NAME

nlsrequest – format and send listener service request message

SYNOPSIS

#include <listen.h>

int nlsrequest(int *fildes***, char** **service_code***);**

extern int _nlslog, t_errno;
extern char *_nlsrmsg;

MT-LEVEL

Unsafe

DESCRIPTION

Given a virtual circuit to a listener process (*fildes*) and a service code of a server process, **nlsrequest( )** formats and sends a *service request message* to the remote listener process requesting that it start the given service. **nlsrequest( )** waits for the remote listener process to return a *service request response message*, which is made available to the caller in the static, null terminated data buffer pointed to by **_nlsrmsg**. The *service request response message* includes a success or failure code and a text message. The entire message is printable.

RETURN VALUES

The success or failure code is the integer return code from **nlsrequest( )**. Zero indicates success, other negative values indicate **nlsrequest( )** failures as follows:

–1 Error encountered by **nlsrequest( )**, see **t_errno**.

Positive values are error return codes from the *listener* process. Mnemonics for these codes are defined in **<listen.h>**.

2 Request message not interpretable.

3 Request service code unknown.

4 Service code known, but currently disabled.

If non-null, **_nlsrmsg** contains a pointer to a static, null terminated character buffer containing the *service request response message*. Note that both **_nlsrmsg** and the data buffer are overwritten by each call to **nlsrequest( )**.

If **_nlslog** is non-zero, **nlsrequest( )** prints error messages on stderr. Initially, **_nlslog** is zero.

FILES

/usr/lib/libnls.a
/usr/lib/libslan.a
/usr/lib/libnsl_s.a

SEE ALSO

nlsadmin(1M), **t_error**(3N)

WARNINGS

nlsrequest( ) cannot always be certain that the remote server process has been successfully started. In this case, **nlsrequest( )** returns with no indication of an error and the caller will receive notification of a disconnect event via a **T_LOOK** error before or during the first **t_snd( )** or **t_rcv( )** call.

NOTES

These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.