NAME

rpc_clnt_calls, clnt_call, clnt_freeres, clnt_geterr, clnt_perrno, clnt_perror, clnt_sperrno, clnt_sperror, rpc_broadcast, rpc_broadcast_exp, rpc_call – library routines for client side calls

MT-LEVEL

MT-Safe

DESCRIPTION

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

The **clnt_call( )**, **rpc_call( )**, and **rpc_broadcast( )** routines handle the client side of the procedure call. The remaining routines deal with error handling in the case of errors.

Some of the routines take a **CLIENT** handle as one of the parameters. A **CLIENT** handle can be created by an RPC creation routine such as **clnt_create( )** (see **rpc_clnt_create**(3N)).

These routines are safe for use in multithreaded applications. **CLIENT** handles can be shared between threads, however in this implementation requests by different threads are serialized (that is, the first request will receive its results before the second request is sent).

Routines

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

#include <rpc/rpc.h>

enum clnt_stat clnt_call(CLIENT **clnt*, **const u_long** *procnum*, **const xdrproc_t** *inproc*, **const caddr_t** *in*, **const xdrproc_t** *outproc*, **caddr_t** *out*, **const struct timeval** *tout***);**

A function macro that calls the remote procedure *procnum* associated with the client handle, *clnt*, which is obtained with an RPC client creation routine such as **clnt_create( )** (see **rpc_clnt_create**(3N)). The parameter *inproc* is the XDR function used to encode the procedure's parameters, and *outproc* is the XDR function used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *tout* is the time allowed for results to be returned, which is overridden by a time-out set explicitly through **clnt_control( )**, see **rpc_clnt_create**(3N).

If the remote call succeeds, the status returned is **RPC_SUCCESS**, otherwise an appropriate status is returned.

bool_t clnt_freeres(CLIENT **clnt*, **const xdrproc_t** *outproc*, **caddr_t** *out***);**

A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The parameter *out* is the address of the results, and *outproc* is the XDR routine describing the results. This routine returns **1** if the results were successfully freed, and **0** otherwise.

void clnt_geterr(const CLIENT **clnt*, **struct rpc_err** **errp***);**

A function macro that copies the error structure out of the client handle to the structure at address *errp*.

void clnt_perrno(const enum clnt_stat *stat***);**

Print a message to standard error corresponding to the condition indicated by *stat*. A newline is appended. Normally used after a procedure call fails for a routine for which a client handle is not needed, for instance **rpc_call( ).**

void clnt_perror(const CLIENT **clnt*, **const char** **s***);**

Print a message to the standard error indicating why an RPC call failed; *clnt* is the handle used to do the call. The message is prepended with string *s* and a colon. A newline is appended. Normally used after a remote procedure call fails for a routine which requires a client handle, for instance **clnt_call( )**.

char *clnt_sperrno(const enum clnt_stat *stat***);**

Take the same arguments as **clnt_perrno( )**, but instead of sending a message to the standard error indicating why an RPC call failed, return a pointer to a string which contains the message.

clnt_sperrno( ) is normally used instead of **clnt_perrno( )** when the program does not have a standard error (as a program running as a server quite likely does not), or if the programmer does not want the message to be output with **printf( )** (see **printf**(3S)), or if a message format different than that supported by **clnt_perrno( )** is to be used. Note: unlike **clnt_sperror( )** and **clnt_spcreaterror( )** (see **rpc_clnt_create**(3N)), **clnt_sperrno( )** does not return pointer to static data so the result will not get overwritten on each call.

char *clnt_sperror(const CLIENT **clnt*, **const char** **s***);**

Like **clnt_perror( )**, except that (like **clnt_sperrno( )**) it returns a string instead of printing to standard error. However, **clnt_sperror( )** does not append a newline at the end of the message.

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

enum clnt_stat rpc_broadcast(const u_long *prognum*, **const u_long** *versnum*,
const u_long *procnum*, **const xdrproc_t** *inproc*, **const caddr_t** *in*,
const xdrproc_t *outproc*, **caddr_t** *out*, **const resultproc_t** *eachresult*,
const char **nettype***);**

Like **rpc_call( )**, except the call message is broadcast to all the connectionless transports specified by *nettype*. If *nettype* is **NULL**, it defaults to "**netpath**. Each time it receives a response, this routine calls **eachresult( )**, whose form is:

bool_t eachresult(caddr_t *out*, **const struct netbuf** **addr*,
const struct netconfig **netconf***);**

where *out* is the same as *out* passed to **rpc_broadcast( )**, except that the remote procedure's output is decoded there; *addr* points to the address of the machine that sent the results, and *netconf* is the netconfig structure of the transport on which the remote server responded. If **eachresult( )** returns **0**, **rpc_broadcast( )** waits for more replies; otherwise it returns with appropriate status.

Warning: broadcast file descriptors are limited in size to the maximum transfer size of that transport. For Ethernet, this value is 1500 bytes. **rpc_broadcast( )** uses **AUTH_SYS** credentials by default (see **rpc_clnt_auth**(3N)).

enum clnt_stat rpc_broadcast_exp(const u_long *prognum*, **const u_long** *versnum*,
const u_long *procnum*, **const xdrproc_t** *xargs*, **caddr_t** *argsp*,
const xdrproc_t *xresults*, **caddr_t** *resultsp*, **const resultproc_t** *eachresult*,
const int *inittime*, **const int** *waittime*, **const char** **nettype***);**

Like **rpc_broadcast( )**, except that the initial timeout, *inittime* and the maximum timeout, *waittime* are specified in milliseconds.

inittime is the initial time that **rpc_broadcast_exp( )** waits before resending the request. After the first resend, the re-transmission interval increases exponentially until it exceeds *waittime*.

enum clnt_stat rpc_call(const char **host*, **const u_long** *prognum*,
const u_long *versnum*, **const u_long** *procnum*, **const xdrproc_t** *inproc*,
const char **in*, **const xdrproc_t** *outproc*, **char** **out*, **const char** **nettype***);**

Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine, *host*. The parameter *inproc* is used to encode the procedure's parameters, and *outproc* is used to decode the procedure's results; *in* is the address of the procedure's argument(s), and *out* is the address of where to place the result(s). *nettype* can be any of the values listed on **rpc**(3N). This routine returns RPC_SUCCESS if it succeeds, or an appropriate status is returned. Use the **clnt_perrno( )** routine to translate failure status into error messages.

Warning: **rpc_call( )** uses the first available transport belonging to the class *nettype*, on which it can create a connection. You do not have control of timeouts or authentication using this routine.

SEE ALSO

printf(3S), **rpc**(3N), **rpc_clnt_auth**(3N), **rpc_clnt_create**(3N)