RPC Paradigm

Lenuta Alboaie (adria@info.uaic.ro)
Andrei Panu (andrei.panu@info.uaic.ro)

Content

- **Remote Procedure Call (RPC)**
  - Preliminaries
  - Characteristics
  - XDR (External Data Representation)
  - Functioning
  - Implementations
  - Uses

Preliminaries

- **Designing distributed applications**
  - Protocol oriented – ***sockets***
    - The protocol is designed, then the applications that implement it
  - Oriented on functionality – **RPC**
    - The applications are developed, then they are divided into components and a communication protocol between the components is added

RPC|Characteristics

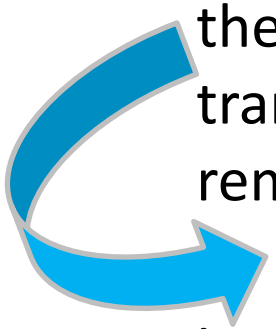
- **Idea:** Instead of accessing remote services by sending and receiving messages, the client calls a procedure that will be executed on another machine
- **Effect**: RPC "hides" the existence of the network for the program
  - The *message-passing* mechanism used in network communication is hidden from the programmer
  - The programmer must no longer open a connection, read and write data, close the connection, etc.
- It is a simpler programming tool than the BSD *socket* interface

RPC|Characteristics

- A RPC application consist of a client and a server, the server being located on the machine on which the procedure is executed
- When making a remote call, the procedure's parameters are transferred over the network to the application that executes the procedure; after the execution finishes, the results are transferred over the network to the client application
- The client and the server -> processes on different machines

RPC|Characteristics

- **RPC performs client/server communication via TCP/IP *sockets* (usually UDP), via two *stub* interfaces**
  - OBS.: The RPC package (*client stub* and *server stub | skeleton*) hides all details related to network programming
- **RPC** involves the following **steps:**
  1. The client invokes a *remote* procedure
     - A local procedure is called, named *client stub*, that packs the arguments in a message and sends it to the transport level, from where it is transferred to the remote *server* machine

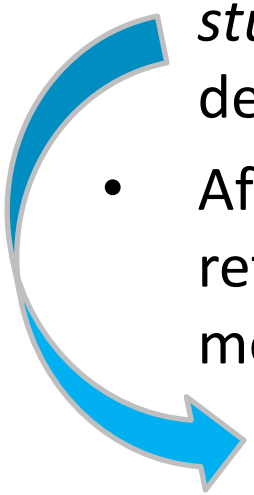


Marshalling (serialization) = a mechanism which includes encoding arguments in a standard format and wrapping them in a message

RPC|Characteristics

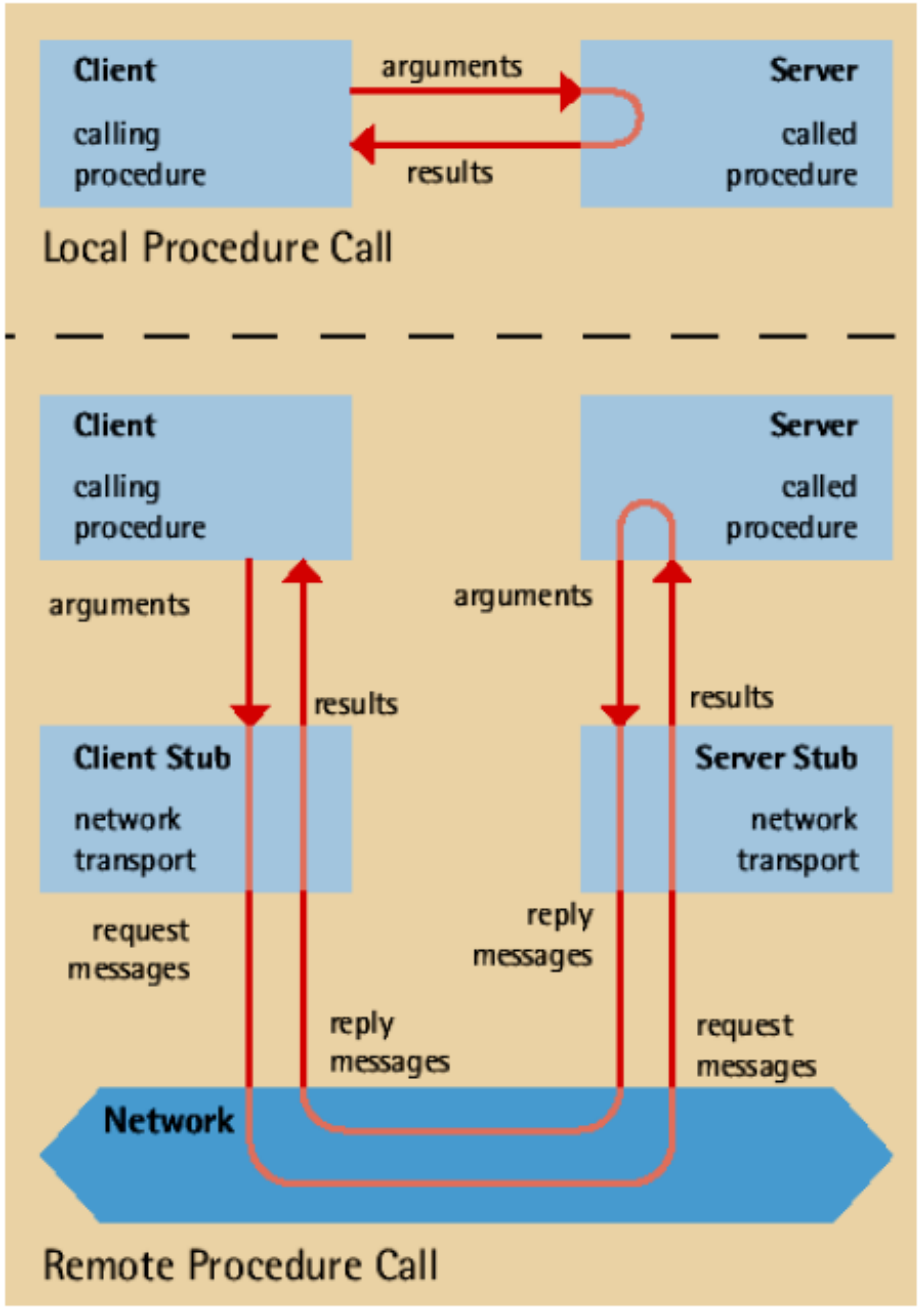
- **RPC** involves the following **steps**:

2. The server:
    - The transport level sends the message to the *server stub*, which unpacks the parameters and calls the desired function
    - After the function returns, the *server stub* takes the returned values, wraps them (marshalling) into a message and sends them to the *client stub*



un-marshalling (deserialization) = decoding

3. The *Client stub* retrieves the received values and returns them to the client application



RPC|Characteristics

- The stub interfaces implement the RPC protocol
- Differences from local calls:
  - Performance can be affected by the transmission time
  - Error handling is more complex
  - The server location must be known (remote procedure identification and access)
  - User authentication may be required

