NAME

rpc_soc, authdes_create, authunix_create, authunix_create_default, callrpc, clnt_broadcast, clntraw_create, clnttcp_create, clntudp_bufcreate, clntudp_create, get_myaddress, getrpcport, pmap_getmaps, pmap_getport, pmap_rmtcall, pmap_set, pmap_unset, registerrpc, svc_fds, svc_getcaller, svc_getreq, svc_register, svc_unregister, svcfd_create, svcraw_create, svctcp_create, svcudp_bufcreate, svcudp_create, xdr_authunix_parms – obsolete library routines for RPC

MT-LEVEL

Unsafe

DESCRIPTION

RPC routines allow C 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. Finally, the procedure call returns to the client.

The routines described in this manual page have been superseded by other routines. The preferred routine is given after the description of the routine. New programs should use the preferred routines, as support for the older interfaces may be dropped in future releases.

File Descriptors

Transport independent RPC uses TLI as its transport interface instead of sockets.

Some of the routines described in this section (such as **clnttcp_create( )**) take a pointer to a file descriptor as one of the parameters. If the user wants the file descriptor to be a socket, then the application will have to be linked with both **librpcsoc** and **libnsl**. If the user passed **RPC_ANYSOCK** as the file descriptor, and the application is linked with **libnsl** only, then the routine will return a TLI file descriptor and not a socket.

Routines

The following routines require that the header **<rpc/rpc.h>** be included. The symbol **PORTMAP** should be defined so that the appropriate function declarations for the old interfaces are included through the header files.

#define PORTMAP
#include <rpc/rpc.h>

AUTH ∗ **authdes_create(char** ∗*name*, **unsigned** *window*, **struct sockaddr** ∗*syncaddr*, **des_block** ∗*ckey*);

authdes_create( ) is the first of two routines which interface to the RPC secure authentication system, known as DES authentication. The second is **authdes_getucred( )**, below. Note: the keyserver daemon **keyserv**(1M) must be running for the DES authentication system to work.

authdes_create( ), used on the client side, returns an authentication handle that will enable the use of the secure authentication system. The first parameter *name* is the network name, or *netname*, of the owner of the server process. This field usually represents a hostname derived from the utility routine **host2netname( )**, but could also represent a user name using **user2netname( )** (see **secure_rpc**(3N)). The second field is window on the validity of the client credential, given in seconds. A small window is more secure than a large one, but choosing too small of a window will increase the frequency of resynchronizations because of clock drift. The third parameter *syncaddr* is optional. If it is NULL, then the authentication system will assume that the local clock is always in sync with the server's clock, and will not attempt resynchronizations. If an address is supplied, however, then the system will use the address for consulting the remote time service whenever resynchronization is required. This parameter is usually the address of the RPC server itself. The final parameter *ckey* is also optional. If it is NULL, then the authentication system will generate a random DES key to be used for the encryption of credentials. If it is supplied, however, then it will be used instead.

Warning: this routine exists for backward compatibility only, and is obsoleted by **authdes_seccreate( )** (see **secure_rpc**(3N)).

AUTH ∗ **authunix_create(char** ∗*host*, **int** *uid* , **int** *gid*, **int** *grouplen*, **int** *gidlistp*);

Create and return an RPC authentication handle that contains authentication information. The parameter *host* is the name of the machine on which the information was created; *uid* is the user's user ID; *gid* is the user's current group ID; *grouplen* and *gidlistp* refer to a counted array of groups to which the user belongs.

Warning: it is not very difficult to impersonate a user.

Warning: this routine exists for backward compatibility only, and is obsoleted by **authsys_create( )** (see **rpc_clnt_auth**(3N)).

AUTH ∗ **authunix_create_default(void)**

Call **authunix_create( )** with the appropriate parameters.

Warning: this routine exists for backward compatibility only, and is obsoleted by **authsys_create_default( )** (see **rpc_clnt_auth**(3N)).

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

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, and *out* is the address of where to place the result(s). This routine returns **0** if it succeeds, or the value of **enum clnt_stat** cast to an integer if it fails. The routine **clnt_perrno( )** (see **rpc_clnt_calls**(3N)) is handy for translating failure statuses into messages.

Warning: you do not have control of timeouts or authentication using this routine. This routine exists for backward compatibility only, and is obsoleted by **rpc_call( )** (see **rpc_clnt_calls**(3N)).

