

# Remote Procedure Call

Giovanni Agosta

Piattaforme Software per la Rete – Modulo 2

# Outline

- 1 The Remote Procedure Call model
- 2 Remote Procedure Call implementation
- 3 The rpcgen Compiler
  - Overview
  - Declarations and Definitions
  - Using the rpcgen Compiler

# Introduction

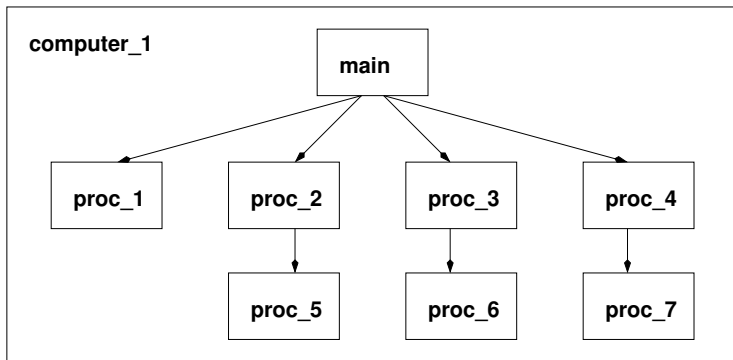
## Designing Distributed Programs

- Communication-Oriented Design
- Application-Oriented Design

## Remote Procedure Call model

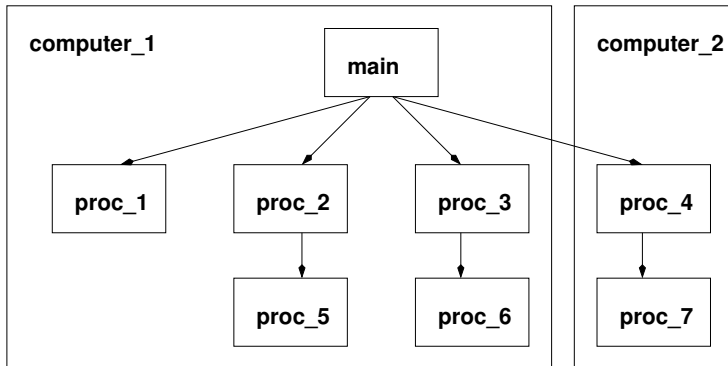
- Support the Client-Server model
- Allow Application-Oriented Design
- Divide the program at procedure boundaries into local and remote parts

# Remote Procedure Call model



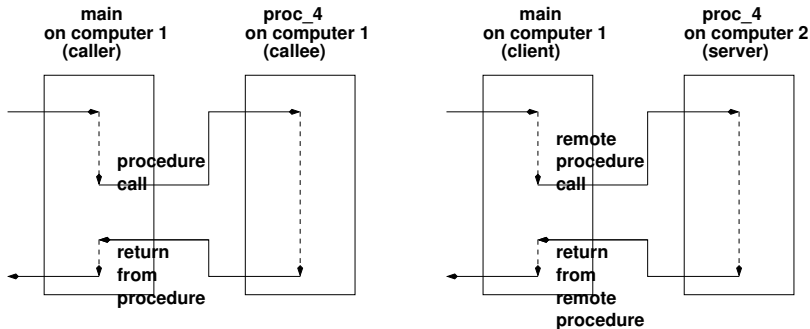
- A typical program, divided into a set of procedures (static view)

## Remote Procedure Call model



- The same program, partitioned between two machines using the RPC model
- A communication protocol is needed between `main` and `proc_4`

# Remote Procedure Call model



- However, remote calls are much slower
- Remote calls also do not happen in the same address space!
- And have no access to the environment (e.g., files)

## Remote Procedure Call Addressing

- Derived from Sun Open Network Computing (ONC)
- Requires a tuple of (*program*, *version*, *procedure*) to work
- Uses unique, registered identifiers (integers) to identify remote programs (i.e., servers)
- RPC identifiers are mapped to IP ports
- The *portmapper* service is used to register programs to ports and obtain the port for a given program

```
rpcinfo -p
```

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper

# Remote Procedure Call Semantics

## Semantics provided by Sun-style RPC

- Guaranteed mutual exclusion (at most one remote procedure active at any time in any server program)
- Only the weakest possible assumptions, based on the underlying protocols
  - **UDP** if the procedure returns, assume *at least once* execution
  - **UDP** if the procedure does not returns, assume *zero or more* execution

## What to do?

- No need to worry about mutual exclusion
- When using UPD, remote procedures need to be *idempotent*



# Remote Procedure Call Messages

## Message Format

- Not fixed
- Uses External Data Representation (XDR) to provide a machine-independent data representation

