# RPC Manual Pages

This appendix lists the RPC library calls in UNIX-style manual page format.

## RPC Library Functions

The RPC library calls are listed in alphabetical order in this chapter.

Following a brief introductory statement summarizing its use, each function is described using the documentation style of UNIX.

This table lists the basic components of each function description:

**Table A-1 RPC Library Functions**

| Function | Description |
|---|---|
| Synopsis | A synopsis of the function is given in C language format. The function prototype statement is listed showing all function arguments. |
| Description | A description of the function is given, including any special rules for specifying arguments, alternative uses of the function, and any results returned. |
| Parameters | Each parameter for the call is described. |
| Files | Any required include files are listed in this section. |
| See Also | References to related functions are given. |

# auth_destroy()

Destroy authentication information.

## Synopsis

```
void auth_destroy(auth)
AUTH *auth;
```

## Description

auth_destroy() is a macro that destroys the authentication information associated with auth. Destruction usually involves deallocation of private data structures. The use of auth is undefined after calling auth_destroy(). This routine is called indirectly based on the pointer to the routine passed in the struct AUTH. This routine may also be called using the upper-case AUTH_DESTROY().

## Parameters

| | |
|---|---|
| auth | RPC authentication handle |

## Files

rpc.h - RPC include file

## See Also

authnone_create(), authunix_create(), authunix_create_default()

# authnone_create()

Create RPC authentication handle.

## Synopsis

```
AUTH *authnone_create()
```

## Description

authnone_create() creates and returns an RPC authentication handle that passes nonusable authentication information with each remote procedure call. This is the default used by RPC.

## Files

rpc.h - RPC include file

## See Also

auth_destroy(), authunix_create(), authunix_create_default()

# authunix_create()

Create RPC authentication handle.

## Synopsis

```
AUTH    *authunix_create(host, uid, gid, len, aup_gids)
char    *host;
int     uid, gid, len, *aup_gids;
```

## Description

authunix_create() creates and returns an RPC authentication handle that contains authentication information. It is easy to impersonate a user.

## Parameters

| | |
|---|---|
| host | The name of the machine on which the information was created. |
| uid | The user's user ID. |
| gid | The user's current group ID. |
| len | Refers to a counted array of groups to which the user belongs. |
| aup_gids | Refers to a counted array of groups to which the user belongs. |

## Files

rpc.h - RPC include file

## See Also

auth_destroy(), authnone_create(), authunix_create_default()

# authunix_create_default()

Call authunix with default parameters.

## Synopsis

```
AUTH *authunix_create_default()
```

## Description

authunix_create_default() calls authunix_create() with the appropriate parameters.

## Files

rpc.h - RPC include file

## See Also

auth_destroy(), authnone_create(), authunix_create()

# callrpc()

Call remote procedure.

## Synopsis

```
int callrpc(host, prognum, versnum, procnum, inproc, in, outproc, out)
char         *host;
u_long       prognum, versnum, procnum;
char         *in, *out;
xdrproc_t    inproc, outproc;
```

## Description

callrpc() calls the remote procedure associated with program number, version number, and remote procedure on the machine host. This routine returns zero if it succeeds, or the value of enum clnt_stat cast to an integer if it fails. The routine clnt_perrno() is handy for translating failure statuses into messages.

Calling remote procedures with this routine uses UDP/IP as a transport; see clntudp_create() for restrictions. You do not have control of time-outs or authentication using this routine.

## Parameters

| | |
|---|---|
| host | Machine name |
| prognum | Program number |
| versnum | Version number |
| procnum | Remote procedure |
| inproc | XDR routine used to encode the procedure's parameters |
| in | The address of the procedure's arguments |
| outproc | XDR routine used to decode the procedure's results |
| out | The address at which to place the result(s) |

## Files

rpc.h - RPC include file

## See Also

clnt_stat(), clnt_perrno(), clnttcp_create(), clntudp;_create()

# clnt_broadcast()

Broadcast RPC call message.

**Note** This routine exists but it currently returns failure (-1).

## Synopsis

```
enum clnt_stat clnt_broadcast(prognum, versnum, procnum, inproc, in, outproc,
                              out, eachresult)
u_long           prognum, versnum, procnum;
char             *in, *out;
xdrproc_t        inproc, outproc;
resultproc_t     eachresult;
```

## Description

clnt_broadcast() is like callrpc(), except the call message is broadcast to all locally connected broadcast nets. Each time it receives a response, this routine calls eachresult(), whose form is:

```
eachresult(out, addr)
char    *out;
struct   sockaddr_in *addr;
```

| | |
|---|---|
| out | The same as out passed to clnt_broadcast(), except that the remote procedure's output is decoded there |
| addr | Points to the address of the machine that sent the results. |

If eachresult() returns zero, clnt_broadcast() waits for more replies; otherwise it returns with appropriate status.

Note that broadcast sockets are limited in size to the maximum transfer unit of the data link. For Ethernet, this value is 1500 bytes.

## Parameters

| | |
|---|---|
| prognum | Program number |
| versnum | Version number |
| procnum | Remote procedure |
| inproc | XDR routine used to encode the procedure's parameters |
| in | The address of the procedure's arguments |
| outproc | XDR routine used to decode the procedure's results |
| out | The address of where to place the result(s) |
| eachresult | Routine called on response receipt (see above) |

## Files

rpc.h - RPC include file

pmapclnt.h - Portmapper include file

## See Also

callrpc()

# clnt_call()

Call remote procedure.

