CIS 620 Remote Procedure Calls (RPC) Spring 2020

To get a general concept of RPC, read Tanenbaum's textbook:

- Middleware Protocols (pages 122–124)
- Subsection 4.2 till 4.2.2 (pages 125–134)

When building a socket-based client/server application, there are many things which programmers need to deal with. One typical example is how to pack data in a message and how to extract them. Furthermore, in order to let the application be portable between big-endian and little-endian machines, the data byte-order issue cannot be ignored. So RPC is such a mechanism to ease the work. Below are eight steps (mainly based on "Internetworking with TCP/IP", Vol 3 by Comer and Stevens) for building a distributed application using RPC:

1. Build and test a conventional application.
2. Divide the program by choosing a set of procedures to move to a remote machine. Place the selected procedure in a separate file.
3. Write an `rpcgen` specification for he remote program, including names and numbers for the remote procedures and the declarations of their arguments. Choose a remote program number and a version number (usually 1). Usually, this interface specification file name has the suffix `.x` .
4. Run rpcgen to check the specification and, if valid, generate the four source code files that will be used in the client and server.
5. Modify each routine's interface (e.g. using call-by-reference for parameters) and some corresponding code for the client side and server side.
6. Compile and link together the client program. It consists of three files: the original application program (with the remote procedures removed), the client-side stub (generated by rpcgen), and the XDR (eXternal Data Representation) procedures (generated by rpcgen). When all these files have been compiled and linked together, the resulting executable program becomes the client.
7. Compile and link together the server program. It consists of three files: the procedures taken from the original application that now comprise the remote program, the server-side stub, (generated by rpcgen), and the XDR-procedures (generated by rpcgen). When all these files have been compiled and linked together, the resulting executable program becomes the server.
8. Start the server on the remote machine and then invoke the client on the local machine.

Let's use the RDB example (in "Power Programming with RPC" by Bloomer) to show how it works. It is a simple application where a client can send a request to a remote database server. Open a terminal and login to the workstation `spirit`. Type the following command to get the rdb example under your directory:

```
spirit> tar xvfz ~cis620s/pub/rdb.tar.gz
./rdb/
```

```
./rdb/rdb.c
./rdb/makefile
./rdb/rdb.x
./rdb/rdb_svc_proc.c
./rdb/personnel.dat
```

Next, to see the contents of the interface specification `rdb.x`, type:

```
spirit> cd rdb
spirit> nl rdb.x
     1 /*
     2  * rdb.x: remote database access protocol
     3  */
     4 /* preprocessor directives */
     5 %#define DATABASE "personnel.dat" /* '%' passes it through */

     6 /* constant definitions */
     7 const MAX_STR = 256;

     8 /* structure definitions, no enumerations needed */
     9 struct record {
    10   string firstName<MAX_STR>; /* <> defines the maximum */
    11   string middleInitial<MAX_STR>;  /* possible length */
    12   string lastName<MAX_STR>;
    13   int phone;
    14   string location<MAX_STR>;
    15 };

    16 /* program definition, no union or typdef definitions needed */
    17 program RDBPROG { /* could manage multiple servers */
    18      version RDBVERS {
    19                  record FIRSTNAME_KEY(string) = 1;
    20                  record LASTNAME_KEY(string) = 2;
    21                  record PHONE_KEY(int) = 3;
    22                  record LOCATION_KEY(string) = 4;
    23                  int ADD_RECORD(record) = 5;
    24      } = 1;
    25 } = 0x20000001;  /* program number ranges established by ONC */
```

You can see that there are five remote procedures (lines 19–23) and each has been assigned a function number (i.e. 1 to 5). Using the remote function `record LASTNAME_KEY(string)` as an example, it takes the last name (through the parameter) as the key to search the database file (i.e. `"personnel.dat"`) and returns a record back. Note that the remote function name must be capitalized in `.x` file. Different applications use different RPC program numbers. Numbers below `0x20000000` are reserved by the system. The RPC specification language extends the C language data type and the angle-bracket `< >` means variable-length.

