Sistemas Distribuidos 2013

Trabajo Práctico de Sockets

El siguiente trabajo práctico tiene por objetivo:

1) Introducirlos en la comunicación entre procesos en Sistemas Distribuidos. Dichos procesos se ejecutan en distintos sistemas operativos que se están ejecutando en distintos nodos de la red.

2) Destacar la importancia de la tecnología de Sockets para la comunicación entre procesos distribuidos mediante el paso de mensajes.

Comencemos

Primer Ejemplo

Aquí tenemos un Mini Servidor de Comandos Linux Debian y Un Mini Cliente Telnet Windows XP para el Mini Servidor.
Es muy simple, el Cliente Mini Telnet envía un comando al Servidor, el Servidor lo ejecuta y le devuelve la salida del comando al Cliente, el cliente lo muestra por pantalla.
El alumno tiene que implementar el ejemplo y modificarlo para convertirlo en un servidor concurrente, usando pid_t fork(void) creando Servidores hijos como procesos pesados, también debe obtener una segunda versión del mismo Servidor modificándolo para convertirlo en un servidor concurrente usando pthread_t pthread_create(…,…,…,…) obteniendo Servidores hijos como procesos livianos.

Servidor en Linux Debian

```
/*
   Mini servidor comandos Debian
   Hay que convertirlo en servidor concurrente con hijos pesados (fork) version 1
   y con hijos livianos (threads) version 2
*/
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>

#define IP "192.168.1.3"
#define PUERTO 6666

void UbicacionDelCliente(struct sockaddr_in);

main(int argc, char *argv[])
{
  struct sockaddr_in s_sock,c_sock;
```

```
 int idsocks,idsockc;
 int lensock = sizeof(struct sockaddr_in);
 idsocks = socket(AF_INET, SOCK_STREAM, 0);
 printf("idsocks %d\n",idsocks);
 s_sock.sin_family      = AF_INET;
 s_sock.sin_port        = htons(PUERTO);
 s_sock.sin_addr.s_addr = inet_addr(IP);
 memset(s_sock.sin_zero,0,8);
 printf("bind %d\n", bind(idsocks,(struct sockaddr *) &s_sock,lensock));
 printf("listen %d\n",listen(idsocks,5));
 while(1)
   {
    printf("esperando conexion\n");
    idsockc = accept(idsocks,(struct sockaddr *)&c_sock,&lensock);
    if(idsockc != -1)
       {
        /* Ubicacion del Cliente */
         printf("conexion aceptada desde el cliente\n");
         UbicacionDelCliente(c_sock);
        /*-------------------------------------------------*/
        char buf[30];
        int nb;
        nb=read(idsockc,buf,30);
        buf[nb]='\0';
        printf(".......recibido del cliente %d : %s\n",idsockc,buf);
        int defout = dup(1);
        dup2(idsockc,1);
        system(buf);
        dup2(defout,1);
        close(idsockc);
        close(defout);
       }
    else
       {
         printf("conexion rechazada %d \n",idsockc);
       }
   }
}

void UbicacionDelCliente(struct sockaddr_in c_sock)
{
 printf(".....c_sock.sin_family %d\n",c_sock.sin_family);
 printf(".....c_sock.sin_port %d\n",c_sock.sin_port);
 printf(".....c_sock.sin_addr.s_addr %s\n\n", inet_ntoa(c_sock.sin_addr));
}
```