## Synopsis

```
enum clnt_stat clnt_call(clnt, procnum, inproc, in, outproc, out, timeout)
CLIENT             *clnt;
ulong              procnum;
xdrproc_t          inproc, outproc;
char               *in, *out;
struct timeval     timeout;
```

## Description

clnt_call() is a 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(). This routine is called indirectly based on the pointer to the routine passed in the struct CLIENT. This call may also be called using the upper-case CLNT_CALL().

## Parameters

| | |
|---|---|
| clnt | Client handle |
| procnum | Remote procedure |
| inproc | XDR routine used to encode the procedure's parameters |
| in | The address of the procedure's argument(s) |
| outproc | XDR routine used to decode the procedure's results |
| out | The address of where to place the result(s) |
| timeout | The time allowed for results to come back |

## Files

rpc.h - RPC include file

## See Also

clnt_stat(), clnt_perrno(), clnttcp_create(), clntudp_create()

# clnt_control()

Change or receive information about client object.

## Synopsis

```
bool_t clnt_control(cl, req, info)
CLIENT       *cl;
int          req;
char         *info
```

## Description

clnt_control() is a macro used 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 UDP and TCP, 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 parameter passed to clnt_call() will be ignored in all future calls.

```
CLGET_SERVER_ADDR   struct sockaddr   get server's address
```

The following operations are valid for UDP 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 UDP RPC waits for the server to reply before retransmitting the request.

This routine is called indirectly based on the pointer to the routine passed in the struct CLIENT. This call may also be called using the upper-case CLNT_CONTROL().

This routine returns TRUE on success and FALSE on failure.

## Parameters

| | |
|---|---|
| cl | Client |
| req | Indicates the type of operation |
| info | A pointer to the information. |

## Files

rpc.h - RPC include file

## See Also

clnt_call()

# clnt_create()

Client creation routine.

## Synopsis

```
CLIENT     *clnt_create(host, prognum, versnum, proto)
char       *host;
u_long     prognum, versnum;
char      *proto;
```

## Description

clnt_create() is a generic client creation routine. Default time-outs are set, but can be modified using clnt_control().

Using UDP has its shortcomings. Since UDP-based RPC messages can only hold up to 8K of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

## Parameters

| | |
|---|---|
| host | Identifies the name of the remote host where the server is located |
| prognum | Remote program number |
| versnum | Remote version number |
| proto | Indicates which kind of transport protocol to use. The currently supported values for this field are udp and tcp. |

## Files

rpc.h - RPC include file

## See Also

clnt_control(), clnt_destroy(), clnttcp_create(), clntudp_create()

# clnt_destroy()

Destroy client's RPC handle.

## Synopsis

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

## Description

clnt_destroy() is a 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 socket, it will close it also. Otherwise, the socket remains open. This routine is called indirectly based on the pointer to the routine passed in the struct CLIENT. This call may also be called using the upper-case CLNT_DESTROY().

## Parameters

| | |
|---|---|
| clnt | Client handle |

## Files

rpc.h - RPC include file

## See Also

clnt_stat(), clnt_perrno(), clntudp_create()

# clnt_freeres()

Free data allocated by result decoding.

## Synopsis

```
bool_t clnt_freeres(clnt, outproc, out)
CLIENT      *clnt;
xdrproc_t   outproc;
char        *out;
```

## Description

clnt_freeres() is a macro that frees any data allocated by the RPC/XDR system when it decoded the results of an RPC call. This routine returns TRUE if the results were successfully freed, and FALSE otherwise. This routine is called indirectly based on the pointer to the routine passed in the struct CLIENT. This routine may also be called using the upper-case CLNT_FREERES().

## Parameters

| | |
|---|---|
| clnt | Client |
| outproc | The XDR routine describing the results |
| out | The address of the results |

## Files

rpc. h - RPC include file

## See Also

clnt_call()

# clnt_geterr()

Get error structure.

## Synopsis

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

## Description

clnt_geterr() is a macro that copies the error structure out of the client handle to the structure at address errp. This routine is called indirectly based on the pointer to the routine passed in the struct CLIENT. This routine may also be called using the upper-case CLNT_GETERR().

## Parameters

| | |
|---|---|
| clnt | Client |
| errp | The address of the error structure |

## Files

rpc.h - RPC include file

## See Also

clnt_call(), clnt_call()

# clnt_pcreateerror()

Print error about client creation.

## Synopsis

```
void clnt_pcreateerror(str)
char       *str;
```

## Description

clnt_pcreateerror() prints a message via the rpclog() facility indicating why a client RPC handle could not be created. The message is prepended with string str and a colon. Used when a clnt_create(), clntraw_create(), clnttcp_create(), or clntudp_create() call fails.

## Parameters

| | |
|---|---|
| str | string to prepend |

## Files

rpc.h - RPC include file

## See Also

clnt_create(), clntraw_create(), clnttcp_create(), clntudp_create(), and clnt_spcreateerror()

# clnt_perrno()

Print standard error.

## Synopsis

```
void clnt_perrno(stat)
enum clnt_stat     stat;
```

## Description

clnt_perrno() prints via the rpclog() facility, a standard error corresponding to the condition indicated by stat. Used after callrpc().

## Parameters

| | |
|---|---|
| stat | error indication |

## Files

rpc.h - RPC include file

## See Also

callrpc(), clnt_call(), clnt_perror(), clnt_sperror(), and clnt_sperror()

# clnt_perror()

Print message for why RPC call failed.

