

Remote Procedure Call (RPC)

Antonio Lioy
< lioy@polito.it >

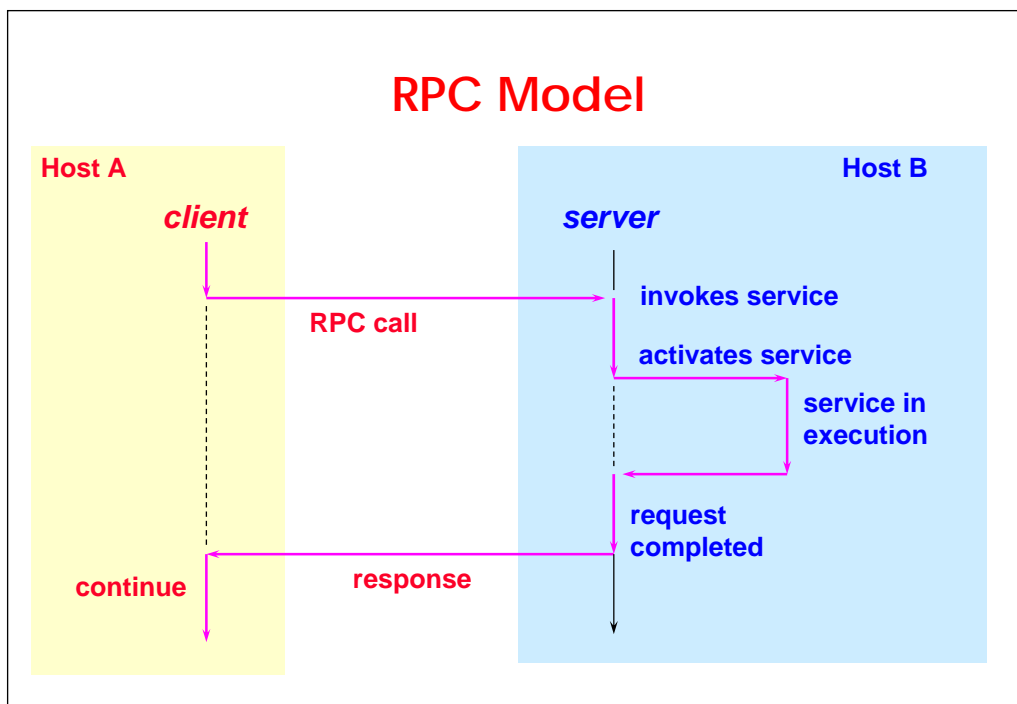
Politecnico di Torino
Dip. Automatica e Informatica

Remote Procedure Call (RPC)

- **SUN's proposal to facilitate the network programming for application-level programmers**
- **communication mechanism at high level**
- **permit to develop applications throughout special procedure calls**
- **implement a client-server model intended for network applications**
- **versions:**
 - **SUN-RPC (ONC - Open Network Computing)**
 - **(DCE-RPC)**
 - **MS-RPC**

RPC Model

- similar to the model for local procedures (LPC)
- differences:
 - the control evolves through two different processes (server and client)
 - the client makes the call and waits for a response message
 - the server receives the request, processes it and sends a response



Program number and procedure number

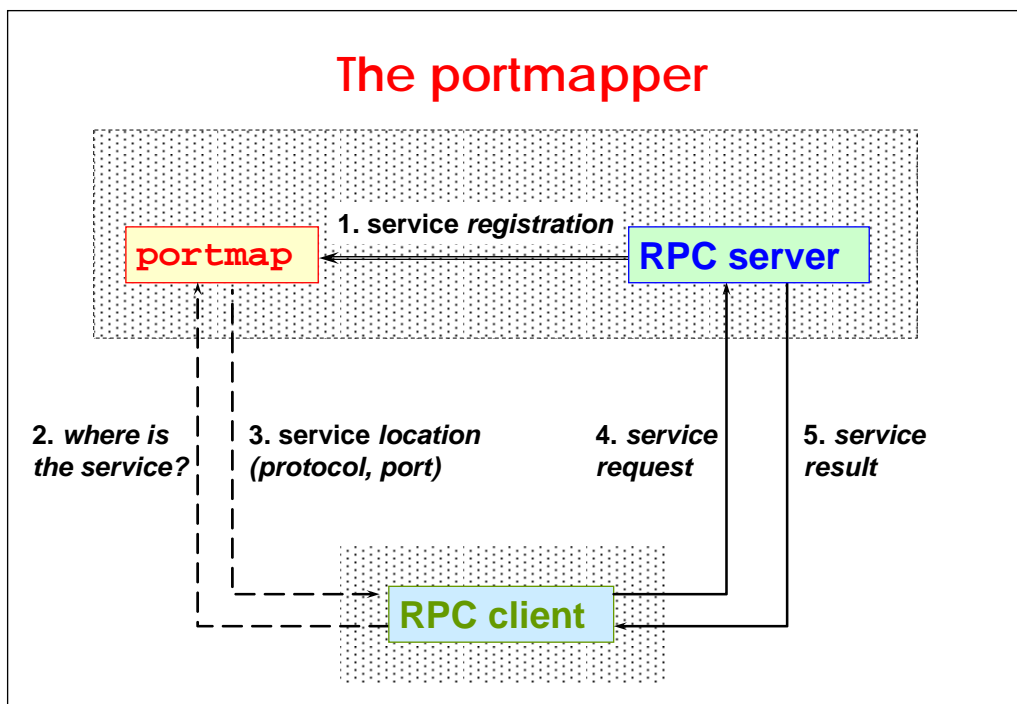
- each RPC procedure is characterized by two parameters:
 - program number
 - procedure number
- each program number characterizes a group of similar procedures, each of which with a different procedure number

