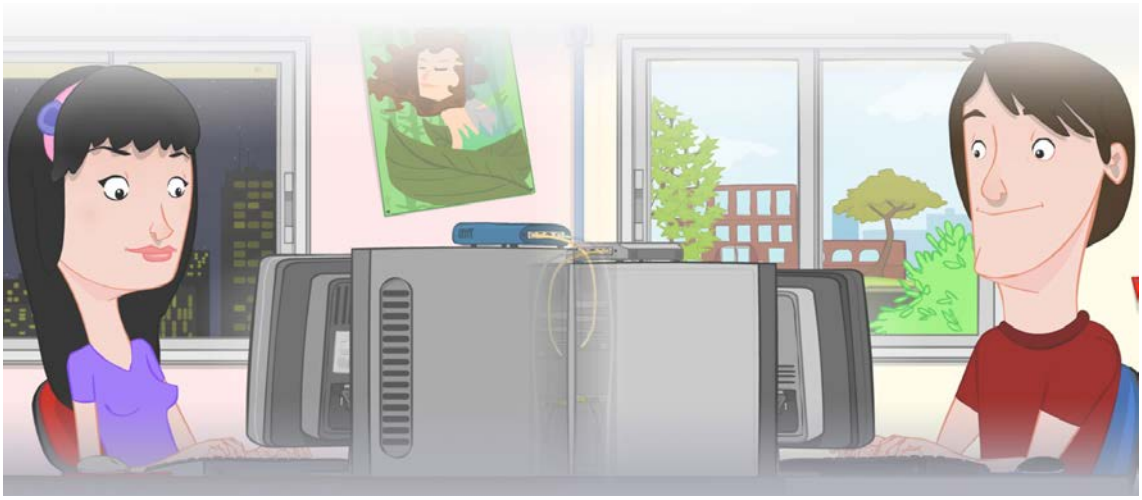
Práctica Sockets

Comunicación entre procesos de máquinas diferentes



Curso 2020-2021

Grado en Ingeniería Técnica de Telecomunicación

(3º curso)

Área de Ingeniería Telemática

Práctica Sockets: Comunicaciones entre procesos de máquinas diferentes

1. Motivación:

Hasta ahora hemos visto como podemos comunicar procesos de una aplicación cuando estos están en la misma máquina. En el mundo de las redes es preciso diseñar aplicaciones donde unos procesos se puedan comunicar con otros incluso aunque estén en máquinas diferentes. Estas interacciones entre procesos se producen utilizando la infraestructura de esas redes de comunicación. En general hablamos de protocolos de comunicaciones que debemos utilizar para trabajar con esas redes. Internet es el ejemplo más claro de ello, pero recordar que hay muchos otros protocolos además de los que se utilizan en internet. Desde el punto de vista del programador, conviene utilizar librerías que se puedan abstraer lo más posible de las problemáticas específicas de los protocolos de esas redes y que permitan una forma fácil y flexible para implementar las comunicaciones entre procesos de máquinas diferentes. Este concepto es el que inspiró en su día a los creadores de la librería de Sockets para facilitar la comunicación entre procesos. La universidad de Berkeley fue pionera y gracias a ellos disponemos ahora de una solución software suficientemente flexible para adaptarnos a las necesidades que hoy en día podemos tener de comunicar procesos con procesos utilizando redes muy diferentes (Internet, AppleTalk, IPX/SPX, Ethernet, NetBeui, X.25, frame-relay, etc).

En ese interés de adaptarse a cualquier tipo de red, la arquitectura de sockets puede tener cosas un poco extrañas que quizás compliquen su aplicación a casos concretos, pero que buscan la flexibilidad para aplicaciones en otros casos. Los alumnos harán estas prácticas utilizando la familia de protocolos de internet (AF_INET), pero conviene que conozca el sentido más general de la librería de sockets. Las ideas planteadas anteriormente son importantes para ello: (1) Los sockets no están diseñados sólo para aplicaciones del mundo de internet, (2) muchas llamadas a servicios de sockets (bind(), listen(), accept()) pueden parecer extrañas, pero obedecen a un sentido más general para otros usos en otras redes y (3) los parámetros que se utilizan en esas llamadas pueden ser muy diferentes en función de la familia de protocolo que se esté utilizando.

Los sockets es una arquitectura pensada para ofrecer a una máquina la posibilidad de comunicarse con otros procesos de la misma u otras máquinas olvidándose un poco de la problemática de los protocolos de las redes que se deben utilizar. Si esto se entiende, se dará sentido a su uso. Un ingeniero, debe entender que dentro de esa arquitectura hay muchas cosas de implementación de los protocolos y de abstracción de los servicios del socket que sería de interés conocer. En este curso no se va a profundizar en ese nivel.

