CU CS 4995-03: Internet Systems Programming

Remote Procedure Calls: Sun RPC

Overview

- why RPC?
- components
- XDR
- rpcgen

Remote Procedure

- server and client
- look like C (or Java or ...)
- Sun RPC = simpler version of Corba, RMI, DCE, . . .
- RPC = transport + data encoding
- expects and returns pointers instead of values
- thus, need `static` for return values

Sun RPC

Look like function calls to programmer, but…

1. function invokes *client stub* procedure
2. client stub packages arguments into packets
3. *server stub* receives message and calls function
4. on function return, server stub sends result
5. client stub returns results to program

Advantages:

- no network programming

- retransmission handled by RPC package
- data translation (no `htonl(),...`) ➠ XDR: (un)signed integers, booleans, floating point, fixed/variable-length arrays, structures

RPC request

field	length
(length field for TCP)	4
transaction ID (XID)	4
call (0)	4
RPC version (2)	4
program number	4
version number	4
procedure number	4
credentials	< 400
verifier	< 400
parameters	

RPC reply

field	length
transaction ID (XID) of request	4
reply (1)	4
status (0=accepted)	4
verifier	< 400
accept status (0=success)	4
procedure results	

Portmapper

- RPC servers use ephemeral ports
- *portmapper* server registers RPC programs via RPC
- always resides at port 111
- client obtains port numbers of desired program via RPC

Portmapper: example

```
$ rpcinfo -p
program vers proto   port  service
 100000    4   tcp    111  rpcbind
 100000    3   tcp    111  rpcbind
 100000    2   tcp    111  rpcbind
 100000    4   udp    111  rpcbind
 100000    3   udp    111  rpcbind
 100000    2   udp    111  rpcbind
 100007    3   udp  32773  ypbind
 100003    2   udp   2049  nfs  [fixed port!]
 100005    1   udp  32828  mountd
 100005    2   udp  32828  mountd
 100005    1   tcp  32793  mountd
 100005    2   tcp  32793  mountd
```

Interface Definition

Example: remote console printing

```
program MESSAGEPROG {
  version PRINTMESSAGEVERS {
    int PRINTMESSAGE(string) = 1;
  } = 1;
} = 0x20000001;
```

XDR (RFC 1832)

integer: 32-bit or 64-bit signed or unsigned

```
int identifier;
unsigned int identifier;
unsigned hyper identifier;
```

enum: 32-bit signed integer

```
enum { RED = 2, YELLOW = 3, BLUE = 5 } colors;
```

boolean: 32-bit signed integer

```
bool identifier;
```

XDR data types

float, double: IEEE floating point ($\pm 2^{e-127} \cdot 1.b$, with 8/11/15 bits exponent, 23/52/112 bits mantissa)

```
float identifier;
double identifier;
quadruple identifier;
```

opaque: any data, not changed; fixed or variable $\longrightarrow$ count + data

```
opaque identifier[m];
opaque identifier<m>;
```

string: ASCII; similar to opaque

XDR

array: `type-name identifier[n];`

struct: encoded as sequence of elements

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

discriminated union: discriminant + structure: union switch (discriminant-declaration) case discriminant-value-A: arm-declaration-A; case discriminant-value-B: arm-declaration-B; ... default: default-declaration; identifier;

Sample Remote Procedure

```
#include <stdio.h>
#include "printmsg.h"  /* generated by rpcgen */

int *printmessage_1(char **msg, struct svc_req *req)
{
  static int result;   /* must be static! */
  FILE *f;

  f = fopen("/dev/console", "w");
  if (f == (FILE *)NULL) {
    result = 0;
    return (&result);
  }
  fprintf(f, "%s\n", *msg);
  fclose(f);
  result = 1;
  return &result;
```

}

Server Procedure

`svc_req` contains context of invocation:

- the program version, procedure numbers, raw and canonical credentials
- transport information

```
struct svc_req {
  rpcprog_t   rq_prog;     /* service program number */
  rpcvers_t   rq_vers;     /* service protocol version */
  rpcproc_t   rq_proc;     /* the desired procedure */
  struct opaque_auth rq_cred; /* raw creds from the wire */
  caddr_t     rq_clntcred;     /* read only cooked cred */
  struct __svcxprt *rq_xprt;  /* associated transport */
};
```

RPC Client

```
#include <stdio.h>
#include "printmsg.h"   /* msg.h generated by rpcgen */

main(int argc, char *argv[])
{
  CLIENT *clnt;
  int *result;
  char *server;
  char *message;

  server = argv[1];
  message = argv[2];
  /*
   * Create client "handle" used for
   * calling MESSAGEPROG on the server
   * designated on the command line.
   */
```

```
clnt = clnt_create(server, MESSAGEPROG, PRINTMESSAGEVERS,
       "visible");
if (clnt == (CLIENT *)NULL) {
  clnt_pcreateerror(server);
  exit(1);
}

result = printmessage_1(&message, clnt);
if (result == (int *)NULL) {
  clnt_perror(clnt, server);
  exit(1);
}
if (*result == 0) {
  fprintf(stderr,
   "%s: could not print your message\n",argv[0]);
  exit(1);
}

printf("Message delivered to %s\n", server);
```

```
  clnt_destroy(clnt);
  exit(0);
}
```

Generating an RPC Client and Server

```
$ rpcgen printmsg.x
$ gcc rprintmsg.c printmsg_clnt.c -o rprintmsg -lnsl
$ gcc msg_proc.c printmsg_svc.c -o msg_server -lnsl
```

on remote host, start server: `msg_server`; on client: `printmsg bart "Hello, world."`

Other rpcgen invocations

-N : C-like, multiple arguments

multithreaded: server creates threads; (-A)

XDR Primitives

```
bool_t xdr_char(xdrs, op)
bool_t xdr_u_char(xdrs, ucp)
bool_t xdr_int(xdrs, ip)
bool_t xdr_u_int(xdrs, up)
bool_t xdr_long(xdrs, lip)
bool_t xdr_u_long(xdrs, lup)
bool_t xdr_short(xdrs, sip)
bool_t xdr_u_short(xdrs, sup)
bool_t xdr_string(xdrs, sp, maxlength)
```

strings, arrays, discriminated unions