EDOC038

EUROGAM PROJECT

NSF DATA ACQUISITION SYSTEM

Remote Procedure Call - C library

Edition 1.0
October 1990

NSF Software Systems Group
UK Science and Engineering Research Council
Daresbury Laboratory

Introduction

This document describes the extensions to the C language library for Sun Remote Procedure Calls (RPC) used by the NSF data acquisition system. The library is an extended version of the standard library made from source code available from Sun. The library is used by the Network Tape System client side implementation described in *Network Tape System — C library* (EDOC039.) The reader is referred to the Sun manual "Network Programming" and to the man (1) pages for RPC (3N) for further details of standard RPC. Sun copyright still applies.

Summary

The library implements Sun RPC from the Sun source with some modifications and additions. The library makes it possible for the client side to call a remote procedure on a known UDP port without calling the portmapper. It is possible for the server side to register a program for use with a given UDP port. The library implements "asynchronous" RPC; in this model after an RPC call has been made control returns to the caller without waiting for a reply. A further call to the library may be made to obtain the next outstanding reply. These changes apply to the intermediate layer and the UDP based lowest layer only.

Description

Some public routines have been created to allow the passing of a *portnum* parameter; they are **callrpcport()**, **registerrpcport()** and **svcudp_createport()**.

Some routines have been written to implement asynchronous RPC; they are **callasync()**, **replyasync()**, **setblockasync()**, **clntudp_ascreate()**, **clntudp_callonly()** and **clntudp_reply()**.

One routine has been modified in the implementation of asynchronous RPC; it is **clnt_sperrno()**.

Two new RPC statuses or error numbers have been added to *clnt.h*. These are RPC_NOREPLYYET (error 18) and RPC_NOCALLS (error 19.)

Asynchronous RPC may be used in blocking or non-blocking mode, by default the library operates in non-blocking mode. In blocking mode programs will block in the reply call until a reply is received from a remote procedure. In non-blocking mode the procedure returns immediately even if no reply has been received yet.

By using asynchronous RPC a form of batching may be implemented in an application program by sending several requests without waiting for a reply. It should be noted however that the remote server will send replies to all the requests it receives. Some action must therefore be taken at the client end, either to receive and act on or to explicitly discard unwanted replies. In the present implementation there can be no more than 16 calls waiting for a reply at any one time.

This library is based on Sun RPC version 3.9. Programs using these facilities must be linked with the library *librpclib.a* and the correct version of the header file (<rpc/clnt.h>) must be used.

Files

There is a new source code file called *clnt_async.c*, it contains all the C source code for the new mode. *Clnt_async.c* is based on the Sun source files *clnt_simple.c* and *clnt_udp.c*. The contents of this file are only of interest to those who are developing the facility further.

Several files have been modified; they are *clnt.h*, *clnt_simple.c*, *clnt_udp.c*, *clnt_perror.c*, *svc_simple.c* and *svc_udp.c*.

The compiled library is in *librpclib.a*.

Functions

int callrpcport(host, portnum, prognum, versnum, procnum, inproc, in, outproc, out)

```
char *host;
u_long portnum, prognum, versnum, procnum;
char *in, *out;
xdrproc_t inproc, outproc;
```

This routine is exactly like the standard **callrpc()** (see RPC (3N)) with the addition of the *portnum* parameter. If *portnum* is 0 the routine behaves just like **callrpc()**. Otherwise the routine assumes the program uses UDP port number *portnum* and calls that port without first calling the remote machine's portmapper to discover the UDP port.

int registerrpcport(portnum, prognum, versnum, procnum, procname, inproc, outproc)

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

This routine is exactly like the standard **registerrpc()** (see RPC (3N)) with the addition of the *portnum* parameter. If *portnum* is 0 the routine behaves just like **registerrpc()**, otherwise the routine assumes the program uses UDP port number *portnum* and registers the program as such.

SVCXPRT *svcudp_createport(sock, portnum)

```
int sock, portnum;
```

This routine is exactly like the standard **svcudp_create()** (see RPC (3N)) with the addition of the *portnum* parameter. If *portnum* is 0 the routine behaves just like **svcudp_create()**, otherwise the routine assumes the program uses UDP port *portnum* and binds the socket *sock* to that port.

int callasync(host, portnum, prognum, versnum, procnum, inproc, in, outproc, out, handle)

```
char *host;
u_long portnum, prognum, versnum, procnum;
char *in, *out;
xdrproc_t inproc, outproc;
caddr_t handle;
```

This routine creates an RPC client. The call is analogous to the creation and calling part of the **callrpcport()** routine, unlike **callrpcport()** , however, it does not wait for a reply. The routine returns (as an integer) the RPC status of the underlying RPC library calls. Private information to identify the caller is passed to the routine in the parameter *handle*. This information is stored internally by the library. If there is no space for a new asynchronous RPC descriptor the error RPC_NOCALLS will be returned. In this implementation there can be up to 16 descriptors active at any one time.

int replyasync(handleptr)

```
caddr_t *handleptr;
```

This routine looks for replies from RPC calls made by **callasync()**. The call is analogous to the reply part of **callrpc()** or **callrpcport()**. If a reply is found the caller's identifier for the original RPC request is returned in *handleptr*. If the library has been set to blocking mode control will remain in this routine until a reply is received or an error (including timeout) occurs. In non-blocking mode the routine returns immediately. In non-blocking mode the error RPC_NOREPLYYET is returned if there are no replies. If there are no calls waiting for replies the error RPC_NOCALLS is returned.

void setblockasync(set)

int set;

This call sets the asynchronous library into blocking or non-blocking mode. If *set* has the value 0 the library operates in non-blocking mode otherwise it operates in blocking mode.

NOTE.

Once the library has been set to blocking mode it catches the signals SIGIO and SIGALRM. The sockets for all calls have their FASYNC flags set (see fcntl (2V).)

CLIENT *clntudp_ascreate (raddr, program, version, wait, sockp)

struct sockaddr_in *raddr;

u_long program;

u_long version;

struct timeval wait;

register int *sockp;

This routine is analogous to **clntudp_create()** except that rather than create a “normal” RPC client it creates an asynchronous RPC client.

static enum clnt_stat clntudp_callonly (cl, proc, xargs, argsp, utimeout)

register CLIENT *cl; /* client handle */

u_long proc; /* procedure number */

xdrproc_t xargs; /* xdr routine for args */

caddr_t argsp; /* pointer to args */

struct timeval utimeout; /* seconds to wait before giving up */

This routine is analogous to **clntudp_call()** (the routine used by the **clnt_call()** macro when CLIENT is using the UDP service) except that rather than call the remote procedure and wait for a reply the routine just makes the call and returns.

static enum clnt_stat clntudp_reply (clptr)

CLIENT **clptr;

This routine carries out the reply functions of **clntudp_call()** in the synchronous model. It looks for a reply to any of the outstanding calls. If there are no calls outstanding RPC_NOCALLS is returned. If there are no replies waiting RPC_NOREPLYYET is returned. If any call has timed out RPC_TIMEDOUT is returned and the CLIENT for the appropriate call is returned at the location pointed to by*clptr*. If there is a successful reply to one of the waiting calls the CLIENT for the appropriate call is returned at the location pointed to by *clptr*.

char *clnt_sperrno (stat)

enum clnt_stat stat;

The list of error messages used by this routine has been enlarged to include RPC_NOREPLYYET and RPC_NOCALLS.