NAME

rpc – library routines for remote procedure calls

SYNOPSIS

cc [*flag* ...] *file* ... **–lnsl** [*library* ...]

#include <rpc/rpc.h>
#include <netconfig.h>

MT-LEVEL

MT-Safe with exceptions

DESCRIPTION

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

All RPC routines require the header **<rpc/rpc.h>**. Routines that take a **netconfig** structure also require that **<netconfig.h>** be included. Applications using RPC and XDR routines should be linked with the **libnsl** library.

Multithread Considerations

In the case of multithreaded applications, the **_REENTRANT** flag must be defined on the command line at compilation time (**-D_REENTRANT**). Defining this flag enables a thread-specific version of **rpc_createerr** (see **rpc_clnt_create**(3N)).

Client-side routines are MT-Safe. CLIENT handles (see **rpc_clnt_create**(3N)) 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).

Server-side routines are mostly MT-Unsafe. In this implementation the service transport handle, SVCXPRT (see **rpc_svc_create**(3N)), contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. Routines that are affected by this restriction are marked as unsafe for MT applications (see **rpc_svc_calls**(3N)).

Nettype

Some of the high-level RPC interface routines take a *nettype* string as one of the parameters (for example, **clnt_create( )**, **svc_create( )**, **rpc_reg( )**, **rpc_call( )**). This string defines a class of transports which can be used for a particular application.

nettype can be one of the following:

netpath Choose from the transports which have been indicated by their token names in the **NETPATH** environment variable. If **NETPATH** is unset or **NULL**, it defaults to **visible**. **netpath** is the default *nettype*.

visible Choose the transports which have the visible flag (**v**) set in the **/etc/netconfig** file.

circuit_v This is same as **visible** except that it chooses only the connection oriented transports (semantics **tpi_cots** or **tpi_cots_ord**) from the entries in the **/etc/netconfig** file.

datagram_v This is same as **visible** except that it chooses only the connectionless datagram transports (semantics **tpi_clts**) from the entries in the **/etc/netconfig** file.

circuit_n This is same as **netpath** except that it chooses only the connection oriented datagram transports (semantics **tpi_cots** or **tpi_cots_ord**).

datagram_n This is same as **netpath** except that it chooses only the connectionless datagram transports (semantics **tpi_clts**).

udp This refers to Internet UDP.

tcp This refers to Internet TCP.

If *nettype* is **NULL**, it defaults to **netpath**. The transports are tried in left to right order in the **NETPATH** variable or in top to down order in the **/etc/netconfig** file.

Data Structures

Some of the data structures used by the RPC package are shown below.

The AUTH Structure

```
union des_block {
    struct {
        u_int32 high;
        u_int32 low;
    } key;
    char c[8];
};
typedef union des_block des_block;
extern bool_t xdr_des_block( );

/*
* Authentication info. Opaque to client.
*/
struct opaque_auth {
    enum_t   oa_flavor;     /* flavor of auth */
    caddr_t  oa_base;       /* address of more auth stuff */
    u_int    oa_length;     /* not to exceed MAX_AUTH_BYTES */
};

/*
* Auth handle, interface to client authenticators.
*/
typedef struct {
    struct   opaque_auth   ah_cred;
    struct   opaque_auth   ah_verf;
    union    des_block     ah_key;
    struct auth_ops {
        void   (*ah_nextverf)( );
        int    (*ah_marshal)( );   /* nextverf & serialize */
        int    (*ah_validate)( );  /* validate verifier */
        int    (*ah_refresh)( );   /* refresh credentials */
```

```
        void    (*ah_destroy)( );    /* destroy this structure */
    } *ah_ops;
    caddr_t ah_private;
} AUTH;
```

The CLIENT Structure

```
/*
* Client rpc handle.
* Created by individual implementations.
* Client is responsible for initializing
*/
typedef struct {
    AUTH              *cl_auth;           /* authenticator */
    struct clnt_ops {
      enum clnt_stat  (*cl_call)( );      /* call remote procedure */
      void            (*cl_abort)( );     /* abort a call */
      void            (*cl_geterr)( );    /* get specific error code */
      bool_t          (*cl_freeres)( );   /* frees results */
      void            (*cl_destroy)( );   /* destroy this structure */
      bool_t          (*cl_control)( );   /* the ioctl( ) of rpc */
    } *cl_ops;
    caddr_t           cl_private;         /* private stuff */
    char              *cl_netid;          /* network identifier */
    char              *cl_tp;             /* device name */
} CLIENT;
```

