Overview [1]

Note: The RPC feature is included in the trusted computing base (TCB) of the Trusted UNICOS system. The operator functions, procedures, and duties outlined in the following subsections are required to maintain a Trusted UNICOS system. No special security administrator or operator functions are necessary for the management of RPC on a Trusted UNICOS system.

Remote procedure calls (RPCs) provide a way for you do the following:

- Distribute program segments across computers in a network
- Communicate with more than one machine on a given network while executing a program
- Communicate with other programs that run on the same machine

The typical configuration for environments that use RPCs consists of workstations connected to a computer through a network. The workstations are used for application interface and high-resolution displays; the computationally intense part of the code runs on the computer.

A program must be registered so that other programs on the network can find it. (See an example of registering in subsection 2.1, page 12.) RPCs use a client/server paradigm in which the client first sends data to a server running in a machine on a network. The server receives the data packet, processes it as required, and returns a result to the client. The server does not have to return any information to the client. In C language context, the server can be a function of type `void`. The same is true for the client; it can call a server without passing data to it. Figure 1 demonstrates the typical RPC paradigm.

RPC and XDR
1.1

The machines on a given network can run in different operating system environments. Programs that use RPC are shielded from the calling conventions of these various operating systems by the use of data translation routines known as External Data Representation (XDR) routines. XDR is a protocol that allows programmers to specify arbitrary data structures that are independent of a specific machine. These routines ensure that data of any type can be passed successfully between machines with potentially different word sizes or other architectural differences.

XDR routines act as filters for the data moving back and forth, ensuring that the data is translated into a form that the receiving machine can interpret correctly. Translating data from the sending machine into XDR format is called *serializing*. When the receiving machine interprets the serialized data, the process is called *deserializing*. The XDR routines do the serializing and deserializing for each communication between server and client.

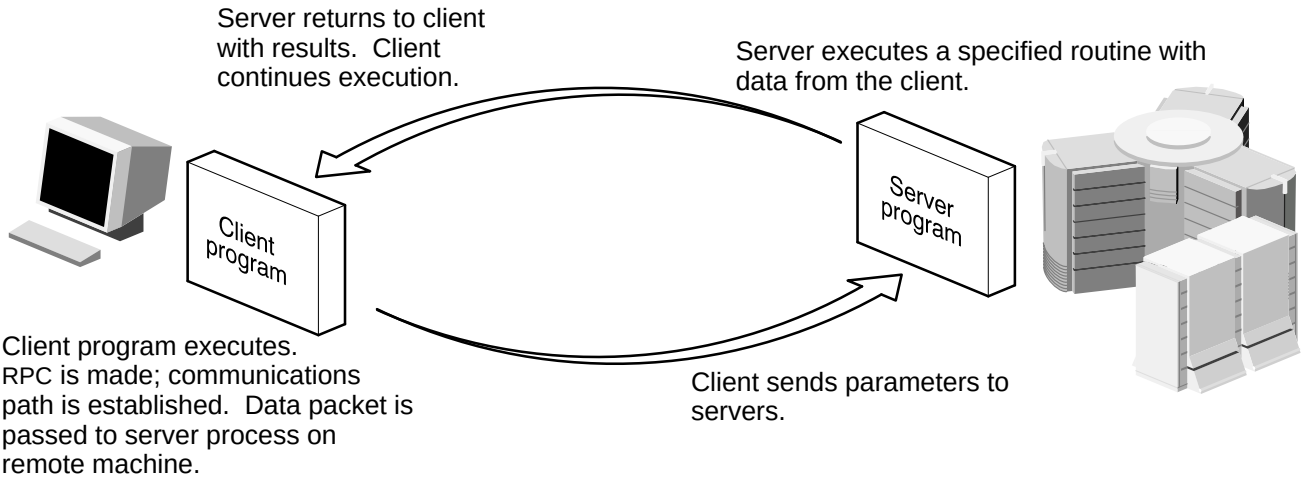

Figure 1. RPC paradigm

The C library (`libc.a`) contains predefined XDR routines for passing most data types. If only one request value is being passed to the server and one result value is being returned, XDR routines from the library can be used. However, when a structure is being passed, the programmer must construct an XDR routine that maps the structure members to predefined XDR routines by member type. The `rpcgen` utility can automate the writing of structure XDR conversion routines. See appendix B, page 97, for information on `rpcgen`. Of course, if

the data going to and from the client and server is the same in type (for example, the client passes a structure of two integers to the server, and the server passes a structure of two integers back to the client), they can share the same user-developed XDR routine.

