

# Práctica 2

## Programación Distribuida y Tiempo Real

Lucas Di Cunzolo  
Santiago Tettamanti

**Abstract**—RPC - Introducción.

**Index Terms**—Programación Distribuida y Tiempo Real, c, L<sup>A</sup>T<sub>E</sub>X, rpc

### 1 PUNTO 1

#### 1.1 A

Si los procedimientos fuesen locales estarían todos dentro de un mismo archivo como cualquier otro programa "común" de C. Como se muestra en C No habría latencia en la comunicación ni perdida de comunicación ya que no habría comunicación alguna con ningún host server/client (sin tener en cuenta el SO local); todas las llamadas se harían dentro del mismo espacio de direcciones.

#### 1.2 B

Capturas de las salidas de tanto los clientes como los servidores de cada uno de los casos:

Caso 1: B B

Caso 2: B No sacamos captura del server ui, no habia output en este caso

Caso 3: B B

Caso 4: B B

#### 1.3 C

Con UDP C: Vemos que al poner en el server en el medio de la comunicación un mensaje de exit C, el servidor termina bruscamente en el medio de la operación C; entonces el cliente, al terminar el timeout definido en su proceso C, termina la comunicacion, tal como se ve en esta captura de la ejecución del cliente C. Lo mismo pasa si en vez de terminar su proceso

el servidor tarda mucho en procesar lo pedido, lo que simularemos con un mensaje de sleep dentro del proceso servidor C de tal manera que el sleep tenga una duracion mayor a la definida en el timeout del cliente; entonces como el servidor no da su respuesta antes que termine el timeout, el cliente cerrará la conexión C. En este caso el servidor, a diferencia de con el mensaje exit, seguirá su ejecución esperando el próximo request ??.

Con TCP C el resultado es exactamente el mismo tanto cuando el servidor termina su ejecución en medio de la comunicación (agregando el mensaje exit) como cuando tarda más de lo debido (agregando mensaje sleep)

### 2 PUNTO 2

#### 2.1 A

El flag -N intenta de generar código en un estándar más nuevo que el ANSI (flag -C). Al intentar compilar el código con ese flag, utilizando el código original, falla. Esto se debe a que rpcgen, genera un .h las funciones con el tipo SIN puntero, a diferencia del flag -C, que los genera como punteros a operand.

#### 2.2 B

El flag -M sirve para generar código seguro para la concurrencia, utilizando un parámetro extra al servicio, de tipo int\*.

El flag -A, es flag por default, que dependiendo el sistema en el que se compila, va a ser (o no), seguro para la concurrencia multihilo.

C

falla en la misma. Esto lo podemos corroborar en el código de llamada al proceso por parte del cliente: C

### 5.3 A.2

## 3 PUNTO 3

rpcgen utiliza la estructura definida en el archivo .x para generar estructuras C en ambos puntos (cliente y servidor), con las cuales va a trabajar casteando.

El servicio recibe punteros a estas estructuras, las cuales va a trabajar y retornar nuevamente casteando a (caddr\_t), el cual es equivalente a un void\*. Esto nos permite trabajar con cualquier tipo de C, siempre volviendo a castear a la estructura definida a partir del .x

## 4 PUNTO 4

Para este punto, se implementaron las funciones básicas, write|add, read|get y list|ls

Como primera instancia, se definió nuestro archivo .x, disponible para ver en el apéndice C

## 5 PUNTO 5

### 5.1 A

### 5.2 A.1

Para el experimento del timeout usaremos el caso 1-simple. Vemos que si ponemos en el proceso servidor un delay de 25 mediante un sleep(25) en el medio de la comunicación: ?? Entonces la comunicación se realiza exitosamente y la llamada al proceso remoto termina de forma correcta: ?? ??

En cambio si ponemos en el proceso servidor un delay de 26 mediante un sleep(26) en el medio de la comunicación: ?? Entonces la comunicación se corta por parte del cliente y la llamada no se completa: ?? ??

