Unit 11
SUN's RPC

Computer Architecture Area (ARCOS)

Distributed Systems

Bachelor in Informatics Engineering

Universidad Carlos III de Madrid

RPC functions library

- RPC Functions library
- RPC services to build applications
  - Create a client handler
  - Destroy a client handler

Functions library

- **rpc** is a function library to develop distributed applicactions using RPCs:

```
#include <rpc/rpc.h>
```

- Multiple access levels:
  - Simpified interface
  - Intermediate level routines
  - High level routines

RPC services

- Create a client handler

```
CLIENT *
clnt_create (const char *host, const u_long prognum,
   const u_long versnum, const char *nettype)
```

Arguments:

- host Host name where the remote program is located on
- prognum Remote program number
- versnum Remote program version number
- nettype Transport protocol:
  NETPATH, VISIBLE, CIRCUIT_V, DATAGRAM_V, CIRCUIT_N, DATAGRAM_N, TCP, UDP

RPC services

- Destroy client handler

```
void clnt_destroy (CLIENT *clnt)
```

Arguments:

- clnt RPC client handler

RPC services

- Indicate error in case of RPC failure:

```
void clnt_perror (CLIENT *clnt, char *s)
void clnt_pcreateerror (CLIENT *clnt, char *s)
```

Arguments:

- clnt RPC client handler
- s Error message

Example: create/destroy a handler

```
#include <stdio.h>
#include <rpc/rpc.h>
#define RMTPROGNUM (u_long)0x3fffffffL  /* Define remote program number and version */
#define RMTPROGVER (u_long)0x1
main()
{
   CLIENT *client;  /* client handler */

   /* Create client handlre */
   client = clnt_create("as400.somewhere.ibm.com", RMTPROGNUM, RMTPROGVER, "TCP");
   if (client == NULL){
          clnt_pcreateerror(client, "Could not create client\n");
          exit(1);
   }
   // Call to remote procedure
   /* Destroy client handler */
   clnt_destroy(client);
   exit(0);
}
```

Call a remote procedure (client)

- A remote procedure is called:

```
type_result       procedure_v   (type_arg1 arg1,
                                 type_arg2 arg2,
                                 …
                                 type_argn argn,
                                 CLIENT *clnt)
```

where:

- procedure_v — Procedure name to call
- arg1,arg2,…,argn — Procedure arguments
- clnt — RPC client handler

Implement a procedure (server)

- For each remote procedure, the server offers an implementation of that procedure following its prototype:

```
type_result  procedure_v_svc        (type_arg1 arg1,
                                     type_arg2 arg2,
                                     …
                                     type_argn argn,
                                     struct svc_req *rqstp)
```

where:

- procedure_v_svc Procedure name to implement
- arg1,arg2,…,argn Procedure arguments
- rqstp Structure that contains request information

Interfaces compiler (rpcgen)

- **rpcgen** is the interface compiler that generates **C** code for:
  - Client stub
  - Server stub and main procedure
  - Procedures for XDR data marshalling and unmarshalling
  - Header file (.h) with types and procedure prototype declarations

rpcgen syntax

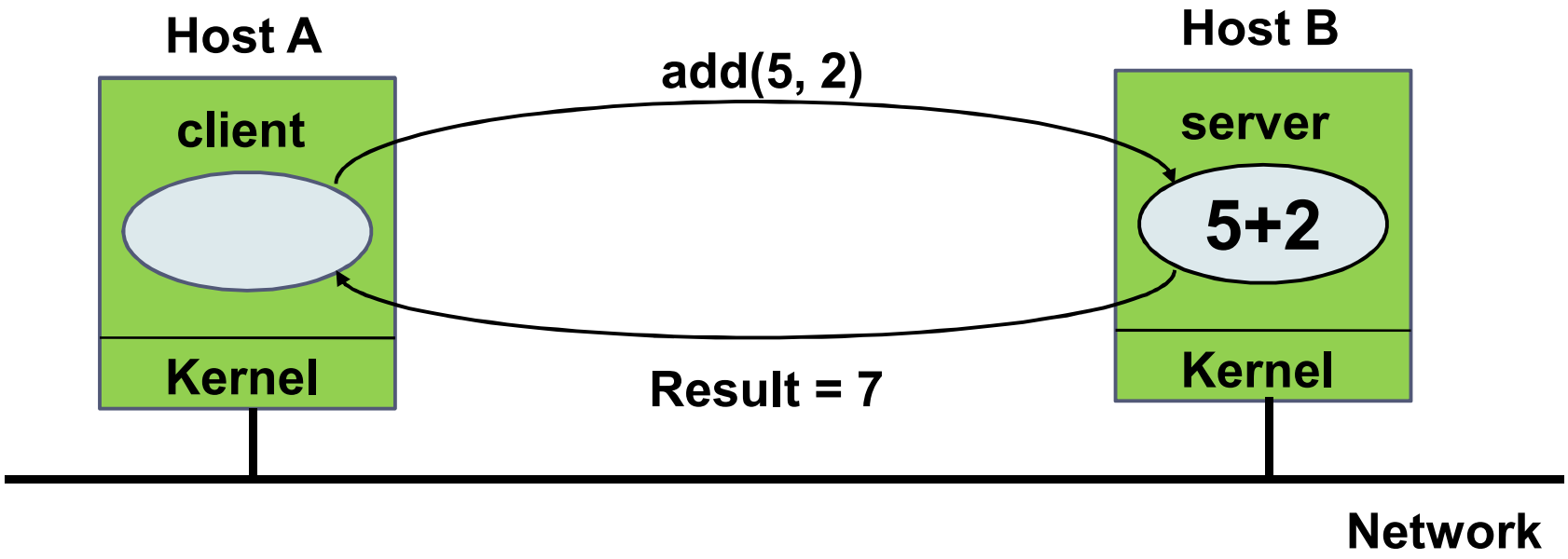
- Compile with rpcgen

```
rpcgen infile
```