RPC and XDR, based on RFC 1057 and RFC 1014, respectively, have been placed in the public domain; they serve as a standard for network application development.

Identifying remote procedures
1.2

An RPC message has three unsigned fields: the remote program number, the remote program version number, and the remote procedure number. These fields uniquely identify the procedure to be called.

Remote program number
1.2.1

The user's remote program number is a unique number in the range 0x20000000 to 0x3fffffff. Numbers outside of this range are reserved for other uses (see Table 1 for assigned ranges).

Table 1. Remote program number assignments

Program number	Assignment
0x0 – 0x1fffffff	Sun
0x20000000 – 0x3fffffff	User
0x40000000 – 0x5fffffff	Transient
0x60000000 – 0x7fffffff	Reserved
0x80000000 – 0x9fffffff	Reserved
0xa0000000 – 0xbfffffff	Reserved
0xc0000000 – 0xdfffffff	Reserved
0xe0000000 – 0xffffffff	Reserved

Sun Microsystems administers the first group of numbers, which should be identical for all RPC users. If a user develops an application that is of general interest, that application can be given an assigned number in the first range. The second group

of numbers is reserved for specific customer applications and will not, in general, be the same across machines. This range is intended primarily for debugging new programs. The third group is reserved for programs that generate program numbers dynamically. The final groups are reserved for future use and should not be used by any user-developed programs.

To register a protocol specification, send a request by electronic mail to `rpc@sun.com`, or write to the following:

RPC Administrator
Sun Microsystems
2550 Garcia Avenue
Mountain View, CA 94043

Please include a compilable `rpcgen.`*x* file that describes your protocol. In return, you will be given a unique program number.

You can find the RPC program numbers and protocol specifications of standard RPC services in the include files in the `/usr/include/rpcsvc` directory. These services, however, constitute only a small subset of those that have already been registered. Table 2 contains the most recent list of registered programs, as of the time of this printing. An asterisk denotes programs that are provided in the UNICOS 9.0 software release.

Table 2. Registered program list

RPC number	Program	Description
100000*	PMAPPROG	Portmapper
100001*	RSTATPROG	Remote statistics
100002*	RUSERSPROG	Remote users
100003*	NFSPROG	NFS daemon
100004*	YPPROG	Network information service (NIS)
100005*	MOUNTPROG	`mount` daemon
100006	DBXPROG	Remote `dbx`
100007*	YPBINDPROG	NIS binder
100008*	WALLPROG	Shutdown message
100009*	YPPASSWDPROG	NIS password server
100010	ETHERSTATPROG	Ethernet statistics server

Table 2. Registered program list (continued)

RPC number	Program	Description
100011	RQUOTAPROG	Disk quota server
100012*	SPRAYPROG	Spray packets server
100013	IBM3270PROG	3270 mapper
100014	IBMRJEPROG	RJE mapper
100015	SELNSVCPROG	Selection service
100016	RDATABASEPROG	Remote database access
100017	REXECPROG	Remote execution server
100018	ALICEPROG	Alice office automation
100019	SCHEDPROG	Scheduling service
100020	LOCKPROG	Local lock manager
100021	NETLOCKPROG	Network lock manager
100022	X25PROG	X.25 inr protocol
100023	STATMON1PROG	Status monitor 1
100024	STATMON2PROG	Status monitor 2
100025	SELNLIBPROG	Selection library
100026	BOOTPARAMPROG	Boot parameters service
100027	MAZEPROG	Mazewars game
100028	YPUPDATEPROG	NIS update server
100029*	KEYSERVEPROG	Key server for secure RPC
100030	SECURECMDPROG	Secure login
100031	NETFWDIPROG	NFS net forwarder initializing
100032	NETFWDTPROG	NFS net forwarder transmission
100033	SUNLINKMAP_PROG	SunLink MAP
100034	NETMONPROG	Network monitor
100035	DBASEPROG	Lightweight database
100036	PWDAUTHPROG	Password authorization
100037	TFSPROG	Translucent file service
100038	NSEPROG	NSE server
100039	NSE_ACTIVATE_PRG	NSE activate daemon

Table 2. Registered program list (continued)

RPC number	Program	Description
150001*	PCNFSDPROG	PC password authentication
200000	PYRAMIDLOCKINGPROG	Pyramid-locking
200001	PYRAMIDSYS5	Pyramid-sys5
200002	CADDS_IMAGE	CV cadds_image
300001	ADT_RFLOCKPROG	ADT file locking