[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

RPC|Characteristics

- Stub procedures can be automatically generated; afterwards, they are “binded” to client and server programs
- The server stub listens to a port and invokes the routines
- The client and the server will communicate through messages, using a network independent and OS independent representation:

External Data Representation (XDR)

RPC|Characteristics

- **External Data Representation (XDR)**

  XDR defines various data types and their transmission mode in RPC Messages (RFC 1014)

  – Usual types:
    - from C: int, unsigned int, float, double, void,…
    - additional: string, fixed-length array, variable-length array, …

  – Conversion functions (rpc/xdr.h)
    - xdrmem_create() – associates a RPC data stream to a memory zone
    - xdr_*typename*() – converts data

RPC|Characteristics

- **External Data Representation (XDR)**

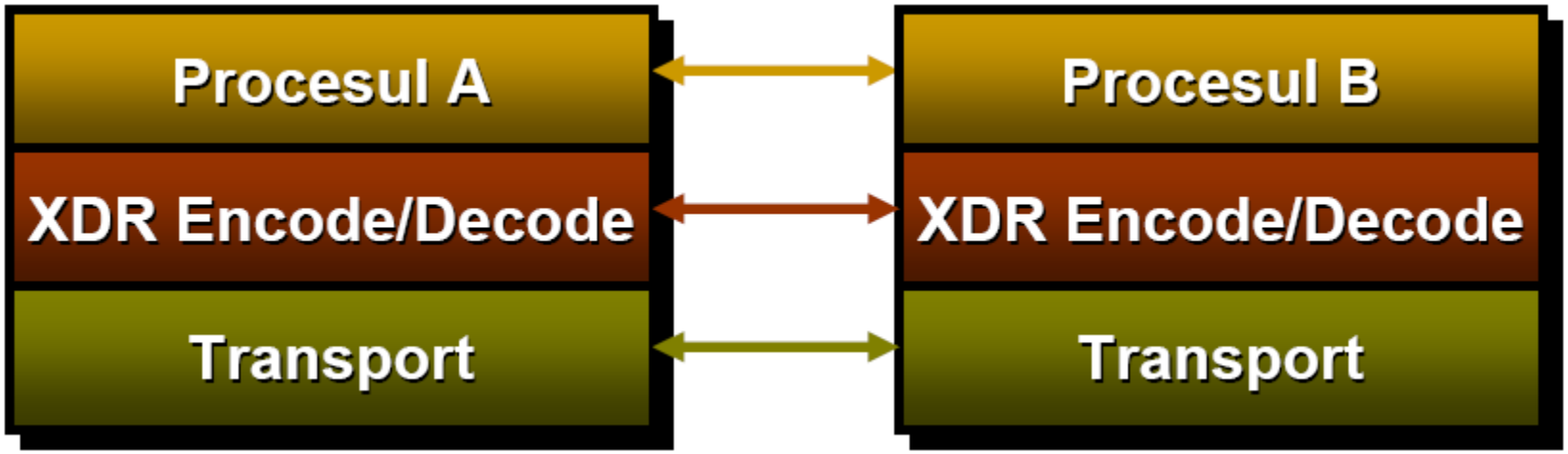
Example

```
#include <rpc/xdr.h>
#define BUFSIZE 400 /* lungimea zonei de memorie */
/* conversia unui intreg in format XDR */
...
XDR *xdrm; /* zona de memorie XDR */
char buf[BUFSIZE];
int intreg;
...
xdrmem_create (xdrm, buf, BUFSIZE, XDR_ENCODE);
...
intreg = 33;
xdr_int (xdrm, &intreg);
...
```

Inlocuit la celalalt capat al comunicatiei cu XDR_DECODE

RPC|Characteristics

- **External Data Representation (XDR)**
  - Can be seen as an additional level between transport level and application level
  - Ensures symmetric conversion of client and server data

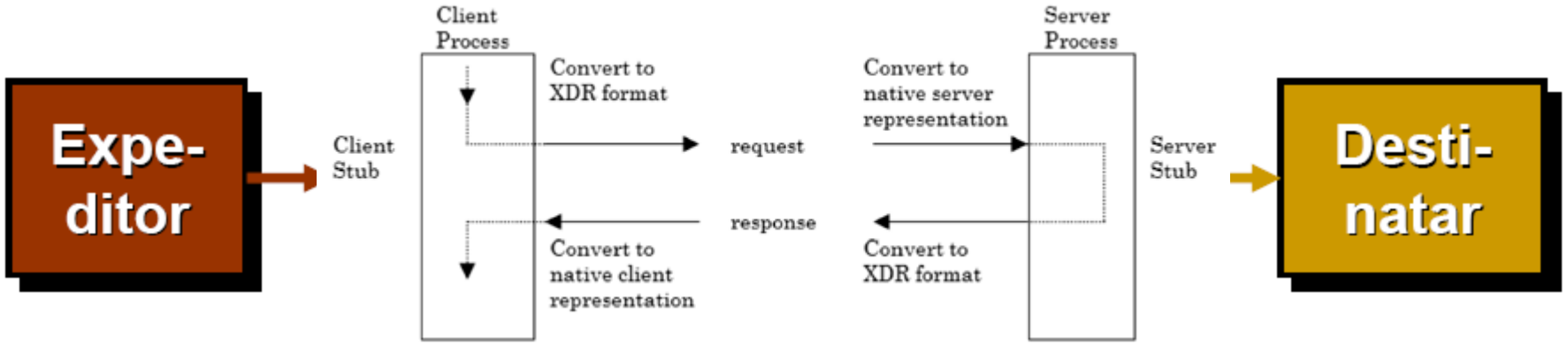


[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

RPC|Characteristics

External Data Representation (XDR)

- Coding/decoding activity



- Currently, it can be replaced by XML-RPC, SOAP, or JSON-RPC representations (in the context of Web services)

see Web Technologies course

RPC|Functioning

Context:

- A network service is identified by the port where a *daemon* runs, waiting for requests
- RPC programs use ephemeral ports

How does the client know where to send the request?

