NAME

rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed, clnt_create_vers, clnt_destroy, clnt_dg_create, clnt_pcreateerror, clnt_raw_create, clnt_spcreateerror, clnt_tli_create, clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr – library routines for dealing with creation and manipulation of CLIENT handles

MT-LEVEL

MT-Safe

DESCRIPTION

RPC library routines allow C language programs to make procedure calls on other machines across the network. First a **CLIENT** handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.

These routines are MT-Safe. In the case of multithreaded applications, the **_REENTRANT** flag must be defined on the command line at compilation time **(-D_REENTRANT).** When the **_REENTRANT** flag is defined, **rpc_createerr** becomes a macro which enables each thread to have its own **rpc_createerr**.

Routines

See **rpc**(3N) for the definition of the **CLIENT** data structure.

#include <rpc/rpc.h>

bool_t clnt_control(CLIENT **clnt*, **const u_int** *req*, **char** **info*);

A function macro to change or retrieve various information about a client object. *req* indicates the type of operation, and *info* is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of *req* and their argument types and what they do are:

CLSET_TIMEOUT	**struct timeval ***	set total timeout
CLGET_TIMEOUT	**struct timeval ***	get total timeout

Note: if you set the timeout using **clnt_control( )**, the timeout argument passed by **clnt_call( )** is ignored in all subsequent calls.

Note: If you set the timeout value to **0 clnt_control( )** immediately returns an error (**RPC_TIMEDOUT**). Set the timeout parameter to **0** for batching calls.

CLGET_FD	**int ***	get the associated file descriptor
CLGET_SVC_ADDR	**struct netbuf ***	get servers address
CLSET_FD_CLOSE	**void**	close the file descriptor when destroying the client handle (see **clnt_destroy( )**)
CLSET_FD_NCLOSE	**void**	do not close the file descriptor when destroying the client handle
CLGET_VERS	**unsigned long ***	get the RPC program's version number associated with the client handle

CLSET_VERS	**unsigned long ***	set the RPC program's version number associated with the client handle. This assumes that the RPC server for this new version is still listening at the address of the previous version.
CLGET_XID	**unsigned long ***	get the XID of the previous remote procedure call
CLSET_XID	**unsigned long ***	set the XID of the next remote procedure call

The following operations are valid for connectionless transports only:

CLSET_RETRY_TIMEOUT	**struct timeval ***	set the retry timeout
CLGET_RETRY_TIMEOUT	**struct timeval ***	get the retry timeout

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request.

clnt_control( ) returns **TRUE** on success and **FALSE** on failure.

CLIENT *clnt_create(const char **host*, **const u_long** *prognum*,
const u_long *versnum*, **const char** **nettype***);**

Generic client creation routine for program *prognum* and version *versnum*. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class of transport protocol to use. The transports are tried in left to right order in **NETPATH** variable or in top to bottom order in the netconfig database.

clnt_create( ) tries all the transports of the *nettype* class available from the **NETPATH** environment variable and the netconfig database, and chooses the first successful one. A default timeout is set and can be modified using **clnt_control( )**. This routine returns **NULL** if it fails. The **clnt_pcreateerror( )** routine can be used to print the reason for failure.

Note: **clnt_create( )** returns a valid client handle even if the particular version number supplied to **clnt_create( )** is not registered with the **rpcbind** service. This mismatch will be discovered by a **clnt_call** later (see **rpc_clnt_calls**(3N)).

CLIENT *clnt_create_timed(const char **host*,
const u_long *prognum*, **const u_long** *versnum*,
const char **nettype*, **const struct timeval** **timeout***);**

Generic client creation routine which is similar to **clnt_create( )** but which also has the additional parameter *timeout* that specifies the maximum amount of time allowed for each transport class tried. In all other respects, the **clnt_create_timed( )** call behaves exactly like the **clnt_create( )** call.

CLIENT *clnt_create_vers(const char ***host*****, const u_long** ***prognum*****, u_long** ***vers_outp*****, const u_long** ***vers_low*****, const u_long** ***vers_high*****, char** ***nettype*****);**

Generic client creation routine which is similar to **clnt_create( )** but which also checks for the version availability. *host* identifies the name of the remote host where the server is located. *nettype* indicates the class transport protocols to be used. If the routine is successful it returns a client handle created for the highest version between *vers_low* and *vers_high* that is supported by the server. *vers_outp* is set to this value. That is, after a successful return *vers_low* **<=** **vers_outp* **<=** *vers_high*. If no version between *vers_low* and *vers_high* is supported by the server then the routine fails and returns NULL. A default timeout is set and can be modified using **clnt_control( )**. This routine returns **NULL** if it fails. The **clnt_pcreateerror( )** routine can be used to print the reason for failure.

Note: **clnt_create( )** returns a valid client handle even if the particular version number supplied to **clnt_create( )** is not registered with the **rpcbind** service. This mismatch will be discovered by a **clnt_call** later (see **rpc_clnt_calls**(3N)). However, **clnt_create_vers( )** does this for you and returns a valid handle only if a version within the range supplied is supported by the server.

void clnt_destroy(CLIENT ***clnt*****);**

A function macro that destroys the client's RPC handle. Destruction usually involves deallocation of private data structures, including *clnt* itself. Use of *clnt* is undefined after calling **clnt_destroy( )**. If the RPC library opened the associated file descriptor, or **CLSET_FD_CLOSE** was set using **clnt_control( )**, the file descriptor will be closed.