The client code is in the file `rdb.c`. Note that some necessary modification (in Step 5, Page 1) has been made in it already. Use the command `nl` to see the code:

```
spirit> nl rdb.c
     1 #include <stdio.h>
     2 #include <ctype.h>
     3 #include <rpc/rpc.h>
     4 #include "rdb.h"

     5 #define PRINTRECORD(pR) {\
     6   printf("first\tmiddle\tlast\tphone\tlocation\n"); \
     7   printf("%s\t%s\t%s\t%d\t%s\n", \
     8     pR->firstName, pR->middleInitial, \
     9     pR->lastName, pR->phone, pR->location); \
    10   }

    11 int main(argc, argv)
    12   int             argc;
    13   char           *argv[];
    14 {
    15   CLIENT         *cl;    /* a client handle */
    16   char           *value;
    17   int             key;

    18   if ((argc != 4) || (!isdigit(argv[2][0]))) {
    19     fprintf(stderr, "Usage: %s server key value\n", argv[0]);
    20     exit(1);
    21   }
    22   if (!(cl = clnt_create(argv[1], RDBPROG, RDBVERS, "tcp"))) {
    23     /*
    24      * CLIENT handle couldn't be created, server not there.
    25      */
    26     clnt_pcreateerror(argv[1]);
    27     exit(1);
    28   }
    29   value = argv[3];
    30   switch (key = atol(argv[2])) {
    31   case FIRSTNAME_KEY:
    32     PRINTRECORD(firstname_key_1(&value, cl));
    33     break;
    34   case LASTNAME_KEY:
    35     PRINTRECORD(lastname_key_1(&value, cl));
    36     break;
    37   case PHONE_KEY:{
    38     int             p;
    39     if (!(sscanf(argv[3], "%d", &p)) != 1) {
    40       fprintf(stderr,"\"PHONE_KEY\" requires integer value\n");
```

```
41        exit(1);
42      }
43      PRINTRECORD(phone_key_1(&p, cl));
44      break;
45    }
46    case LOCATION_KEY:
47      PRINTRECORD(location_key_1(&value, cl));
48      break;
49    case ADD_RECORD:{
50      record  *pR = (record *) malloc(sizeof(record));
51      pR->firstName = (char *) malloc(MAX_STR);
52      pR->middleInitial = (char *) malloc(MAX_STR);
53      pR->lastName = (char *) malloc(MAX_STR);
54      pR->location = (char *) malloc(MAX_STR);
55      if (sscanf(argv[3], "%s%s%s%d%s", pR->firstName,
56          pR->middleInitial, pR->lastName, &(pR->phone),
57          pR->location) != 5) {
58        fprintf(stderr, "\"ADD_RECORD\" requires a complete quoted record\n");
59        exit(1);
60      }
61      if (!(*add_record_1(pR, cl))) {
62        fprintf(stderr, "couldn't add record\n");
63        exit(1);
64      }
65      break;
66    }
67    default:
68      fprintf(stderr, "%s: unknown key\n", argv[0]);
69      exit(1);
70    }
71 }
```

The first thing to add into the conventional program is in line 22, which we use

`cl = clnt_create(argv[1], RDBPROG, RDBVERS, "tcp")`

to get a client handle `cl`. The `argv[1]` is the remote server name. Next, we also need to modify the invocation of the remote function. Using the code in line 35 as an example,

`PRINTRECORD(lastname_key_1(&value, cl));`

the letters of the function name `lastname_key` are small, not capitalized as in `rdb.x`. The function name has `_1` appended at the end where `1` is the version number. Furthermore, we have to use call-by-reference for the string parameter `value` and add the client handle `cl` as the second parameter. The return type specified in the `rdb.x` is `record` and it should be also changed to call-by-reference, i.e., a pointer to a record. So the macro `PRINTRECORD(pR)` uses this pointer to print each field in the record out.

Now we can use `nl` again to see the server code which is in the file `rdb_svc_proc.c`.

```
spirit> nl rdb_svc_proc.c
     1 #include <stdio.h>
```

```
 2 #include <string.h>
 3 #include <rpc/rpc.h>
 4 #include "rdb.h"

 5 FILE           *fp = NULL;
 6 static record  *pR = NULL;

 7 int
 8 readRecord()
 9 {
10   char            buf[MAX_STR];

11   if (!pR) {
12     pR = (record *) malloc(sizeof(record));
13     pR->firstName = (char *) malloc(MAX_STR);
14     pR->middleInitial = (char *) malloc(MAX_STR);
15     pR->lastName = (char *) malloc(MAX_STR);
16     pR->location = (char *) malloc(MAX_STR);
17   }
18   if (!fgets(buf, MAX_STR - 1, fp)) return (0);
19   if (sscanf(buf, "%s%s%s%d%s", pR->firstName, pR->middleInitial,
20      pR->lastName, &(pR->phone), pR->location) != 5) return (0);
21   return (1);
22 }

23 record         *
24 firstname_key_1_svc(char **name, struct svc_req *rqp)
25 {
26   return ((record *) pR);
27 }

28 record         *
29 lastname_key_1_svc(char **name, struct svc_req *rqp)
30 {
31   if (!(fp = fopen(DATABASE, "r")))
32     return ((record *) NULL);

33   while (readRecord())
34     if (!strcmp(pR->lastName, *name)) break;
35   if (feof(fp)) {
36     fclose(fp);
37     return ((record *) NULL);
38   }
39   fclose(fp);
40   return ((record *) pR);
41 }
```

```
42 record         *
43 phone_key_1_svc(int *phone, struct svc_req *rqp)
44 {
45   return ((record *) pR);
46 }

47 record         *
48 location_key_1_svc(char **name, struct svc_req *rqp)
49 {
50   return ((record *) pR);
51 }

52 int            *
53 add_record_1_svc(record *r, struct svc_req *rqp)
54 {
55   static int             status;
56   return ((int *) &status);
57 }
```