Portmapper = network service responsible for associating services to different ports

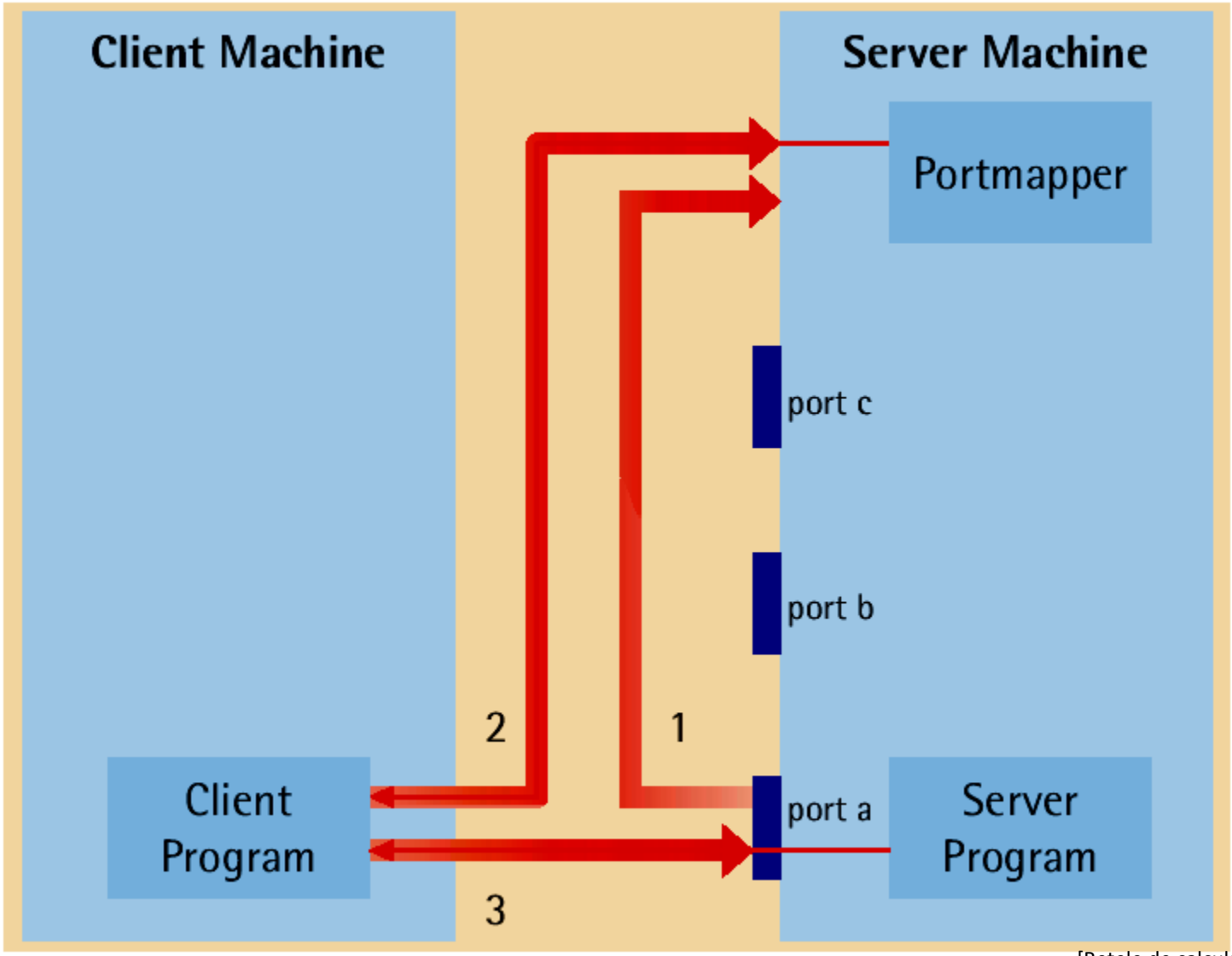
=> The port numbers for a particular service are not fixed

- It is available at port 111 (well-known port)

```
[adria@thor ~] $ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  56660  status
    100024    1   tcp  48918  status
```

RPC|Functioning

General mechanism



[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

RPC|Functioning

General mechanism:

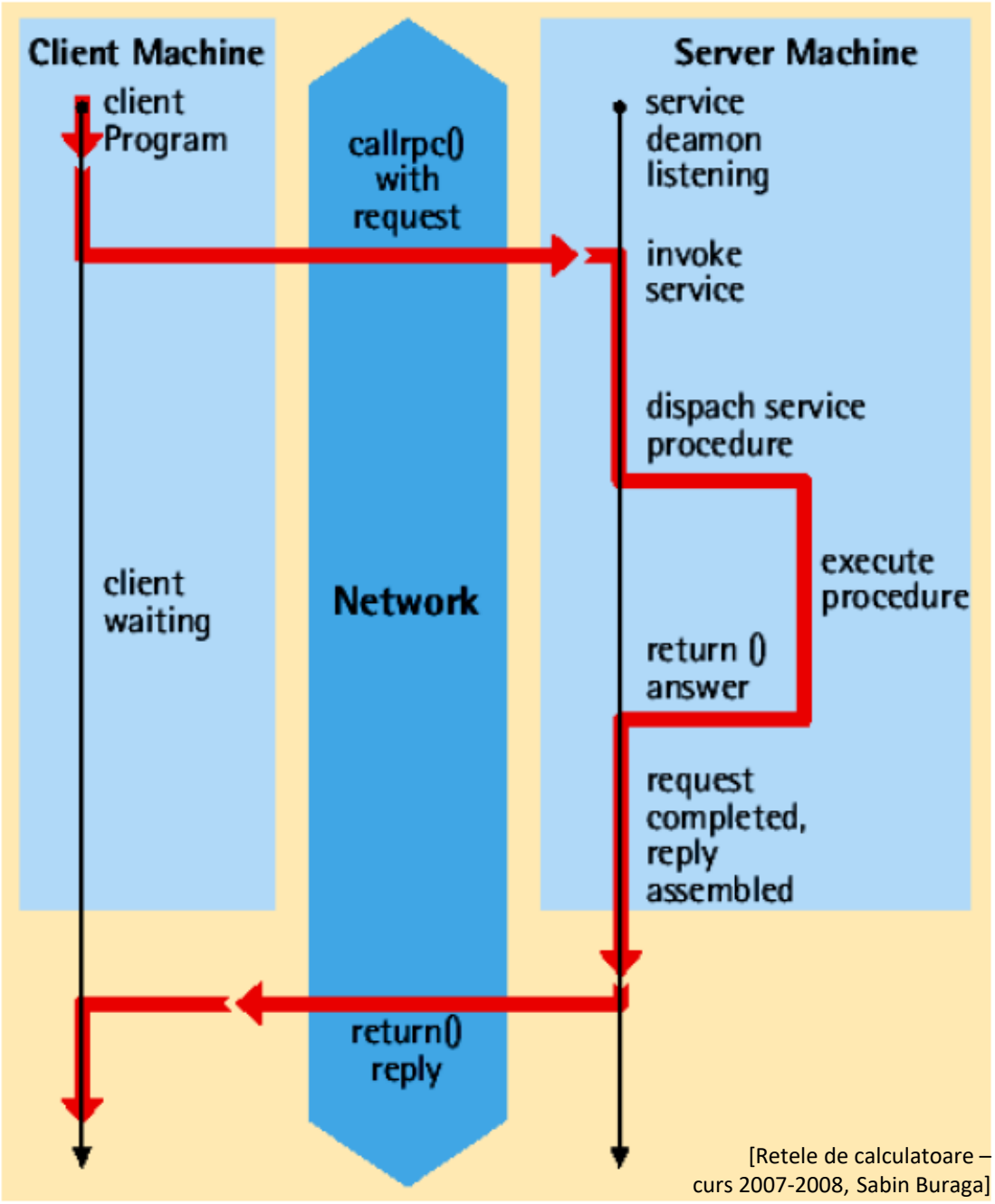
Step 1: Determine the address at which the server will provide the service