The SVCXPRT Structure

```
enum xprt_stat {
    XPRT_DIED,
    XPRT_MOREREQS,
    XPRT_IDLE
};

/*
* Server side transport handle
*/
typedef struct {
    int               xp_fd;              /* file descriptor for the
                                             server handle */
    u_short           xp_port;            /* obsolete */
    struct xp_ops {
      bool_t          (*xp_recv)( );      /* receive incoming requests */
      enum xprt_stat  (*xp_stat)( );      /* get transport status */
      bool_t          (*xp_getargs)( );   /* get arguments */
      bool_t          (*xp_reply)( );     /* send reply */
      bool_t          (*xp_freeargs)( );  /* free mem allocated
                                             for args */
```

```
        void                 (*xp_destroy)( );  /* destroy this struct */
    } *xp_ops;
    int                  xp_addrlen;        /* length of remote addr.
                                               Obsolete */
    char                 *xp_tp;            /* transport provider device
                                               name */
    char                 *xp_netid;         /* network identifier */
    struct netbuf        xp_ltaddr;         /* local transport address */
    struct netbuf        xp_rtaddr;         /* remote transport address */
    char                 xp_raddr[16];      /* remote address. Obsolete */
    struct opaque_auth   xp_verf;           /* raw response verifier */
    caddr_t              xp_p1;             /* private: for use
                                               by svc ops */
    caddr_t              xp_p2;             /* private: for use
                                               by svc ops */
    caddr_t              xp_p3;             /* private: for use
                                               by svc lib */
    int                  xp_type            /* transport type */
} SVCXPRT;
```

The svc_reg Structure

```
struct svc_req {
    u_long               rq_prog;       /* service program number */
    u_long               rq_vers;       /* service protocol version */
    u_long               rq_proc;       /* the desired procedure */
    struct opaque_auth   rq_cred;       /* raw creds from the wire */
    caddr_t              rq_clntcred;   /* read only cooked cred */
    SVCXPRT              *rq_xprt;      /* associated transport */
};
```

The XDR Structure

```
/*
 * XDR operations.
 * XDR_ENCODE causes the type to be encoded into the stream.
 * XDR_DECODE causes the type to be extracted from the stream.
 * XDR_FREE can be used to release the space allocated by an XDR_DECODE
 * request.
 */
enum xdr_op {
   XDR_ENCODE=0,
   XDR_DECODE=1,
   XDR_FREE=2
};
```

```
/*
 * This is the number of bytes per unit of external data.
 */
#define BYTES_PER_XDR_UNIT(4)
#define RNDUP(x)  ((((x) + BYTES_PER_XDR_UNIT - 1) /
            BYTES_PER_XDR_UNIT) \ * BYTES_PER_XDR_UNIT)

/*
 * A xdrproc_t exists for each data type which is to be encoded or
 * decoded.  The second argument to the xdrproc_t is a pointer to
 * an opaque pointer.  The opaque pointer generally points to a
 * structure of the data type to be decoded.  If this points to 0,
 * then the type routines should allocate dynamic storage of the
 * appropriate size and return it.
 * bool_t  (*xdrproc_t)(XDR *, caddr_t *);
 */
typedef  bool_t (*xdrproc_t)( );

/*
 * The XDR handle.
 * Contains operation which is being applied to the stream,
 * an operations vector for the particular implementation
 */
typedef struct {
    enum xdr_op    x_op;      /* operation; fast additional param */
    struct xdr_ops {
      bool_t     (*x_getlong)( );     /* get a long from underlying stream */
      bool_t     (*x_putlong)( );     /* put a long to underlying stream */
      bool_t     (*x_getbytes)( );    /* get bytes from underlying stream */
      bool_t     (*x_putbytes)( );    /* put bytes to underlying stream */
      u_int      (*x_getpostn)( );    /* returns bytes off from beginning */
      bool_t     (*x_setpostn)( );    /* lets you reposition the stream */
      long *     (*x_inline)( );      /* buf quick ptr to buffered data */
      void       (*x_destroy)( );     /* free privates of this xdr_stream */
    } *x_ops;
    caddr_t    x_public;              /* users' data */
    caddr_t    x_private;             /* pointer to private data */
    caddr_t    x_base;                /* private used for position info */
    int        x_handy;               /* extra private word */
} XDR;
```

Index to Routines

The following table lists RPC routines and the manual reference pages on which they are described:

