Remote Procedure Call Implementations

- Sun ONC(Open Network Computing) RPC. Implements at-most-once semantics by default. At-least-once (idempotent) can also be chosen as an option for some procedures.
  - NFS (Network File Service)
  - NIS (Network Information Service)
- DCE (Distributed Computing Environment) RPC. Implements at-most-once semantics by default. At-least-once (idempotent) can also be chosen as an option for some procedures.
  - Distributed file service.
  - Directory service.
  - Security service.
  - Distributed Time Service.

SUN ONC RPC Details (1)

- Each RPC procedure is uniquely defined by program and version numbers.
- The program number specifies a group of related remote procedures each of which has a different procedure number.
- Each program also has a version number so that, when a minor change is made to a remote service, a new program number does not have to be assigned.
- The file `/etc/rpc` lists some well known RPC programs. The program `rpcinfo` lists the registered RPC programs on a particular system.
- The `portmap` or `rpcbind` network service runs on each system on a well known port. To find the port for a remote program, a client sends an RPC call message to the portmap server.
- Can use either TCP/IP or UDP underneath to make remote calls. The selection depends on the application requirements.

SUN ONC RPC Details (2)

- UDP is good choice for idempotent operations, arguments/reply size $<$ 8K, capacity to handle several hundred clients. TCP is a good choice for non-idempotent operations, or if arguments/results exceed 8K and reliable transport is desired.
- XDR (eXternal Data Representation) is used to handle data layout in a machine-independent manner.
- Program numbers are assigned in blocks of 0x2000000 according to the following table.

0x0-0x1FFFFFFF	defined by Sun
0x20000000-0x3FFFFFFF	defined by user
0x40000000-0x5FFFFFFF	transient
0x60000000-0xFFFFFFFF	reserved

SUN ONC RPC Details (3)

- The IDL compiler is named rpcgen. Includes options to generate thread-safe server/client stub code. Some implementations can also generate multi-threaded servers with an option. The server is single-threaded for DCE RPC and some other implementations.
- RPC has three authentication mechanisms: `AUTH_NONE`, `AUTH_UNIX`, and `AUTH_DES`. The last option requires a public-key cryptosystem (a separate system).
- Header files are in /usr/include/rpc/ and /usr/include/rpcsvc. The latter one has header files for common services implemented using RPC.

Observing RPC on a server

In the `onyx` lab, you can try the following commands:

- Use the following command to get a summary of rpc services available on a server:
  `rpcinfo -s onyx`
- Use the following command to make a RPC call to procedure 0 (the null procedure) for RPC program number 100003 and report whether a response was received.
  `rpcinfo -T tcp onyx 100003`
- See man page for more options.

Developing RPC based Client/Servers

- Develop a protocol header file describing the interface to the remote procedures.
- Compile the protocol header file using the rpcgen compiler. It then produces C language output consisting of skeleton versions of the client routines, a server skeleton, XDR filter routines for both parameters and results, a header file that contains common definitions, and optionally, dispatch tables that the server uses to invoke routines that are based on authorization checks.
- Then write the server functions in any language that supports system calling conventions. Compile the server along with the server skeleton generated by `rpcgen`.
- To create an executable program for a remote program, write an ordinary main program that makes local procedure calls to the client skeletons, and link the program with the rpcgen skeletons.

RPC and XDR languages

The XDR language is for describing data and is similar to C declarations. The RPC language is an extension to the XDR language, with the addition of `program` and `version` types.
An RPC file is a series of definitions. The following definition types are recognized.

- `enum-definition`
- `typedef-definition`
- `const-definition`
- `declaration-definition`
- `struct-definition`
- `union-definition`
- `program-definition`

XDR Enum, Typedef, and Const Definitions

- XDR enum before compilation.

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

- C enum resulting after compilation.

```
enum colortype {
    RED = 0,
    GREEN = 1,
    BLUE = 2
};
typedef enum colortype colortype;
```

- An XDR typedef followed by the compiled C typedef.

```
typedef string fname_stype<255>;
typedef char *fname_type;
```

- An XDR constant declaration followed by compiled C version.

```
const DOZEN = 12;
#define DOZEN 12
```

XDR Declarations

- Simple declarations. Same syntax as in C.
- Fixed-length array declarations. Same syntax as in C.
- Variable-length array declarations. None in C, so XDR uses its own notation using angle brackets.

```
int heights<12>  /* at most 12 items */
int widths<>     /* any number of items */
```

  The compiled C version.

```
struct {
  u_int heights_len; /* number of items in the array */
  int *heights_val;  /* pointer to array */
} heights;
```

- Pointer declarations. Same as in C. Pointers cannot be sent over the network, but we can use XDR pointers to send recursive data types, such as lists and trees. In XDR, this is known as optional-data, instead of a pointer. For example:

```
listitem * next;
```

Special Cases

- Booleans. The `bool` type in XDR maps to `bool_t` type in C with TRUE/FALSE defined in the header files.
- Strings. XDR has a `string` type, followed by angle brackets to denote the maximum size of the string (not including the terminating null character).
- Opaque data. XDR uses `opaque` type to describe untyped data, consisting of sequences of arbitrary bytes. The `opaque` type gets translated ino `char` type in C.
- Voids. In a `void` declaration, the variable is not named. Declarations of voids occur only in union and program definitions.

XDR Struct's and Union's

- An XDR struct is almost like in C. For example.

```
/* XDR version */
struct coord {
    int x;
    int y;
}

/* here is the compiled C version */
struct coord {
    int x;
    int y;
}
typedef struct coord coord;
```

- XDR Unions are discriminated unions unlike C unions. An example.

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

/* the compiled C version */
struct read_result {
    int errno;
    union {
        char data[1024];
    } read_result_U;
};
tyepdef struct read_result read_result;
```

Programs

- An example program definition.

```
program TIMEPROG {
    version TIMEVERS {
        unsigned int TIMEGET(void) = 1;
        void TIMESET(unsigned) = 2;
    } = 1;
} = 44;
```

- The corresponding C declarations.

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

Examples

See the folder `rpc` in the examples.

- square
- msg
- sort
- userlookup
- linked list
- thread-safe-square