Mediante estos resultados podemos afirmar que el timeout definido en el proceso cliente es el 25 segundos, por los que si el proceso servidor o la comunicación tardan más de ese tiempo entonces el cliente finalizará la comunicación abruptamente dando por sentado una

## APPENDIX

### .1 A

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

int add( int x, int y ) {
    return(x+y);      /* Note the use of the '+' operator to achieve addition! */
}

int subtract( int x, int y ) {
    return(x-y);      /* This is a little harder, we have to use '-' */
}

int main( int argc, char *argv[]) {

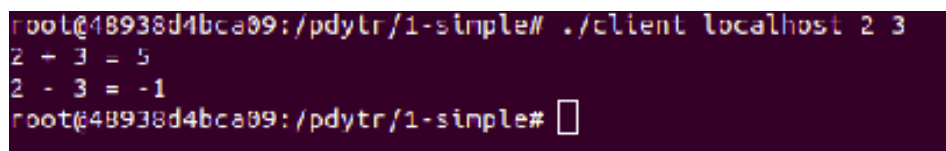
    int x,y;

    if (argc!=3) {
        fprintf(stderr,"Usage: simp num1 num\n");
        fprintf(stderr,"        num1 and num2 must be integer values (for now)\n");
        fprintf(stderr,"        Floating point arithmetic is coming soon – preregis
        fprintf(stderr,"        receive our new integrated multiplication program a
        exit(0);
    }

    x = atoi(argv[1]);
    y = atoi(argv[2]);

    printf("%d + %d = %d\n",x,y,add(x,y));
    printf("%d - %d = %d\n",x,y,subtract(x,y));
    return(0);
}
```

### .2 B



```
root@48938d4bca09:/pdytr/1-simple# ./client localhost 2 3
2 + 3 = 5
2 - 3 = -1
root@48938d4bca09:/pdytr/1-simple#
```

Fig. 1. Simple client

```

root@48938d4bca09:/pdytr/1-simple# ./server
Got request: adding 2, 3
Got request: subtracting 2, 3

```

Fig. 2. Simple server

```

root@48938d4bca09:/pdytr/2-u1# ./client localhost santi
Name santi, UID is -1
root@48938d4bca09:/pdytr/2-u1# ./client localhost root
Name root, UID is 0
root@48938d4bca09:/pdytr/2-u1# ./client localhost 1
UID 1, Name is daemon
root@48938d4bca09:/pdytr/2-u1# ./client localhost 0
UID 0, Name is root
root@48938d4bca09:/pdytr/2-u1#

```

Fig. 3. UI client

```

root@48938d4bca09:/pdytr/3-array# ./vadd_client localhost 12 15
12 + 15 = 27
root@48938d4bca09:/pdytr/3-array# ./vadd_client localhost 1 7
1 + 7 = 8
root@48938d4bca09:/pdytr/3-array#

```

Fig. 4. Array client

```

root@48938d4bca09:/pdytr/3-array# ./vadd_service
Got request: adding 2 numbers
Got request: adding 2 numbers

```

Fig. 5. Array server

```

root@48938d4bca09:/pdytr/4-list# ./client localhost 2 5
2 5
Sum is 7
root@48938d4bca09:/pdytr/4-list# ./client localhost 9878 324
9878 324
Sum is 10202
root@48938d4bca09:/pdytr/4-list#

```

Fig. 6. List client

```
root@48938d4bca09:/pdytr/4-list# ./server
```

Fig. 7. List server

### .3 C

```
/* Create a CLIENT data structure that reference the RPC
   procedure SIMP_PROG, version SIMP_VERSION running on the
   host specified by the 1st command-line arg. */

clnt = clnt_create(argv[1], SIMP_PROG, SIMP_VERSION, "udp");

/* Make sure the create worked */
if (clnt == (CLIENT *) NULL) {
    clnt_pcreateerror(argv[1]);
    exit(1);
}

/* get the 2 numbers that should be added */
x = atoi(argv[2]);
y = atoi(argv[3]);

printf("%d + %d = %d\n", x, y, add(clnt, x, y));
printf("%d - %d = %d\n", x, y, sub(clnt, x, y));
return(0);
}
```

Fig. 8. UDP client

```
int *
add_1_svc(operands *argp, struct svc_req *rqstp)
{
    static int result;

    printf("Got request: adding %d, %d\n",
           argp->x, argp->y);

    result = argp->x + argp->y;
    exit(0);

    return (&result);
}
```

Fig. 9. Server exit

```

root@48938d4bca09:/pdytr/1-simple# ./server
Got request: adding 1, 5
root@48938d4bca09:/pdytr/1-simple# █

```

Fig. 10. server trouble

