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

- Ariel Rossanigo
arielrossanigo@gmail.com

- Román Zenobi
rozenobi@hotmail.com

Desarrollo del Práctico

Cliente PING TCP (ClienteTcp.c)

```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)*

```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

```
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

```
#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;
}
```