NAME

rpc_clnt_create, **clnt_control**, **clnt_create**, **clnt_create_timed**, **clnt_create_vers**, **clnt_create_vers_timed**, **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

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <rpc/rpc.h>

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

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

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

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

*CLIENT **
clnt_create_vers_timed(*const char * host, const rpcprog_t prognum, rpcvers_t *vers_outp, const rpcvers_t vers_low, const rpcvers_t vers_high, const char *nettype, const struct timeval *timeout*);

void
clnt_destroy(*CLIENT *clnt*);

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

void
clnt_pcreateerror(*const char *s*);

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

*CLIENT **
clnt_raw_create(*const rpcprog_t prognum*, *const rpcvers_t versnum*);

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

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

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

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

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.

Routines

clnt_control()

A function macro to change or retrieve various information about a client object. The *req* argument 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 argument to 0 for batching calls.

CLGET_SVC_ADDR	struct netbuf *	get servers address
CLGET_FD	int *	get fd from handle
CLSET_FD_CLOSE	void	close fd on destroy
CLSET_FD_NCLOSE	void	do not close fd on destroy
CLGET_VERS	uint32_t *	get RPC program version
CLSET_VERS	uint32_t *	set RPC program version
CLGET_XID	uint32_t *	get XID of previous call
CLSET_XID	uint32_t *	set XID of next 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
CLSET_CONNECT	*int* *	use connect(2)

The retry timeout is the time that RPC waits for the server to reply before retransmitting the request. The **clnt_control**() function returns TRUE on success and FALSE on failure.

clnt_create() Generic client creation routine for program *prognum* and version *versnum*. The *host* argument identifies the name of the remote host where the server is located. The *nettype* argument indicates the class of transport protocol to use. The transports are tried in left to right order in NETPATH environment variable or in top to bottom order in the netconfig database. The **clnt_create**() function 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(8) service. This mismatch will be discovered by a **clnt_call**() later (see rpc_clnt_calls(3)).

clnt_create_timed()

Generic client creation routine which is similar to **clnt_create**() but which also has the additional argument *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.

clnt_create_vers()

Generic client creation routine which is similar to **clnt_create**() but which also checks for the version availability. The *host* argument identifies the name of the remote host where the server is located. The *nettype* argument 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. The *vers_outp* argument 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(8) service. This mismatch will be discovered by a **clnt_call**() later (see rpc_clnt_calls(3)). 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.

clnt_create_vers_timed()

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

clnt_destroy()

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(3)).

clnt_dg_create()

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 *fildes* argument 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(3)). 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 *sendsz* and *recvsz* arguments; values of 0 choose suitable defaults. This routine returns NULL if it fails.

clnt_pcreateerror()

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.

clnt_spcreateerror()

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.

clnt_raw_create()

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(3)). 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. The **clnt_raw_create**() function should be called after **svc_raw_create**().

clnt_tli_create()

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. The *fildes* argument 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 *sendsz* and *recvsz* arguments; 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(8)) is not consulted for the address of the remote service.

clnt_tp_create()

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

clnt_tp_create_timed()

Like **clnt_tp_create**() except **clnt_tp_create_timed**() has the extra argument *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.

clnt_vc_create()

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 *fildes* argument is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the *sendsz* and *recvsz* arguments; 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. The **clnt_vc_create**() function 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.

SEE ALSO

rpc(3), rpc_clnt_auth(3), rpc_clnt_calls(3), rpcbind(8)