

PDyTR 2018

Informe 2

Integrantes:

Aparicio Natalia Elizabeth, Legajo 12667/7

Eusebi Cirano, Legajo 12469/2

Repository

1 Para los ejemplos de RPC proporcionados (*.tar, analizar en el orden dado a los nombres de los archivos):

a.- Mostrar cómo serían los mismos procedimientos si fueran locales, es decir haciendo el proceso inverso del realizado en la clase de explicación de RPC.

En caso de ser locales, los métodos estarían implementados todos en un mismo ejecutable, donde se llama a la rutina RPC se llamaría la rutina implementada localmente. No serían necesarios los rpcgen files, stubs ni xdr.

b.- Ejecutar los procesos y mostrar la salida obtenida (del "cliente" y del "servidor") en cada uno de los casos.

Ejemplo: **1-simple**

```
root@pdytr:/pdytr/Practicas/Practica2/Ejercicio1/1-simple# ./server
Got request: adding 10, 5
Got request: subtracting 10, 5
--
root@client:/pdytr/Practicas/Practica2/Ejercicio1/1-simple# ./client compiler 10
50
10 + 50 = 60
10 - 50 = -40
```

Ejemplo: **2-ui**

```

root@pdytr:/pdytr/Practicas/Practica2/Ejercicio1/2-u1# useradd -m -s /bin/bash
"dockercito"
root@pdytr:/pdytr/Practicas/Practica2/Ejercicio1/2-u1# ./server

--

root@client:/pdytr/Practicas/Practica2/Ejercicio1/2-u1# ./client compiler
dockercito
Name dockercito, UID is 1000

root@client:/pdytr/Practicas/Practica2/Ejercicio1/2-u1# ./client compiler root
Name root, UID is 0

root@client:/pdytr/Practicas/Practica2/Ejercicio1/2-u1# ./client compiler pdytr
Name pdytr, UID is -1

```

Ejemplo: 3-array

```

root@pdytr:/pdytr/Practicas/Practica2/Ejercicio1/3-array# ./vadd_service
Got request: adding 3 numbers
Got request: adding 4 numbers

--

root@client:/pdytr/Practicas/Practica2/Ejercicio1/3-array# ./vadd_client compiler
1 2 3
1 + 2 + 3 = 6

root@client:/pdytr/Practicas/Practica2/Ejercicio1/3-array# ./vadd_client compiler
1 2 3 4
1 + 2 + 3 + 4 = 10

```

Ejemplo: 4-list

```

root@pdytr:/pdytr/Practicas/Practica2/Ejercicio1/4-list# ./server

-
root@client:/pdytr/Practicas/Practica2/Ejercicio1/4-list# ./client compiler 1 2 5
10
1 2 5 10
Sum is 18

```

c.- Mostrar experimentos donde se produzcan errores de conectividad del lado del cliente y del lado del servidor. Si es necesario realice cambios mínimos para, por ejemplo incluir sleep() o exit(), de forma tal que no se reciban comunicaciones o no haya receptor para las comunicaciones. Verifique con UDP y con TCP.

