Pràctica 7: Programació d'un servidor TCP.

Autor: Enrique Bonet

Objectiu i descripció general.

L'objectiu d'esta pràctica és la programació d'un servidor amb el protocol de transport TCP. Per tal de fer això, es proposa el desenrotllament d'una versió simplificada d'un servidor d'IRC.

L'Internet Relay Xat és un protocol d'aplicació que permet establir converses entre grups d'usuaris en Internet, tenint les possibilitats de distints grups d'usuaris, conversacions privades, existència de moderadors en els grups, etc. L'IRC es va definir originàriament en el RFC 1459, podent trobar-se les actuals especificacions en els RFC 2810, 2811, 2812 i 2813.

Desenrotllament de la pràctica.

Desenrotllar un programa en C/C++ que implemente una versió molt simplificada d'un servidor d'IRC. El servidor haurà de respondre a la següent línia de comandaments:

servidor <port>

On <*port*> indica el port on es trobarà instal·lat el servidor.

L'anàlisi bàsica de desenrotllament del programa és:

1. Creeu un socket, associar-lo en escolta al port desitjat, posar-lo en mode bloquejat i inicialitzar la cua d'espera de peticions.
2. Comproveu si hi ha alguna petició de connexió al socket i, en cas afirmatiu, acceptar-la en una cua de clients connectats. Perquè el programa no es bloquege esperant una connexió, heu d'utilitzar la funció *select()* amb un timeout d'1 milisegon, per exemple, perquè la petició de connexió es rep com un senyal de dades disponibles per a lectura en el socket.
3. Comproveu si podeu llegir dades d'algun dels clients connectats, utilitzant, per a no bloquejar l'execució del programa, la funció *select()* amb el timeout del punt anterior. En cas afirmatiu, llegir les dades de cadascun dels clients.
    a. Si les dades llegides són d'una grandària major que 0 bytes, hem realitzat una lectura correcta.
    b. No obstant això, si llegim 0 bytes de dades, a pesar que la crida a la funció *select()* indicava que teníem dades a llegir, açò indica que el client ha tancat la connexió. En este cas hem de tancar eixa connexió i eliminar el client de la llista de clients connectats.

4. Si hem llegides dades d'algun client, hem d'escriure-les en tots els clients que tenim connectats. Per a això, i novament utilitzant la funció *select()*, comprovem si podem escriure en un client, i en cas afirmatiu, escrivim les dades[1].
5. Tornem al punt 2.

Com a ajuda se vos proporciona la següent funció *main()*[2]:

```
int main(int argc,char *argv[])
{
  char buffer[GRAND_CUA][GRAND_TEXT + 1];
  int sock, cua[GRAND_CUA], num, cont;

  if (argc!=2)
    Us(argv[0]);

  sock=CrearSocket((unsigned short)atoi(argv[1])); /* Creació del socket. */

  num=0;
  while (EnterPolsat()==0) /* Mentre no polsem la tecla [Enter]. */
  {
    num=AcceptarConexio(sock, cua, num);          /* Comprovar si hi ha
                                                     connexions noves. */

    if ((cont=Llegir(sock, cua, &num, buffer))>0) /* Comprovar si tenim dades
                                                     per a llegir. */
      Escriure(sock, cua, num, buffer, cont);     /* Enviament de les dades
                                                     llegides. */
  }

  TancarSockets(sock, cua, num); /* Tanquem totes les connexions. */
  return 0;
}
```

La variable *sock* de les funcions *Llegir()* i *Escriure()* només es fa servir per a poder tancar eixe socket si es produeix un error.

Com pot veure's en el codi anterior, la cua de connexions s'ha implementat per mitjà d'un senzill vector, la grandària del qual és de 10 (*GRAND_CUA* té un valor de 10). D'altra banda, el valor de *GRAND_TEXT* és de 80, i correspon amb la grandària màxima de la línia que pot enviar el client.

Finalment, la funció *EntePolsat()* té el codi següent:

```
int EnterPolsat(void)
{
  fd_set conjunt;
  struct timeval t;

  FD_ZERO(&conjunt);
  FD_SET(0, &conjunt); /* 0 és l'identificador del socket de stdin */

  t.tv_sec=0;
  t.tv_usec=1000; /* 1 milisegon */

  return select(1, &conjunt, NULL, NULL, &t); /* En cas d'error torna -1 i
                                                 eixirem del programa. */
}
```

[1] Si un client no permet l'escriptura en eixe instant, les dades no són enviades al cliente i, per tant, es perdran.

[2] No cal fer el programa amb esta configuració del *main()*, només s'ofereix com a ajuda i no com a requisit per al desenvolupament del codi.

Podeu trobar un exemple compilat del programa servidor, així com un xicotet client X que permet utilitzar eixe servidor en la pàgina web de l'assignatura. Els programes tenen el nom de *servidor* i *xat*, respectivament.

El programa servidor respon a la línia de comandaments proposada, mentre que el client respon a la línia de comandaments:

xat <adreça IP> <port>

On <*adreça IP*> especifica l'adreça IP de l'ordinador on es troba el servidor, i <*port*> especifica el port d'eixe ordinador on es troba a l'escolta el servidor.

Nota: Pel fet que els ordinadors de pràctiques tenen configurat el tallafocs de Linux en un format molt restrictiu, és possible que els clients no puguen connectar-se al servidor[3], per la qual cosa es recomana desactivar el tallafocs per mitjà del comandament "*service iptables 'stop'*".

[3] La connexió dins del propi ordenador, a l'adreça de loopback (127.0.0.1), hauria de funcionar sempre.

Práctica 7: Desarrollo de un servidor bajo TCP.