The caller should call **auth_destroy(***clnt*→**cl_auth)** (before calling **clnt_destroy( )**) to destroy the associated AUTH structure (see **rpc_clnt_auth**(3N)).

CLIENT *clnt_dg_create(const int *fildes***, const struct netbuf** ***svcaddr*****, const u_long** *prognum***, const u_long** *versnum***, const u_int** *sendsz***, const u_int** *recvsz***);**

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connectionless transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. This routine will resend the call message in intervals of 15 seconds until a response is received or until the call times out. The total time for the call to time out is specified by **clnt_call( )** (see **clnt_call( )** in **rpc_clnt_calls**(3N)). The retry time out and the total time out periods can be changed using **clnt_control( )**. The user may set the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of **0** choose suitable defaults. This routine returns **NULL** if it fails.

void clnt_pcreateerror(const char **s***);**

Print a message to standard error indicating why a client RPC handle could not be created. The message is prepended with the string *s* and a colon, and appended with a newline.

CLIENT *clnt_raw_create(const u_long *prognum***, const u_long** *versnum***);**

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding RPC server should live in the same address space; (see **svc_raw_create( )** in **rpc_svc_create**(3N)). This allows simulation of RPC and measurement of RPC overheads, such as round trip times, without any kernel or networking interference. This routine returns **NULL** if it fails. **clnt_raw_create( )** should be called after **svc_raw_create( )**.

char *clnt_spcreateerror(const char **s***);**

Like **clnt_pcreateerror( )**, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.

Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.

CLIENT *clnt_tli_create(const int *fildes***, const struct netconfig** **netconf***, const struct netbuf** **svcaddr***, const_long** *prognum***, const u_long** *versnum***, const u_int** *sendsz***, const u_int** *recvsz***);**

This routine creates an RPC client handle for the remote program *prognum* and version *versnum*. The remote program is located at address *svcaddr*. If *svcaddr* is **NULL** and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if *svcaddr* is **NULL**, **RPC_UNKNOWNADDR** error is set. *fildes* is a file descriptor which may be open, bound and connected. If it is **RPC_ANYFD**, it opens a file descriptor on the transport specified by *netconf*. If *fildes* is **RPC_ANYFD** and *netconf* is **NULL**, a **RPC_UNKNOWNPROTO** error is set. If *fildes* is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters *sendsz* and *recvsz*; values of **0** choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), **clnt_tli_create( )** calls appropriate client creation routines. This routine returns **NULL** if it fails. The **clnt_pcreateerror( )** routine can be used to print the reason for failure. The remote **rpcbind** service (see **rpcbind**(1M)) is not consulted for the address of the remote service.

CLIENT *clnt_tp_create(const char **host*, **const u_long** *prognum*,
const u_long *versnum*, **const struct netconfig** **netconf*);

Like **clnt_create( )** except **clnt_tp_create( )** tries only one transport specified through *netconf*.

clnt_tp_create( ) creates a client handle for the program *prognum*, the version *versnum*, and for the transport specified by *netconf*. Default options are set, which can be changed using **clnt_control( )** calls. The remote **rpcbind** service on the host *host* is consulted for the address of the remote service. This routine returns **NULL** if it fails. The **clnt_pcreateerror( )** routine can be used to print the reason for failure.

CLIENT *clnt_tp_create_timed(const char **host*, **const u_long** *prognum*,
const u_long *versnum*, **const struct netconfig** **netconf*, **const struct timeval** **timeout*);

Like **clnt_tp_create( )** except **clnt_tp_create_timed( )** has the extra parameter *timeout* which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the **clnt_tp_create_timed( )** call behaves exactly like the **clnt_tp_create( )** call.

CLIENT *clnt_vc_create(const int *fildes*, **const struct netbuf** **svcaddr*,
const u_long *prognum*, **const u_long** *versnum*, **const u_int** *sendsz*,
const u_int *recvsz*);

This routine creates an RPC client for the remote program *prognum* and version *versnum*; the client uses a connection-oriented transport. The remote program is located at address *svcaddr*. The parameter *fildes* is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters *sendsz* and *recvsz*; values of **0** choose suitable defaults. This routine returns **NULL** if it fails.

The address *svcaddr* should not be **NULL** and should point to the actual address of the remote program. **clnt_vc_create( )** does not consult the remote **rpcbind** service for this information.

struct rpc_createerr rpc_createerr;

A global variable whose value is set by any RPC client handle creation routine that fails. It is used by the routine **clnt_pcreateerror( )** to print the reason for the failure.

In multithreaded applications, **rpc_createerr** becomes a macro which enables each thread to have its own **rpc_createerr**.

SEE ALSO

rpc(3N), **rpc_clnt_auth**(3N), **rpc_clnt_calls**(3N), **rpcbind**(1M)