## Synopsis

```
void clnt_perror(clnt, str)
CLIENT        *clnt;
char            *str;
```

## Description

clnt_perror() prints a message via the rpclog() facility indicating why an RPC call failed. The message is prepended with string str and a colon. Used after clnt_call() or callrpc().

## Parameters

| | |
|---|---|
| clnt | The handle used to do the call |
| str | String prepended to message |

## Files

rpc.h - RPC include file

## See Also

clnt_call(), callrpc(), clnt_perrno(), clnt_sperrno(), and clnt_sperror()

# clnt_specreaterror()

Returns a string for why RPC handle could not be created.

## Synopsis

```
char *clnt_spcreateerror(str)
char *str;
```

## Description

clnt_spcreateerror() returns a string indicating why a client RPC handle could not be created. clnt_spcreateerror() is like clnt_pcreateerror(), except that it returns a string instead of using the rpclog() facility.

**Note** Returns pointer to static data that is overwritten on each call.

## Parameters

| | |
|---|---|
| str | String to be prepended to message |

## Files

rpc.h - RPC include file

## See Also

clnt_pcreateerror()

# clnt_sperrno()

Returns pointer to string for why an RPC call failed.

## Synopsis

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

## Description

clnt_sperrno() takes the same arguments as clnt_perrno(), but instead of sending a message to the rpclog() facility indicating why an RPC call failed, returns a pointer to a string which contains the message. The string ends with a newline.

clnt_sperrno() is used instead of clnt_perrno() if the program does not want to use the rpclog() facility, or if a message format different than that supported by clnt_perrno() is to be used.

---

**Note** Unlike clnt_sperror() and clnt_spcreateerror(), clnt_sperrno() does not return a pointer to static data, so the result will not get overwritten on each call.

---

## Parameters

| | |
|---|---|
| stat | error condition |

## Files

rpc.h - RPC include file

## See Also

clnt_perrno(), clnt_sperror(), and clnt_spcreateerror()

# clnt_sperror()

Returns a string for why an RPC call failed.

## Synopsis

```
char *clnt_sperrno(rpch, str)
CLIENT    *rpch;
char      *str;
```

## Description

clnt_sperror( )is like clnt_perror(), except that, like clnt_sperrno(), it returns a string instead of using the rpclog() facility.

**Note** Returns a pointer to static data that is overwritten on each call.

## Parameters

| | |
|---|---|
| rpch | Handle |
| str | String |

## Files

rpc.h - RPC include file

## See Also

clnt_perrno(), and clnt_perror()

# clntraw_create()

Creates an RPC client.

## Synopsis

```
CLIENT   *clntraw_create(prognum, versnum)
u_long   prognum, versnum;
```

## Description

clntraw_create() creates an RPC client for the remote program prognum, version versnum. The transport used to pass messages to the service is actually a buffer within the task's address space, so the corresponding RPC server should live in the same address space; see svcraw_create(). This allows simulation of RPC and acquisition of RPC overheads, such as round trip times, without any network interference. This routine returns NULL if it fails.

## Parameters

| | |
|---|---|
| prognum | remote program |
| versnum | version |

## Files

rpc.h - RPC include file

## See Also

svcraw_create()

# clnttcp_create()

Creates an RPC client which uses TCP.

## Synopsis

```
CLIENT *clnttcp_create(addr, prognum, versnum, sockp, sendsz, recvsz)
struct sockaddr_in     *addr;
u_long                 prognum, versnum;
int                    *sockp;
u_int                  sendsz, recvsz;
```

## Description

clnttcp_create() creates an RPC client for the remote program prognum, version versnum; the client uses TCP/IP as a transport. The remote program is located at Internet address *addr. If addr->sin_port is zero, then it is set to the actual port that the remote program is listening on (the remote portmap service is consulted for this information). The parameter sockp is a socket; if it is RPC_ANYSOCK, then this routine opens a new one and sets sockp. Since TCP-based RPC uses buffered I/O, the user may specify the size of the send and receive buffers with the parameters sendsz and recvsz; values of zero choose suitable defaults. This routine returns NULL if it fails.

## Parameters

| | |
|---|---|
| addr | Internet address |
| prognum | Remote program |
| versnum | Version |
| sockp | Pointer to a socket |
| sendsz | Size of send buffer |
| recvsz | Size of receive buffer |

## Files

rpc.h - RPC include file

## See Also

clntudp_create()

# clntudp_create()

Creates an RPC client which uses TCP.

## Synopsis

```
CLIENT *clntudp_create(addr, prognum, versnum, wait, sockp)
struct sockaddr_in      *addr;
u_long                  prognum, versnum;
struct timeval          wait;
int                     *sockp;
```

## Description

clntudp_create() creates an RPC client for the remote program prognum, version versnum; the client uses UDP/IP as a transport. The remote program is located at Internet address *addr. If addr->sin_port is zero, then it is set to the actual port that the remote program is listening on (the remote portmap service is consulted for this information). The parameter sockp is a socket; if it is RPC_ANYSOCK, then this routine opens a new one and sets sockp. The UDP transport resends the call message in intervals of wait time 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().

Since UDP-based RPC messages can only hold up to 8K of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

## Parameters

| | |
|---|---|
| addr | Internet address |
| prognum | Remote program |
| versnum | Version |
| wait | Time interval to resend message |
| sockp | Socket pointer |

## Files

rpc.h - RPC include file

## See Also

clnttcp_create()

# get_myaddress()

Get machine's IP address.

