SOCKETS

Nombres: Laura Catalina Preciado Ballen

Daniel Santiago Arcila Martinez

Universidad Distrital Francisco José de Caldas

Programación Avanzada
Fecha: Viernes 7 de agosto

Colombia, Bogotá D.C

RESUMEN

Este mecanismo surge a principios de los 80 con el sistema Unix de Berkeley, para proporcionar un medio de comunicación entre procesos y presentan la misma funcionalidad que tiene la

comunicación por correo o por teléfono (de un buzón se extraen mensajes completos, mientras que el teléfono permite el envío de flujos de información que no tienen una estructura claramente definida), es decir permiten que un proceso hable (emita o reciba información) con otro incluso estando estos en distintas máquinas. Esta característica de ínter conectividad hace que el concepto de socket sea de gran utilidad

Palabras clave:Socket, TCP, UDP, proceso,servidor, cliente, número de puerto, dirección IP

¿QUÉ ES UN SOCKET?

Un socket (enchufe), es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un socket se define como el punto final en una conexión. Los sockets se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets.

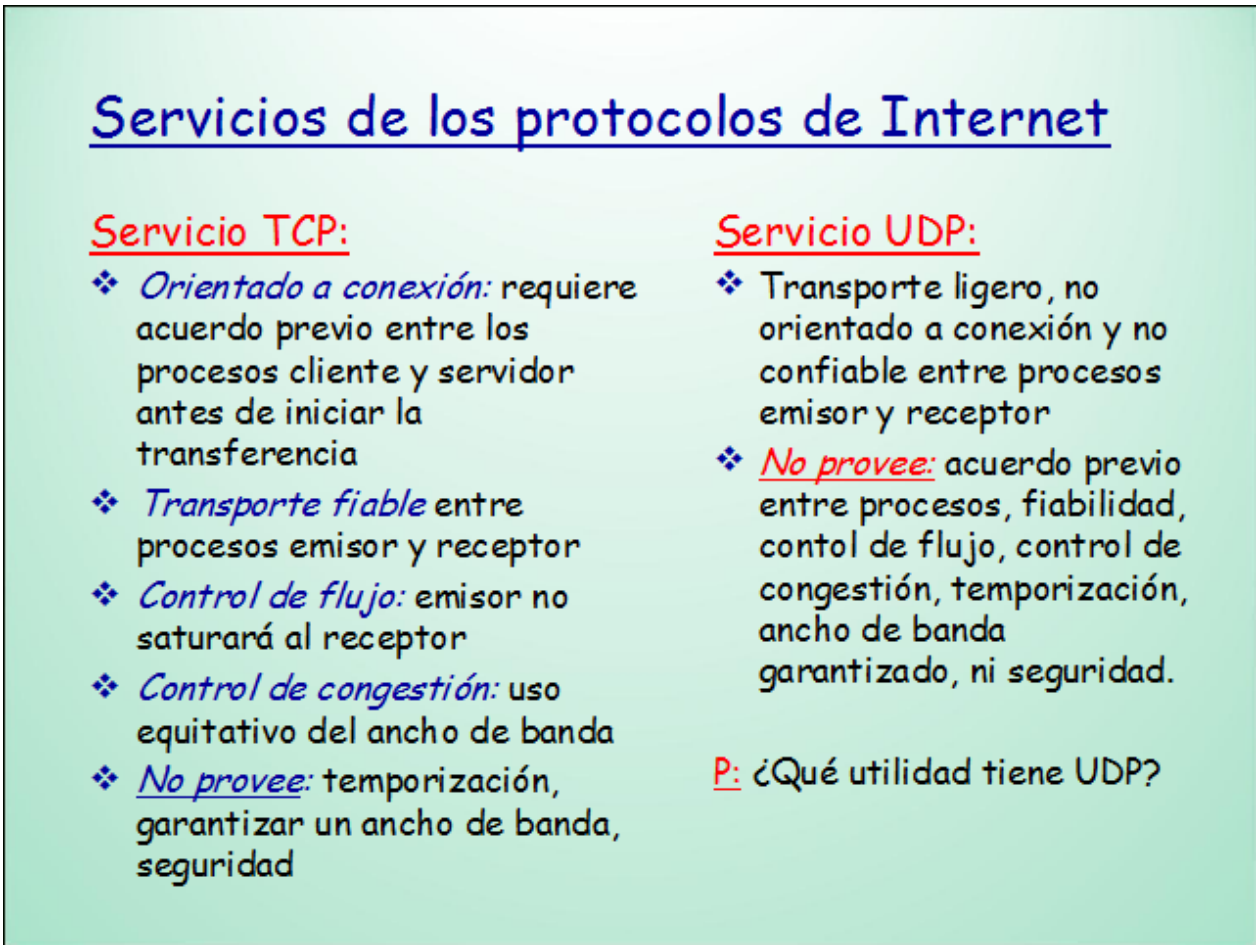
Un socket es también una dirección de Internet, combinando una dirección IP (la dirección numérica única de cuatro partes que identifica a un ordenador particular en Internet) y un número de puerto (el número que identifica una aplicación de Internet particular, como FTP, Gopher, o WWW).