El conocimiento de los sockets permitirá a los alumnos diseñar aplicaciones cliente/servidor para comunicar procesos de la misma máquina o entre máquinas remotas.

2. Programación de sockets para AF_INET

Esta práctica tiene un tiempo de desarrollo de tres semanas y está organizada en dos pruebas. La primera prueba está diseñada para realizar un test previo de los conocimientos del alumno en el manejo de los sockets de la familia de Internet (AF_INET). La segunda prueba es similar aunque un poco más exigente. Se pretende que el alumno avance en el diseño de aplicaciones servidoras en modo concurrente y en la utilización de funciones específicas de la familia AF_INET para facilitar el acceso remoto a servicios externos de internet (DNS, etc,…).

En ambos casos el alumno realizará unos programas que se comunicarán con una aplicación de evaluación (el monitor). Este apartado permitirá al alumno comprobar su nivel de dominio de los conceptos y habilidades que se trabajan en esta práctica.

2.1 Descripción de la primera prueba práctica

El alumno debe realizar un programa denominado <client.c> que se ejecutará como <./client> en el directorio de trabajo donde reside el programa <./monitor>. El programa monitor propone la realización de 7 ejercicios que pueden ejecutarse de forma independiente pero que están enunciados pensando en que se hagan de forma progresiva.

Ejercicio 1: Establecer conexión TCP

El proceso monitor pondrá un socket TCP a la escucha de conexiones en un puerto determinado (3000). Se espera que el programa cliente realice una conexión a dicho socket. Se revelará el secreto <1> cuando se realice dicha conexión.

Ejercicio 2: Recibir datos por TCP

El proceso monitor enviará un mensaje por la conexión TCP establecida en el apartado anterior. El programa cliente DEBE ALMACENAR este mensaje para su posterior uso. El secreto <2> se revelará si se cumple con éxito el apartado siguiente. Ese mensaje realmente tiene la clave <2> en formato “<999>”, pero se espera que en el apartado siguiente la devuelva en eco.

Ejercicio 3: Enviar datos por TCP

El proceso monitor espera recibir el mensaje que se envió en el apartado anterior. Tras ello, enviará un nuevo mensaje que debe ser reenviado de vuelta de manera inmediata. Se revelará el secreto <2> si se recibe el mensaje del apartado (2) correctamente. Se revelará el secreto <3> si se recibe en eco el mensaje enviado en este apartado. Ese mensaje realmente tiene la clave <3> en formato “<999>”, pero se espera que se devuelva en eco.

Ejercicio 4: Recibir conexión en servidor TCP

El proceso monitor intentará conectarse a un servidor de la máquina que esté en un puerto determinado (3001). Se revelará el secreto <4> si se consigue conectar con éxito.

Ejercicio 5: Recibir/enviar datos en servidor TCP

El proceso monitor enviará un mensaje por la conexión TCP establecida en el apartado anterior. El programa cliente debe enviar de forma inmediata el mismo mensaje en eco de vuelta. Se revelará el secreto <5> si se recibe el eco del mensaje de forma correcta.

Ejercicio 6: Conectar con servidor UDP y enviar datos

El proceso monitor pondrá un socket UDP a la escucha de conexiones en un puerto determinado (3000). Se espera que el programa cliente envíe un saludo ("HOLA!" o "KAIXO!"). Se revelará el secreto <6> cuando se realice dicha conexión y envío de datos

Ejercicio 7: Crear servidor UDP y recepción de datos

El proceso monitor enviará un mensaje al servicio UDP en la máquina en un puerto determinado (3001). Este mensaje contendrá el secreto <7> con formato "<999>. El programa client debe instalar un servicio UDP de recepción de datagramas que devuelva en eco el mensaje recibido. Se revelará el secreto <7> si se recibe el eco correctamente.

2.2 Descripción de la segunda prueba práctica

Esta prueba es algo más completa que la anterior. El ejercicio 1 y el 3 son similares a lo realizado en la prueba anterior, pero con una sesión tcp y udp completa en cada caso. El Ejercicio 2 propone la realización de un servidor tcp concurrente completo y el 4 un servidor udp. Los ejercicios 5 y 6 proponen la realización de un cliente tcp y otro udp para conexión a un servidor externo referenciado por dominio o ip. Esto obligará a los alumnos a utilizar funciones específicas de la familia AF_INET.