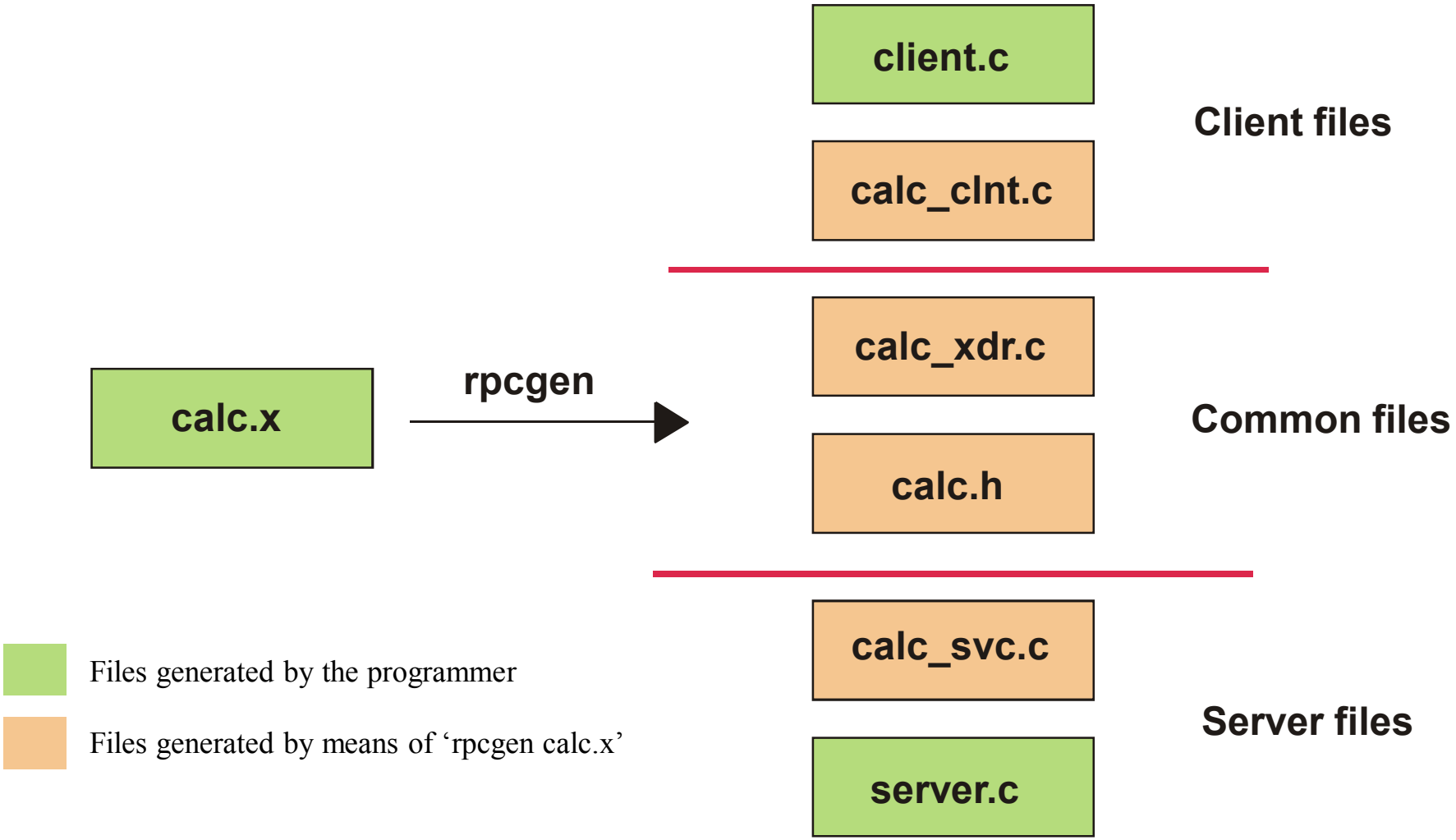
```
rpcgen [-abkCLNTM][-Dname[=value]] [-i size] [-I [-K seconds]] [-Y path] infile
rpcgen [-c | -h | -l | -m | -t | -Sc | -Ss | -Sm] [-o outfile] [infile]
rpcgen [-s nettype]* [-o outfile] [infile]
rpcgen [-n netid]* [-o outfile] [infile]
```

