Facultad de Ingeniería y Ciencias
Escuela de Informática y Telecomunicaciones

Laboratorio N6: Redes de Datos
'Sockets'

Profesor: José Pérez
Ayudante: Alexis Inzunza
Fecha: 09-06-2017

Integrantes:
Benjamín Alvarez
Nicolás Reyes

ÍNDICE

ÍNDICE DE FIGURAS

Actividades de Laboratorio

El objetivo de este laboratorio es crear un algoritmo simple de cliente y servidor para un chat usando sockets, pero primero, ¿Qué es un socket? Un socket es un organismo que permite el intercambio de información entre un cliente y un servidor. Para poder usarlo necesitaremos ciertas herramientas. Estás serán los identificadores o IP's de dos computadores, un protocolo de transporte (TCP o UDP) y dos números de puerto, uno local y otro remoto. En este caso usaremos el protocolo TCP ya que este ofrece muchas mas confiabilidad cuando mandamos un mensaje, y es lo que generelamente se busca tener en un chat, ya que este debe ser entre dos personas o más pero no salir de ese grupo de conversación.

I. Actividad 1

Para crear el cliente y el servidor usaremos el lenguaje de programación Python por un tema de comodidad. También haremos uso de API conocida como "Sockets de Berkeley":

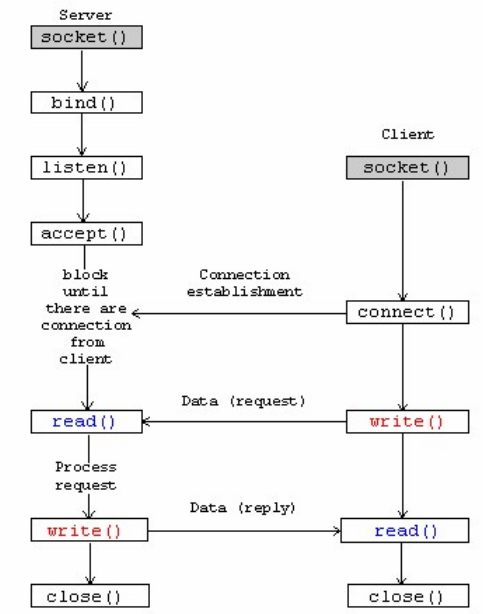

Figura 1. Sockets de Berkeley

Las funciones que se utilizarán son las siguientes:

- socket.socket([family[, type[, proto]]]): Es el constructor del socket y sirve para instanciarlo. En este caso vamos a usar sockets de internet, por lo que el primer parámetro será socket.AF_INET y el segundo dependerá del protocolo que usemos, si es UDP se usa socket.SOCK_DGRAM y para TCP será socket.SOCK_STREAM.
- socket.bind(address): Este recibe una tupla que contiene el string de la IP host y el int del puerto a usar en nuestro servidor.
- socket.listen(backlog): Deja a nuestro server escuchando por conexiones. El parámetro que usemos dependerá de cuantas conexiones queremos mantener en cola, por ejemplo si ponemos 5 el servidor esperará a tener 5 conexiones para aceptarlas a todas. En general el número usado es 1.
- socket.accept(): Acepta una conexión. Retorna un socket y su address, que corresponde a la información del cliente.
- socket.recv(bufsize[, flags]): Se utiliza para recibir los datos del cliente. El bufsize indica la cantidad de bytes máximo admitidos, en general se usa 1024.
- socket.send(string[, flags]): Lo utiizaremos para enviar los datos. El parámetro que enviaremos es un string.
- socket.close(): Cierra el socket. Se utiliza una vez se termina el intercambio de información.

Ahora que conocemos las funciones empezaremos a crear el cliente. Lo primero que debemos hacer tanto en el cliente como en el servidor es importar socket escribiendo 'import socket' al principio de nuestro código.
Después definimos el host y el puerto a usar, en este caso el host será la IP de Loopback y el puerto será 5000. Después de eso crearemos nuestro socket usando la primera función nombrada anteriormente, entregandole los parámetros socket.AF_INET y socket.SOCK_STREAM, el segundo es porque trabajaremos con TCP.
Luego de tener creado nuestro socket usaremos la función socket.connect() y le pasaremos el host y el puerto en una tupla. Ahora pediremos en mensaje que se va a enviar con raw_input y lo enviamos con socket.send(1024), luego cerraremos el client con socket.close(). El código quedará de esta forma:

```
import socket
host = '127.0.0.1'
port = 5000

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect((host, port))

message = raw_input("-> ")
s.send(message)

s.close()
```