TIPOS DE SOCKET

En la actualidad existen varios tipos de socket y cada uno por lo regular se asocia a un tipo de protocolo, por ejemplo:

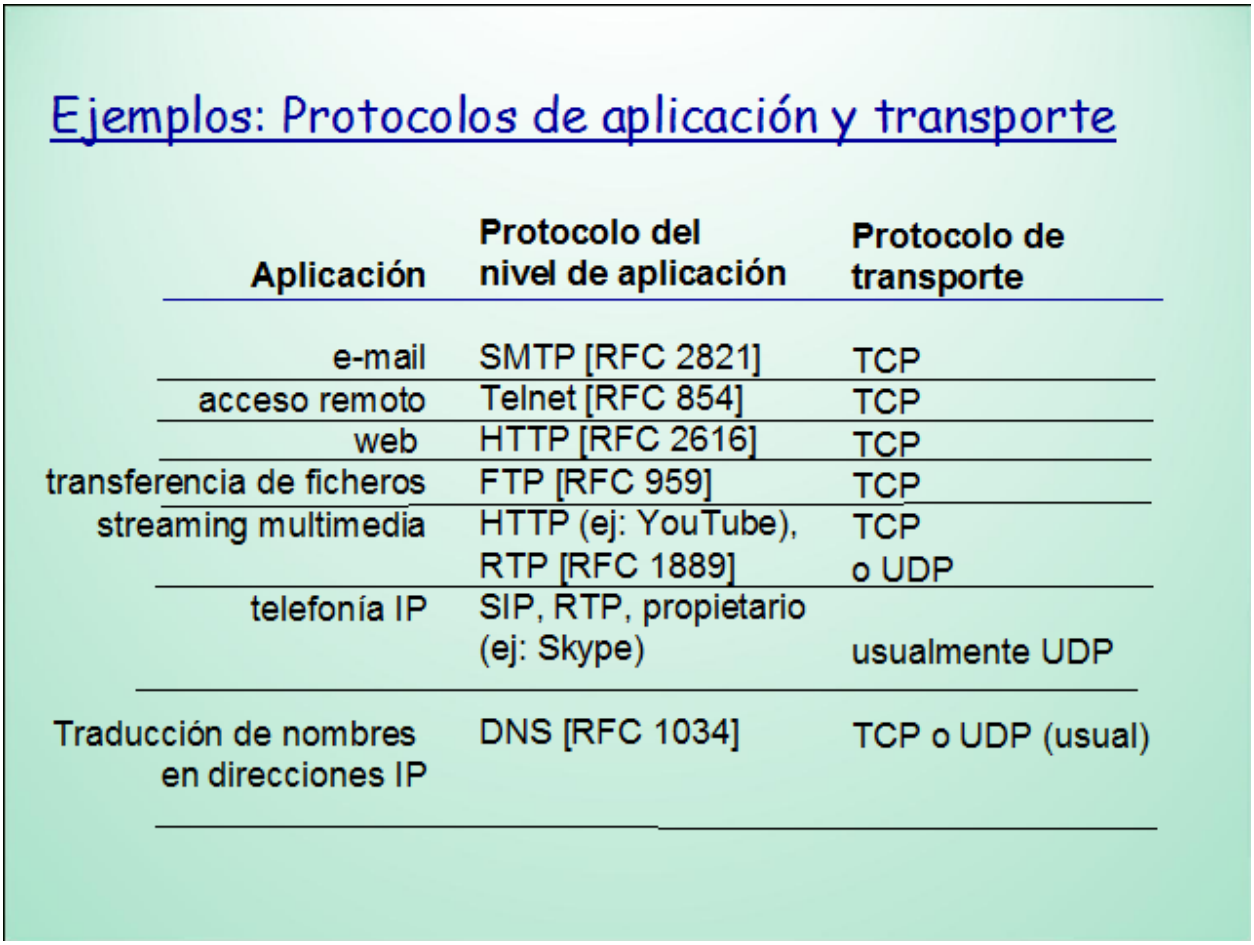
SOCK_STREAM: está asociado al protocolo TCP, este brinda seguridad en la transmisión de datos, seguridad en la recepción, en la integridad y en la secuencia, entre otros.