  - **infile** Input file written in RPC language
  - Some options:
    - -a Generates all files, including example codes for client and server
    - -N Allows procedures to have multiple arguments
    - -M Generates multithreaded code

Example: application using RPC



Application scheme

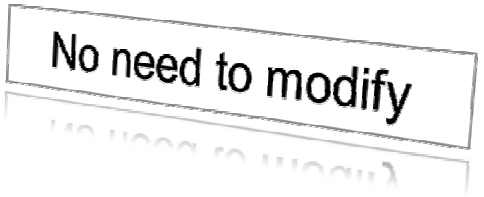


Example: calc.x

```
struct arguments {
  int a;
  int b;
};

program CALC {
   version CALCVER {
       int ADD(arguments)      = 1;
       int SUBTRACT(arguments) = 2;
   } = 1;
} = 99;
```

Example: calc.h

```
#ifndef _CALC_H_RPCGEN
#define _CALC_H_RPCGEN

#include <rpc/rpc.h>

struct arguments {
   int a;
   int b;
};
#define CALC    99
#define CALCVER  1

#define ADD      1
extern int * add_1(arguments *, CLIENT *);
extern int * add_1_svc(arguments *, struct svc_req *);
#define SUBTRACT 2
...

#endif /* !_CALC_H_RPCGEN */
```



Example: server.c

```
#include "calc.h"

int * add_1_svc(arguments *argp, struct svc_req *rqstp)
{
   // TODO: implementation of the procedure
   static int result;
   result = argp->a + argp->b;
   return(&result);
}
int * subtract_1_svc(arguments *argp, struct svc_req *rqstp)
{
   // TODO: implementation of the procedure
   static int result;
   result = argp->a - argp->b;
   return(&result);
}
```

Example: client.c (I)

```
#include "calc.h"

main( int argc, char* argv[] )
{
   CLIENT *clnt;
   int *res;
   arguments  add_1_arg;
   char *host;

   if(argc < 2) {
        printf("usage: %s server_host\n", argv[0]);
        exit(1);
   }
   host = argv[1];
```

Example: client.c (II)

```
    /* Step 1: locate the server */
    clnt = clnt_create(host, CALC, CALCVER, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror(host);
        exit(1);
    }

    add_1_arg.a = 5;
    add_1_arg.b = 2;

    /* Step 2: Call remote procedure */
    res = add_1(&add_1_arg, clnt);
    if (res == NULL) {
        clnt_perror(clnt, "call failed:");
    }
    printf("The sum is %d\n", *res);

    /* Step 3: Destroy handler */
    clnt_destroy( clnt );
}
```

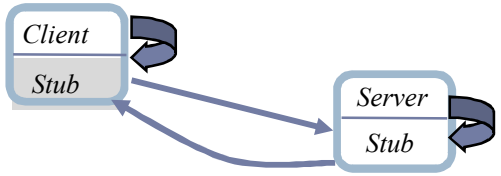
Binding:
The *host's portmapper* is contacted.
The host:API:version:protocol are indicated
The *portmapper* indicates the location (port, etc.)

calc_xdr.c

No need to modify

```
#include "calc.h"

bool_t
xdr_arguments (XDR *xdrs, arguments *objp)
{
        register int32_t *buf;

         if (!xdr_int (xdrs, &objp->a))
                 return FALSE;
         if (!xdr_int (xdrs, &objp->b))
                 return FALSE;
        return TRUE;
}
```