- Upon initialization, the server sets and registers via *portmapper* the port to which the service will be provided (port **a**)

Step 2: The client interrogates *portmapper* on the server machine to identify the port to which it must send the RPC request

Step 3: The client and the server can communicate to perform the remote procedure execution

- The requests and the responses are encoded/decoded through XDR



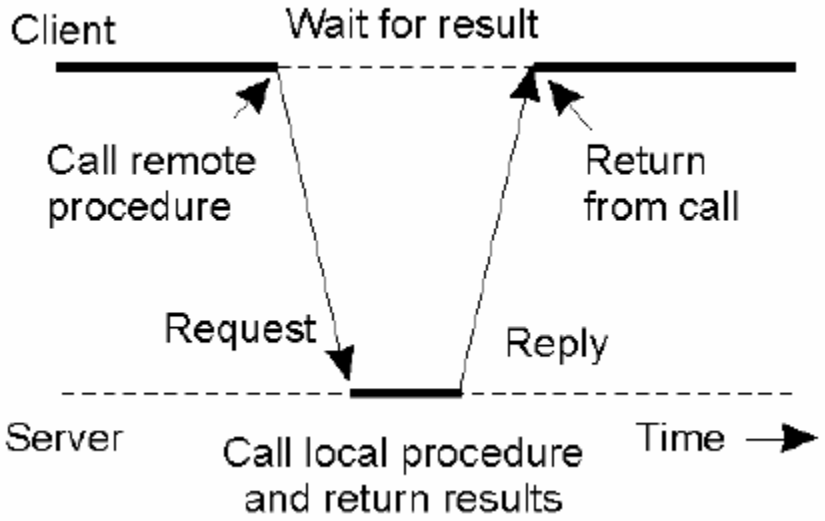
[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

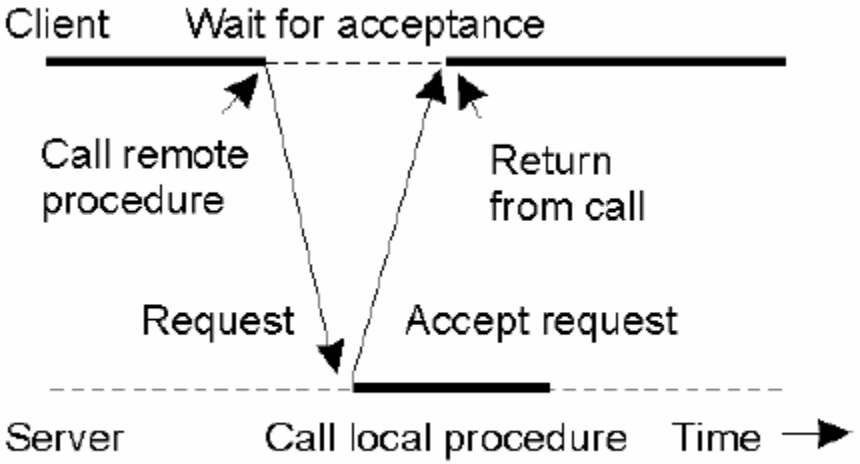
RPC|Functioning

- When a server provides multiple services, a dispatcher routine is usually used
- The dispatcher identifies specific requests and calls the appropriate procedure, after which the result is sent back to the client to continue its execution

RPC|Functioning

- RPC data transfers can be:
  - Synchronous
  - Asynchronous





[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

RPC|Implementation

- Open Network Computing RPC (ONC RPC) – the most widespread implementation in Unix environments (Sun Microsystems)
  - RFC 1057
  - The RPC interface is structured on 3 levels:
    - **Superior**: system, hardware, and network independent
      - Example: man **rcmd** -> *routines for returning a stream to a remote command ….*
    - **Intermediary**: calls the functions defined in the RPC library:
      - **registerrpc**() – regiters a procedure to be executed remotely
      - **callrpc(**) – calls a remote procedure
      - **svc_run(**) – runs a RPC service
    - **Inferior:** offer the possibility to control the RPC mechanisms (e.g., choosing the data transport mode, etc.)

RPC|Implementation

- Open Network Computing RPC (ONC RPC)
  - Remote procedures will be included in a **remote program** – software unit that will run on a remote machine
  - Each remote program corresponds to a server: it can contain remote procedures + global data; the procedures may share common data;
  - Each remote program is identified by a unique 32 bit identifier; according to the Sun RPC implementation, we have the following identifier values:
    - 0x00 00 00 00 – 0x1F FF FF FF – system's RPC applications
    - 0x20 00 00 00 – 0x3F FF FF FF – user programs
    - 0x40 00 00 00 – 0x5F FF FF FF – temporary identifiers
    - 0x60 00 00 00 – 0xFF FF FF FF – reserved values
  - Each procedure (within a program) is identified by an index (1..n)

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

  Examples:

  - 10000 *portmapper* meta-server
  - 10001 for rstatd, which provides information about the remote system; rstat() or perfmeter() can be used
  - 10002 for rusersd, which provides information about the users connected to the remote machine
  - 10003 nfs server, which provides access to the NFS (*Network File System*)

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

  Each remote program has a version number associated

  - Initially version 1
  - Next versions are uniquely identified by other version numbers

  It offers the possibility to change the implementation details or to expand the capabilities of the application, without assigning another program identifier

  A remote program is a 3-tuple, of the form:

  <program_id, version, procedure_index>

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

High-level programming:

fenrir.info.uaic.ro - PuTTY

```
    I A  prog.c (Modified) (c)  }        Row 18    Col
#include <rpc/rpc.h>
#include <string.h>
#include <rpcsvc/rusers.h>
#include <stdio.h>

int main(int argc, char * argv[])
{
  int nrUsers;
  if (argc!=2){/*eroare*/}
  if ((nrUsers=rusers(argv[1]))<0){/*eroare RPC */}
  printf("Sunt %d utilizatori pe masina %s\n",
                 nrUsers, argv[1]);
  exit(0);
}
```

Compiling: gcc prog.c – lrpcsvc –o prog

Execution: ./prog fenrir.infoiasi.ro

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

Intermediate level programming:

```
callrpc (char *host,    /* server name */
        u_long prognum,  /* server program number */
        u_long versnum,  /* version number */
         u_long procnum, /* procedure number */
         xdrproc_t inproc,  /* used for XDR encoding */,
        char *in,  /* procedure's arguments address */,
        xdrproc_t outproc,  /* used for decoding  */,
        char *out,  /* address for placing results */
            );
```

Called by the RPC client

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

Intermediate level programming:

registerrpc(

u_long **prognum** /* server program number */,
u_long **versnum** /* version number */,
u_long **procnum** /* procedure number */,
void *(***procname**)*() /* *remote* function name */,
xdrproc_t **inproc** /* used for param. decoding */,
xdrproc_t **outproc** /* used for results encoding */
);

Called by the RPC server

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

  Intermediate level programming:

  svc_run ()