Autor: Enrique Bonet

Objetivo y descripción general.

El objetivo de esta práctica es el desarrollo de un servidor bajo el protocolo de transporte TCP. Para ello, se propone el desarrollo de una versión simplificada de un servidor de IRC.

El Internet Relay Chat es un protocolo de aplicación que permite la conversación entre grupos de usuarios en Internet, teniendo las posibilidades de distintos grupos de usuarios, conversaciones privadas, existencia de moderadores en los grupos, etc. El IRC se definió originalmente en el RFC 1459, pudiendo encontrarse las actuales especificaciones en los RFC 2810, 2811, 2812 y 2813.

Desarrollo de la práctica.

Desarrollar un programa en C/C++ que implemente una versión muy simplificada de un servidor de IRC. El servidor deberá responder a la siguiente línea de comandos:

servidor <puerto>

Donde <*puerto*> indica el puerto donde se encontrará instalado el servidor.

El análisis básico de desarrollo del programa es:

1. Crear un socket, asociarlo en escucha al puerto deseado, ponerlo en modo bloqueante e inicializar la cola de espera de peticiones.
2. Mirar si existe alguna petición de conexión al socket y, en caso afirmativo, aceptar dicha petición en una cola de clientes conectados. Para que el programa no se bloquee esperando una conexión, debéis utilizar la función *select()* con un timeout de 1 milisegundo, por ejemplo, pues la petición de conexión se recibe como una señal de datos disponibles para lectura en el socket.
3. Comprobar si podemos leer datos de alguno de los clientes conectados, utilizando, para no bloquear la ejecución del programa, la función *select()* con el timeout del punto anterior. En caso afirmativo, leer los datos de cada uno de los clientes.
    a. Si los datos leídos son de un tamaño mayor que 0 bytes, hemos realizado una lectura correcta.
    b. Sin embargo, si leemos 0 bytes de datos, a pesar de que la llamada a la función *select()* indicaba que teníamos datos a leer, esto indica que el cliente ha cerrado la conexión. En tal caso debemos cerrar esa conexión y eliminar el cliente de la lista de clientes conectados.
4. Si hemos leído datos de algún cliente, debemos escribir dichos datos en todos los clientes que tenemos conectados. Para ello, y nuevamente utilizando la

función *select()*, comprobamos si podemos escribir en un cliente, y en caso afirmativo, escribimos los datos[4].

5. Volvemos al punto 2.

Como ayuda se os proporciona la siguiente función *main()*[5]:

```
int main(int argc,char *argv[])
{
  char buffer[TAM_COLA][TAM_TEXTO+1];
  int sock,cola[TAM_COLA],num,cont;

  if (argc!=2)
    Uso(argv[0]);

  sock=CrearSocket((unsigned short)atoi(argv[1])); /* Creamos el socket. */

  num=0;
  while (EnterPulsado()==0) /* Mientras no pulsemos la tecla [Enter]. */
  {
    num=AceptarConexion(sock,cola,num); /* Miramos si hay conexiones nuevas. */

    if ((cont=Leer(sock,cola,&num,buffer))>0) /* Miramos si tenemos datos para leer. */
      Escribir(sock,cola,num,buffer,cont); /* Enviamos los datos leidos. */
  }

  CerrarSockets(sock,cola,num); /* Cerramos todas las conexiones. */
  return 0;
}
```

Donde en las funciones *Leer()* y *Escribir()* la variable *sock* solo se utiliza para poder cerrar ese socket si se produce un error.

Como puede verse en el código anterior, la cola de conexiones se ha implementado mediante un sencillo vector, cuyo tamaño es de 10 (*TAM_COLA* tiene un valor de 10). Por otro lado, el valor de *TAM_TEXTO* es de 80, y debe así especificarse pues se corresponde con el tamaño máximo de la línea que puede enviar el cliente.

Por último, la función *EnterPulsado()* tiene el siguiente código:

```
int EnterPulsado(void)
{
  fd_set conjunto;
  struct timeval t;

  FD_ZERO(&conjunto);
  FD_SET(0,&conjunto); /* 0 es el identificador del socket de stdin */

  t.tv_sec=0;
  t.tv_usec=1000; /* 1 milisegundo */

  return select(1,&conjunto,NULL,NULL,&t); /* En caso de error devuelve -1 y
                                              saldremos del programa. */
}
```

Podéis encontrar un ejemplo compilado del programa servidor, así como un pequeño cliente X que permite utilizar ese servidor en la página web de la asignatura. Los programas tienen el nombre de *servidor* y *chat*, respectivamente.

El programa servidor responde a la línea de comandos propuesta, mientras que el cliente responde a la línea de comandos:

[4] Como simplificación, si un cliente no permite la escritura en ese instante, los datos no son enviados a ese cliente y por tanto, se pierden.

[5] No es necesario realizar el programa con esta configuración del *main()* aquí dada. Solo se ofrece como ayuda y no como requisito para el desarrollo del programa.

chat <dirección IP> <puerto>

Donde *<dirección IP>* especifica la dirección IP del ordenador donde se encuentra el servidor, y *<puerto>* especifica el puerto de ese ordenador donde se encuentra a la escucha el servidor.

Nota: Debido a que los ordenadores de prácticas tienen configurado el cortafuegos de Linux en un formato muy restrictivo, es posible que los clientes no puedan conectarse al servidor[6], por lo que se recomienda desactivar el cortafuegos mediante el comando "*service iptables stop*".

[6] Siempre es posible intentar la conexión, dentro del propio ordenador, a la dirección de loopback (127.0.0.1), la cual debe funcionar.