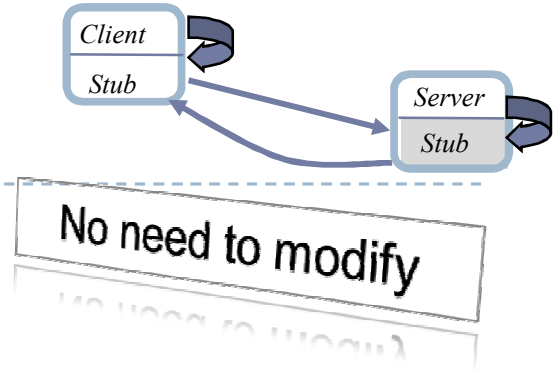
calc_clnt.c (client stub)



No need to modify

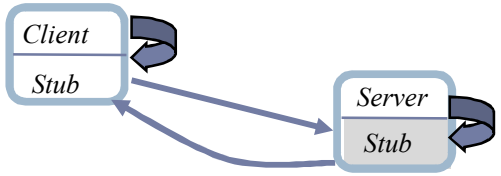
```
#include <memory.h> /* for memset */
#include "calc.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

int * add_1(arguments *argp, CLIENT *clnt)
{
   static int clnt_res;

   memset((char *)&clnt_res, 0, sizeof(clnt_res));
   if (clnt_call (clnt, ADD,
          (xdrproc_t) xdr_arguments, (caddr_t) argp,
          (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
          TIMEOUT) != RPC_SUCCESS) {
          return (NULL);
   }
   return (&clnt_res);
}
int * subtract_1(arguments *argp, CLIENT *clnt)
{
   …
}
```

calc_svc.c (server stub)




```
int main (int argc, char **argv)
{
        register SVCXPRT *transp;
        pmap_unset (CALC, CALCVER);

        transp = svcudp_create(RPC_ANYSOCK);
       svc_register(transp, CALC, CALCVER, calc_1, IPPROTO_UDP;

        transp = svctcp_create(RPC_ANYSOCK, 0, 0);
       svc_register(transp, CALC, CALCVER, calc_1, IPPROTO_TCP;

        svc_run ();

        fprintf (stderr, "%s", "svc_run returned");
        exit (1);
        /* NOTREACHED */
}
```

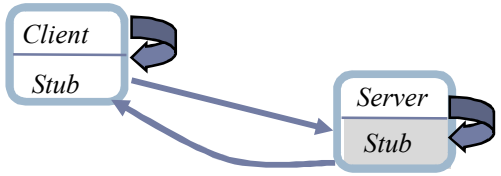
calc_svc.c (II) (server stub)




```
static void calc_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
   union {
         arguments add_1_arg;
         arguments subtract_1_arg;
   } argument;
   char *result;
   xdrproc_t _xdr_argument, _xdr_result;
   char *(*local)(char *, struct svc_req *);

   switch (rqstp->rq_proc) {
      case NULLPROC:
         (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
         return;

      case add:
         _xdr_argument = (xdrproc_t) xdr_arguments;
         _xdr_result = (xdrproc_t) xdr_int;
         local = (char *(*)(char *, struct svc_req *)) add_1_svc;
         break;
```

No need to modify

calc_svc.c (III) (server stub)