  Called by the RPC server, represents the *dispatcher*

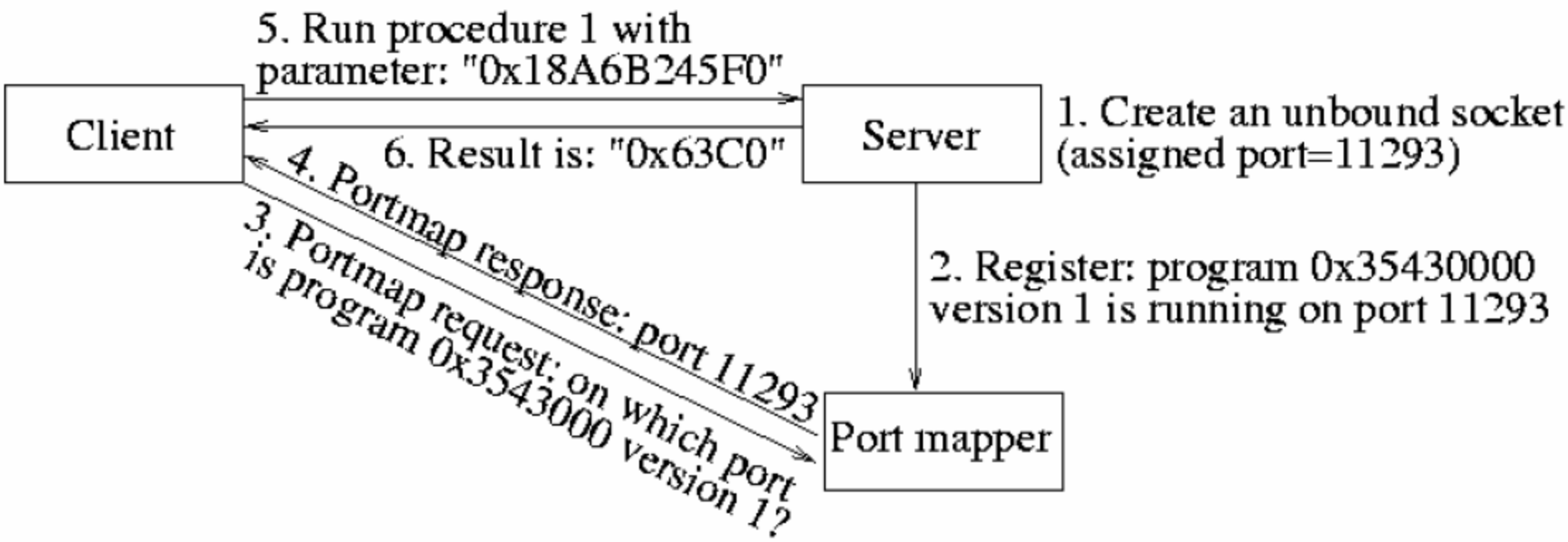
  - Waits for RPC requests, then calls the appropriate procedure using *svc_getreq*()

  OBS.: Intermediate level functions only use UDP

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

Lower level programming:



[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

  Developing RPC applications using **rpcgen**

  - Create a RPC specifications file (Q.x)
    - Declarations of constants used by the client and the server
    - Declarations of global data types
    - Declarations of remote programs, procedures, parameter types, result type, unique program identifier
  - The server.c program that contains procedures
  - The client.c program which invokes procedures

  For server: gcc server.c Q_svc.c Q_xdr.c –o server

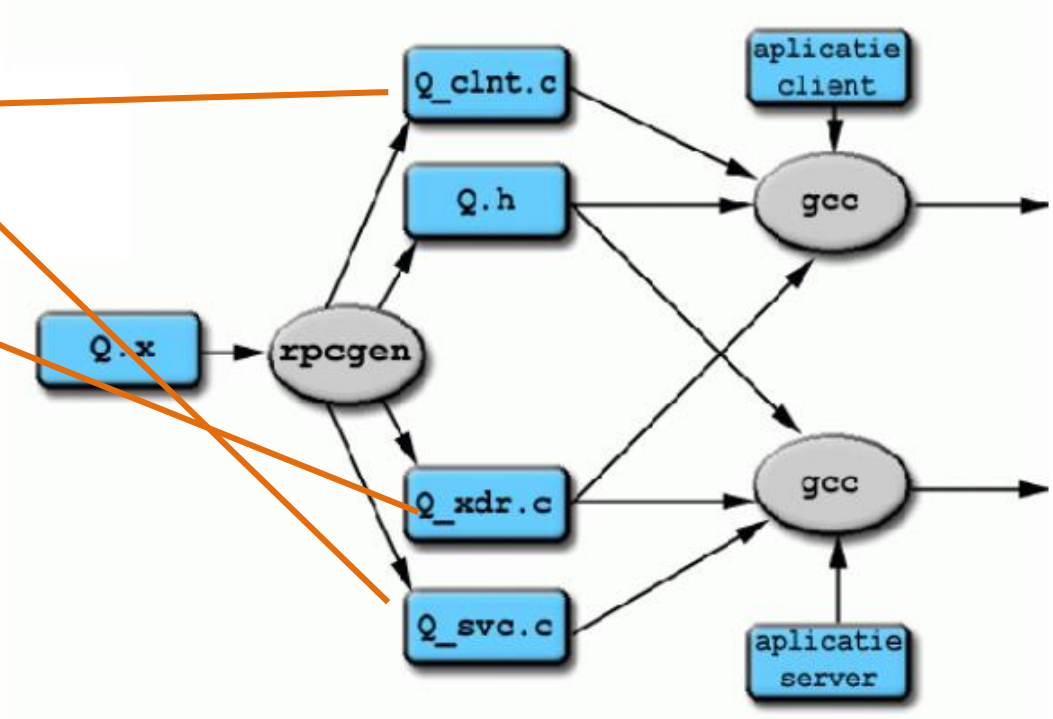
  For client: gcc client.c Q_clnt.c Q_xdr.c –o client

RPC|Implementation

- Open Network Computing RPC (ONC RPC)