Note that among the five remote functions, only the function `lastname_key()` is completely implemented in this example. Let's look the modification we need to make (in Step 5) for the server code. Lines 28 and 29 show the modified function interface for the function which uses the last name as the search key. The function name should be appended `_1_svc` at the end. Remember that the C language uses `char *` for the string. Because of the call-by-reference requirement, we have to use `char **` for the first parameter. We also need to add a second parameter. Furthermore, the result returned back to client should be through a pointer. Note that RPC does NOT really pass addresses between the client and the server because an address in one process is meaningless to another process, let alone they are on different machines. In fact, RPC uses the marshaling technique. Through the address, RPC merges all necessary data into a packet and then sends it to the remote. The server unpacks the message to its buffer and calls the corresponding function with the address of the buffer. Returning results will be done in a similar way. Refer Tanenbaum's book for detailed explanation.

You might be curious to know that how the client executable can be built because the functions have been moved to the remote side and hence the linker usually will generate error messages like

`undefined reference to 'remote_func_name'.`

Similarly, if you look the code `rdb_svc_proc.c`, there is no `main()` function. Then, how the server executable can be built? To answer these questions, let's use the provided makefile to build the application.

```
spirit> make
rpcgen rdb.x
gcc -c -o rdb.o -DDEBUG rdb.c
gcc -DDEBUG   -c -o rdb_clnt.o rdb_clnt.c
gcc -DDEBUG   -c -o rdb_xdr.o rdb_xdr.c
gcc -DDEBUG -o rdb rdb.o rdb_clnt.o rdb_xdr.o \

gcc -c -o rdb_svc_proc.o -DDEBUG rdb_svc_proc.c
```

```
gcc -DDEBUG   -c -o rdb_svc.o rdb_svc.c
gcc -DDEBUG -o rdb_svc rdb_svc_proc.o rdb_svc.o \
rdb_xdr.o
```

From above, the first command uses the RPC protocol compiler `rpcgen` to check the interface specification `rdb.x` and generate automatically four source code files (actually, this is the Step 4 on Page 1):

- a header file `rdb.h` which contains the interface information and must be included by both the client, `rdb.c`, and the server, `rdb_svc_proc.c`
- the client stub `rdb_clnt.c`. You can find the stub functions there.
- the server stub `rdb_svc.c`. You can find the server's `main()` there.
- an XDR file `rdb_xdr.c`. Needed for both the client and the server. So programmers do not need to handle the Little/Big byte-order conversion issue.

Then, you can see that each of the C files will be compiled separately. As stated in Step 6 and Step 7, three files `rdb.o, rdb_clnt.o, rdb_xdr.o` will be linked together to build the client `rdb`, and `rdb_svc_proc.o rdb_svc.o rdb_xdr.o` will be linked to build the server `rdb_svc`, respectively.

Now, open another terminal to login to a Linux workstation (e.g. `degas`) by way of either `spirit` or `grail`. Run the server executable first:

```
degas> cd rdb
degas> ./rdb_svc
```

Next, run the client on `spirit` and get the result back (see below).

```
spirit> ./rdb degas 2 HUA
first middle last phone location
MULAN B HUA 4777 UC486
```

The value 2 (i.e. `argv[2]`, Line 30 in `rdb.c`) in the command line tells the client to use last name (i.e. `"HUA"` as the key to search the database.

We can use the command `rpcinfo` to report RPC information. The following example shows that we probe the RPC daemon on `degas` to display a list of all registered RPC programs there. In the last entry, you can find the decimal program number `536870913` which is equal to the hex number `0x20000001` specified in `rdb.x`. Note that if some application has registered and used a program number, another application cannot register the same number at the same time. Therefore, I would suggest students replace the last 4-digit of the program number `0x20000001` in `rdb.x` with the last 4-digit of their CSU-ID before building the `rdb` application.

```
spirit> rpcinfo -p degas
   program vers proto   port  service
    100000    4   tcp    111  portmapper
    100000    3   tcp    111  portmapper
```

```
   100000    2   tcp    111  portmapper
   100000    4   udp    111  portmapper
   100000    3   udp    111  portmapper
   100000    2   udp    111  portmapper
   100007    2   udp    909  ypbind
   100007    1   udp    909  ypbind
   100007    2   tcp    910  ypbind
   100007    1   tcp    910  ypbind
536870913    1   udp  59429
536870913    1   tcp  33771
```