## Synopsis

```
void get_myaddress(addr)
struct sockaddr_in  *addr;
```

## Description

get_myaddress() returns the machine's IP address in *addr, without consulting the library routines that deal with /etc/hosts. The port number is always set to htons(PMAPPORT).

## Parameters

| | |
|---|---|
| addr | Internet address |

## Files

rpc. h - RPC include file

## See Also

htons()

# getrpcbyname()

Get RPC entry by name.

## Synopsis

```
struct rpcent *getrpcbyname(name)
char *name;
```

## Description

getrpcbyname() returns a pointer to an object with the following structure containing the information returned by the Domain Name Resolver (DNR).

```
structrpcent{
    char  *r_name;                  /* name of server for this rpc program */
    char  **r_aliases;              /* alias list */
    long  r_number;                 /* rpc program number */
};
```

r_name The name of the server for this RPC program.

r_aliases A zero terminated list of alternate names for the RPC program.

r_number The RPC program number for this service.

getrpcbyname() queries DNR with a DFRPCBYN request.

## Parameters

name Name of server

## Files

rpc.h - RPC include file

## See Also

getrpcbynumber()

# getrpcbynumber()

Get RPC entry by number.

## Synopsis

```
struct rpcent *getrpcbynumber(number)
int number;
```

## Description

getrpcbynumber() queries the Domain Name Resolver (DNR) and returns a pointer to an object with the following structure:

```
structrpcent{
    char  *r_name;                  /* name of server for this rpc program */
    char  **r_aliases;              /* alias list */
    long  r_number;                 /* rpc program number */
    };
```

r_name The name of the server for this RPC program.

r_aliases A zero terminated list of alternate names for the RPC program.

r_number The RPC program number for this service.

getrpcbynumber() queries DNR with a DFRPCBYV request.

## Parameters

number RPC program number

## Files

rpc.h - RPC include file

## See Also

getrpcbyname()

# mvs_svc_run()

Call the appropriate service routine for RPC requests.

## Synopsis

```
int mvs_svc_run(ecblistp, ecbcount)
unsigned long    **ecblistp;
int              ecbcount;
```

## Description

mvs_svc_run() waits for RPC requests to arrive, and calls the appropriate service procedure using svc_getreq() when one arrives. mvs_svc_run() is similar to svc_run() but can wait on an ECB list in addition to the socket wait. This call returns if any ECB is posted. This procedure is usually waiting for a select() system call to return. mvs_svc_run() also returns if the API shuts down or encounters a system error.

## Parameters

| | |
|---|---|
| ecblistp | pointer to ECB list |
| ecbcount | ECB count to wait on |

## Files

rpc.h - RPC include file

## See Also

svc_run()

# pmap_getmaps()

Get list of port mappings.

## Synopsis

```
struct pmaplist *pmap_getmaps(addr)
struct sockaddr_in     *addr;
```

## Description

pmap_getmaps() is a user interface to the portmap service, which returns a list of the current RPC program-to-port mappings on the host located at IP address *addr. This routine can return NULL. The rpcinfo utility uses this routine.

## Parameters

| | |
|---|---|
| addr | Internet address |

## Files

rpc.h - RPC include file

pmapclnt.h - Portmapper include file

## See Also

pmap_getport(), pmap_set(), pmap_unset()

# pmap_getport()

Get port number for a service.

## Synopsis

```
u_short pmap_getport(addr, prognum, versnum, protocol)
struct sockaddr_in   *addr;
u_long               prognum, versnum, protocol;
```

## Description

pmap_getmaps() is a user interface to the portmap service, which returns the port number on which a service waits that supports program number prognum, version versnum, and speaks the transport protocol associated with protocol. The value of protocol is most likely IPPROTO_UDP or IPPROTO_TCP. A return value of zero means that the mapping does not exist or that the RPC system failed to contact the remote portmap service. In the latter case, the global variable rpc_createerr() contains the RPC status.

## Parameters

| | |
|---|---|
| addr | Internet address |
| prognum | Remote program number |
| versnum | Version number |
| protocol | Transport protocol |

## Files

rpc.h - RPC include file

pmapclnt.h - Portmapper include file

## See Also

pmap_getmaps(), pmap_set(), pmap_unset()

# pmap_rmtcall()

Tell portmapper to make an RPC call.

## Synopsis

```
enum clnt_stat pmap_rmtcall(addr, prognum, versnum, procnum, inproc, in,
                            outproc, out, timeout, portp)
struct sockaddr_in     *addr;
u_long                 prognum, versnum, procnum;
char                   *in, *out;
xdrproc_t              inproc, outproc;
struct timeval         timeout;
u_long                 *portp;
```

## Description

pmap_rmtcall() is a user interface to the portmap service, which instructs portmap on the host at IP address *addr to make an RPC call on your behalf to a procedure on that host. The parameter *portp will be modified to the program's port number if the procedure succeeds. The definitions of other parameters are discussed in callrpc() and clnt_call(). This procedure should be used for a ping and nothing else. See also clnt_broadcast().

## Parameters

| | |
|---|---|
| addr | Internet address |
| prognum | Remote program number |
| versnum | Version number |
| procnum | Procedure number |
| inproc | XDR procedure used to encode the procedure's parameters |
| in | The address of the procedure's arguments |
| outproc | XDR procedure used to decode the procedure's results |
| out | The address of where to place the result(s) |
| timeout | The time allowed for results to come back |
| portp | Pointer to program's port number |

## Files

rpc.h - RPC include file

pmapclnt.h - Portmapper include file

## See Also

pmap_getmaps(), pmap_getport(), pmap_set(), pmap_unset()

# pmap_set()

Set portmapping.