No need to modify

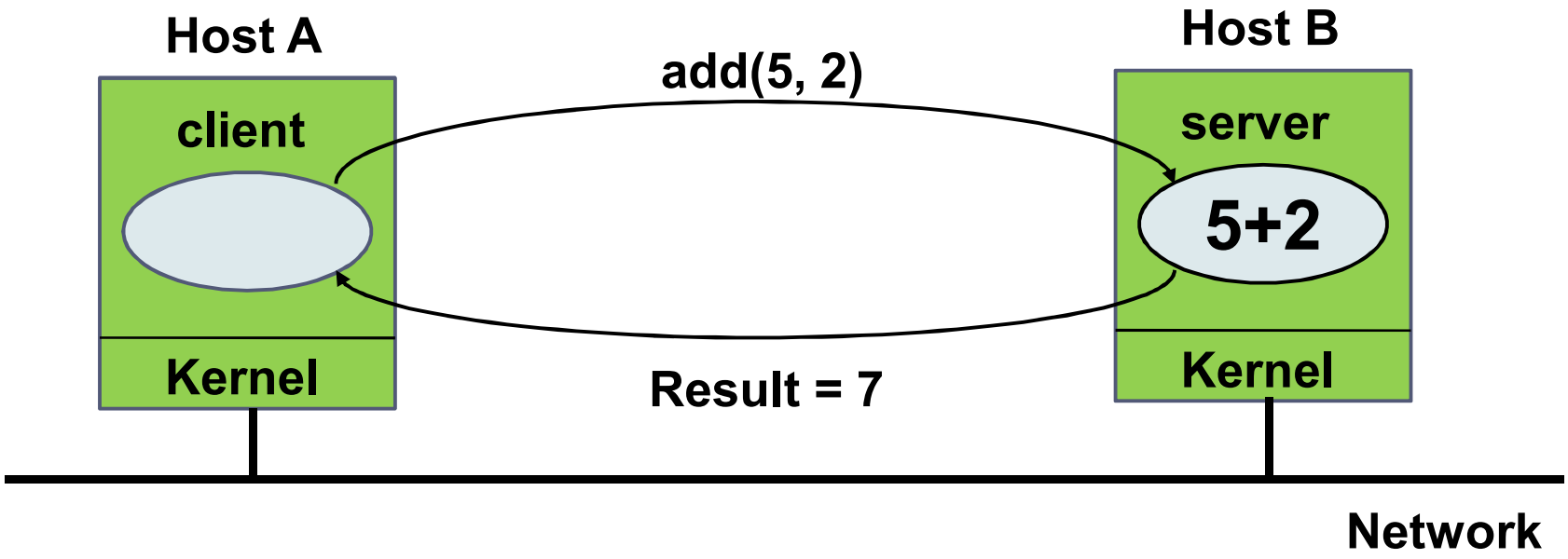
```
        case SUBTRACT:
            _xdr_argument = (xdrproc_t) xdr_arguments;
            _xdr_result = (xdrproc_t) xdr_int;
            local = (char *(*)(char *, struct svc_req *)) subtract_1_svc;
            break;

        default:
            svcerr_noproc (transp);
            return;
    }
    memset ((char *)&argument, 0, sizeof (argument));
    if (!svc_getargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument))
    { … }
    result = (*local)((char *)&argument, rqstp);
    if (result != NULL && !svc_sendreply(transp, (xdrproc_t) _xdr_result, result))
    { … }
    if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {
            fprintf (stderr, "%s", "unable to free arguments");  exit (1);    }
    return;
}
```

Compilation

- gcc –c calc_xdr.c
- gcc –c calc_clnt.c
- gcc –c calc_svc.c
- gcc –c client.c
- gcc –c server.c

- gcc calc_xdr.o calc_clnt.o client.o –o **client**
- gcc calc_xdr.o calc_svc.o server.o –o **server**