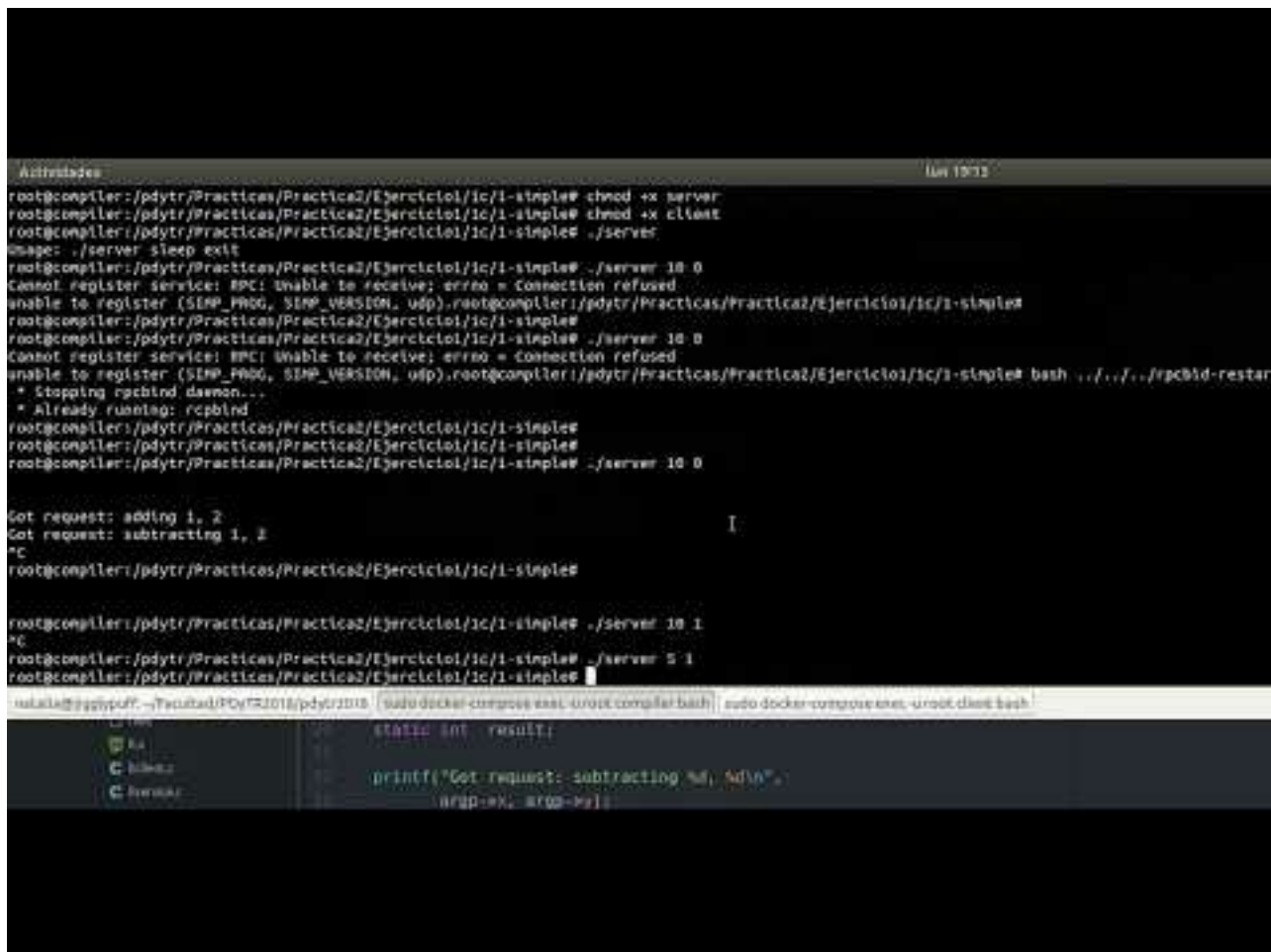
A partir de agregar sleeps al cliente no se detectan errores o interrupciones en las conexiones.

Si se agregan sleeps al servidor en cambio, verificamos que los clientes tienen un timeout, configurable a partir del método `clnt_control()` y en caso de exeder el timeout el cliente envía un mensaje de error.

Si se agregan `exit()` al servidor el cliente envía un mensaje de error y la comunicación se corta.

Tanto en TCP como en UDP se detectan los mismos resultados.

Video de muestra: [link](#)



```

root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# chmod +x server
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# chmod +x client
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server
usage: ./server sleep exit
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server 10 0
Cannot register service: RPC: Unable to receive: errno = connection refused
unable to register (SIMP_PROG, SIMP_VERSION, udp).root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server 10 0
Cannot register service: RPC: Unable to receive: errno = connection refused
unable to register (SIMP_PROG, SIMP_VERSION, udp).root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# bash .../rcpbind-restart
* Stopping rcpbind daemon...
* Already running: rcpbind
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server 10 1
^C
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server 5 1
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#
Got request: adding 1, 2
Got request: subtracting 1, 2
^C
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server 10 1
^C
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple# ./server 5 1
root@compiler:~/Practicas/Practica2/Ejercicio1/1c/1-simple#

```

2 Describir/analizar las opciones

a.- -N

Utilizado para aceptar multiples argumentos y argumentos por valor en las rutinas RPC.

b.- -M y -A

-M sirve para habilitar el modo Multithread-safe, donde el valor de retorno es pasado como argumento a la rutina del servidor en vez de utilizar variables estáticas (no thread-safe). -A habilita el modo Multithread-safe-auto que genera un thread por cada request nueva.

Lo que se debe tener en cuenta del lado del servidor es que al habilitar las funciones Multithread, se debe hacer la sincronización correspondiente para que todas las rutinas sean thread-safe.

Si se decide agregar multiples threads del lado del cliente (a travez de cualquier implementación standard ej: pthreads/openMP), es el usuario el que debe asegurar la seguridad del mismo ya que RPC no influye de ninguna manera en este flujo de ejecución.

3 Analizar la *transparencia* de RPC en cuanto al manejo de parámetros de los procedimientos remotos. Considerar lo que sucede en caso de los valores de retorno. Puede aprovechar los ejemplos provistos.

RPC se encarga de mantener al usuario abstracto de los llamados remotos. No hay diferencia entre un llamado a función local y uno remoto en cuanto al código (a excepción de la inicialización y configuración del cliente). Del mismo modo, el servidor recibe argumentos como si fueran de un llamado local, por lo que muy facilmente se pueden portar aplicaciones monolíticas en funcionamiento a RPC extrayendo las rutinas a distribuir y generando el archivo de descripción.