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

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

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

where *out* is 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 **0 clnt_broadcast( )** waits for more replies; otherwise it returns with appropriate status. If **eachresult( )** is NULL, **clnt_broadcast( )** returns without waiting for any replies.

Warning: broadcast packets are limited in size to the maximum transfer unit of the transports involved. For Ethernet, the callers argument size is approximately 1500 bytes. Since the call message is sent to all connected networks, it may potentially lead to broadcast storms. **clnt_broadcast( )** uses SB AUTH_SYS credentials by default (see **rpc_clnt_auth**(3N)).

Warning: this routine exists for backward compatibility only, and is obsoleted by **rpc_broadcast( )** (see **rpc_clnt_calls**(3N)).

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

This routine creates an internal, memory-based RPC client for the remote program *prognum*, version *versnum*. The transport used to pass messages to the service is actually a buffer within the process'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 kernel interference. This routine returns NULL if it fails.

Warning: this routine exists for backward compatibility only, and has the same functionality as **clnt_raw_create( )** (see **rpc_clnt_create**(3N)), which obsoletes it.

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

This routine 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 **0**,, then it is set to the actual port that the remote program is listening on (the remote **rpcbind** service is consulted for this information). The parameter **fdp* is a file descriptor, which may be open and bound; if it is **RPC_ANYSOCK**, then this routine opens a new one and sets **fdp*. Refer to the **File Descriptor** section for more information. 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 **0** choose suitable defaults. This routine returns NULL if it fails.

Warning: this routine exists for backward compatibility only. **clnt_create( )**, **clnt_tli_create( )**, or **clnt_vc_create( )** (see **rpc_clnt_create**(3N)) should be used instead.

CLIENT * clntudp_bufcreate(struct sockaddr_in **addr*, **u_long** *prognum*, **u_long** *versnum*, **struct timeval** *wait*, **int** **fdp*, **u_int** *sendsz*, **u_int** *recvsz***);**

Create a client handle for the remote program *prognum*, on *versnum*; the client uses UDP/IP as the transport. The remote program is located at the Internet address *addr*. If *addr→sin_port* is **0**, it is set to port on which the remote program is listening on (the remote **rpcbind** service is consulted for this information). The parameter **fdp* is a file descriptor, which may be open and bound; if it is **RPC_ANYSOCK**, then this routine opens a new one and sets **fdp*. Refer to the **File Descriptor** section for more information. 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( )** (see **rpc_clnt_calls**(3N)). If successful it returns a client handle, otherwise it returns NULL. The error can be printed using the **clnt_pcreateerror( )** (see **rpc_clnt_create**(3N)) routine.

The user can specify the maximum packet size for sending and receiving by using *sendsz* and *recvsz* arguments for UDP-based RPC messages.

Warning: if *addr→sin_port* is **0** and the requested version number *versnum* is not registered with the remote portmap service, it returns a handle if at least a version number for the given program number is registered. The version mismatch is discovered by a **clnt_call( )** later (see **rpc_clnt_calls**(3N)).

Warning: this routine exists for backward compatibility only. **clnt_tli_create( )** or **clnt_dg_create( )** (see **rpc_clnt_create**(3N)) should be used instead.

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

This routine creates an RPC client handle 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 **0**, then it is set to actual port that the remote program is listening on (the remote **rpcbind** service is consulted for this information). The parameter **fdp* is a file descriptor, which may be open and bound; if it is **RPC_ANYSOCK**, then this routine opens a new one and sets **fdp*. Refer to the **File Descriptor** section for more information. 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( )** (see **rpc_clnt_calls**(3N)). **clntudp_create( )** returns a client handle on success, otherwise it returns NULL. The error can be printed using the **clnt_pcreateerror( )** (see **rpc_clnt_create**(3N)) routine.

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

Warning: this routine exists for backward compatibility only. **clnt_create( )**, **clnt_tli_create( )**, or **clnt_dg_create( )** (see **rpc_clnt_create**(3N)) should be used instead.

void get_myaddress(struct sockaddr_in **addr*);

Places the local system's IP address into **addr*, without consulting the library routines that deal with **/etc/hosts**. The port number is always set to **htons(PMAPPORT)**.