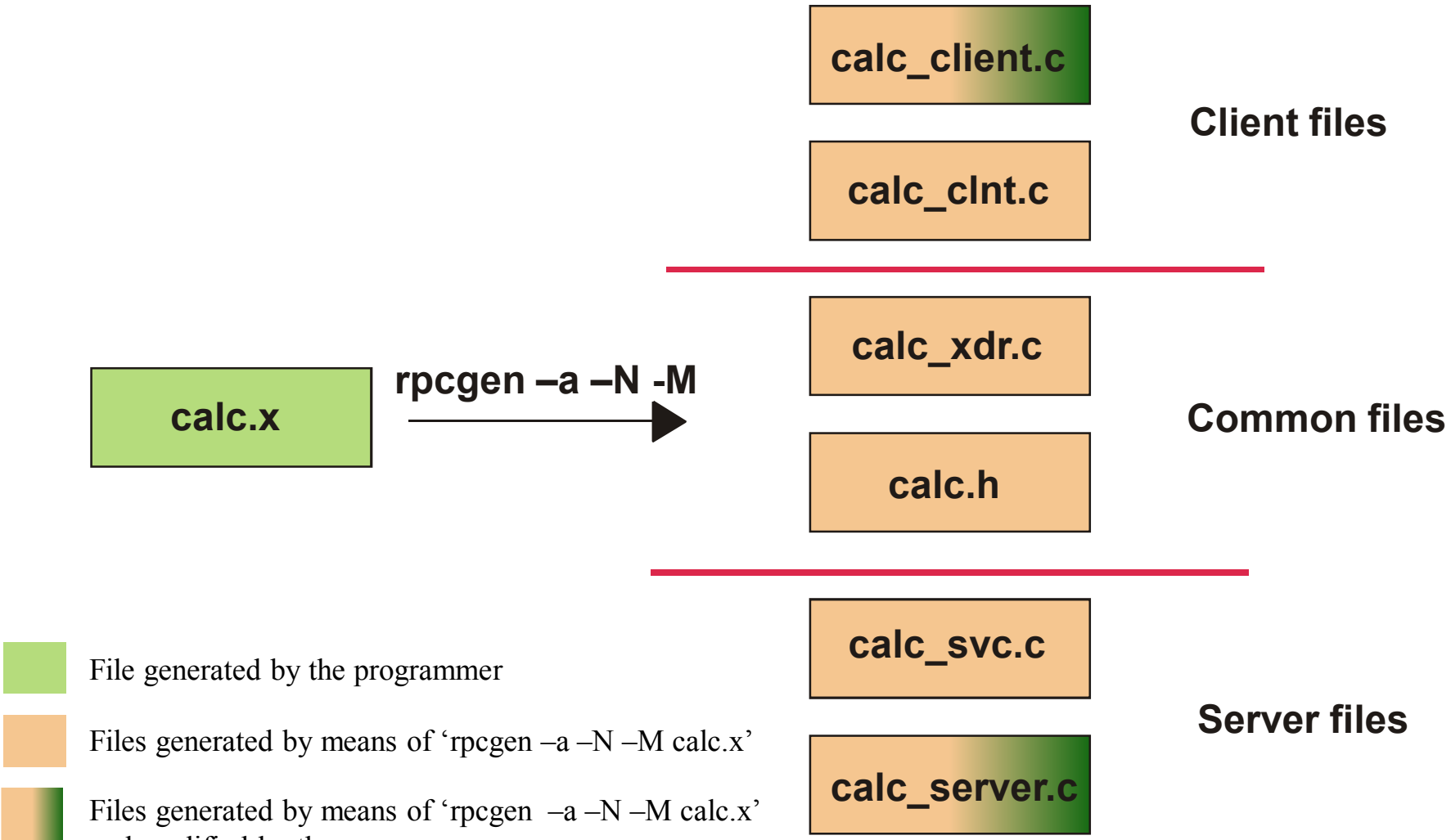
Example: new application using RPC



Compilation

- **rpcgen –a –N –M** calc.x
- Options:
  - -a: Generates all files including examples for client and server
  - -N: Allows procedures to have several arguments
  - -M: generate *multithreaded* code (code that can be used in applications with several *threads*)

New application scheme



Example using new options

```
program CALC {
    version CALCVER {
        int add(int a, int b)      = 1;
        int subtract(int a, int b) = 2;
    } = 1;
} = 99;
```

Generated files

- **calc.h**: includes functions prototypes
- **calc_client.c**: client example
- **calc_clnt.c**: client *stub*
- **calc_server.c**: template with the functions to be implemented in the server
- **calc_svc.c**: server *stub*
- **calc_xdr.c**: XDR functions

calc.h (I)

```
#include <rpc/rpc.h>
#include <pthread.h>

struct add_1_argument {
        int a;
        int b;
};
typedef struct add_1_argument add_1_argument;

struct subtract_1_argument {
        int a;
        int b;
};
typedef struct subtract_1_argument subtract_1_argument;
```

calc.h (II)

```
#define CALC 99
#define CALCVER 1

extern enum clnt_stat add_1(int , int , int *,
                            CLIENT *);
extern  bool_t add_1_svc(  int , int , int *,
                           struct svc_req *);

extern  enum clnt_stat subtract_1(int , int , int *,
                                  CLIENT *);
extern  bool_t subtract_1_svc(    int , int , int *,
                                  struct svc_req *);
```

calc_client.c: client example (I)

```
#include "calc.h"

void calc_1(char *host)
{
    CLIENT *clnt;
    enum clnt_stat retval_1;
    int result_1;   int add_1_a;   int add_1_b;
    enum clnt_stat retval_2;
    int result_2;   int subtract_1_a;  int subtract_1_b;

    clnt = clnt_create (host, CALC, CALCVER, "udp");
    if (clnt == NULL) {
           clnt_pcreateerror (host);
           exit (1);
    }
```

calc_client.c: client example (II)

```
 retval_1 = add_1(add_1_a, add_1_b, &result_1, clnt);
       if (retval_1 != RPC_SUCCESS) {
         clnt_perror (clnt, "call failed");
 }

 retval_2 = subtract_1(subtract_1_a, subtract_1_b,
  &result_2, clnt);
  if (retval_2 != RPC_SUCCESS) {
                clnt_perror (clnt, "call failed");
  }
 clnt_destroy (clnt);
}
```

calc_client.c: client example (III)