Versions

- each RPC program has a version number associated with it
- when a remote service is modified, the version number is changed instead of assigning a new program number
- for programming it is enough to check the numbers on the manual

The portmapper

- in UNIX the daemon *portmap* (ports TCP/111 and UDP/111) maintains:
 - the list of registered RPC programs
 - the port and the protocol on which the RPC program is listening
- the command *rpcinfo* provides information about registered programs:
 - `rpcinfo -p [hostname]`
 - `rpcinfo -d program versnum`



Independence from transport level

- the RPC protocol is *independent from the transport protocol*
- RPC does not care about *how* a message is transmitted
- the protocol manages only the specification and the interpretation of messages

Independence from transport level

Attention!

- the RPC calls do not implement any type of control
- the application must know the type of the lower transport protocol
- the type of protocol determines *the semantics* of the remote procedure calls

RPC Semantics (I)

If the protocol is not reliable (e.g. UDP):

- the **receiving** of a reply message implies the execution of the procedure **once** or more times
- the **retransmission** of an RPC message implies only that the procedure was executed **zero** or more times

RPC Semantics (II)

If the protocol is reliable (e.g. TCP):

- the **receiving** of a reply message implies the execution of the procedure exactly one time
- the **non receiving** of a reply message **DOES NOT** imply the procedure was not executed

Consequently also in case of TCP are necessary time-out and reconnections

UDP vs. TCP for RPC

UDP is convenient to be used if:

- multiple executions of the procedure are not damaging
- arguments and result have dimension less than the UDP packet (8KB)
- the server must manage many clients (UDP does not maintain information on clients)

UDP vs. TCP for RPC

TCP is convenient to be used if:

- the application requires a reliable connection
- the procedures **CANNOT** be executed several times
- the dimensions of the arguments or of the result exceed 8KB

Data representation

Problem: lack of portability due to different data encoding on different platforms

Solution: representation of data independently of the hardware and the operating system

Representation of data

Sources of incompatibility among representations in different platforms:

- byte order in integers (Little Endian vs. Big Endian)
- different floating-point formats (IEEE/non-IEEE)
- structure alignment on boundaries of word

eXternal Data Representation (XDR)

- standard for the description and encoding of data independently from the machine
- RPC converts the data in XDR before sending them on the network: *serializing (o marshalling)*
- the inverse process is called *deserializing (o unmarshalling)*

RPC – high level

- transparent to the O.S and to the network
- RPC = calls to C functions

```
rnusers()  number of remote users
rusers()   information on remote users
havedisk() the remote machine has the disk ?
rstat()    data performances remote kernel
rwall()    write to remote machines
yppasswd() update database password NIS
```

```
#include <stdio.h>

int main (int argc, char *argv[])
{
    int num;

    if (argc != 2) {
        fprintf(stderr,
            "usage: %s hostname\n", argv[0]);
        exit (1);
    }
    if ((num=rusers(argv[1])) < 0) {
        fprintf(stderr,
            "error: rusers(%s) failed\n",argv[1]);
        exit (1);
    }
    printf("%d users on host %s\n",num,argv[1]);
}
```

RPC – intermediate level

- freed from details on socket and O.S.
- RPC = indirect calls through intermediate procedures
 - `callrpc` (`server, prognum, versnum, procnum, infilter, invar, outfilter, outvar`);
 - `registerrpc` (`prognum, versnum, procnum, procname, infilter, outfilter`);
 - `svc_run` (`void`)

```
/*
file: rusersd.c
scopo: server (fictious) for rusers
*/

#include <stdio.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>

unsigned long *users (void)
{
    static unsigned long users = 0;
    users++;
    return &users;
}
```

```
int main (int argc, char *argv[])
{
    if (registerrpc(
        RUSERSPROG, RUSERSVERS, RUSERSPROC_NUM,
        users, xdr_void, xdr_u_long) < 0)
    {
        fprintf (stderr,
            "error: registerrpc() failed\n");
        exit (1);
    }
    svc_run();
    fprintf (stderr,
        "error: svc_run() returned!\n");
    exit (1);
}
```

```
/*
file: rnusers.c
scopo: client for rnusers (RPC medium level)
*/

#include <stdio.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>

int main (int argc, char *argv[])
{
    unsigned long num;
    int status;
    if (argc != 2) {
        fprintf (stderr,
            "usage: %s hostname\n", argv[0]);
        exit (1);
    }

```

```

    if ((status=callrpc(argv[1],
        RUSERSPROG,RUSERSVERS,RUSERSPROC_NUM,
        xdr_void,NULL,xdr_u_long,&num)) != 0)
    {
        fprintf (stderr,
            "error: callrpc(%s) failed\n",argv[1]);
        clnt_perrno(status);
        exit (1);
    }
    printf ("%lu users on host %s\n",
        num, argv[1]);
}

```

RPC – low level

- **control of details:**
 - transport protocol (TCP vs. UDP)
 - authentication
 - memory management
- **RPC functions:**
 - similar to those of intermediate level + control of authentication and creation of client and server

Standard for SUN-RPC

- RFC-1831 “RPC: Remote Procedure Call protocol specification - version 2”
- RFC-1832 “XDR: External Data Representation standard”
- RFC-1833 “Binding protocols for ONC RPC version 2”
- RFC-2695 “Authentication mechanisms for ONC RPC”

- nota: ONC = Open Network Computing