

6 Remote Procedure Call (RPC)

- RFC 1831 (Sun)
- invoke procedure on remote host in analogy to local procedure call
- programming interface: (local) procedure call
- remote procedure call
 - active client calls
 - server sends reply
- network service = collection of remote programs
- remote program implements several remote procedures
- more than one version of remote program active on server

Differences RPC vs local calls

1. Error handling: failures of the remote server or network
2. Global variables and side-effects: not available
3. Performance: RPC usually one or more orders of magnitude slower
4. Authentication: may be necessary for RPC

RPC Overview

- client: calls *stub* for remote procedure
- client *stub*:
 - marshals (*converts*) parameters to external representation (XDR)
 - builds request from procedure id and parameters
 - sends request to server (UDP or TCP)
 - waits for result (synchronous)
- server: determines local procedure from id, calls its skeleton
- *procedure skeleton*:
 - unmarshals parameters from external representation
 - makes local call to procedure
 - marshals result to XDR
 - returns datagram to client
- client stub:
 - receives result
 - unmarshals from XDR
 - returns result to local procedure call

XDR

- RFC 1832 (Sun)
- language for describing data
- semantics: data representation in terms of octet stream
- a data item of n bytes is represented by $4 * i$ octets
where $r = 4 * i - n < 4$ (padding)
- fixes “network” byte order:
big-endian (most significant byte sent first)

XDR data types

- signed/unsigned 32 bit integer (`int`)
- C-style enumeration; represented as signed 32 bit integer
- Boolean (`bool`); equivalent to
`enum { FALSE = 0; TRUE = 1 } ident;`
- signed/unsigned 64 bit integer (`hyper`)
- floating point number: `float`, `double`, `quadruple`
- fixed length opaque data; length is `n`
`opaque ident[n];`
- variable length opaque data; max length is `n` or $2^{32} - 1$ if unspecified
`opaque ident1<n>; opaque ident2<>;`
- string type, like opaque, but with ASCII characters
`string str1<n>; string str2<>;`

- arrays:
 - fixed length typename obj[n];
 - variable length typename obj<n>
- structure

```
struct {  
    component-declaration-A;  
    component-declaration-B;  
    ...  
} identifier;
```
- discriminated union

```
union switch (discriminant-declaration) {  
    case discriminant-value-A:  
        arm-declaration-A;  
    case discriminant-value-B:  
        arm-declaration-B;  
    ...  
    default:  
        default-declaration;  
} identifier;
```

- `void` (no elements, no space)
- constant declaration
`const name-identifier = n;`
- type definition, modifies declaration to bind the type to a name, rather than binding a variable to a type
`typedef declaration;`
- optional data
`typename *identifier;`

XDR Example

```
const MAXUSERNAME = 32;    /* max length of a user name */
const MAXFILELEN = 65535; /* max length of a file      */
const MAXNAMELEN = 255;   /* max length of a file name */

/*
 * Types of files:
 */
enum filekind {
    TEXT = 0,    /* ascii data */
    DATA = 1,   /* raw data   */
    EXEC = 2     /* executable */
};

/*
 * File information, per kind of file:
 */
union filetype switch (filekind kind) {
case TEXT:
    void;          /* no extra information */
case DATA:
    string creator<MAXNAMELEN>; /* data creator      */
case EXEC:
    string interpretor<MAXNAMELEN>; /* program interpretor */
};
```



```
/*
 * A complete file:
 */
struct file {
    string filename<MAXNAMELEN>; /* name of file */
    filetype type;                /* info about file */
    string owner<MAXUSERNAME>;   /* owner of file */
    opaque data<MAXFILELEN>;     /* file data */
};
```

Requirements for RPC protocol

1. unique specification of called procedure; by triple
(program number, version number, procedure number)
2. matching responses to requests
3. authentication

opaque to the RPC protocol:

```
enum auth_flavor {  
    AUTH_NONE      = 0,  
    AUTH_SYS       = 1,  
    AUTH_SHORT     = 2  
    /* and more to be defined */  
};
```

```
struct opaque_auth {  
    auth_flavor flavor;  
    opaque body<400>;  
};
```

RPC message

```
struct rpc_msg {
    unsigned int xid;
    /* message id; used to match request/reply */
    union switch (msg_type mtype) {
    case CALL:
        call_body cbody;
    case REPLY:
        reply_body rbody;
    } body;
};
```

RPC calls

```
struct call_body {
    unsigned int rpcvers;           /* must be 2 */
    unsigned int prog;
    unsigned int vers;
    unsigned int proc;
    opaque_auth cred;
    opaque_auth verf;
    /* procedure specific parameters start here */
};
```

RPC replies

```
union reply_body switch (reply_stat stat) {  
    case MSG_ACCEPTED:  
        accepted_reply areply;  
    case MSG_DENIED:  
        rejected_reply rreply;  
} reply;
```

RPC accepted replies

```
struct accepted_reply {
    opaque_auth verf;
    union switch (accept_stat stat) {
    case SUCCESS:
        opaque results[0];
        /*
         * procedure-specific results start here
         */
    case PROG_MISMATCH:
        struct {
            unsigned int low;
            unsigned int high;
        } mismatch_info;
    default:
        /*
         * Void. Cases include PROG_UNAVAIL, PROC_UNAVAIL, GARBAGE_ARGS, and SYSTEM_ERR.
         */
        void;
    } reply_data;
};
```