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

    if (argc < 2) {
        printf ("usage: %s server_host\n", argv[0]);
        exit (1);
    }
    host = argv[1];

    calc_1 (host);

    exit (0);
}
```

calc_server: server template (I)

```
#include "calc.h"

bool_t add_1_svc( int a, int b, int *result,
                  struct svc_req *rqstp)
{
    bool_t retval;
    /*
     * insert server code here
     */
    *result = a + b;
    retval = TRUE;

    return retval;
}
```

calc_server: server template (II)

```
bool_t subtract_1_svc(  int a, int b, int *result,
                        struct svc_req *rqstp)
{
    bool_t retval;
    /*
     * insert server code here
     */
    *result = a - b;
    retval = TRUE;

    return retval;
}
```

Authentication

- Request and reply messages have fields to specify authentication information
- The server is responsible of access control
- There are different authentication protocols:
  - None
  - UNIX-like, based on *uid* and *gid*
  - Kerberos authentication
  - By means of a shared key used to sign RPC messages

Protocol definition syntax

```
definition-list:
  definition;
   definition; definition-list

definition:
  const-definition
  enum-definition
  struct-definicion
  union-definition
  typedef-definition
  program-definition
```

Symbolic constant definition

- In XDR:

```
const MAX_SIZE = 8192;
```

- Translation to C:

```
#define MAX_SIZE 8192
```

Enumerated types

- In XDR:

```
enum colour{
   RED = 0;
   GREEN = 1;
   BLUE = 2;
};
```

- Translation to C:

```
enum colour{
   RED = 0;
   GREEN = 1;
   BLUE = 2
};
typdef enum colour colour;
bool_t xdr_colour();
```

Structures

- In XDR:

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

- Translation to C:

```
struct point{
  int x;
  int y;
};
typdef struct point point;
bool_t xdr_point();
```

Unions

- In XDR:

```
union result switch (int error){
   case 0:
     int n;
   default:
     void;
};
```

- Translation to C:

```
struct result {
  int error;
  union {
     int n;
  } result_u;
};
typdef struct result result;
bool_t xdr_result();
```

Types definition

- In XDR:

```
typedef point points[2];
```

- Translation to C:
  - Do not change

```
typedef point points[2];
```

Program definition

```
program-def:
"program" identifier "{"
      version-def
      version-def *
  "}" "=" constant ";"

version-def:
  "version" identifier "{"
      procedude-def
      procedure-def *
  "}" "=" constant ";"

procedure-def:
  type-ident procedure-ident "("type-ident")" "="
  value
```

```
program CALC {
   version CALCVER {
       int add(int a, int b)  = 1;
       int subtract(int a, int b) = 2;
   } = 1;
} = 99;
```

Basic data types

- C basic data types

- Examples:

Signed integers:

Declaration: `int a;`

C equivalent: `int a;`

Unsigned integers :

Declaration: `unsigned a;`

C equivalent: `unsigned a;`

Floating-point numbers:

Declaration: `float a;`

C equivalent: `float c;`

Main XDR data types

Floating-point numbers:

Declaration:
```
float a;
```
C equivalent:
```
float c;
```

Fixed-length byte arrays:

Declaration:
```
opaque a[20];
```
C equivalent:
```
char a[20];
```

Variable-length byte arrays:

Declaration:
```
opaque a<37>;
opaque b<>;
```
C equivalent:
```
struct {
        int a_len;
        char *a_val;
} a;
```

Main XDR data types

Character strings:

Declaration:

```
string a<37>;
string b<>;
```

C equivalent:

```
char *a;
chat *b;
```

Fixed-length vectors:

Declaration:

```
int a[12];
```

C equivalent:

```
int a[12];
```

Main XDR data types

Variable-length vectors:

Declaration:

```
int a<12>;
float b<>;
```

C equivalent:

```
struct {
        int a_len;
        int *a_val;
} a;
struct {
        int b_len;
        float *b_val;
} b;
```

Main XDR data types

Structures:

Declaration:

```
struct t {
        int c1;
        string c2<20>;
};
t a;
```

C equivalent:

```
struct t {
        int c1;
        char *c2;
};
typedef struct t t;
t a;
```

Constants:

Declaration:

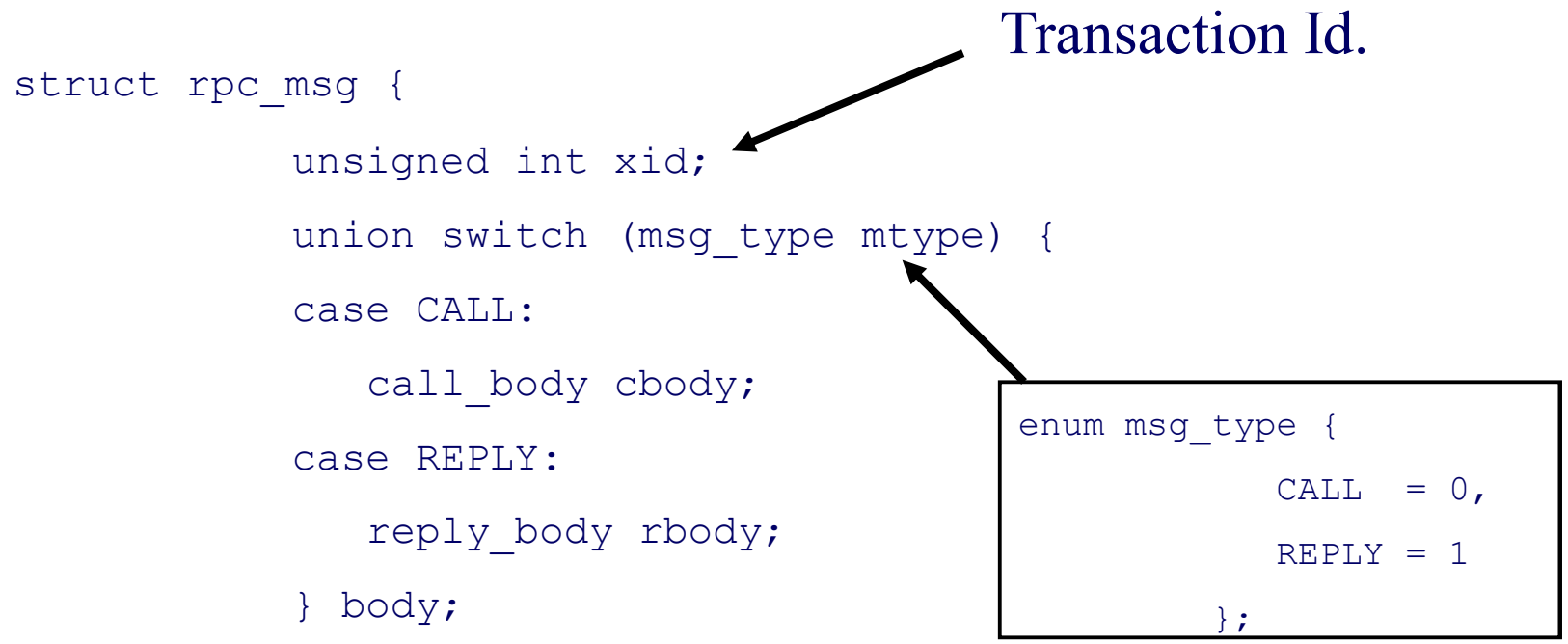
```
const MAX = 12;
```

C equivalent:

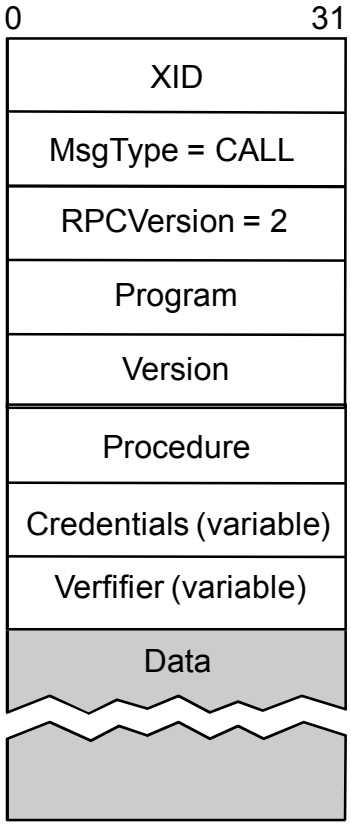
```
#define MAX 12
```

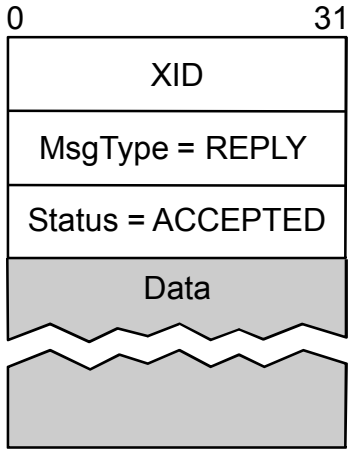
Call-reply messages

- Call-reply messages



Messages format





RPC evolution