**Tecnologías de la Información.**

**Redes, Comunicaciones y Arquitecturas Distribuidas.**

**AÑO 2010**

**Trabajo Práctico N° 2**

**Programación de Clientes y Servidores utilizando RPC en LINUX**

**Profesor:**

* MSc. Pablo Pessolani

**Integrantes:**

* Lorena Diorio

[lorenadiorio@gmail.com](mailto:lorenadiorio@gmail.com)

* Ariel Rossanigo

[arielrossanigo@gmail.com](mailto:arielrossanigo@gmail.com)

* Román Zenobi

[rozenobi@hotmail.com](mailto:rozenobi@hotmail.com)

**Desarrollo del Práctico**

***Configuración de rpc (pingRPC.x)***

program DATE\_PROG {

version DATE\_VERS {

string ping(string) = 1; /\* procedure number = 1 \*/

} = 1; /\* version number = 1 \*/

} = 0x31234567; /\* program number = 0x31234567 \*/

***Servidor RPC (servidorRPC.c)***

#include "pingRPC.h"

#include "utiles\_ping.c"

**char** \*\*

ping\_1\_svc(**char** \*\*argp, **struct** svc\_req \*rqstp)

{

**static** **char** \* result;

printf("Se recibio peticion en el puerto %d, desde la IP: %s\n",

(svc\_getcaller(rqstp->rq\_xprt))->sin\_port,

inet\_ntoa((svc\_getcaller(rqstp->rq\_xprt))->sin\_addr));

printf("TS: %s, Datos:%s \n", str\_hora\_actual(), \*argp);

result= \*argp;

**return** &result;

}

***Cliente RPC (clienteRPC.c)***

#include "pingRPC.h"

#include "utiles\_ping.c"

void date\_prog\_1(char \*host, int tamanio, int repeticiones)

{

CLIENT \*clnt;

char \* \*result\_1;

char \*msg = generar\_paquete(tamanio);

clnt = clnt\_create (host, DATE\_PROG, DATE\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

int i;

for (i=0; i< repeticiones;i++)

{

iniciar\_temporizador();

result\_1 = ping\_1(&msg, clnt);

if (result\_1 == (char \*\*) NULL) {clnt\_perror (clnt, "call failed");}

printf("RTT paquete %d: %d uSeg\n", i+1,finalizar\_temporizador());

}

imprimir\_resultados\_ping(host);

clnt\_destroy (clnt);

}

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

{

char \*ip\_server;

int puerto;

int repeticiones = 5;

int tamanio=100;

if (parsear\_parametros\_cliente(argc, argv, &ip\_server, &puerto, &repeticiones,

&tamanio) < 0)

{

return -1;

}

date\_prog\_1 (ip\_server, tamanio, repeticiones);

exit (0);

}

***PING RPC (PingRpc\_svc.c)***

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include "pingRPC.h"

#include <stdio.h>

#include <stdlib.h>

#include <rpc/pmap\_clnt.h>

#include <string.h>

#include <memory.h>

#include <sys/socket.h>

#include <netinet/in.h>

#ifndef SIG\_PF

#define SIG\_PF void(\*)(int)

#endif

static void

date\_prog\_1(struct svc\_req \*rqstp, register SVCXPRT \*transp)

{

union {

char \*ping\_1\_arg;

} argument;

char \*result;

xdrproc\_t \_xdr\_argument, \_xdr\_result;

char \*(\*local)(char \*, struct svc\_req \*);

switch (rqstp->rq\_proc) {

case NULLPROC:

(void) svc\_sendreply (transp, (xdrproc\_t) xdr\_void, (char \*)NULL);

return;

case ping:

\_xdr\_argument = (xdrproc\_t) xdr\_wrapstring;

\_xdr\_result = (xdrproc\_t) xdr\_wrapstring;

local = (char \*(\*)(char \*, struct svc\_req \*)) ping\_1\_svc;

break;

default:

svcerr\_noproc (transp);

return;

}

memset ((char \*)&argument, 0, sizeof (argument));

if (!svc\_getargs (transp, (xdrproc\_t) \_xdr\_argument, (caddr\_t) &argument)) {

svcerr\_decode (transp);

return;

}

result = (\*local)((char \*)&argument, rqstp);

if (result != NULL && !svc\_sendreply(transp, (xdrproc\_t) \_xdr\_result, result)) {

svcerr\_systemerr (transp);