RPC Routine	**Manual Reference Page**
auth_destroy	**rpc_clnt_auth**(3N)
authdes_create	**rpc_soc**(3N)
authdes_getucred	**secure_rpc**(3N)
authdes_seccreate	**secure_rpc**(3N)
authkerb_getucred	**kerberos_rpc**(3N)
authkerb_seccreate	**kerberos_rpc**(3N)
authnone_create	**rpc_clnt_auth**(3N)
authsys_create	**rpc_clnt_auth**(3N)
authsys_create_default	**rpc_clnt_auth**(3N)
authunix_create	**rpc_soc**(3N)
authunix_create_default	**rpc_soc**(3N)
callrpc	**rpc_soc**(3N)
clnt_broadcast	**rpc_soc**(3N)
clnt_call	**rpc_clnt_calls**(3N)
clnt_control	**rpc_clnt_create**(3N)
clnt_create	**rpc_clnt_create**(3N)
clnt_destroy	**rpc_clnt_create**(3N)
clnt_dg_create	**rpc_clnt_create**(3N)
clnt_freeres	**rpc_clnt_calls**(3N)
clnt_geterr	**rpc_clnt_calls**(3N)
clnt_pcreateerror	**rpc_clnt_create**(3N)
clnt_perrno	**rpc_clnt_calls**(3N)
clnt_perror	**rpc_clnt_calls**(3N)
clnt_raw_create	**rpc_clnt_create**(3N)
clnt_spcreateerror	**rpc_clnt_create**(3N)
clnt_sperrno	**rpc_clnt_calls**(3N)
clnt_sperror	**rpc_clnt_calls**(3N)
clnt_tli_create	**rpc_clnt_create**(3N)
clnt_tp_create	**rpc_clnt_create**(3N)
clnt_udpcreate	**rpc_soc**(3N)
clnt_vc_create	**rpc_clnt_create**(3N)
clntraw_create	**rpc_soc**(3N)
clnttcp_create	**rpc_soc**(3N)
clntudp_bufcreate	**rpc_soc**(3N)
get_myaddress	**rpc_soc**(3N)
getnetname	**secure_rpc**(3N)
host2netname	**secure_rpc**(3N)
key_decryptsession	**secure_rpc**(3N)
key_encryptsession	**secure_rpc**(3N)
key_gendes	**secure_rpc**(3N)
key_setsecret	**secure_rpc**(3N)

netname2host	**secure_rpc**(3N)
netname2user	**secure_rpc**(3N)
pmap_getmaps	**rpc_soc**(3N)
pmap_getport	**rpc_soc**(3N)
pmap_rmtcall	**rpc_soc**(3N)
pmap_set	**rpc_soc**(3N)
pmap_unset	**rpc_soc**(3N)
rac_drop	**rpc_rac**(3N)
rac_poll	**rpc_rac**(3N)
rac_recv	**rpc_rac**(3N)
rac_send	**rpc_rac**(3N)
registerrpc	**rpc_soc**(3N)
rpc_broadcast	**rpc_clnt_calls**(3N)
rpc_broadcast_exp	**rpc_clnt_calls**(3N)
rpc_call	**rpc_clnt_calls**(3N)
rpc_reg	**rpc_svc_calls**(3N)
svc_create	**rpc_svc_create**(3N)
svc_destroy	**rpc_svc_create**(3N)
svc_dg_create	**rpc_svc_create**(3N)
svc_dg_enablecache	**rpc_svc_calls**(3N)
svc_fd_create	**rpc_svc_create**(3N)
svc_fds	**rpc_soc**(3N)
svc_freeargs	**rpc_svc_reg**(3N)
svc_getargs	**rpc_svc_reg**(3N)
svc_getcaller	**rpc_soc**(3N)
svc_getreq	**rpc_soc**(3N)
svc_getreqset	**rpc_svc_calls**(3N)
svc_getrpccaller	**rpc_svc_calls**(3N)
svc_kerb_reg	**kerberos_rpc**(3N)
svc_raw_create	**rpc_svc_create**(3N)
svc_reg	**rpc_svc_calls**(3N)
svc_register	**rpc_soc**(3N)
svc_run	**rpc_svc_reg**(3N)
svc_sendreply	**rpc_svc_reg**(3N)
svc_tli_create	**rpc_svc_create**(3N)
svc_tp_create	**rpc_svc_create**(3N)
svc_unreg	**rpc_svc_calls**(3N)
svc_unregister	**rpc_soc**(3N)
svc_vc_create	**rpc_svc_create**(3N)
svcerr_auth	**rpc_svc_err**(3N)
svcerr_decode	**rpc_svc_err**(3N)
svcerr_noproc	**rpc_svc_err**(3N)
svcerr_noprog	**rpc_svc_err**(3N)
svcerr_progvers	**rpc_svc_err**(3N)
svcerr_systemerr	**rpc_svc_err**(3N)

svcerr_weakauth	**rpc_svc_err**(3N)
svcfd_create	**rpc_soc**(3N)
svcraw_create	**rpc_soc**(3N)
svctcp_create	**rpc_soc**(3N)
svcudp_bufcreate	**rpc_soc**(3N)
svcudp_create	**rpc_soc**(3N)
user2netname	**secure_rpc**(3N)
xdr_accepted_reply	**rpc_xdr**(3N)
xdr_authsys_parms	**rpc_xdr**(3N)
xdr_authunix_parms	**rpc_soc**(3N)
xdr_callhdr	**rpc_xdr**(3N)
xdr_callmsg	**rpc_xdr**(3N)
xdr_opaque_auth	**rpc_xdr**(3N)
xdr_rejected_reply	**rpc_xdr**(3N)
xdr_replymsg	**rpc_xdr**(3N)
xprt_register	**rpc_svc_calls**(3N)
xprt_unregister	**rpc_svc_calls**(3N)

FILES

/etc/netconfig

SEE ALSO

getnetconfig(3N), **getnetpath**(3N), **kerberos_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), **rpc_xdr**(3N), **rpcbind**(3N), **secure_rpc**(3N), **xdr**(3N), **netconfig**(4), **rpc**(4), **environ**(5)