAllocRecvData**, **NgSetDebug**, **NgSetErrLog** - netgraph user library

LIBRARY

Netgraph User Library (libnetgraph, -lnetgraph)

SYNOPSIS

#include <netgraph.h>

int
NgMkSockNode(*const char *name, int *csp, int *dsp*);

int
NgNameNode(*int cs, const char *path, const char *fmt, ...*);

int
NgSendMsg(*int cs, const char *path, int cookie, int cmd, const void *arg, size_t arglen*);

int
NgSendAsciiMsg(*int cs, const char *path, const char *fmt, ...*);

int
NgSendReplyMsg(*int cs, const char *path, struct ng_mesg *msg, const void *arg, size_t arglen*);

int
NgRecvMsg(*int cs, struct ng_mesg *rep, size_t replen, char *path*);

int
NgAllocRecvMsg(*int cs, struct ng_mesg **rep, char *path*);

int
NgRecvAsciiMsg(*int cs, struct ng_mesg *rep, size_t replen, char *path*);

int
NgAllocRecvAsciiMsg(*int cs, struct ng_mesg **rep, char *path*);

int
NgSendData(*int ds, const char *hook, const u_char *buf, size_t len*);

int
NgRecvData(*int ds, u_char *buf, size_t len, char *hook*);

int
NgAllocRecvData(*int ds, u_char **buf, char *hook*);

int
NgSetDebug(*int level*);

void
NgSetErrLog(*void (*log)(const char *fmt, ...), void (*logx)(const char *fmt, ...)*);

DESCRIPTION

These functions facilitate user-mode program participation in the kernel netgraph(4) graph-based networking system, by utilizing the netgraph *socket* node type (see ng_socket(4)).

The **NgMkSockNode**() function should be called first, to create a new *socket* type netgraph node with associated control and data sockets. If *name* is non-NULL, the node will have that global name assigned to it. The *csp* and *dsp* arguments will be set to the newly opened control and data sockets associated with the node; either *csp* or *dsp* may be NULL if only one socket is desired. The **NgMkSockNode**() function loads the *socket* node type KLD if it is not already loaded.

The **NgNameNode**() function assigns a global name to the node addressed by *path*.

The **NgSendMsg**() function sends a binary control message from the *socket* node associated with control socket *cs* to the node addressed by *path*. The *cookie* indicates how to interpret *cmd*, which indicates a specific command. Extra argument data (if any) is specified by *arg* and *arglen*. The *cookie*, *cmd*, and argument data are defined by the header file corresponding to the type of the node being addressed. The unique, non-negative token value chosen for use in the message header is returned. This value is typically used to associate replies.

Use **NgSendReplyMsg**() to send reply to a previously received control message. The original message header should be pointed to by *msg*.

The **NgSendAsciiMsg**() function performs the same function as **NgSendMsg**(), but adds support for ASCII encoding of control messages. The **NgSendAsciiMsg**() function formats its input a la printf(3) and then sends the resulting ASCII string to the node in a NGM_ASCII2BINARY control message. The node returns a binary version of the message, which is then sent back to the node just as with **NgSendMsg**(). As with **NgSendMsg**(), the message token value is returned. Note that ASCII conversion may not be supported by all node types.

The **NgRecvMsg**() function reads the next control message received by the node associated with control socket *cs*. The message and any extra argument data must fit in *replen* bytes. If *path* is non-NULL, it must point to a buffer of at least NG_PATHSIZ bytes, which will be filled in (and NUL terminated) with the path to the node from which the message was received.

The length of the control message is returned. A return value of zero indicates that the socket was closed.

The **NgAllocRecvMsg**() function works exactly like **NgRecvMsg**(), except that the buffer for a message is dynamically allocated to guarantee that a message is not truncated. The size of the buffer is equal to the socket's receive buffer size. The caller is responsible for freeing the buffer when it is no longer required.

The **NgRecvAsciiMsg**() function works exactly like **NgRecvMsg**(), except that after the message is received, any binary arguments are converted to ASCII by sending a NGM_BINARY2ASCII request back to the originating node. The result is the same as **NgRecvMsg**(), with the exception that the reply arguments field will contain a NUL-terminated ASCII version of the arguments (and the reply header argument length field will be adjusted).

The **NgAllocRecvAsciiMsg**() function works exactly like **NgRecvAsciiMsg**(), except that the buffer for a message is dynamically allocated to guarantee that a message is not truncated. The size of the buffer is equal to the socket's receive buffer size. The caller is responsible for freeing the buffer when it is no longer required.

The **NgSendData**() function writes a data packet out on the specified hook of the node corresponding to data socket *ds*. The node must already be connected to some other node via that hook.

The **NgRecvData**() function reads the next data packet (of up to *len* bytes) received by the node corresponding to data socket *ds* and stores it in *buf*, which must be large enough to hold the entire packet. If *hook* is non-NULL, it must point to a buffer of at least NG_HOOKSIZ bytes, which will be filled in (and NUL terminated) with the name of the hook on which the data was received.

The length of the packet is returned. A return value of zero indicates that the socket was closed.

The **NgAllocRecvData**() function works exactly like **NgRecvData**(), except that the buffer for a data packet is dynamically allocated to guarantee that a data packet is not truncated. The size of the buffer is equal to the socket's receive buffer size. The caller is responsible for freeing the buffer when it is no longer required.

The **NgSetDebug**() and **NgSetErrLog**() functions are used for debugging. The **NgSetDebug**() function

sets the debug level (if non-negative), and returns the old setting. Higher debug levels result in more verbosity. The default is zero. All debug and error messages are logged via the functions specified in the most recent call to **NgSetErrLog**(). The default logging functions are vwarn(3) and vwarnx(3).

At debug level 3, the library attempts to display control message arguments in ASCII format; however, this results in additional messages being sent which may interfere with debugging. At even higher levels, even these additional messages will be displayed, etc.

Note that select(2) can be used on the data and the control sockets to detect the presence of incoming data and control messages, respectively. Data and control packets are always written and read atomically, i.e., in one whole piece.

User mode programs must be linked with the **-l**netgraph flag to link in this library.

INITIALIZATION

To enable netgraph in your kernel, either your kernel must be compiled with **options NETGRAPH** in the kernel configuration file, or else the netgraph(4) and ng_socket(4) KLD modules must have been loaded via kldload(8).

RETURN VALUES

The **NgSetDebug**() function returns the previous debug setting.

The **NgSetErrLog**() function has no return value.

All other functions return -1 if there was an error and set *errno* accordingly.

A return value of zero from **NgRecvMsg**() or **NgRecvData**() indicates that the netgraph socket has been closed.

For **NgSendAsciiMsg**() and **NgRecvAsciiMsg**(), the following additional errors are possible:

[ENOSYS]	The node type does not know how to encode or decode the control message.
[ERANGE]	The encoded or decoded arguments were too long for the supplied buffer.
[ENOENT]	An unknown structure field was seen in an ASCII control message.
[EALREADY]	The same structure field was specified twice in an ASCII control message.
[EINVAL]	ASCII control message parse error or illegal value.

[E2BIG] ASCII control message array or fixed width string buffer overflow.

SEE ALSO

select(2), socket(2), warnx(3), kld(4), netgraph(4), ng_socket(4)

HISTORY

The **netgraph** system was designed and first implemented at Whistle Communications, Inc. in a version of FreeBSD 2.2 customized for the Whistle InterJet.

AUTHORS

Archie Cobbs <*archie@FreeBSD.org*>