}

if (!svc\_freeargs (transp, (xdrproc\_t) \_xdr\_argument, (caddr\_t) &argument)) {

fprintf (stderr, "%s", "unable to free arguments");

exit (1);

}

return;

}

int

main (int argc, char \*\*argv)

{

register SVCXPRT \*transp;

pmap\_unset (DATE\_PROG, DATE\_VERS);

transp = svcudp\_create(RPC\_ANYSOCK);

if (transp == NULL) {

fprintf (stderr, "%s", "cannot create udp service.");

exit(1);

}

if (!svc\_register(transp, DATE\_PROG, DATE\_VERS, date\_prog\_1, IPPROTO\_UDP)) {

fprintf (stderr, "%s", "unable to register (DATE\_PROG, DATE\_VERS, udp).");

exit(1);

}

transp = svctcp\_create(RPC\_ANYSOCK, 0, 0);

if (transp == NULL) {

fprintf (stderr, "%s", "cannot create tcp service.");

exit(1);

}

if (!svc\_register(transp, DATE\_PROG, DATE\_VERS, date\_prog\_1, IPPROTO\_TCP)) {

fprintf (stderr, "%s", "unable to register (DATE\_PROG, DATE\_VERS, tcp).");

exit(1);

}

svc\_run ();

fprintf (stderr, "%s", "svc\_run returned");

exit (1);

/\* NOTREACHED \*/

}

***PING RPC (PingRpc\_cInt.c)***

/\*

\* Please do not edit this file.

\* It was generated using rpcgen.

\*/

#include <memory.h> /\* for memset \*/

#include "pingRPC.h"

/\* Default timeout can be changed using clnt\_control() \*/

static struct timeval TIMEOUT = { 25, 0 };

char \*\*

ping\_1(char \*\*argp, CLIENT \*clnt)

{

static char \*clnt\_res;

memset((char \*)&clnt\_res, 0, sizeof(clnt\_res));

if (clnt\_call (clnt, ping,

(xdrproc\_t) xdr\_wrapstring, (caddr\_t) argp,

(xdrproc\_t) xdr\_wrapstring, (caddr\_t) &clnt\_res,

TIMEOUT) != RPC\_SUCCESS) {

return (NULL);

}

return (&clnt\_res);

}

***Utilies PING (Utiles\_Ping.c)***

#include <stdio.h>

#include <sys/time.h>

#include <time.h>

#include <unistd.h>

//completa puerto de acuerdo a los parametros de entrada

int parsear\_parametros\_servidor(int argc, char\* argv[], int \*puerto)

{

int c;

while ((c= getopt(argc, argv, "p::")) != -1)

{

switch(c)

{

case 'p':

{

\*puerto = atoi(optarg);

if (\*puerto <1024 || \*puerto > 65535)

{

printf ("El puerto debe estar comprendido entre 1024 y 65535\n");

return -1;

}

break;

}

}

}

return 0;

}

//completa IP, puerto, repeticiones y tamanio de acuerdo a los parametros de entrada

int parsear\_parametros\_cliente(int argc, char\* argv[], char \*\*ip, int \*puerto, int \*repeticiones, int \*tamanio)

{

if (argc < 2)

{

printf ("Error. Uso: %s <IPservidor> [-p <puerto>] [-r <repeticiones>] [-s <tamaño>]\n", argv[0]);

return -1;

}

\*ip= argv[1];

int c;

while ((c= getopt(argc, argv, "r::p::s::")) != -1)

{

switch(c)

{

case 'r':

{

\*repeticiones = atoi(optarg);

if (\*repeticiones < 1 || \*repeticiones > 101)

{

printf ("El nro de repeticiones debe estar comprendido entre 1 y 101\n");

return -1;

}

break;

}

case 's':

{

\*tamanio = atoi(optarg);

if (\*tamanio <1 || \*tamanio > 10001)

{

printf ("El tamano debe estar comprendido entre 1 y 10001\n");

return -1;

}

break;

}

case 'p':

{

\*puerto = atoi(optarg);

if (\*puerto <1024 || \*puerto > 65535)

{

printf ("El puerto debe estar comprendido entre 1024 y 65535\n");

return -1;

}

break;

}

}

}

return 0;