## Synopsis

```
bool_t pmap_set(prognum, versnum, protocol, port)
u_long    prognum, versnum, protocol;
u_long    port;
```

## Description

pmap_set() is a user interface to the portmap service, which establishes a mapping between the triple (prognum, versnum, protocol) and port on the machine's portmap service. The value of protocol is most likely IPPROTO_UDP or IPPROTO_TCP. This routine returns TRUE if it succeeds, FALSE otherwise. It is automatically done by svc_register().

## Parameters

| | |
|---|---|
| prognum | Program number |
| versnum | Version number |
| protocol | Transport protocol |
| port | Program's port number |

## Files

rpc.h - RPC include file

pmapclnt.h - Portmapper include file

## See Also

pmap_getmaps(), pmap_getport(), pmap_unset()

# pmap_unset()

Unset portmapping.

## Synopsis

```
bool_t pmap_unset(prognum, versnum)
u_long  prognum, versnum;
```

## Description

pmap_unset() is a user interface to the portmap service, which destroys all mapping between the triple (prognum, versnum,*) and ports on the machine's portmap service. This routine returns TRUE if it succeeds, FALSE otherwise.

## Parameters

| | |
|---|---|
| prognum | Program number |
| versnum | Version number |

## Files

rpc.h - RPC include file

pmapclnt.h - Portmapper include file

## See Also

pmap_getmaps(), pmap_getport(), pmap_set()

# registerrpc()

Register a procedure with RPC.

## Synopsis

```
int registerrpc(prognum, versnum, procnum, procname, inproc, outproc)
u_long          prognum, versnum, procnum;
char            *(*procname)();
xdrproc_t       inproc, outproc;
```

## Description

registerrpc() registers procedure procname with the RPC service package. If a request arrives for program prognum, version versnum, and procedure procnum, procname is called with a pointer to its parameter(s); procname should return a pointer to its static result(s); inproc is used to decode the parameters; outproc is used to encode the results. This routine returns zero if the registration succeeded, -1 otherwise.

Remote procedures registered in this form are accessed using the UDP/IP transport; see svcudp_create() for restrictions.

## Parameters

| | |
|---|---|
| prognum | Program number |
| versum | Version number |
| procnum | Procedure number |
| procname | Procedure name |
| inproc | XDR procedure used to decode the procedure's parameters |
| outproc | XDR procedure used to encode the procedure's results |

## Files

rpc.h - RPC include file

## See Also

svcudp_create()

# rpc_createerr

Global variable for unsuccessful client creation.

## Synopsis

```
struct rpc_createerr rpc_createerr
```

## Description

rpc_createerr is a global variable whose value is set by any RPC client creation routine that does not succeed. Use the routine clnt_pcreateerror() to print the reason why.

## Files

rpc.h - RPC include file

## See Also

clnt_pcreateerror()

# svc_destroy()

Destroy RPC transport handle.

## Synopsis

```
void svc_destroy(xprt)
SVCXPRT    *xprt;
```

## Description

svc_destroy() is a macro that destroys the RPC service transport handle, xprt. Destruction usually involves deallocation of private data structures, including xprt itself. Use of xprt is undefined after calling this routine. This routine is called indirectly based on the pointer to the routine passed in the struct SVCXPRT. This routine may also be called using the upper-case SVC_DESTROY().

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(),svc_run(), svc_sendreply() svc_unregister(), svcudp_create(), svctcp_create()

# svc_fdset

Global variable for RPC's file descriptor bit mask.

## Synopsis

```
fd_set svc_fdset;
```

## Description

svc_fdset is a global variable reflecting the RPC service side's read file descriptor bit mask. It is suitable as a parameter to the select system call. This is only of interest if a service implementor does not call svc_run(), but rather does his own asynchronous event processing. This variable may change after calls to svc_getreqset() or any creation routines.

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(),svc_run(), svc_sendreply() svc_unregister(), svctcp_create(), svcudp_create()

# svc_freeargs()

Free data allocated by svc_getargs argument decoding.

## Synopsis

```
bool_t svc_freeargs(xprt, inproc, in)
SVCXPRT      *xprt;
xdrproc_t    inproc;
char         *in;
```

## Description

svc_freeargs() is a macro that frees any data allocated by the RPC/XDR system when it decoded the arguments to a service procedure using svc_getargs(). This routine returns TRUE if the results were successfully freed, and FALSE otherwise. This routine is called indirectly based on the pointer to the routine passed in the struct SVCXPRT. This routine may also be called with the upper-case SVC_FREEARGS().

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |
| inproc | used to encode the procedure's parameters |
| in | the address of the procedure's arguments |

## Files

rpc.h - RPC include file

## See Also

svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(),svc_run(), svc_sendreply() svc_unregister()

# svc_getargs()

Decode RPC request arguments.

## Synopsis

```
bool_t svc_getargs(xprt, inproc, in)
SVCXPRT     *xprt;
xdrproc_t   inproc;
char        *in;
```

## Description

svc_getargs() is a macro that decodes the arguments of an RPC request associated with the RPC service transport handle, xprt. The parameter in is the address where the arguments will be placed; inproc is the XDR routine used to decode the arguments. This routine returns TRUE if decoding succeeds, and FALSE otherwise. This routine is called indirectly based on the pointer to the routine passed in the struct SVCXPRT. This routine may also be called with the upper-case SVC_GETARGS().

