## Programs using RPC

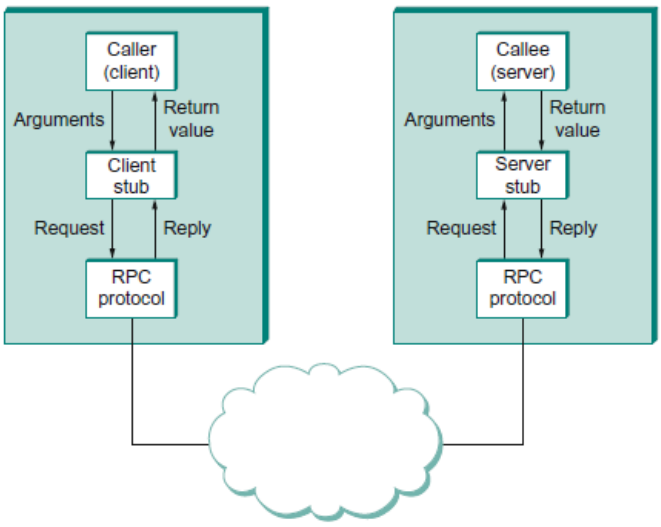
**Aim:** To implement remote procedure call

**Description:**

- RPC is actually more than just a protocol and a popular mechanism for structuring distributed systems
- It is based on the semantics of local procedure call, without knowing that procedure is local or remote
- In object oriented languages it is known as RMI
- RPC protocol is distributed with several kernels and runs on top of a network device driver and treated as a transport protocol
- RPC refers to a type of protocol rather than a specific standard like TCP, specific RPC protocol varies in the functions they perform.

**Functions to be performed by RPC are:**

- Provide a name space for uniquely identifying the procedure to be called
- Match each reply message to the corresponding request message




**Sequence of events in RPC**

1.The client calls a local procedure, called the client stub.

2.The client stub packages the arguments to the remote procedure (this may involve converting them to a standard format) and builds

one or more network messages. The packaging of arguments into a network message is called marshaling and sends to the kernel(by writing to a socket via system calls to the local kernel).

3.Network messages are transferred by the kernel to the remote system via some protocol (either connectionless or connection-oriented).

4.A server stub process, sometimes called a skeleton, on the server receives the messages. It unmarshals the arguments from the messages and, if needed, converts them from a standard format into a machine-specific format.

5.The server stub calls a local procedure call to the actual server function, passing it the arguments that it received from the client. The server function gets the illustion that it was called locally by the client since the server stub calls it locally with the same parameters that the client transmitted.

6.When the server is finished, it returns to the server stub with its return values.

7.The server stub converts the return values to a standard format (if necessary) and marshals them into one or more network messages to send to the client stub.

8.Messages get sent back across the network to the client stub.

9.The client stub reads the messages from the local kernel.

10.It then returns the results to the client function (possibly converting them first- unmarshals). The client feels that it just received a return value from the remote function, unaware that all the network messaging took place.

**Experimentation results and discussion:**

Running *rpcgen*

*date.x* generates date.h, date_clnt.c and date_svc.c. The header file is included with both client and server. The respective C source files are linked with client and server code.

```
/*
 * date.x  Specification of the remote date and time server
```

```
 */

/*
 * Define two procedures
  *          bin_date_1()  returns  the  binary  date  and  time  (no
arguments)
 *      str_date_1() takes a binary time and returns a string
 *
 */

program DATE_PROG {
version DATE_VERS {
long BIN_DATE(void) = 1;    /* procedure number = 1 */
string STR_DATE(long) = 2;  /* procedure number = 2 */
    } = 1;                           /* version number = 1 */
} = 0x31234567;                       /* program number = 0x31234567
*/
```

Notes:

- Start numbering procedures at 1 (procedure 0 is always the ``null procedure''.
- Program number is defined by the user. Use range 0x20000000 to 0x3fffffff.
- Provide a prototype for each function. Sun RPC allows only a single parameter and a single result. Must use a structure for more parameters or return values (see XDRC++ example).
- use clnt_create() to get handle to remote procedure.
- do not have to use rpcgen. Can handcraft own routines.

```
/*
 * rdate.c  client program for remote date program
 */

#include <stdio.h>

#include <rpc/rpc.h>    /* standard RPC include file */
#include "date.h"       /* this file is generated by rpcgen */

main(int argc, char *argv[])
{
    CLIENT *cl;         /* RPC handle */
char *server;
long *lresult;      /* return value from bin_date_1() */
char **sresult;     /* return value from str_date_1() */

if (argc != 2) {
fprintf(stderr, "usage: %s hostname\n", argv[0]);
exit(1);
    }

server = argv[1];
```

```
    /*
     * Create client handle
     */

if ((cl = clnt_create(server, DATE_PROG, DATE_VERS, "udp")) ==
NULL) {
        /*
         * can't establish connection with server
         */
         clnt_pcreateerror(server);
exit(2);
    }

    /*
     * First call the remote procedure "bin_date".
     */

if ( (lresult = bin_date_1(NULL, cl)) == NULL) {
        clnt_perror(cl, server);
exit(3);
    }
printf("time on host %s = %ld\n",server, *lresult);

    /*
     * Now call the remote procedure str_date
     */

if ( (sresult = str_date_1(lresult, cl)) == NULL) {
        clnt_perror(cl, server);
exit(4);
    }
printf("time on host %s = %s", server, *sresult);

    clnt_destroy(cl);         /* done with the handle */
exit(0);
}
/*
 * dateproc.c   remote procedures; called by server stub
 */

#include <rpc/rpc.h>        /* standard RPC include file */
#include "date.h"           /* this file is generated by rpcgen */

/*
 * Return the binary date and time
 */

long *bin_date_1()
 {
static long timeval;    /* must be static */
```

```
timeval = time((long *) 0);
return(&timeval);
 }

/*
 * Convert a binary time and return a human readable string
 */

char **str_date_1(long *bintime)
{
static char *ptr;       /* must be static */
ptr = ctime(bintime);   /* convert to local time */
return(&ptr);
}
```

**Execution Procedure:**

Eg: for day time service using RPC

$rpcgen date.x

COMPILING

$gcc -o server date_proc.c date_svc.c

$gcc -o client rdate.c date_clnt.c

EXECUTION

$./server

$./client 127.0.0.1

**TASKS to be performed**

1. **Execute the above program and write the conclusions**

2. **Write RPC for finding square of a number**

**References:**

1. https://web.cs.wpi.edu/~rek/DCS/D04/SunRPC.html