}

//Genera un string con los digitos del 0 al 9, de tamanio caracteres

char\* generar\_paquete(int tamanio)

{

char\* res = malloc((tamanio+1)\*sizeof(char));

int i;

for (i = 0; i<tamanio; i++)

{

res[i] = (char)(i%10 + 48);

}

res[tamanio]='\0';

return res;

}

//Variables utilizadas en los temporizadores

struct timeval start;

struct timeval stop;

struct timezone tz;

//Calcula la diferencia entre dos valores de tiempo y la almacena en out

void tvsub( out, in )

struct timeval \*out, \*in;

{

if( (out->tv\_usec -= in->tv\_usec) < 0 )

{

out->tv\_sec--;

out->tv\_usec += 1000000;

}

out->tv\_sec -= in->tv\_sec;

}

struct ResultadosTemporizador

{

int Minimo;

int Maximo;

int Total;

int Cantidad;

float Promedio;

};

//variable para llevar los resultados acumulados del todas las mediciones

struct ResultadosTemporizador res;

//inicializamos res

void iniciar\_resultados\_temporizador()

{

res.Minimo = 0;

res.Maximo = -1;

res.Cantidad= 0;

res.Total = 0;

}

//Iniciamos una medicion, colocamos en start la hora del dia

void iniciar\_temporizador()

{

gettimeofday( &start, &tz );

}

//finalizamos una medicion, calculamos la diferencia entre la hora del dia y start

//actualizamos los valores acumulados

int finalizar\_temporizador()

{

int tiempo;

gettimeofday( &stop, &tz );

tvsub( &stop, &start );

tiempo = stop.tv\_sec \* 1000000 + stop.tv\_usec;

if (res.Minimo > tiempo || res.Cantidad==0) res.Minimo = tiempo;

if (res.Maximo < tiempo || res.Cantidad==0) res.Maximo = tiempo;

res.Total += tiempo;

res.Cantidad++;

return tiempo;

}

//Obtenemos los resultados acumulados de todas las mediciones

struct ResultadosTemporizador obtener\_resultados\_temporizador()

{

if (res.Cantidad!= 0)

res.Promedio = (float)res.Total/res.Cantidad;

else

res.Promedio = 0;

return res;

}

//Imprimimos los resultados acumulados

void imprimir\_resultados\_ping(char \*IP)

{

struct ResultadosTemporizador r= obtener\_resultados\_temporizador();

printf("========================================================\n");

printf("Ping a: %s\n", IP);

printf("RTT min: %d uSeg, max: %d uSeg, prom: %.2f uSeg\n", r.Minimo, r.Maximo, r.Promedio);

printf("========================================================\n");

}

char \* str\_hora\_actual()

{

struct tm \*ptr;

time\_t lt;

//obtenemos tiempo actual

lt = time(NULL);

//obtenemos tiempo local

ptr = localtime(&lt);

//obtenemos un string formateado con el tiempo local

char \* res= malloc(9\*sizeof(char));

strftime(res, 9, "%H:%M:%S", ptr);

return res;

}

***Makefile***

TP2:

gcc -o ../obj/rpcping ../src/clienteRPC.c ../src/pingRPC\_clnt.c

gcc -o ../obj/rpcpingd ../src/servidorRPC.c ../src/pingRPC\_svc.c

clean:

rm ../obj/\*

copy:

if test -d /home/sod/2010/tp2/Zenobi-DIorio-Rossanigo;

then echo "el path ya existe";

else mkdir /home/sod/2010/tp2/Zenobi-DIorio-Rossanigo;

fi

cp -R -f -u ../\* /home/sod/2010/tp2/Zenobi-DIorio-Rossanigo

***Mediciones***

|  |  |
| --- | --- |
| Tamaño | RTT Promedio (uSeg) |
| 1 | 85 |
| 10 | 72 |
| 100 | 86 |
| 300 | 83 |
| 500 | 204 |
| 1000 | 255 |
| 3000 | 2838 |
| 5000 | 2773 |
| 8500 | 3757 |

***Descargos.***

En las mediciones no pudimos utilizar paquetes de 10.000 caracteres porque RPC no podía codificar los argumentos. Este valor lo reemplazamos por 8.500 caracteres.

El error devuelto por RPC era: ***call failed: RPC: Can't encode arguments.***