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |
| inproc | used to encode the procedure's parameters |
| in | the address of the procedure's arguments |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(),svc_run(), svc_sendreply() svc_unregister()

# svc_getcaller()

Get network address of caller.

## Synopsis

```
struct sockaddr_in *svc_getcaller(xprt)
SVCXPRT    *xprt;
```

## Description

svc_getcaller() is the approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle, xprt.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle. |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getreqset(), svc_getreq(), svc_register(),svc_run(), svc_sendreply() svc_unregister()

# svc_getreq()

Service an RPC request on a socket.

## Synopsis

```
svc_getreq(rdfds)
int rdfds;
```

## Description

svc_getreq() is similar to svc_getreqset(). This interface is obsoleted by svc_getreqset().

## Parameters

| | |
|---|---|
| rdfds | read file descriptor bit mask |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_register(),svc_run(), svc_sendreply() svc_unregister()

# svc_getreqset()

Service an RPC request that arrived on a socket.

## Synopsis

```
svc_getreqset(rdfds)
fd_set   *rdfds;
```

## Description

svc_getreqset() is only of interest if a service implementor does not call svc_run(), but instead implements custom asynchronous event processing. It is called when the select() system call has determined that an RPC request has arrived on some RPC socket(s); rdfds is the resultant read file descriptor bit mask. The routine returns when all sockets associated with the value of rdfds have been serviced.

## Parameters

| | |
|---|---|
| rdfds | read file descriptor bit mask. |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreq(), svc_register(), svc_run(), svc_sendreply(), svc_unregister()

# svc_register()

Register procedure with service dispatch procedure.

## Synopsis

```
bool_t svc_register(xprt, prognum, versnum, dispatch, protocol)
SVCXPRT    *xprt;
u_long     prognum, versnum;
void       (*dispatch) ();
u_long     protocol;
```

## Description

svc_register() associates prognum and versnum with the service dispatch procedure, dispatch. If protocol is zero, the service is not registered with the portmap service. If protocol is non-zero, then a mapping of the triple [prognum, versnum, protocol] to xprt->xp_port is established with the local portmap service (generally protocol is zero, IPPROTO_UDP or IPPROTO_TCP). The procedure dispatch has the following form:

```
dispatch(request, xprt)
struct svc_req *request;
SVCXPRT *xprt;
```

The svc_register() routine returns TRUE if it succeeds, and FALSE otherwise.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |
| prognum | program number |
| versnum | version number |
| dispatch | service dispatch procedure |
| protocol | transport protocol |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_run(), svc_sendreply() svc_unregister()

# svc_run()

Call the appropriate service routine for RPC requests.

## Synopsis

```
void svc_run()
```

## Description

svc_run() waits for RPC requests to arrive, and calls the appropriate service procedure using svc_getreq() when one arrives. This procedure is usually waiting for a select() system call to return. svc_run() doesn't return unless the API shuts down or encounters a system error.

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(), svc_sendreply() svc_unregister()

# svc_sendreply()

Send results of remote procedure call.

## Synopsis

```
bool_t svc_sendreply(xprt, outproc, out)
SVCXPRT     *xprt;
xdrproc_ t      outproc;
char                *out;
```

## Description

svc_sendreply() is called by an RPC service's routine to send the results of a remote procedure call. The parameter xprt is the request's associated transport handle; outproc is the XDR routine which is used to encode the results; and out is the address of the results. This routine returns TRUE if it succeeds, FALSE otherwise.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |
| outproc | XDR routine used to decode the procedure's results |
| out | The address of where to place the result(s) |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(), svc_sendreply() svc_unregister()

# svc_unregister()

Remove mapping to dispatch routines.

## Synopsis

```
void svc_unregister(prognum, versnum)
u_long    prognum, versnum;
```

## Description

svc_unregister() removes all mapping of the double [prognum, versnum] to dispatch routine, and of the triple [prognum, versnum,*] to port number.

## Parameters

| | |
|---|---|
| prognum | Program number |
| versnum | Version number |

## Files

rpc.h - RPC include file

## See Also

svc_freeargs(), svc_getargs(), svc_getcaller(), svc_getreqset(), svc_getreq(), svc_register(), svc_sendreply()

# svcerr_weakauth()

Called when insufficient authentication parameters are given.

## Synopsis

```
void svcerr_weakauth(xprt)
SVCXPRT *xprt;
```

## Description

svcerr_weakauth() is called by a service dispatch routine that refuses to perform a remote procedure call due to insufficient (but correct) authentication parameters. The routine calls svcerr_auth(xprt, AUTH_TOOWEAK).

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svcerr_auth(), svcerr_decode(), svcerr_noproc(), svcerr_noprog(), svcerr_progvers(), svcerr_systemerr()

# svcerr_auth()

Called after an authentication error.

## Synopsis

```
void svcerr_auth(xprt, why)
SVCXPRT           *xprt
enum auth_stat   why;
```

## Description

svcerr_auth() is called by a service dispatch routine that refuses to perform a remote procedure call due to an authentication error.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |
| why | Error |

## Files

rpc.h - RPC include file

## See Also

svcerr_decode(), svcerr_noproc(), svcerr_noprog(), svcerr_progvers(), svcerr_systemerr(), svcerr_weakauth()

# svcerr_decode()

Called for parameter decoding error.

## Synopsis

```
void svcerr_decode(xprt)
SVCXPRT *xprt;
```

## Description

svcerr_decode() is called by a service dispatch routine that cannot successfully decode its parameters. See also svc_getargs().