Cliente en Windows XP

```
/*
 Cliente mini Telnet de un servidor de comandos Debian
 Hay que agregarle el ingreso de un usuario y contraseña
 para que sea validado por el servidor.
*/
#include <windows.h>
#include <winsock2.h> //Cabecera para los socket windows
#include <stdio.h>
#include <stdlib.h>

void UbicacionDelServidor(struct in_addr);

int main ()
{
   WSADATA wsa; //Variable inicialización
   SOCKET dccliente; //Contiene el descriptor de socket cliente
   struct sockaddr_in remoto; ////Contiene los datos de conexion del equipo remoto
   WSAStartup (MAKEWORD(2,0),&wsa); //Inicializamos winsock
   dccliente = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);
   if (dccliente == -1 )
    {
        printf ("socket fallo----\n");
        exit(0);
    }
   //Rellenamos la estructura sockaddr_in remoto
   remoto.sin_family = AF_INET; //Para6ckets en entorno windows
   remoto.sin_port = htons (6666); // Rellenamos el puerto
   remoto.sin_addr.s_addr = inet_addr("192.168.1.3");
   memset(remoto.sin_zero,0,8); //Completamos la estructura con ceros
   //Establecemos conexion
   if (connect(dccliente, (SOCKADDR *)&remoto, sizeof(SOCKADDR)) == -1)
   {
      printf ("Conexion fallo...\n");
      exit(0);
   }
   //--------------------------------------
   printf ("Conexion establecida con exito...Datos del Servidor...\n");
   UbicacionDelServidor(remoto.sin_addr);
   //---------------------------
   printf("Ingrese comando Linux ");
   char bufin[512];
   scanf("%s",bufin);
   if (send(dccliente, bufin,strlen(bufin), 0) == -1)
   {
      printf("send fallo...\n");
      exit(0);
   }
```

```
   char buf[4096];
   if (recv(dccliente, buf,4095, 0) == -1)
   {
      printf("recv fallo...\n");
      exit(0);
   }
   else
   {
      printf("%s\n",buf);
   }
   system ("PAUSE");
   return (0);
}

void UbicacionDelServidor(struct in_addr addr )
{
  struct hostent *remoteHost;
  char *host_name;
  char **pAlias;

  remoteHost = gethostbyaddr((char *)&addr, 4, AF_INET);

  if (remoteHost == NULL)
    printf("Fallo gethostbyaddr\n");
  else
   {
     printf("Nombre Servidor: %s\n", remoteHost->h_name);
     for (pAlias = remoteHost->h_aliases; *pAlias != 0; pAlias++)
      {
         printf("Nombre Alternativo: %s\n", *pAlias);
      }
      printf("Tipo Address: ");
      switch (remoteHost->h_addrtype) {
      case AF_INET:
          printf("AF_INET\n");
          break;
      case AF_INET6:
          printf("AF_INET6\n");
          break;
      case AF_NETBIOS:
          printf("AF_NETBIOS\n");
          break;
      default:
          printf(" %d\n", remoteHost->h_addrtype);
          break;
       }
       printf("Bytes Address: %d\n", remoteHost->h_length);

       if (remoteHost->h_addrtype == AF_INET)
       {
```

```
            int i = 0 ;
            while (remoteHost->h_addr_list[i] != 0)
             {
                addr.s_addr = *(u_long *) remoteHost->h_addr_list[i++];
                printf("IPv4 Address #%d: %s\n", i, inet_ntoa(addr));
            }
        }
        else
        if (remoteHost->h_addrtype == AF_INET6) printf("IPv6 address\n");
    }
}
```

Segundo Ejemplo

Aquí tenemos un Mini Cliente Apache ejecutándose en un SO Linux Debian que hace conexión con un Servidor Apache corriendo en un SO Windows XP.
Es muy simple, el Cliente le pide un archivo al Servidor Apache, y este le envía el archivo al Cliente, y el Cliente lo muestra por pantalla.
El alumno tiene que modificar el Cliente para que decodifique el archivo recibido quitando todo lo que corresponde al código html.

Cliente que se ejecuta en un SO Linux Debian

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

#define IP "192.168.1.2"
#define PUERTO 80

int main (int argc, char *argv[])
 {
   char file_name[256] = "index.html\0";
   char buf[8192];
   char message[256];
   int sd;
   struct sockaddr_in pin;
   pin.sin_family = AF_INET;
   pin.sin_addr.s_addr = inet_addr(IP);
   pin.sin_port = htons(PUERTO);
   bzero(&pin.sin_zero, sizeof(pin.sin_zero));
   if ((sd=socket(AF_INET,SOCK_STREAM,0)) == -1)
       {
                   printf("Error al abrir el socket\n");
                   exit(1);
     }
    if(connect(sd, (void *)&pin,sizeof(pin)) == -1)
       {
                   printf("Error al conectar el socket\n");
                   exit(1);
      }
     sprintf(message, "GET /%s \n",file_name);
     if(send(sd,message,strlen(message),0) == -1)
       {
                   printf("Error al enviar");
                   exit(1);
```

```
        }
    printf("mensaje %s enviado al servidor apache...\n",message);
    if(recv(sd,buf,8192,0) == -1)
    {
            printf("Error al recibir la repuesta..\n");
            exit(1);
        }
        printf("Respuesta del servidor: \n%s\n",buf);
        close(sd);
        return EXIT_SUCCESS;
}
```