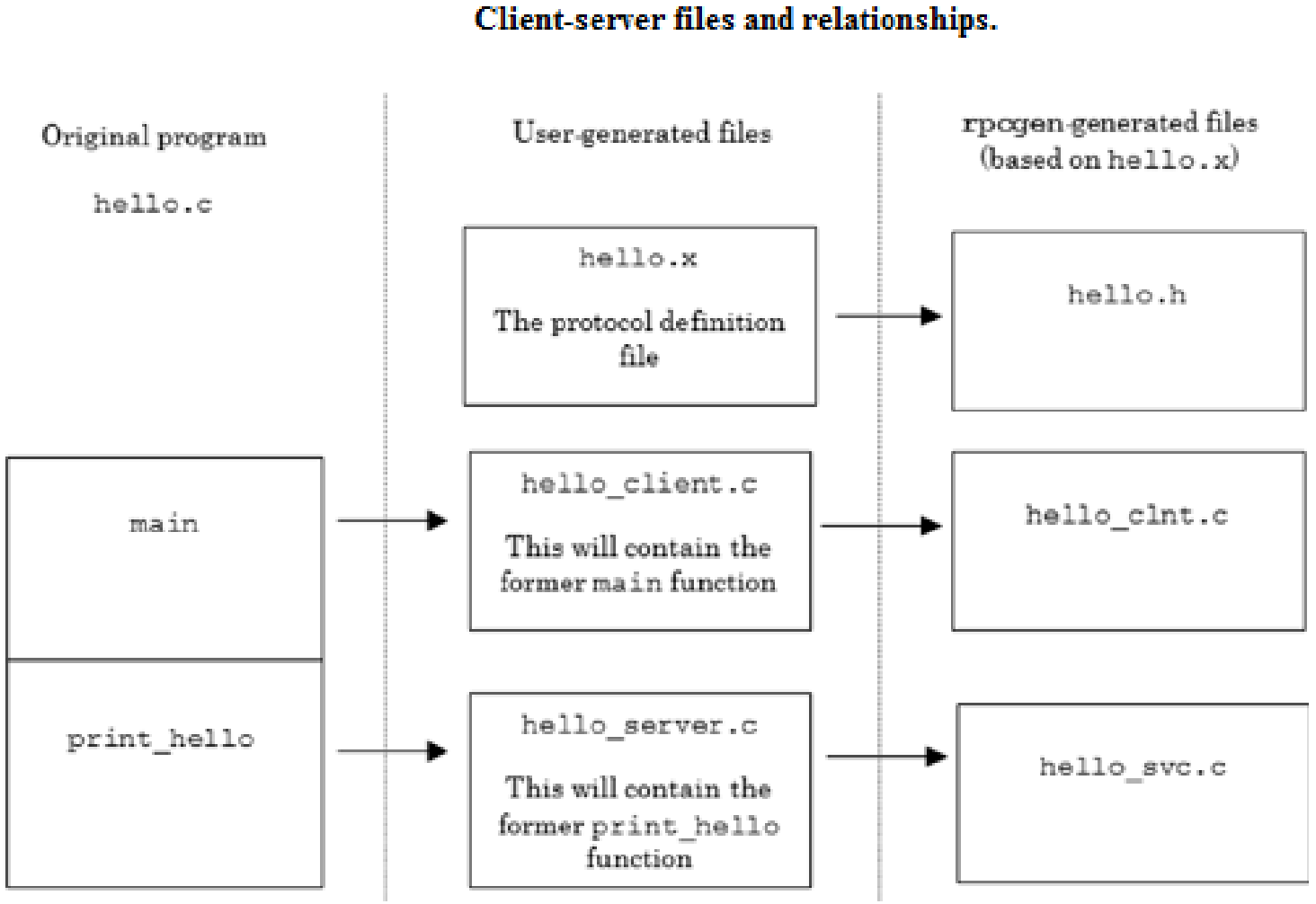
  In the implementation of a RPC application, the **rpcgen** tool is used

- Generates the *client stub* and the *server stub*
- Generates XDR encoding/decoding functions
- Generates the *dispatcher* routine



[Retele de calculatoare – curs 2007-2008, Sabin Buraga]

RPC|Implementation



[Interprocess Communications in Linux, J.S. Gray]

RPC|Implementation

thor.info.uaic.ro - PuTTY

I A hello.c (c) Row 20

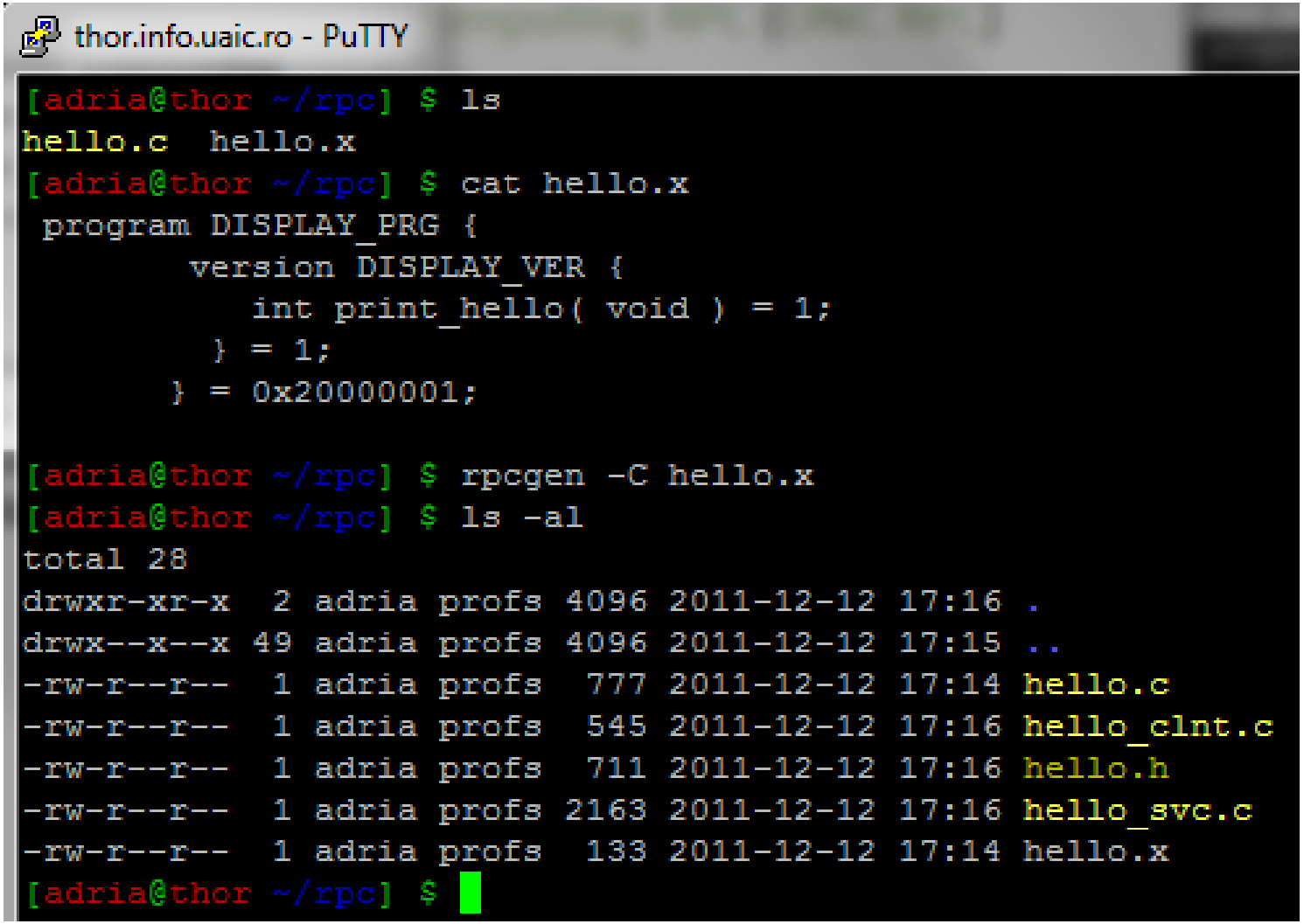
```c
/*
        A C program with a local function
*/
#include <stdio.h>
int print_hello( );

int main( )
{
   printf("main : Calling function.\n");
   if (print_hello())
     printf("main : Mission accomplished.\n");
   else
     printf("main : Unable to display message.");
    return 0;
}

int print_hello( )
{
      return printf("funct: Hello, world.\n");
}
```