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svcerr_auth(), svcerr_noproc(), svcerr_noprog(), svcerr_progvers(), svcerr_systemerr(), svcerr_weakauth()

# svcerr_noproc()

Called for procedure number error.

## Synopsis

```
void svcerr_noproc(xprt)
SVCXPRT *xprt;
```

## Description

svcerr_noproc() is called by a service dispatch routine that does not implement the procedure number that the caller requests.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svcerr_auth(), svcerr_decode(), svcerr_noprog(), svcerr_progvers(), svcerr_systemerr(), svcerr_weakauth()

# svcerr_noprog()

Called when program is not registered.

## Synopsis

```
void svcerr_noprog(xprt)
SVCXPRT *xprt;
```

## Description

svcerr_noprog() is called when the desired program is not registered with the RPC package. Service implementors usually do not need this routine.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svcerr_auth(), svcerr_decode(), svcerr_noproc(), svcerr_progvers(), svcerr_systemerr(), svcerr_weakauth()

# svcerr_progvers()

Called when program version is not registered.

## Synopsis

```
void svcerr_progvers(xprt)
SVCXPRT *xprt;
```

## Description

svcerr_progvers() is called when the desired version of a program is not registered with the RPC package. Service implementors usually do not need this routine.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svcerr_auth(), svcerr_decode(), svcerr_noproc(), svcerr_noprog(),svcerr_systemerr(), svcerr_weakauth()

# svcerr_systemerr()

Called when a system error is detected.

## Synopsis

```
void svcerr_systemerr(xprt)
SVCXPRT *xprt;
```

## Description

svcerr_systemerr() is called by a service dispatch routine when it detects a system error not covered by any particular protocol. For example, if a service can no longer allocate storage, it may call this routine.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

svcerr_auth(), svcerr_decode(), svcerr_noproc(), svcerr_noprog(), svcerr_progvers(), svcerr_weakauth()

# svcfd_create()

Create a service on top of any open descriptor.

## Synopsis

```
void svcfd_create(fd, sendsize, recvsize)
int     fd;
u_int   sendsize;
u_int    recvsize;
```

## Description

svcfd_create() creates a service on top of any open descriptor. Typically, this descriptor is a connected socket for a stream protocol such as TCP. sendsize and recvsize indicate sizes for the send and receive buffers. If they are zero, a reasonable default is chosen.

## Parameters

| | |
|---|---|
| fd | Descriptor |
| endsize | Size of send buffer |
| recvsize | Size of receive buffer |

## Files

rpc.h - RPC include file

## See Also

svctcp_create(), svcudp_create()

# svcraw_create()

Create an RPC service transport.

## Synopsis

```
SVCXPRT *svcraw_create()
```

## Description

svcraw_create() creates an RPC service transport, to which it returns a pointer. The transport is really a buffer within the process's address space, so the corresponding RPC client should live in the same address space; see clntraw_create(). This routine allows simulation of RPC and acquisition of RPC overheads (such as round trip times), without any kernel interference. This routine returns NULL if it fails.

## Files

rpc.h - RPC include file

## See Also

clntraw_create()

# svctcp_create()

Create a TCP/IP based service transport.

## Synopsis

```
SVCXPRT *svctcp_create(sock, send_buf_size, recv_buf_size)
int    sock;
u_int  send_buf_size, recv_buf_size;
```

## Description

svctcp_create() creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the socket sock, which may be RPC_ANYSOCK, in which case a new socket is created. If the socket is not bound to a local TCP port, then this routine binds it to an arbitrary port. Upon completion, xprt->xp_sock is the transport's socket descriptor; xprt->xp_port is the transport's port number. This routine returns NULL if it fails. Since TCP- based RPC uses buffered I/O, users may specify the size of buffers; values of zero choose suitable defaults.

## Parameters

| | |
|---|---|
| sock | Socket |
| send_buf_size | Size of send buffer |
| recv_buf_size | Size of receive buffer |

## Files

rpc.h - RPC include file

## See Also

svcudp_create()

# svcudp_create()

Create a UDP/IP based service transport.

## Synopsis

```
SVCXPRT *svcudp_create(sock)
int sock;
```

## Description

svcudp_create() creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the socket sock, which may be RPC_ANYSOCK, in which case a new socket is created. If the socket is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, xprt->xp_sock is the transport's socket descriptor; whereas the field xprt->xp_port is the transport's port number. This routine returns NULL if it fails.

Since UDP-based RPC messages can only hold up to 8K of encoded data, this transport cannot be used for procedures that take large arguments or return huge results.

## Parameters

| | |
|---|---|
| sock | Socket |

## Files

rpc.h - RPC include file

## See Also

svctcp_create()

# xdr_accepted_reply()

Encode RPC reply messages.

## Synopsis

```
bool_t xdr_accepted_reply(xdrs, ar)
XDR                       *xdrs;
struct accepted_reply       *ar;
```

## Description

xdr_accepted_reply() is used for encoding RPC reply messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

This routine returns TRUE if successful, otherwise it returns FALSE.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| ar | Reply |

## Files

rpc.h - RPC include file

## See Also

xdr_authunix_parms(), xdr_callhdr(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmap(), xdr_pmaplist(), xdr_rejected_reply(), xdr_replymsg()

# xdr_authunix_parms()

Describe UNIX credentials.

## Synopsis

```
bool_t xdr_authunix_parms(xdrs, aupp)
XDR                       *xdrs;
struct authunix_parms       *aupp;
```

## Description

xdr_authunix_parms() is used for describing UNIX credentials. This routine is useful for users who wish to generate these credentials without using the RPC authentication package.