Ejercicio 1: Comunicación TCP con servidor interno

El programa monitor esperará la conexión TCP del cliente en un puerto determinado (3000). Si se produce esa conexión, el monitor desvelará el secreto <1>. Posteriormente el monitor enviará, a través de la comunicación realizada, el secreto <2> como un entero en formato binario. El cliente debe leer ese dato y devolverlo a través de la conexión, y de forma inmediata pero en formato ascii <999>. Posteriormente el cliente debe cerrar la conexión. Si se devuelve correctamente el valor leído se descubrirá el secreto <2> y el <3>.

Si no se conociese el secreto 2 para devolverlo en eco se puede enviar <???>. En ese caso se puede descubrir el secreto <3>.

Ejercicio 2: Servidor TCP concurrente

El cliente debe crear un servidor TCP concurrente que escuche en un puerto determinado (3001), en espera de conexiones. El monitor solicitará conexiones al servidor y una vez obtenida cada sesión enviará un mensaje con un entero en formato binario y esperará su eco con un mensaje en formato ascii <999>. La sesión creada por el programa client en modo servidor debe permanecer activa hasta que el monitor decida cerrarla.

La clave <4> se desvelará cuando el monitor detecte que se ha respondido a la primera sesión solicitada y se han creado los procesos adecuados en el programa client. La clave <5> se obtendrá cuando se completen n sesiones consecutivas, y la clave <6> cuando se cierren bien todas las sesiones creadas

Ejercicio 3: Comunicación UDP con servidor interno

El programa monitor arrancará un servidor UDP en un puerto determinado (3000), en espera de recibir datos. El cliente enviará un mensaje con la cadena <???> al monitor que se interpretará como una petición de la clave <8>. El monitor desvelará la clave <7> por seguir el procedimiento y responderá con un mensaje en formato <999> indicando el valor de la clave <8>. Se espera que el cliente responda, inmediatamente, con un mensaje en eco indicando la clave 8 recibida como un entero en formato binario. Si se hace así, el monitor desvelará la clave <9>.

Ejercicio 4: Servidor UDP

El cliente debe crear un servidor UDP en un puerto determinado (3001), en espera de recibir datos. El monitor enviará un entero en formato binario al servidor UDP con la clave <10> y esperará su eco con un mensaje en formato ascii <999>. Si la respuesta a ese mensaje es correcta se desvelarán las claves <10> y <11>. Posteriormente se enviarán otros mensajes con similar formato a los que debe responder el servidor adecuadamente. Si es así se desvelará la clave <12>.

Ejercicio 5: Comunicación TCP con servidor externo

El alumno debe crear un fichero fuente <client_tcp.c> que sirva para realizar conexiones tcp a cualquier servidor indicado por dominio o ip en un puerto determinado. La estructura de comando debe ser “client_tcp <DNI@dominio:puerto>".

Este programa será compilado por monitor y lo lanzará contra un servidor de prueba externo. El programa client_tcp, cuando se conecte al servidor remoto debe enviar un comando "user DNI\n" y esperará una respuesta "ok\n". Posteriormente mandará un mensaje "get clave\n" y esperará a recibir la clave <14> en formato ascii "CLAVE:<999>\n".

El programa client_tcp debe devolver esa clave en eco con formato en ascii "<999>\n". Posteriormente recibirá un mensaje "bye\n" al que debe responder cortando la comunicación.

Los mensajes que se envían deben presentarse en una línea en pantalla con el formato "TX:<texto transmitido>\n" y los recibidos "RX:<texto recibido>\n"

Si se realiza la conexión de forma adecuada y se recibe el “ok” se descubrirá la clave <13>. Si se mantiene el diálogo de sesión se descubrirá la clave <15> y la clave <14> si se lee correctamente.

Ejercicio 6: Comunicación UDP con servidor externo

El alumno debe crear un fichero fuente <cliente_udp.c> que sirva para realizar conexiones udp a cualquier servidor indicado por dominio o ip en un puerto determinado. La estructura de comando debe ser "cliente_udp <DNI@dominio:puerto>".

Este programa será compilado por monitor y lo lanzará contra un servidor de prueba externo.

Al lanzar el programa client_udp, éste mandará un datagrama al servidor con el comando "GET CLAVE DNI" y recibirá como respuesta la clave <17> en formato ascii "CLAVE:<999>\n".

El cliente UDP debe devolver esa clave en eco en formato ascii "<999>\n".

Los mensajes que se envían deben presentarse en una línea de pantalla con el formato "TX:<texto transmitido>" y los recibidos "RX:<texto recibido>".

SI se realiza la conexión de forma adecuada y con un DNI correcto, se descubrirá la clave <16>.
La clave <18> se obtendrá si se realiza bien el protocolo completo.