RPC rejected replies

```
union rejected_reply switch (reject_stat stat) {
case RPC_MISMATCH:
    struct {
        unsigned int low;
        unsigned int high;
    } mismatch_info;
case AUTH_ERROR:
    auth_stat stat;
};
```

The RPC Language

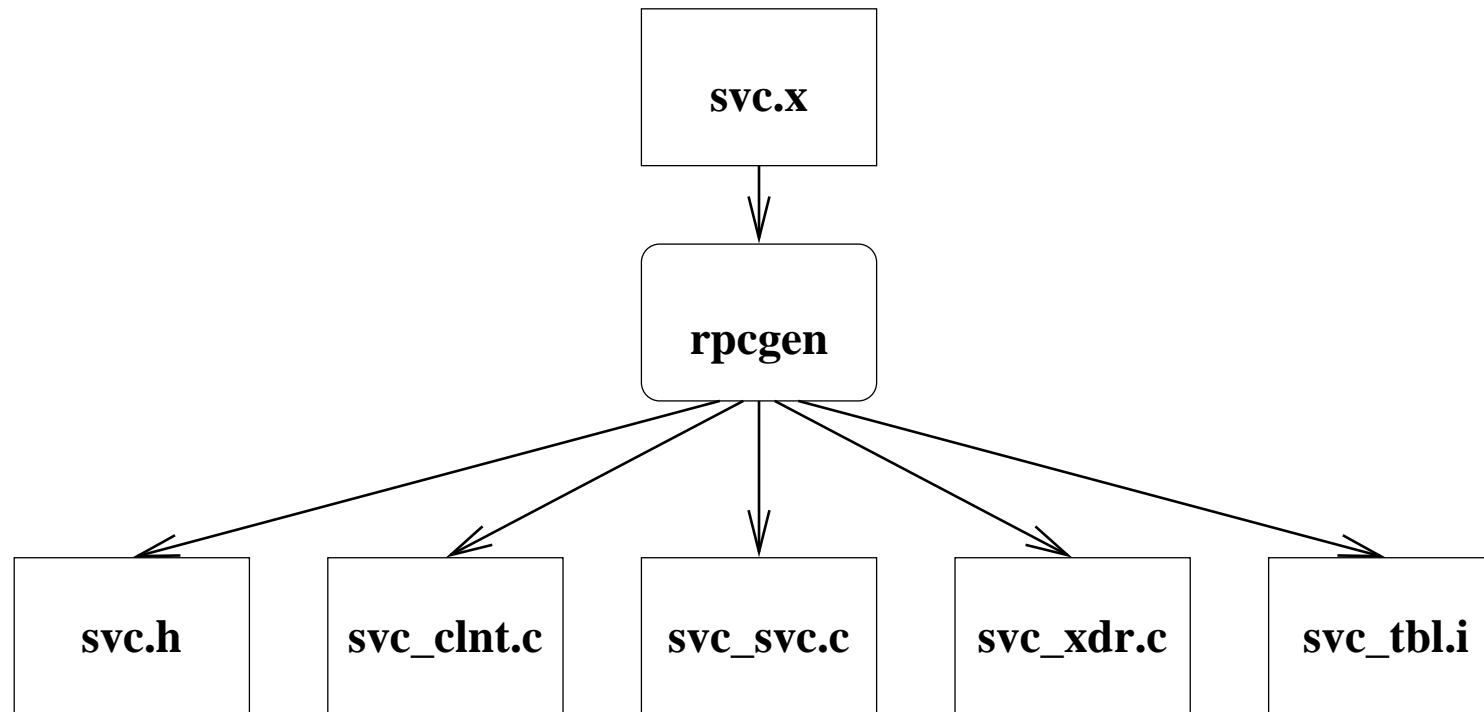
```
program PING_PROG {
  /*** Latest and greatest version */
  version PING_VERS_PINGBACK {
    void
    PINGPROC_NULL(void) = 0;

    /*
     * Ping the client, return the round-trip time
     * (in microseconds). Returns -1 if the operation
     * timed out.
     */
    int
    PINGPROC_PINGBACK(void) = 1;
  } = 2;

  /*** Original version */
  version PING_VERS_ORIG {
    void
    PINGPROC_NULL(void) = 0;
  } = 1;
} = 1;

const PING_VERS = 2;      /* latest version */
```


RPC programming



RPC example client code

```
#include "ping.h"

void
ping_prog_2(char *host)
{
    CLIENT *clnt;
    void *result_1;
    char *pingproc_null_2_arg;
    int *result_2;
    char *pingproc_pingback_2_arg;

    clnt = clnt_create (host, PING_PROG, PING_VERS_PINGBACK, "udp");
    if (clnt == NULL) { clnt_pcreateerror (host); exit (1);}

    result_1 = pingproc_null_2((void*)&pingproc_null_2_arg, clnt);
    if (result_1 == (void *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    result_2 = pingproc_pingback_2((void*)&pingproc_pingback_2_arg, clnt);
    if (result_2 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    clnt_destroy (clnt);
}
```