[Interprocess Communications in Linux, J.S. Gray]

RPC|Implementation



[Interprocess Communications in Linux, J.S. Gray]

RPC|Implementation

```
I A  hello_client.c (c)          Row 1    Col 1   10:01  Ctrl-K H for help
/*
        The CLIENT program:  hello_client.c
        This will be the client code executed by the local client process.
 */
#include <stdio.h>
 #include "hello.h"             /* Generated by rpcgen from hello.x  */
 int
 main(int argc, char *argv[]) {
   CLIENT         *client;
  int            *return_value, filler;
   char           *server;
 /*
     We must specify a host on which to run.  We will get the host name
     from the command line as argument 1.
  */
   if (argc != 2) {
     fprintf(stderr, "Usage: %s host_name\n", *argv);
     exit(1);
   }
   server = argv[1];
 /*
     Generate the client handle to call the server
  */
    if ((client=clnt_create(server, DISPLAY_PRG, DISPLAY_VER, "tcp")) ==
     clnt_pcreateerror(server);
     exit(2);
   }
   printf("client : Calling function.\n");
 return_value = print_hello_1((void *) &filler, client);
   if (*return_value)
     printf("client : Mission accomplished.\n");
   else
     printf("client : Unable to display message.\n");
  return 0;
 }
```

[Interprocess Communications in Linux, J.S. Gray]

RPC|Implementation

```
I A  hello_server.c (c)          Row 1    Col 1   10:02  Ctrl-K H for help

/*
    The SERVER program: hello_server.c
    This will be the server code executed by the "remote" process
*/
#include <stdio.h>
#include "hello.h"          /* is generated by rpcgen from hello.x  */
int *
print_hello_1_svc(void * filler, struct svc_req * req) {
  static int  ok;
   ok = printf("server : Hello, world.\n");
  return (&ok);
}
```

[Interprocess Communications in Linux, J.S. Gray]

RPC|Implementation

```
[adria@thor ~/html/teach/courses/net/files/NetEx/S11/RPC/tempRPC] $ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  56604  status
    100024    1   tcp  34914  status
[adria@thor ~/html/teach/courses/net/files/NetEx/S11/RPC/tempRPC] $
```

```
/tempRPC] $ gcc hello_client.c hello_clnt.c -o client
/tempRPC] $ gcc hello_server.c hello_svc.c -o server
/tempRPC] $ ./server
```

```
[adria@thor ~/html/teach/courses/net/files/NetEx/S11/RPC/tempRPC] $ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100024    1   udp  56604  status
    100024    1   tcp  34914  status
 536870913    1   udp  37547
 536870913    1   tcp  43833
[adria@thor ~/html/teach/courses/net/files/NetEx/S11/RPC/tempRPC] $
```

```
[adria@thor ~/html/teach/courses/net/files/NetEx/S11/RPC/tempRPC] $ ./client 127.0.0.1
client : Calling function.
client : Mission accomplished.
```

```
[adria@thor ~/html/teach/courses/net/files/NetEx/S11/RPC/tempRPC] $ ./server
server : Hello, world.
```

[Interprocess Communications in Linux, J.S. Gray]

RPC|Implementation

- Other implementations:
  - DCE/RPC (Distributed Computing Environment/RPC)
    - Alternative for Sun ONC RPC
    - Used also by Windows servers
  - ORPC (Object RPC)
    - Remote request/response messages are encapsulated into objects
    - Direct descendants:
    - (D)COM (Distributed Component Object Model) & CORBA (Common Object Request Broker Architecture)
    - In Java: RMI (Remote Method Invocation)
    - .Net Remoting , WCF
  - SOAP (Simple Object Access Protocol)
    - XML as XDR, HTTP transfer protocol
    - The foundation for implementing a certain category of Web services

RPC|Uses

- **Remote file access - NFS (Network File System)**
  - Protocol designed to be independent of the machine, OS and protocol – implemented over RPC (…XDR convention)
  - Protocol that enables file sharing => NFS provides transparent access to files for the clients
    - OBS.: Different from FTP (see previous course)
  - The NFS directory hierarchy uses UNIX terminology (tree, directory, path, file, etc.)
  - NFS is a protocol => client - nfs , server –nfsd communicating through RPC
  - NFS model
    - Operations on a remote file: I/O operations, create / rename / delete, stat, listing entries
    - **mount** command – specifies the remote host, the file system that must be accessed, and where to mount it in the local file hierarchy
  - RFC 1094

RPC|Uses

- Remote file access - **NFS** (**Network File System**)
  - It is transparent to the user
  - The NFS client sends a RPC request to the RPC server, using TCP/IP
    - OBS.: NFS was used predominantly with UDP
  - The NFS server receives requests at port 2049 and sends them to local file access module

OBS.: For treating the clients faster, the NFS servers are generally *multi-threading* or for UNIX systems that are not *multi-threading*, multiple instances are created, which stay in the kernel (*nfsd*s)

