**Tecnologías de la Información.**

**Redes, Comunicaciones y Arquitecturas**

**Distribuidas.**

**AÑO 2010**

**Trabajo Práctico N° 1**

**Programación de Clientes y Servidores utilizando SOCKETs 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**

***Cliente PING TCP (ClienteTcp.c)***

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include "utiles\_ping.c"

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

{

char \*ip\_server;

int puerto = 2222;

int repeticiones = 5;

int tamanio=100;

iniciar\_resultados\_temporizador();

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

&tamanio) < 0)

{

return -1;

}

int socket\_tcp;

struct sockaddr\_in direccion;

//generamos IP del server

direccion.sin\_family = AF\_INET;

direccion.sin\_port = htons(puerto);

direccion.sin\_addr.s\_addr = inet\_addr(ip\_server);

//generamos el paquete a enviar

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

int l\_msg= strlen(msg);

//creamos buffer de lectura

char \*lee;

lee= malloc(tamanio \* sizeof(char));

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

int i;

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

{

//abrimos un socket

socket\_tcp = socket (AF\_INET, SOCK\_STREAM, 0);

if (socket\_tcp == -1)

{

printf("Error, no se pudo crear el socket\n");

return -1;

}

//establecemos la conexion

if (connect (socket\_tcp, (struct sockaddr \*) &direccion, sizeof (direccion)) == -1)

{

printf("Error, no se pudo conectar con el server\n");

return -1;

}

//iniciamos medicion

iniciar\_temporizador();

//escribimos el msg y esperamos la respuesta

write(socket\_tcp, msg, l\_msg);

read(socket\_tcp, lee, tamanio);

//finalizamos la medicion e imprimimos el RTT

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

close(socket\_tcp);

}

//imprimimos resultados acumulados

imprimir\_resultados\_ping(ip\_server);

return 0;

}

***Servidor PING TCP (ServidorTcp.c)***

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include "utiles\_ping.c"

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

{

int puerto= 2222;

if (parsear\_parametros\_servidor(argc, argv, &puerto)<0) return -1;

int socket\_tcp;

struct sockaddr\_in direccion;

direccion.sin\_family = AF\_INET;

direccion.sin\_port = htons(puerto);

direccion.sin\_addr.s\_addr = INADDR\_ANY;

//abrimos un socket

socket\_tcp = socket (AF\_INET, SOCK\_STREAM, 0);

if (socket\_tcp == -1)

{

printf("Error, no se pudo crear el socket\n");

return -1;

}

if (bind (socket\_tcp, (struct sockaddr \*)&direccion, sizeof (direccion)) == -1)

{

close (socket\_tcp);

printf("Error, no funciono el bind\n");

return -1;

}

// Se avisa al sistema que comience a atender llamadas de clientes

if (listen (socket\_tcp, 1) == -1)

{

close (socket\_tcp);

printf("Error, no pudo escuchar\n");

return -1;

}

struct sockaddr\_in cliente;

int socket\_retorno;

socklen\_t largo = sizeof(struct sockaddr\_in);

printf("Servidor a la espera de paquetes. Escuchando en puerto %d\n", puerto);

while(1)

{

//llamamos al aceptar para esperar una coneccion

socket\_retorno= accept(socket\_tcp, (struct sockaddr \*) &cliente, &largo);

if (socket\_retorno == -1)

{

close(socket\_tcp);

printf("Error en el aceptar\n");

return -1;

}

char \*leido;

int leido\_l;

leido = malloc(10002\*sizeof(char));

leido\_l= read(socket\_retorno, leido, 10002);

printf("Cliente: %s, TS: %s, Datos: %s \n", inet\_ntoa( cliente.sin\_addr),

str\_hora\_actual(), leido);

write(socket\_retorno, leido, leido\_l);

close(socket\_retorno);

}

close(socket\_tcp);

return 0;

}

***Cliente PING UDP (ClienteUdp.c)***

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include "utiles\_ping.c"

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

{

char \*ip\_server;

int puerto = 2222;

int repeticiones = 5;

int tamanio=100;

iniciar\_resultados\_temporizador();

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

&tamanio) < 0)

{

return -1;

