Exemplo de uso de RPC

Em RPC, o único vínculo explícito entre o servidor e seus clientes é a especificação da interface do serviço oferecido. Essa especificação segue um formato relativamente simples. O exemplo a seguir define um serviço composto de duas operações: ADD e SUB:

```
/* addsub.x : definição da interface */

#define PROGRAM_NUMBER 12345678
#define VERSION_NUMBER 1

/* tipo de dado que será passado aos procedimentos remotos */

struct operands
{
        int x;
        int y;
};

/* Definição da interface que será oferecida aos clientes */

program ADDSUB_PROG
{
   version ADDSUB_VERSION
   {
     int ADD (operands) = 1;
     int SUB (operands) = 2;
   }
   = VERSION_NUMBER;
}
= PROGRAM_NUMBER;
```

Essa definição de interface deve ser compilada pelo utilitário `rpcgen`, gerando diversos arquivos de código e cabeçalho que serão compilados juntamente com o cliente e o servidor:

```
espec:rpc> rpcgen -C addsub.x
espec:rpc> ll
total 28
-rw-r--r--    1 prof          842 Out 16 21:36 addsub_clnt.c
-rw-r--r--    1 prof         1113 Out 16 21:36 addsub.h
-rw-r--r--    1 prof         2360 Out 16 21:36 addsub_svc.c
-rw-r--r--    1 prof          423 Out 16 21:34 addsub.x
-rw-r--r--    1 prof          287 Out 16 21:36 addsub_xdr.c
```

A seguir devem ser escritos o cliente e o servidor que irão se comunicar usando essa especificação. Como o servidor é passivo e só aguarda, processa e responde pedidos, seu código é bastante simples:

```
#include <stdio.h>
```

```
#include "addsub.h"

/* implementação da função add */
int * add_1_svc (operands *argp, struct svc_req *rqstp)
{
   static int result;

   printf ("Recebi chamado: add %d %d\n", argp->x, argp->y);
   result = argp->x + argp->y;
   return (&result);
}

/* implementação da função sub */
int * sub_1_svc (operands *argp, struct svc_req *rqstp)
{
   static int result;

   printf ("Recebi chamado: sub %d %d\n", argp->x, argp->y);
   result = argp->x - argp->y;
   return (&result);
}
```

O código do cliente é um pouco mais complexo, pois ele precisa obter uma referência do servidor antes de submeter seus pedidos:

```
/* Cliente RPC simples */

#include <stdio.h>
#include "addsub.h"

/* função que chama a RPC add_1 */
int add (CLIENT *clnt, int x, int y)
{
   operands ops;
   int *result;

   /* junta os parâmetros em um struct */
   ops.x = x;
   ops.y = y;

   /* chama a função remota */
   result = add_1 (&ops,clnt);
   if (result == NULL)
   {
     printf ("Problemas ao chamar a função remota\n");
     exit (1);
   }

   return (*result);
```

```
}

/* função que chama a RPC sub_1 */
int sub (CLIENT *clnt, int x, int y)
{
   operands ops;
   int *result;

   /* junta os parâmetros em um struct */
   ops.x = x;
   ops.y = y;

   /* chama a função remota */
   result = sub_1 (&ops,clnt);
   if (result == NULL)
   {
      printf ("Problemas ao chamar a função remota\n");
      exit (1);
   }
   return (*result);
}

int main( int argc, char *argv[])
{
   CLIENT *clnt;
   int x,y;

   /* verifica se o cliente foi chamado corretamente */
   if (argc!=4)
   {
      fprintf (stderr,"Usage: %s hostname num1
num2\n",argv[0]);
      exit (1);
   }

   /* cria uma struct CLIENT que referencia uma interface RPC
*/
   clnt = clnt_create (argv[1], ADDSUB_PROG, ADDSUB_VERSION,
"udp");

   /* verifica se a referência foi criada */
   if (clnt == (CLIENT *) NULL)
   {
      clnt_pcreateerror (argv[1]);
      exit(1);
   }

   /* obtém os dois inteiros que serão passados via RPC */
   x = atoi (argv[2]);
```

```
    y = atoi (argv[3]);

    /* executa os procedimentos remotos */
    printf ("%d + %d = %d\n", x, y, add (clnt,x,y));
    printf ("%d - %d = %d\n", x, y, sub (clnt,x,y));

    return (0);
}
```

Em seguida, cliente e servidor podem ser compilados separadamente:

```
espec:~> cc server.c addsub_svc.c  addsub_xdr.c -o server
-lnsl
espec:~> cc client.c addsub_clnt.c addsub_xdr.c -o client
-lnsl
espec:rpc> ll
total 60
-rw-r--r--    1 prof          842 Out 16 21:36 addsub_clnt.c
-rw-r--r--    1 prof         1113 Out 16 21:36 addsub.h
-rw-r--r--    1 prof         2360 Out 16 21:36 addsub_svc.c
-rw-r--r--    1 prof          423 Out 16 21:34 addsub.x
-rw-r--r--    1 prof          287 Out 16 21:36 addsub_xdr.c
-rwxr-xr-x    1 prof        13523 Out 16 21:37 client
-rw-r--r--    1 prof         1583 Out 16 21:29 client.c
-rwxr-xr-x    1 prof        14598 Out 16 21:37 server
-rw-r--r--    1 prof          507 Out 16 21:28 server.c
```

O servidor deve ser lançado antes do cliente, de preferência em uma janela separada.