This routine returns TRUE if successful, otherwise it returns FALSE.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| aupp | UNIX credentials |

## Files

rpc.h - RPC include file

## See Also

xdr_accepted_reply(), xdr_callhdr(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmap(), xdr_pmaplist(), xdr_rejected_reply(), xdr_replymsg()

# xdr_callhdr()

Describe RPC call header messages.

## Synopsis

```
void xdr_callhdr(xdrs, chdr)
XDR              *xdrs;
struct rpc_msg     *chdr;
```

## Description

xdr_callhdr() is used for describing RPC call header messages. It encodes the static part of the call message header in the XDR language format. It includes information such as transaction ID, RPC version number, program number, and version number. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| chdr | Call header message |

## Files

rpc.h - RPC include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmap(), xdr_pmaplist(), xdr_rejected_reply(), xdr_replymsg()

# xdr_callmsg()

Describe RPC call messages.

## Synopsis

```
bool_t xdr_callmsg(xdrs, cmsg)
XDR               *xdrs;
struct rpc_msg      *cmsg;
```

## Description

xdr_callmsg() is used for describing RPC call messages. It includes all the RPC call information such as transaction ID, RPC version number, program number and version number, authentication information, etc. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

This routine returns TRUE if successful, FALSE otherwise.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| cmsg | Call message |

## Files

rpc.h - RPC include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callhdr(), xdr_opaque_auth(), xdr_pmap(), xdr_pmaplist(), xdr_rejected_reply(), xdr_replymsg()

# xdr_opaque_auth()

Describe RPC authentication information message.

## Synopsis

```
bool_t xdr_opaque_auth(xdrs, ap)
XDR                    *xdrs;
struct opaque_auth       *ap;
```

## Description

xdr_opaque_auth() is used for describing RPC authentication information messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

This routine returns TRUE if successful, FALSE otherwise.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| ap | Authentication information message |

## Files

rpc.h - RPC include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callhdr(), xdr_callmsg(), xdr_pmap(), xdr_pmaplist(), xdr_rejected_reply(), xdr_replymsg()

# xdr_pmap()

Describe parameters to portmap procedures.

## Synopsis

```
bool_t xdr_pmap(xdrs,regs)
XDR             *xdrs;
struct pmap       *regs;
```

## Description

xdr_pmap() is used for describing parameters to various portmap procedures, externally. This routine is useful for users who wish to generate RPC-style messages without using the pmap package.

This routine returns TRUE if successful, FALSE otherwise.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| regs | Portmap parameters |

## Files

rpc.h - RPC include file

pmapport.h - Portmap include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callhdr(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmaplist(), xdr_rejected_reply(), xdr_replymsg()

# xdr_pmaplist()

Describe list of portmappings.

## Synopsis

```
bool_t xdr_pmaplist(xdrs, rp)
XDR                 *xdrs;
struct pmaplist       **rp;
```

## Description

xdr_pmaplist() is used for describing a list of port mappings, externally. This routine is useful for users who wish to generate RPC-style messages without using the pmap interface.

This routine returns TRUE if successful, FALSE otherwise.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| rp | port mapping list |

## Files

rpc.h -RPC include file

pmapport.h - portmap include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callhdr(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmap(), xdr_rejected_reply(), xdr_replymsg()

# xdr_rejected_reply()

Describe RPC reply messages.

## Synopsis

```
bool_t xdr_rejected_reply(xdrs, rr)
XDR                      *xdrs;
struct rejected_reply      *rr;
```

## Description

xdr_rejected_reply() is used for describing RPC reply messages. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

This routine returns TRUE if successful, FALSE otherwise.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| rr | Reply message. |

## Files

rpc.h - RPC include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callhdr(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmap(), xdr_pmaplist(), xdr_replymsg()

# xdr_replymsg()

Describe RPC reply messages.

## Synopsis

```
bool_t xdr_replymsg(xdrs, rmsg)
XDR                 *xdrs;
struct rpc_msg        *rmsg;
```

## Description

xdr_replymsg() is used for describing RPC reply messages. This reply could be an acceptance, rejection, or NULL. This routine is useful for users who wish to generate RPC-style messages without using the RPC package.

This routine returns TRUE if successful, FALSE otherwise.

## Parameters

| | |
|---|---|
| xdrs | XDR structure |
| rms | Reply message |

## Files

rpc.h - RPC include file

## See Also

xdr_accepted_reply(), xdr_authunix_parms(), xdr_callhdr(), xdr_callmsg(), xdr_opaque_auth(), xdr_pmap(), xdr_pmaplist(), xdr_rejected_reply()

# xprt_register()

Register RPC service transport handle.

## Synopsis

```
void xprt_register(xprt)
SVCXPRT *xprt;
```

## Description

xprt_register() is used after RPC service transport handles are created, to register them with the RPC service package. This routine modifies the global variable svc_fds. Service implementors usually do not need this routine.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle. |

## Files

rpc.h - RPC include file

## See Also

xprt_unregister()

# xprt_unregister()

Unregister RPC service transport handle.

## Synopsis

```
void xprt_unregister(xprt)
SVCXPRT *xprt;
```

## Description

xprt_unregister() is used before an RPC service transport handle is destroyed, to unregister it with the RPC service package. This routine modifies the global variable svc_fds. Service implementors usually do not need this routine.

## Parameters

| | |
|---|---|
| xprt | RPC service transport handle |

## Files

rpc.h - RPC include file

## See Also

xprt_unregister()