## External Data Representation (XDR)

- A symmetric data conversion solution:
  - Avoid having one conversion procedure for each server machine
  - However, double computational overhead
- XDR data structures are similar to C data structures

## XDR Data types (1)

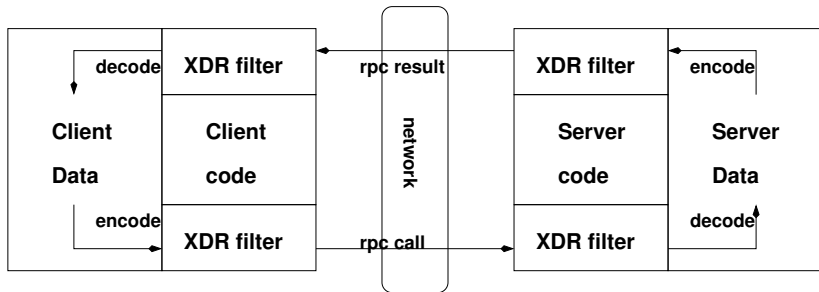
Type	Size	Description
int	32	32-bit integer
unsigned int	32	32-bit unsigned integer
bool	32	0 or 1
hyper	64	64-bit integer
unsigned hyper	64	64-bit unsigned integer
float	32	single precision float
double	64	double precision float
enum	arb	enumeration
string	arb	ASCII string
opaque	arb	raw data

## XDR Data types (2)

Type	Size	Description
fixed array	arb	fixed-size array
counted array	arb	max-sized array
structure	arb	C struct
union	arb	Pascal variant records
void	0	no data
constant	arb	symbolic constant
optional data	arb	0 or 1 occurrences of other type

- optional data used to represent pointers
- union uses type tags

# Remote Procedure Call Mechanism



- Data flow through XDR filters in both directions

# Remote Procedure Call Mechanism

## A rather cumbersome mechanism...

- 1 Remote program registers to portmap
- 2 Caller program gets port from portmap
- 3 Caller program encodes input parameters
- 4 Caller program passes the data to remote host
- 5 Remote callee decodes the input parameters
- 6 Remote callee operates
- 7 Remote callee encodes results
- 8 Remote host returns the results to caller program
- 9 Caller program decodes the results and resumes execution

# The rpcgen Compiler

## Overview

- Writing calls to the XDR library directly is cumbersome
- Use a *domain specific language* for both XDR and RPC
- Use rpcgen compiler to compile XDR *and* RPC program specification

# The rpcgen Compiler

## Structures

### Sample XDR Declaration

```
struct coord {  
    int x;  
    int y;  
};
```

### C translation

```
struct coord {  
    int x;  
    int y;  
};  
typedef struct coord coord;
```

# The rpcgen Compiler

## Discriminated Union

### Sample XDR Declaration

```
union foo switch (int n)
{
    case 0:
        opaque data[1024];
    default:
        void;
};
```

### C translation

```
struct foo {
    int n;
    union {
        char data[1024];
    } foo_u;
};
typedef struct foo foo;
```



# The rpcgen Compiler

## Constant, array, typedef and string

### Sample XDR Declaration

```
const DOZEN = 12;  
typedef string s<24>;  
int palette[8];  
int heights<12>;
```

### C translation

```
#define DOZEN 12  
typedef char *s;  
int palette[8];  
struct {  
    u_int heights_len;  
    int *heights_val;  
} heights;
```

# The rpcgen Compiler

## Booleans, pointers and raw data

- Boolean type **typedef int** bool\_t;
- Pointers are represented as *optional data*

### Sample XDR Declaration

```
bool married;  
listitem *next;  
opaque diskblock [512];  
opaque filedata <1024>;
```

### C translation

```
bool_t married;  
listitem *next;  
char diskblock [512];  
struct {  
    u_int filedata_len;  
    char *filedata_val;  
} filedata;
```

# The rpcgen Compiler

Defining remote procedures

## Sample rpcgen Declaration

```
program TIMEPROG {  
    version TIMEVERS {  
        unsigned int TIMEGET(v  
        void TIMESET(unsigned)  
    } = 1;  
} = 44;
```

## C translation

```
#define TIMEPROG 44  
#define TIMEVERS 1  
#define TIMEGET 1  
#define TIMESET 2
```

# Using the rpcgen Compiler

- Write an rpcgen source file (e.g., `avg.x`)
- Compile it: `rpcgen avg.x`
  - ① `avg.h` generated header file
  - ② `avg_clnt.c` generated client stub
  - ③ `avg_svc.c` generated server stub
  - ④ `avg_xdr.c` common xdr routines
- Write server and client logic
- Compile server and client programs