```

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

int *
add_1(operands *argp, CLIENT *clnt)
{
    static int clnt_res;

    memset((char *)&clnt_res, 0, sizeof(clnt_res));
    if (clnt_call (clnt, ADD,
                  (xdrproc_t) xdr_operands, (caddr_t) argp,
                  (xdrproc_t) xdr_int, (caddr_t) &clnt_res,
                  TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&clnt_res);
}

```

Fig. 11. Client timeout

```

root@48938d4bca09:/pdytr/1-simple# ./client localhost 1 5
Trouble calling remote procedure
root@48938d4bca09:/pdytr/1-simple# █

```

Fig. 12. Client trouble

```

/* Here is the actual remote procedure */
/* The return value of this procedure must be a pointer to int! */
/* we declare the variable result as static so we can return a
   pointer to it */

int *
add_1_svc(operands *argp, struct svc_req *rqstp)
{
    static int result;

    printf("Got request: adding %d, %d\n",
           argp->x, argp->y);

    result = argp->x + argp->y;
    sleep(40);
    return (&result);
}

```

Fig. 13. Server sleep

```
root@48938d4bca09:/pdytr/1-simple# ./server
Got request: adding 3, 4
|
```

Fig. 14. Server waiting

```
/* Create a CLIENT data structure that reference the RPC
   procedure SIMP_PROG, version SIMP_VERSION running on the
   host specified by the 1st command line arg. */

clnt = clnt_create(argv[1], SIMP_PROG, SIMP_VERSION, "tcp");

/* Make sure the create worked */
if (clnt == (CLIENT *) NULL) {
    clnt_pcreateerror(argv[1]);
    exit(1);
}

/* get the 2 numbers that should be added */
x = atoi(argv[2]);
y = atoi(argv[3]);

printf("%d + %d = %d\n", x, y, add(clnt, x, y));
printf("%d - %d = %d\n", x, y, sub(clnt, x, y));
return(0);
}
```

Fig. 15. TCP client

```
gcc -c -Wall -DRPC_SVC_FG simp_clnt.c
gcc -c -Wall -DRPC_SVC_FG simp_xdr.c
simp_xdr.c: In function 'xdr_operands':
simp_xdr.c:12:20: warning: unused variable 'buf' [-Wunused-variable]
    register int32_t *buf;
                    ^
gcc -o client simpclient.o simp_clnt.o simp_xdr.o -lnsl
gcc -c -Wall -DRPC_SVC_FG simpservice.c
gcc -c -Wall -DRPC_SVC_FG simp_svc.o
gcc -o server simpservice.o simp_svc.o simp_xdr.o -lrpcsvc -lnsl
simp_svc.o: In function 'simp_prog_1':
simp_svc.c:(.text+0x153): undefined reference to 'simp_prog_1_freeresult'
collect2: error: ld returned 1 exit status
Makefile:15: recipe for target 'server' failed
make: *** [server] Error 1
```

Fig. 16. Servidor

```

int *
add_1_svc(operands *argp, struct svc_req *rqstp)
{
    static int result;

    printf("Got request: adding %d, %d\n",
           argp->x, argp->y);
    sleep(25);
    result = argp->x + argp->y;

    return (&result);
}

```

Fig. 17. Experiment timeout

```

root@48938d4bca09:~# cd /pdytr/1-simple/
root@48938d4bca09:/pdytr/1-simple# ./client localhost 1 4
1 + 4 = 5
1 - 4 = -3
root@48938d4bca09:/pdytr/1-simple# █

```

Fig. 18. Experiment timeout client1

```

root@48938d4bca09:/pdytr/1-simple# ./server
Got request: adding 1, 4
Got request: subtracting 1, 4
█

```

Fig. 19. Experiment timeout server1

```

root@48938d4bca09:/pdytr/1-simple# ./client localhost 1 4
Trouble calling remote procedure
root@48938d4bca09:/pdytr/1-simple# █

```

Fig. 20. Experiment timeout client2

```

root@48938d4bca09:/pdytr/1-simple# ./server
Got request: adding 1, 4
█

```

Fig. 21. Experiment timeout serve2