Warning: this routine is only intended for use with the RPC library. It returns the local system's address in a form compatible with the RPC library, and should not be taken as the system's actual IP address. In fact, the **addr* buffer's host address part is actually zeroed. This address may have only local significance and should **NOT** be assumed to be an address that can be used to connect to the local system by remote systems or processes.

Warning: this routine remains for backward compatibility only. The routine **netdir_getbyname( )** (see **netdir**(3N)) should be used with the name **HOST_SELF** to retrieve the local system's network address as a *netbuf* structure.

void getrpcport(char **host*, **int** *prognum*, **int** *versnum*, **int** *proto*)

getrpcport( ) returns the port number for the version *versnum* of the RPC program *prognum* running on *host* and using protocol *proto*. **getrpcport( )** returns **0** if the RPC system failed to contact the remote portmap service, the program associated with *prognum* is not registered, or there is no mapping between the program and a port.

Warning: This routine exists for backward compatibility only. Enhanced functionality is provided by **rpcb_getaddr( )** (see **rpcbind**(3N)).

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

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 command '**rpcinfo** −**p**' uses this routine.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_getmaps( )** (see **rpcbind**(3N)).

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

A user interface to the **portmap** service, which returns the port number on which waits a service that supports program *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 **0** means that the mapping does not exist or that the RPC system failured to contact the remote **portmap** service. In the latter case, the global variable **rpc_createerr** contains the RPC status.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_getaddr( )** (see **rpcbind**(3N)).

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

Request that the **portmap** on the host at IP address **addr* make an RPC on the behalf of the caller to a procedure on that host. **portp* is modified to the program's port number if the procedure succeeds. The definitions of other parameters are discussed in **callrpc( )** and **clnt_call( )** (see **rpc_clnt_calls**(3N)).

Note: this procedure is only available for the UDP transport.

Warning: if the requested remote procedure is not registered with the remote **portmap** then no error response is returned and the call times out. Also, no authentication is done.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_rmtcall( )** (see **rpcbind**(3N)).

bool_t pmap_set(u_long *prognum*, **u_long** *versnum*, **u_long** *protocol*, **u_short** *port***);**

A user interface to the **portmap** service, that establishes a mapping between the triple [*prognum*, *versnum*, *protocol*] and *port* on the machine's **portmap** service. The value of *protocol* may be **IPPROTO_UDP** or **IPPROTO_TCP**. Formerly, the routine failed if the requested *port* was found to be in use. Now, the routine only fails if it finds that *port* is still bound. If *port is not bound, the routine* completes the requested registration. This routine returns **1** if it succeeds, **0** otherwise. Automatically done by **svc_register( )**.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_set( )** (see **rpcbind**(3N)).

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

A user interface to the **portmap** service, which destroys all mapping between the triple [*prognum*, *versnum*, *all-protocols*] and *port* on the machine's **portmap** service. This routine returns one if it succeeds, **0** otherwise.

Warning: this routine exists for backward compatibility only, enhanced functionality is provided by **rpcb_unset( )** (see **rpcbind**(3N)).

int svc_fds;

A global variable reflecting the RPC service side's read file descriptor bit mask; it is suitable as a parameter to the **select( )** 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 is read-only (do not pass its address to **select( )**!), yet it may change after calls to **svc_getreq( )** or any creation routines. Similar to **svc_fdset**, but limited to 32 descriptors.

Warning: this interface is obsoleted by **svc_fdset** (see **rpc_svc_calls**(3N)).

struct sockaddr_in * svc_getcaller(SVCXPRT *_xprt_);

This routine returns the network address, represented as a **struct sockaddr_in**, of the caller of a procedure associated with the RPC service transport handle, *xprt*.

Warning: this routine exists for backward compatibility only, and is obsolete. The preferred interface is **svc_getrpccaller( )** (see **rpc_svc_reg**(3N)), which returns the address as a **struct netbuf**.

void svc_getreq(int _rdfds_);

This routine 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( )** call has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.
This routine is similar to **svc_getreqset( )** but is limited to 32 descriptors.

Warning: this interface is obsoleted by **svc_getreqset( ).**

SVCXPRT * svcfd_create(int _fd_, u_int _sendsz_, u_int _recvsz_);

Create a service on top of any open and bound descriptor. Typically, this descriptor is a connected file descriptor for a stream protocol. Refer to the **File Descriptor** section for more information. *sendsz* and *recvsz* indicate sizes for the send and receive buffers. If they are **0**, a reasonable default is chosen.

