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

SYNOPSIS

 #include <rpc/rpc.h>

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

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

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

 void
 clnt_perrno(*const enum clnt_stat stat*);

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

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

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

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

 enum clnt_stat
 rpc_broadcast_exp(*const rpcprog_t prognum*, *const rpcvers_t versnum*,
  const rpcproc_t 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*);

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

DESCRIPTION

 RPC library routines allow C language programs to make procedure calls on other machines across the net-
 work. 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 arguments. A *CLIENT* handle can be created by an RPC creation routine such as **clnt_create**() (see rpc_clnt_create(3)).

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

clnt_call()

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(3)). The *inproc* argument is the XDR function used to encode the procedure's arguments, 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). The *tout* argument 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(3). If the remote call succeeds, the status returned is RPC_SUCCESS, otherwise an appropriate status is returned.

clnt_freeres()

A function macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. The *out* argument 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.

clnt_geterr()

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

clnt_perrno()

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**().

clnt_perror()

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**().

clnt_sperrno()

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. The **clnt_sperrno**() function 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(3)), or if a message format different than that supported by **clnt_perrno**() is to be used. Note: unlike **clnt_sperror**() and **clnt_spcreateerror**() (see rpc_clnt_create(3)), **clnt_sperrno**() does not return pointer to static data so the result will not get overwritten on each call.

clnt_sperror()

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.

rpc_broadcast()

> Like **rpc_call**(), except the call message is broadcast to all the connectionless transports speci-
> fied 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**() re-
> turns 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. The **rpc_broadcast**() function uses AUTH_SYS creden-
> tials by default (see rpc_clnt_auth(3)).

rpc_broadcast_exp()

> Like **rpc_broadcast**(), except that the initial timeout, *inittime* and the maximum timeout,
> *waittime* are specified in milliseconds. The *inittime* argument is the initial time that
> **rpc_broadcast_exp**() waits before resending the request. After the first resend, the re-trans-
> mission interval increases exponentially until it exceeds *waittime*.

rpc_call()

> Call the remote procedure associated with *prognum*, *versnum*, and *procnum* on the machine,
> *host*. The *inproc* argument is used to encode the procedure's arguments, 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). The *nettype* argument can be any of the values listed on
> rpc(3). 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 cre-
> ate a connection. You do not have control of timeouts or authentication using this routine.

AVAILABILITY

These functions are part of libtirpc.

SEE ALSO

printf(3), rpc(3), rpc_clnt_auth(3), rpc_clnt_create(3)