Figura 2. Cliente

Ahora crearemos nuestro servidor:
Nuevamente al principio escribiremos 'import socket' y definiremos nuestra IP host y el puerto, que debe ser el mismo que pongamos en nuestro cliente. Con eso listo usaremos la función socket.bind() para bindear el host y el puerto a nuestro servidor. Luego usaremos socket.listen para empezar a aceptar conexiones, en este caso el parámetro que le pasaremos será 1. Despúes usaremos socket.accept() para recibir el socket del cliente y su información, esta información la imprimiremos en el servidor. Por último recibiremos el mensaje del cliente usando socket.recv(1024) y también lo imprimiremos. Por último usaremos socket.close() para cerrar nuestro servidor. El código debería quedar así:

```
import socket
host = '127.0.0.1'
port = 5000

s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.bind((host,port))
s.listen(1)

c, addr = s.accept()
print "Conexion desde: " + str(addr)

data = c.recv(1024)
print "De usuario conectado: " + str(data)

c.close()
```

Figura 3. Servidor

II. ACTIVIDAD 2

Utilizamos Wireshark para ver que sucedía al ejecutar nuestro programa y los resultados fueron los siguientes:

53 226.233288000	127.0.0.1	127.0.0.1	TCP	74 49727→5000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3227288 TSecr=0 WS=128
54 226.233301000	127.0.0.1	127.0.0.1	TCP	74 5000→49727 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3227288 TSecr=3227288 WS=128

Figura 4. Wireshark conectarse el cliente al servidor

65 245.815144000	127.0.0.1	127.0.0.1	IPA	69 unknown 0x64 [Malformed Packet]
66 245.815198000	127.0.0.1	127.0.0.1	TCP	66 5000→49727 [ACK] Seq=1 Ack=4 Win=43776 Len=0 TSval=3232183 TSecr=3232183
67 248.913280000	127.0.0.1	127.0.0.1	IPA	69 unknown 0x64 [Malformed Packet]
68 248.913310000	127.0.0.1	127.0.0.1	TCP	66 5000→49727 [ACK] Seq=1 Ack=7 Win=43776 Len=0 TSval=3232958 TSecr=3232958

Figura 5. Wireshark al enviar mensajes desde el cliente

79 300.057662000	127.0.0.1	127.0.0.1	TCP	66 49727→5000 [FIN, ACK] Seq=7 Ack=1 Win=43776 Len=0 TSval=3245744 TSecr=3232958
80 300.057726000	127.0.0.1	127.0.0.1	TCP	66 5000→49727 [FIN, ACK] Seq=1 Ack=8 Win=43776 Len=0 TSval=3245744 TSecr=3245744

Figura 6. Wireshark cuando el cliente se desconecta

III. Análisis y preguntas

1. Utilizando la información obtenida en la actividad 2, explique el funcionamiento del envío y recepción de información para su algoritmo, respondiendo a las interrogantes ¿Qué se envía? ¿Cómo se envía?
   R: Como el protocolo que usamos es TCP, lo que se envia no es solamente el puerto de origen y de destino más el mensaje, sinó que se envian una serie de verificaciones para comprobar que el mensaje llegue correctamente a su destino.
2. ¿Por qué el puerto que muestra el servidor al generar la conexión no es el mismo que el escrito en el algoritmo del ciente?
   R: Esto es porque el cliente generalmente usa un puerto aletorio para conectarse a un servidor con un puerto específico. El servidor responde usando este puerto aleatorio. Por lo tanto el cliente y el servidor no están en el mismo puerto.
3. Si usted tuviese que realizar un videojuego con soporte multijugador utilizando sockets ¿Qué protocolo utilizaría? ¿Por qué?
   R:Si hablamos de un multijugador masivo el protocolo que se usa generalmente es UDP, ya que este aunque no asegura la llegada del paquete si asegura mucha rápidez y en juegos donde hay 64 personas al mismo tiempo la rápidez es necesaria. En otros juegos como por ejemplo los de cartas online que son entre dos personas creo que el mejor protocolo sería TCP ya que no se necesita rápidez, sinó lo que se necesita es mandar bastante información y asegurarse de que este llegue a su destino.
4. ¿Se puede utilizar cualquier número de puerto para cualquier aplicación? Explique.
   R: No, los puertos que se pueden utilizar pueden ir desde 1024 hasta 49151. Esto es porque los puertos inferiores a 1024 son de uso exclusivo del sistema operativo, y los mayores de 48151 hasta 65535 son para uso privado y no pueden ser usados por cualquier aplicación.

IV. Conclusión

En esta actividad aprendimos como crear un cliente y un servidor para un chat, usando el protocolo TCP y además como se envia este mensaje. También pudimos conocer la diferencia entre los protocolos TCP y UDP y revisar cual es el que serviría mas en distintas situaciones. También conocimos los puertos que se pueden o no usar para crear estos servidores. Por último conocimos los socket y sus funciones y como operan.

REFERENCIAS

[1] omicrono, *¿Por qué una partida online ocupa menos ancho de banda que una página web?* http://omicrono.elespanol.com/2016/09/videojuego-online-tiempo-de-carga/

[2] superuser, *Is the same port acting as both client and server?* https://superuser.com/questions/683965/is-the-same-port-acting-as-both-client-and-server