NAME

t_alloc – allocate a library structure

SYNOPSIS

cc [*flag* ...] *file* ... **–lnsl** [*library* ...]

#include <tiuser.h>

char *t_alloc(int *fildes*, **int** *struct_type*, **int** *fields***);**

MT-LEVEL

MT-Safe

DESCRIPTION

The **t_alloc( )** function dynamically allocates memory for the various transport function argument structures as specified below. This function will allocate memory for the specified structure, and will also allocate memory for buffers referenced by the structure.

The structure to allocate is specified by *struct_type*, and can be one of the following:

T_BIND	**struct t_bind**
T_CALL	**struct t_call**
T_OPTMGMT	**struct t_optmgmt**
T_DIS	**struct t_discon**
T_UNITDATA	**struct t_unitdata**
T_UDERROR	**struct t_uderr**
T_INFO	**struct t_info**

where each of these structures may subsequently be used as an argument to one or more transport functions.

Each of the above structures, except **T_INFO**, contains at least one field of type **struct netbuf**. **netbuf** is described in **t_connect**(3N). For each field of this type, the user may specify that the buffer for that field should be allocated as well. The *fields* argument specifies this option, where the argument is the bitwise-OR of any of the following:

T_ADDR	The **addr** field of the **t_bind**, **t_call**, **t_unitdata**, or **t_uderr** structures.
T_OPT	The **opt** field of the **t_optmgmt**, **t_call**, **t_unitdata**, or **t_uderr** structures.
T_UDATA	The **udata** field of the **t_call**, **t_discon**, or **t_unitdata** structures.
T_ALL	All relevant fields of the given structure.

For each field specified in *fields*, **t_alloc( )** will allocate memory for the buffer associated with the field, and initialize the **buf** pointer and **maxlen** (see **netbuf** in **t_connect**(3N) for description of **buf** and **maxlen**) field accordingly. The length of the buffer allocated will be based on the same size information that is returned to the user on **t_open**(3N) and **t_getinfo**(3N). Thus, *fildes* must refer to the transport endpoint through which the newly allocated structure will be passed, so that the appropriate size information can be accessed. If the size value associated with any specified field is –1, the underlying service provider can support a buffer of unlimited size. If this is the case, **t_alloc( )** will allocate a buffer with the default size 1024 bytes. See the **NOTES** section for information regarding memory allocation for buffers other than 1024 bytes. If the size value is –2, **t_alloc( )** will set the buffer pointer to **NULL** and the buffer maximum size to 0, and then will return

success (see **t_open**(3N) or **t_getinfo**(3N)).
For any field not specified in *fields*, **buf** will be set to NULL and **maxlen** will be set to zero.

Use of **t_alloc( )** to allocate structures will help ensure the compatibility of user programs with future releases of the transport interface.

RETURN VALUES

On successful completion, **t_alloc( )** returns a pointer to the newly allocated structure. On failure, NULL is returned, **t_errno** is set to indicate the error, and possibly **errno** is set.

ERRORS

On failure, **t_errno** will be set to one of the following:

TBADF The specified file descriptor does not refer to a transport endpoint.

TSYSERR A system error has occurred during execution of this function, **errno** will be set to the specific error.

SEE ALSO

t_connect(3N), **t_free**(3N), **t_getinfo**(3N), **t_open**(3N)

Transport Interfaces Programming Guide

NOTES

If the underlying service provider supports a buffer of unlimited size in the **netbuf** structure (see **t_connect**(3N)), **t_alloc( )** will return a buffer of size 1024 bytes. If a larger size buffer is required, it will need to be allocated separately using a memory allocation routine such as **malloc**(3C). The **buf** and **maxlen** fields of the **netbuf** data structure can then be updated with the address of the new buffer and the 1024 byte buffer originally allocated by **t_alloc( )** can be freed using **free**(3C).

This interface is safe in multithreaded applications.