RPC|Uses

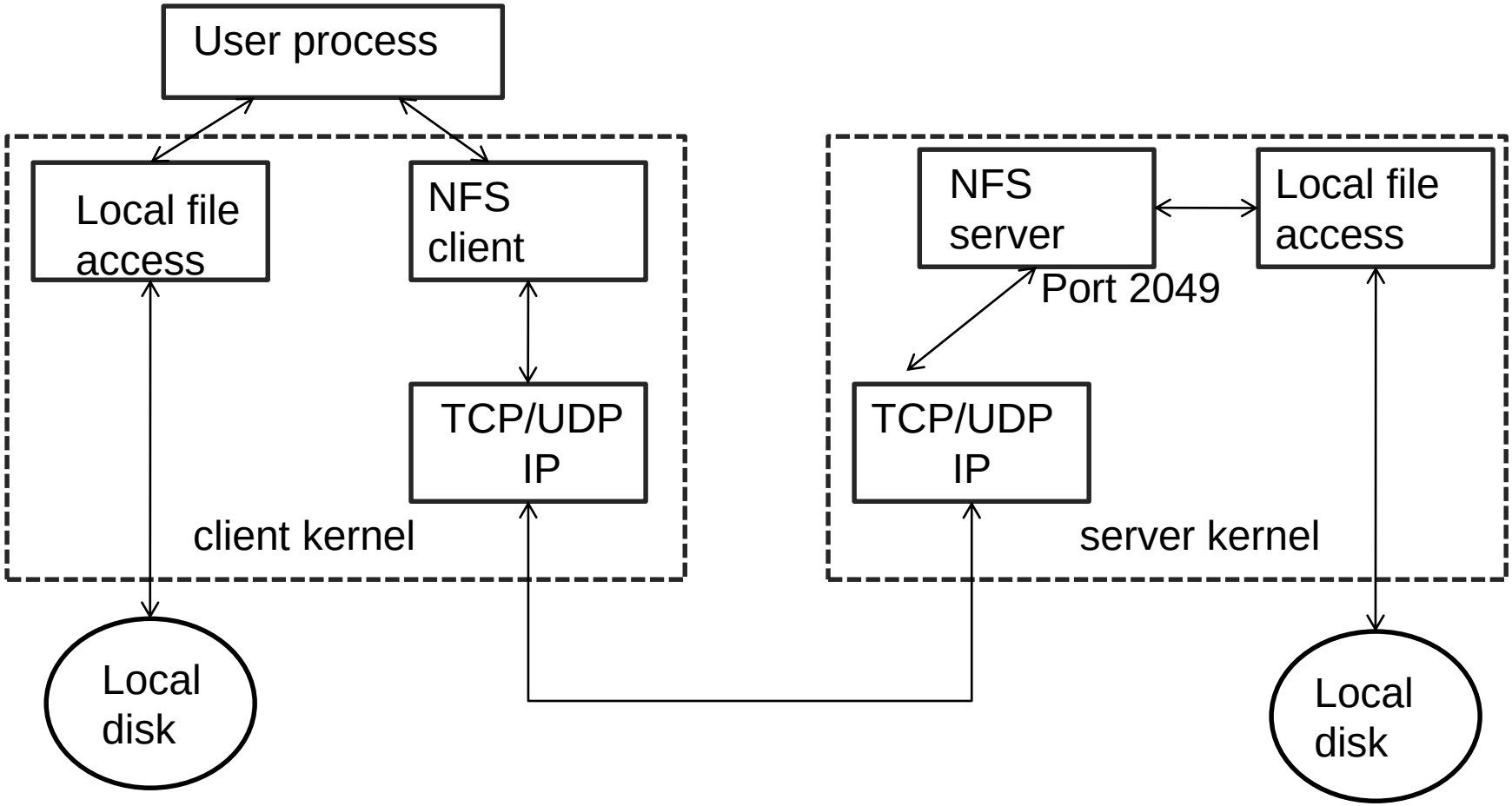


Figure: NFS architecture

RPC|Uses

- Remote file access - **NFS** (**Network File System**)
  - (0) *portmapper* is started at system *boot*
  - (1) *mountd* daemon is started on the server; creates TCP and UDP *endpoints*, assigns them ephemeral ports, and calls *portmapper* to register them
  - (2) *mount* command is executed and a request to *portmapper* is made, in order to obtain the *mount* server port
  - (3) *portmapper* returns the answer
  - (4) a RPC request is created for mounting a filesystem
  - (5) the server returns a *file handle* for the requested filesystem
  - (6) A local mount point is associated to this *file handle* on the client (*file handle* is stored in NFS client's code and any request for the respective filesystem will use this *file handle*)

RPC|Uses

- Remote file access - **NFS** (**Network File System**)
  - The mounting process (**mount** protocol)
    - In order for a client to be able to access files in a filesystem, it must use the **mount** protocol

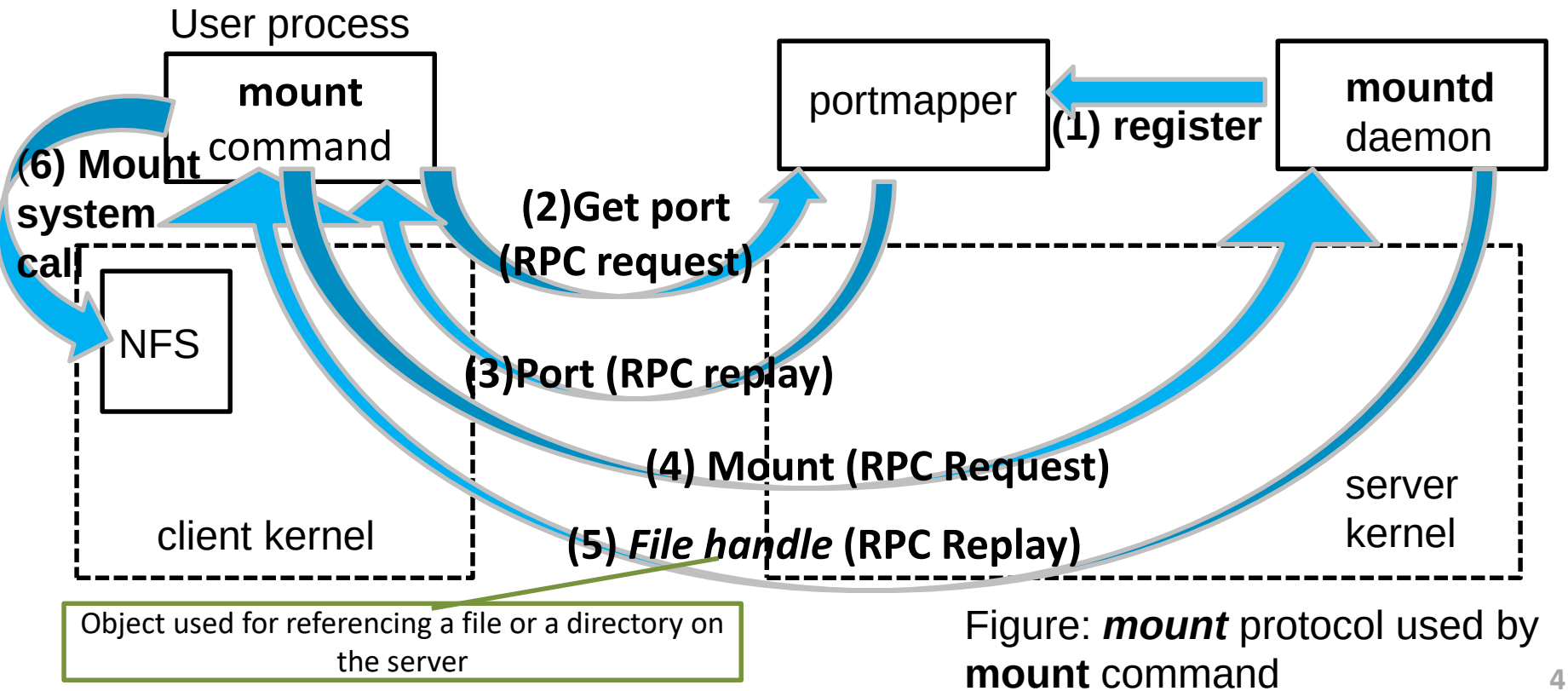


Figure: ***mount*** protocol used by **mount** command

Rezumat

- **Remote Procedure Call (RPC)**
  - Preliminaries
  - Characteristics
  - XDR (External Data Representation)
  - Functioning
  - Implementations
  - Uses

Questions?