}

int socket\_udp;

struct sockaddr\_in direccion;

//generamos IP del server

direccion.sin\_family = AF\_INET;

direccion.sin\_port = htons(puerto);

direccion.sin\_addr.s\_addr = inet\_addr(ip\_server);

//generamos el paquete a enviar

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

int l\_msg= strlen(msg);

//creamos buffer de lectura

char \*lee;

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

int i;

int longitud\_direccion = sizeof(direccion);

int bytes\_leidos;

//abrimos un socket

socket\_udp = socket (AF\_INET, SOCK\_DGRAM, 0);

if (socket\_udp == -1)

{

printf("Error, no se pudo crear el socket\n");

return -1;

}

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

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

{

//iniciamos medicion

iniciar\_temporizador();

//escribimos el msg y esperamos la respuesta

sendto (socket\_udp, msg, l\_msg+1, 0, (struct sockaddr\*) &direccion,

longitud\_direccion);

bytes\_leidos= recvfrom(socket\_udp, lee, 10002, 0, (struct sockaddr\*) &direccion,

&longitud\_direccion);

if (bytes\_leidos == -1)

{

close(socket\_udp);

printf("Error en la lectura\n");

return -1;

}

//finalizamos la medicion e imprimimos el RTT

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

}

close(socket\_udp);

//imprimimos resultados acumulados

imprimir\_resultados\_ping(ip\_server);

return 0;

}

***Servidor PING UDP (ClienteUdp.c)***

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include "utiles\_ping.c"

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

{

int puerto= 2222;

if (parsear\_parametros\_servidor(argc, argv, &puerto)<0)

return -1;

int socket\_udp;

struct sockaddr\_in direccion;

direccion.sin\_family = AF\_INET;

direccion.sin\_port = htons(puerto);

direccion.sin\_addr.s\_addr = INADDR\_ANY;

socket\_udp = socket (AF\_INET, SOCK\_DGRAM, 0);

if (socket\_udp == -1)

{

printf("Error, no se pudo crear el socket\n");

return -1;

}

if (bind (socket\_udp, (struct sockaddr \*)&direccion, sizeof (direccion)) == -1)

{

close (socket\_udp);

printf("Error, no funciono el bind\n");

return -1;

}

//direccion del cliente

struct sockaddr\_in cliente;

int bytes\_leidos;

int longitud\_cliente = sizeof (cliente);

printf("Servidor a la espera de paquetes. Escuchando en puerto %d\n", puerto);

//buffer para la lectura

char \*leido;

leido = malloc(10002\*sizeof(char));

int leido\_l;

while(1)

{

bytes\_leidos= recvfrom(socket\_udp, leido, 10002, 0, (struct sockaddr\*) &cliente, &longitud\_cliente);

if (bytes\_leidos == -1)

{

close(socket\_udp);

printf("Error en la lectura\n");

return -1;

}

printf("Cliente: %s, TS: %s, Datos: %s \n", inet\_ntoa( cliente.sin\_addr), str\_hora\_actual(), leido);

sendto(socket\_udp, leido, bytes\_leidos, 0, (struct sockaddr\*) &cliente, longitud\_cliente);

}

close(socket\_udp);

return 0;

}

[***Makefile***](http://code.google.com/p/pracmaestriaemisi/source/browse/trunk/Sistemas%20distribuidos/guia1/src/Makefile)

TP1:

gcc -o ../obj/tcpping ../src/ClienteTcp.c

gcc -o ../obj/tcppingd ../src/ServidorTcp.c

gcc -o ../obj/udppingd ../src/ServidorUdp.c

gcc -o ../obj/udpping ../src/ClienteUdp.c

clean:

rm ../obj/\*

copy:

if test -d /home/sod/2010/tp1/Zenobi-DIorio-Rossanigo;

then echo "el path ya existe";

else mkdir /home/sod/2010/tp1/Zenobi-DIorio-Rossanigo;

fi

cp -R -f -u ../\* /home/sod/2010/tp1/Zenobi-DIorio-Rossanigo

[***utiles\_ping.c***](http://code.google.com/p/pracmaestriaemisi/source/browse/trunk/Sistemas%20distribuidos/guia1/src/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;

}