Warning: this interface is obsoleted by **svc_fd_create( )** (see **rpc_svc_create**(3N)).

SVCXPRT * svcraw_create(void);

This routine creates an internal, memory-based 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.

Warning: this routine exists for backward compatibility only, and has the same functionality of **svc_raw_create( )** (see **rpc_svc_create**(3N)), which obsoletes it.

SVCXPRT * svctcp_create(int *fd*, **u_int** *sendsz*, **u_int** *recvsz***);**

This routine creates a TCP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor *fd*, which may be **RPC_ANYSOCK**, in which case a new file descriptor is created. If the file descriptor is not bound to a local TCP port, then this routine binds it to an arbitrary port. Refer to the **File Descriptor** section for more information. Upon completion, *xprt→xp_fd* is the transport's file descriptor, and *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 **0** choose suitable defaults.

Warning: this routine exists for backward compatibility only. **svc_create( )**, **svc_tli_create( )**, or **svc_vc_create( )** (see **rpc_svc_create**(3N)) should be used instead.

SVCXPRT * svcudp_bufcreate(int *fd*, **u_int** *sendsz*, **u_int** *recvsz***);**

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor *fd*. If *fd* is **RPC_ANYSOCK ,** then a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, *xprt→xp_fd* is the transport's file descriptor, and *xprt→xp_port* is the transport's port number. Refer to the **File Descriptor** section for more information. This routine returns NULL if it fails.

The user specifies the maximum packet size for sending and receiving UDP-based RPC messages by using the *sendsz* and *recvsz* parameters.

Warning: this routine exists for backward compatibility only. **svc_tli_create( )**, or **svc_dg_create( )** (see **rpc_svc_create**(3N)) should be used instead.

SVCXPRT * svcudp_create(int *fd***);**

This routine creates a UDP/IP-based RPC service transport, to which it returns a pointer. The transport is associated with the file descriptor *fd*, which may be **RPC_ANYSOCK ,** in which case a new file descriptor is created. If the file descriptor is not bound to a local UDP port, then this routine binds it to an arbitrary port. Upon completion, *xprt→xp_fd* is the transport's file descriptor, and *xprt→xp_port* is the transport's port number. This routine returns NULL if it fails.

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

Warning: this routine exists for backward compatibility only. **svc_create( )**, **svc_tli_create( )**, or **svc_dg_create( )** (see **rpc_svc_create**(3N)) should be used instead.

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

Register program *prognum*, procedure *procname*, and version *versnum* 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 while *outproc* is used to encode the results. This routine returns **0** if the registration succeeded, −1 otherwise.

svc_run( ) must be called after all the services are registered.

Warning: this routine exists for backward compatibility only, and is obsoleted by **rpc_reg( )**.

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

Associates *prognum* and *versnum* with the service dispatch procedure, *dispatch*. If *protocol* is **0**, 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 **0**, **IPPROTO_UDP** or **IPPROTO_TCP**). The procedure *dispatch* has the following form:

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

The **svc_register( )** routine returns one if it succeeds, and **0** otherwise.

Warning: this routine exists for backward compatibility only; enhanced functionality is provided by **svc_reg( )**.

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

Remove all mapping of the double [*prognum*, *versnum*] to dispatch routines, and of the triple [*prognum*, *versnum*, *all-protocols*] to port number from **portmap**.

Warning: this routine exists for backward compatibility, enhanced functionality is provided by **svc_unreg( )**.

xdr_authunix_parms(XDR ∗*xdrs***, struct authunix_parms** ∗*aupp***);**

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

Warning: this routine exists for backward compatibility only, and is obsoleted by **xdr_authsys_parms( )** (see **rpc_xdr**(3N)).

SEE ALSO

keyserv(1M), **rpcbind**(1M), **rpcinfo**(1M), **rpc**(3N), **rpc_clnt_auth**(3N), **rpc_clnt_calls**(3N), **rpc_clnt_create**(3N), **rpc_svc_calls**(3N), **rpc_svc_create**(3N), **rpc_svc_err**(3N), **rpc_svc_reg**(3N), **rpcbind**(3N), **secure_rpc**(3N), **select**(3C)

NOTES These interfaces are unsafe in multithreaded applications. Unsafe interfaces should be called only from the main thread.