NAME

rpc — library routines for remote procedure calls

SYNOPSIS

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

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 *struct netconfig* also require that <netconfig.h> be included.

Nettype

Some of the high-level RPC interface routines take a *nettype* string as one of the arguments (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.

The *nettype* argument can be one of the following:

netpath	Choose from the transports which have been indicated by their token names in the NETPATH environment variable. 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, both version 4 and 6.
tcp	This refers to Internet TCP, both version 4 and 6.

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.

Derived Types

The derived types used in the RPC interfaces are defined as follows:

```
typedef u_int32_t rpcprog_t;
typedef u_int32_t rpcvers_t;
typedef u_int32_t rpcproc_t;
typedef u_int32_t rpcprot_t;
typedef u_int32_t rpcport_t;
typedef   int32_t rpc_inline_t;
```

Data Structures

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

The AUTH Structure

```
/*
 * 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 side authenticators.
 */
typedef struct {
    struct    opaque_auth    ah_cred;
    struct    opaque_auth    ah_verf;
    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 auth.
 */

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 */
    const 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 */
    struct sockaddr_in    xp_raddr; /* Obsolete */
    const struct xp_ops2 {
        bool_t    (*xp_control)();    /* catch-all function */
    } *xp_ops2;
    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 */
    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 {
    rpcprog_t    rq_prog;    /* service program number */
    rpcvers_t    rq_vers;    /* service protocol version */
    rpcproc_t    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 */
    u_int    x_handy;    /* extra private word */
} XDR;

/*
 * The netbuf structure. This structure is defined in <xti.h> on SysV
 * systems, but NetBSD / FreeBSD do not use XTI.
```

```
 *
 * Usually, buf will point to a struct sockaddr, and len and maxlen
 * will contain the length and maximum length of that socket address,
 * respectively.
 */
struct netbuf {
        unsigned int maxlen;
        unsigned int len;
        void *buf;
};

/*
 * The format of the address and options arguments of the XTI t_bind call.
 * Only provided for compatibility, it should not be used other than
 * as an argument to svc_tli_create().
 */

struct t_bind {
        struct netbuf   addr;
        unsigned int    qlen;
};
```

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(3)
authdes_create()	rpc_soc(3)
authnone_create()	rpc_clnt_auth(3)
authsys_create()	rpc_clnt_auth(3)
authsys_create_default()	rpc_clnt_auth(3)
authunix_create()	rpc_soc(3)
authunix_create_default()	rpc_soc(3)
callrpc()	rpc_soc(3)
clnt_broadcast()	rpc_soc(3)
clnt_call()	rpc_clnt_calls(3)
clnt_control()	rpc_clnt_create(3)
clnt_create()	rpc_clnt_create(3)
clnt_create_timed()	rpc_clnt_create(3)
clnt_create_vers()	rpc_clnt_create(3)
clnt_create_vers_timed()	rpc_clnt_create(3)
clnt_destroy()	rpc_clnt_create(3)
clnt_dg_create()	rpc_clnt_create(3)
clnt_freeres()	rpc_clnt_calls(3)
clnt_geterr()	rpc_clnt_calls(3)
clnt_pcreateerror()	rpc_clnt_create(3)
clnt_perrno()	rpc_clnt_calls(3)
clnt_perror()	rpc_clnt_calls(3)
clnt_raw_create()	rpc_clnt_create(3)

clnt_spcreateerror()	rpc_clnt_create(3)
clnt_sperrno()	rpc_clnt_calls(3)
clnt_sperror()	rpc_clnt_calls(3)
clnt_tli_create()	rpc_clnt_create(3)
clnt_tp_create()	rpc_clnt_create(3)
clnt_tp_create_timed()	rpc_clnt_create(3)
clnt_udpcreate()	rpc_soc(3)
clnt_vc_create()	rpc_clnt_create(3)
clntraw_create()	rpc_soc(3)
clnttcp_create()	rpc_soc(3)
clntudp_bufcreate()	rpc_soc(3)
get_myaddress()	rpc_soc(3)
pmap_getmaps()	rpc_soc(3)
pmap_getport()	rpc_soc(3)
pmap_rmtcall()	rpc_soc(3)
pmap_set()	rpc_soc(3)
pmap_unset()	rpc_soc(3)
registerrpc()	rpc_soc(3)
rpc_broadcast()	rpc_clnt_calls(3)
rpc_broadcast_exp()	rpc_clnt_calls(3)
rpc_call()	rpc_clnt_calls(3)
rpc_reg()	rpc_svc_calls(3)
svc_create()	rpc_svc_create(3)
svc_destroy()	rpc_svc_create(3)
svc_dg_create()	rpc_svc_create(3)
svc_dg_enablecache()	rpc_svc_calls(3)
svc_fd_create()	rpc_svc_create(3)
svc_fds()	rpc_soc(3)
svc_freeargs()	rpc_svc_reg(3)
svc_getargs()	rpc_svc_reg(3)
svc_getcaller()	rpc_soc(3)
svc_getreq()	rpc_soc(3)
svc_getreqset()	rpc_svc_calls(3)
svc_getrpccaller()	rpc_svc_calls(3)
svc_kerb_reg()	kerberos_rpc(3)
svc_raw_create()	rpc_svc_create(3)
svc_reg()	rpc_svc_calls(3)
svc_register()	rpc_soc(3)
svc_run()	rpc_svc_reg(3)
svc_sendreply()	rpc_svc_reg(3)
svc_tli_create()	rpc_svc_create(3)
svc_tp_create()	rpc_svc_create(3)
svc_unreg()	rpc_svc_calls(3)
svc_unregister()	rpc_soc(3)
svc_vc_create()	rpc_svc_create(3)
svcerr_auth()	rpc_svc_err(3)
svcerr_decode()	rpc_svc_err(3)
svcerr_noproc()	rpc_svc_err(3)
svcerr_noprog()	rpc_svc_err(3)

svcerr_progvers()	rpc_svc_err(3)
svcerr_systemerr()	rpc_svc_err(3)
svcerr_weakauth()	rpc_svc_err(3)
svcfd_create()	rpc_soc(3)
svcraw_create()	rpc_soc(3)
svctcp_create()	rpc_soc(3)
svcudp_bufcreate()	rpc_soc(3)
svcudp_create()	rpc_soc(3)
xdr_accepted_reply()	rpc_xdr(3)
xdr_authsys_parms()	rpc_xdr(3)
xdr_authunix_parms()	rpc_soc(3)
xdr_callhdr()	rpc_xdr(3)
xdr_callmsg()	rpc_xdr(3)
xdr_opaque_auth()	rpc_xdr(3)
xdr_rejected_reply()	rpc_xdr(3)
xdr_replymsg()	rpc_xdr(3)
xprt_register()	rpc_svc_calls(3)
xprt_unregister()	rpc_svc_calls(3)

FILES

/etc/netconfig

AVAILABILITY

These functions are part of libtirpc.

SEE ALSO

getnetconfig(3), getnetpath(3), rpcbind(3), rpc_clnt_auth(3), rpc_clnt_calls(3), rpc_clnt_create(3), rpc_svc_calls(3), rpc_svc_create(3), rpc_svc_err(3), rpc_svc_reg(3), rpc_xdr(3), xdr(3), netconfig(5)