4 Implementar una versión muy restringida de un sistema de archivos remoto. Documente todas las decisiones tomadas.

Se definieron 3 métodos de comunicación RPC:

- `FileSize(string size)` que retorna la cantidad de bytes de un archivo.
- `Read(read_request)` que a partir de un filename, offset y size retorna la cantidad de bytes leídos y un buffer con esos bytes.
- `Write(write_request)` que da un filename, un buffer y una cantidad de bytes y retorna la cantidad de bytes correctamente escritos al final del archivo dado (crea en caso de que no exista).

Se decidió utilizar el flag -M para tener un mejor manejo de la memoria utilizada por el servidor y aceptar peticiones en concurrente.

Para la comunicación se usaron structs nativos de C con el tipo de dato XDR string<>, que es el más básico para manejo de buffers.

Lectura

Para la lectura el cliente pide el tamaño del archivo y aloca un buffer de tamaño fijo. Itera pidiendo lecturas actualizando el offset y utilizando el mismo buffer de para leer y escribir. Esto ahorra peticiones de memoria extra y permite transmitir archivos de tamaños arbitrariamente grandes. Del mismo modo, el archivo se mantiene lo más actualizado posible permitiendo (si se quisiera) desarrollar la reasunción de una descarga desde un punto.

Del lado del servidor, cada petición pide un file handler nuevo al SO en modo texto, se lee sólo el pedazo de archivo requerido y se libera el handle cuanto antes. Esto nos permite correspondientemente soportar peticiones de archivos arbitrariamente grandes. Se tuvo en cuenta mantener las peticiones de manera thread safe.

Escritura

El cliente abre un archivo, obtiene su tamaño y aloca un buffer único de tamaño determinado. Itera leyendo de disco y enviando a red desde el mismo buffer actualizando el offset a partir de la respuesta satisfactoria del servidor.

El servidor acepta los bytes, busca el archivo (o lo crea en caso de no existir) y agrega los bytes al mismo. En caso de peticiones concurrentes el SO deniega el acceso al archivo.

Todos los archivos leídos por el cliente reciben el prefijo `client`.

Todos los archivos escritos por el servidor reciben el prefijo `server`.

5 Timeouts en RPC

a. Desarrollar un experimento que muestre el timeout definido para las llamadas RPC y el promedio de tiempo de una llamada RPC.

En la carpeta del ejercicio5 hemos programado un cliente y un servidor muy sencillos para probarlo. Leyendo la documentación de RPC, sabemos que el timeout por defecto que usa es de 25 segundos.

Promedio de tiempo de una llamada RPC

```
# 0
CLIENT: to_prog_1 start
Send time 0.110507 ms
CLIENT: to_prog_1 finish

# 1
CLIENT: to_prog_1 start
Send time 0.087976 ms
CLIENT: to_prog_1 finish

# 2
CLIENT: to_prog_1 start
Send time 0.070453 ms
CLIENT: to_prog_1 finish

# 3
CLIENT: to_prog_1 start
Send time 0.066042 ms
CLIENT: to_prog_1 finish

# 4
CLIENT: to_prog_1 start
Send time 0.045419 ms
CLIENT: to_prog_1 finish

# 5
CLIENT: to_prog_1 start
Send time 0.042915 ms
CLIENT: to_prog_1 finish

# 6
CLIENT: to_prog_1 start
Send time 0.044942 ms
CLIENT: to_prog_1 finish

# 7
CLIENT: to_prog_1 start
Send time 0.042081 ms
CLIENT: to_prog_1 finish

# 8
CLIENT: to_prog_1 start
Send time 0.044465 ms
CLIENT: to_prog_1 finish
```

```
# 9
CLIENT: to_prog_1 start
Send time 0.041008 ms
CLIENT: to_prog_1 finish

Average time 0.059581 ms

CLIENT: some_function_to_22_1 start
CLIENT: some_function_to_22_1 finish
timeout_22: Send time 22000.349998 ms

CLIENT: some_function_to_26_1 start
call failed - should be timeout: RPC: Timed out

CLIENT: some_function_to_26_1 finish
some_function_to_26_1: Send time 27005.294085 ms
```

b. Reducir el timeout de las llamadas RPC a un 10% menos del promedio encontrado anteriormente.

Seteando el timeout del servidor en 22 segundo, funciona correctamente sin devolver ningún timeout.

c. Desarrollar un cliente/servidor RPC de forma tal que siempre se supere el tiempo de timeout. Una forma sencilla puede utilizar el tiempo de timeout como parámetro del procedimiento remoto, donde se lo utiliza del lado del servidor en una llamada a sleep(), por ejemplo.

La tercer función declarada en el timeout.x (some_function_to_26) siempre que se llame, dará timeout.