Remote program version number
1.2.2

The version number is the release number for the RPC procedure. By convention, the first version number of any program *PROG* is *PROG*VERS_ORIG, and the most recent version is *PROG*VERS. In the following example, *PROG*=RUSERS. Suppose a new version of the user program returns an unsigned short rather than a long value. If this version is named RUSERSVERS_SHORT, a server that wants to support the first version and this version would register twice, as follows:

```
if (!svc_register(transp, RUSERSPROG, RUSERSVERS_ORIG,
 nuser, IPPROTO_TCP)) {
    fprintf(stderr, “can’t register RUSER service\n”);
exit(1);
}
if (!svc_register(transp, RUSERSPROG, RUSERSVERS_SHORT,
 nuser, IPPROTO_TCP)) {
    fprintf(stderr, “can’t register RUSER service\n”);
    exit(1);
}
```

The same C procedure can handle both versions, as follows:

```
nuser(rqstp, tranp)
struct svc_req *rqstp;
SVCXPRT *transp;
{
        unsigned long nusers;
        unsigned short nusers2;

        switch(rqstp->rq_proc) {

        case NULLPROC:
                if (!svc_sendreply(transp, xdr_void, 0)) {
                        fprintf(stderr, "can't reply to RPC call\n");
                }
                return;

        case RUSERSPROC_NUM:
                /*
                 * Code here to compute the number of users
                 * and put it in the variable 'nusers'
                 */

                if (rqstp->rq_vers == RUSERSVERS_ORIG) {
                        if (!svc_sendreply(transp, xdr_u_long, &nusers) {
                                fprintf(stderr,"can't reply to RPC call\n");
                        }
                } else if (rqstp->rq_vers == RUSERSVERS_SHORT) {
                        nusers2 = (unsigned short) nusers;
                        if (!svc_sendreply(transp, xdr_u_short, &nusers2) {
                                fprintf(stderr,"can't reply to RPC call\n");
                       }
                } else {
                  /* send "bad version" error reply */
            svcerr_progvers(transp, RUSERSVERS_ORIG, RUSERSVERS_SHORT);
                }
                return;

                default:
            /* send "bad procedure" error reply */
                svcerr_noproc(transp);
                return;
        } /* end switch */
}
```

Remote procedure number
1.2.3

The procedure number is the number of the routine being referenced in the server; it identifies the procedure to be called. Servers can register many RPC routines, which would typically be numbered in order, 1, 2, 3, ... *n*. Procedure numbers are documented in the specific program's protocol specification. For example, a file service's protocol specification might state that its procedure number 5 is read, and its procedure number 12 is write.

Registering with the portmapper
1.2.4

RPC servers can register themselves with the portmapper. This capability is useful if, for some reason, it becomes necessary to restart the portmapper while the RPC servers continue. The `-r` option of the `portmap` command specifies restart for standard RPC servers. The `-f` option directs the portmapper to send a `SIGHUP` signal to each of the process/UID pairs found in the file. See `portmap`(8) for a list of the servers that can be restarted and for examples of the `-r` and `-f` parameters.

Transports and semantics
1.3

The RPC protocol deals only with the specification and interpretation of messages; it is independent of transport protocols. Because of transport independence, the RPC protocol does not attach specific semantics to the remote procedures or their execution. Some semantics can be inferred from (but should be explicitly specified by) the underlying transport protocol.

For example, when UDP/IP is used, RPC message passing is unreliable. Thus, if the client retransmits call messages after short time-outs, it can only infer from no reply message that the remote procedure was executed zero or more times (and from a reply message, one or more times). On the other hand, when TCP/IP is used, RPC message passing is reliable. No reply message means that the remote procedure was executed at most once; a reply message means that the remote procedure was executed exactly once.

Error messages
1.4

The reply message to a request message contains enough information to distinguish the following error conditions:

- The remote implementation of RPC is not compatible with protocol version 2. The lowest and highest supported RPC version numbers are returned.
- The remote program is unavailable on the remote system.
- The remote program does not support the requested version number. The lowest and highest supported remote program version numbers are returned.
- The requested procedure number does not exist (this is usually a client-side protocol or programming error).
- The parameters to the remote procedure are invalid from the server's perspective. (Again, this is usually caused by a difference in the protocol between client and server.)
- An authentication error occurred.