SOCK_DGRAM: está asociado al protocolo UDP, e indica que los paquetes viajarán en tipo datagramas, el cual tiene una comunicación asíncrona.



¿COMO SE IDENTIFICA UN SOCKET?

Como se había mencionado anteriormente un socket queda identificado por una Dirección IP y un puerto. De modo que se tiene que aclarar la existencia varios puertos que brindan diferentes servicios y que usan diferentes protocolos de transporte como se ve en la imagen



(El uso de un protocolo u otro dependerá mas de las necesidades de la aplicacion en cuestion)

FUNCIONAMIENTO

Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto. Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

- Que un programa sea capaz de localizar al otro.

- Que ambos programas sean capaces de intercambiarse cualquier secuencia de octetos, es decir, datos relevantes a su finalidad.

Para ello son necesarios los tres recursos que originan el concepto de socket:

- Un protocolo de comunicaciones (TCP o UDP), que permite el intercambio de octetos.
- Un par de direcciones del Protocolo de Red (Dirección IP, si se utiliza el Protocolo TCP/IP), que identifica la computadora de origen y la remota.
- Un par de números de puerto, que identifica a un programa dentro de cada computadora.

Los sockets permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa cliente. El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa servidor. Cuando un cliente conecta con el servidor se crea un nuevo socket, de esta forma, el servidor puede seguir esperando conexiones en el socket principal y comunicarse con el cliente conectado, de igual manera se establece un socket en el cliente en un puerto local.

Una aplicación servidor normalmente escucha por un puerto específico esperando una petición de conexión de un cliente, una vez que se recibe, el cliente y el servidor se conectan de forma que les sea posible comunicarse entre ambos. Durante este proceso, el cliente es asignado a un número de puerto, mediante el cual envía peticiones al servidor y recibe de éste las respuestas correspondientes.

EJEMPLO DE SOCKET EN PYTHON

En este tutorial vamos a hacer un pequeño servidor python y un cliente python que se conectan con socket, se intercambian unas cadenas de texto y cierran la conexión.

El servidor

En el servidor, primero establecemos el socket servidor. Para ello, usamos el módulo socket de python, haciendo el import correspondiente. Damos los siguientes pasos:

- Llamada a la función socket(), a la que le pasamos el tipo de socket que queremos abrir (en el ejemplo, AF_INET, SOCK_STREAM que es el habitual).
- Llamada a la función bind(), pasándole un address compuesto por (host, puerto). Como hacemos de servidor, pasamos "" como host y puerto el que queramos que esté libre (8000). La llamada a bind() le indica al sistema operativo que nosotros vamos a atender las conexiones por el puerto 8000.
- Llamada a listen(). Esta llamada indica al sistema operativo que ya estamos listos para admitir conexiones. El número 1 de parámetro indica cuantos clientes podemos tener encolados en espera simultáneamente. Este número no debería ser grande, puesto que es el número máximo de clientes que quedarán encolados desde que aceptamos un cliente hasta que estamos dispuestos a aceptar el siguiente. Si el código está bien hecho, este tiempo debería ser realmente pequeño, ya que al conectarse un cliente, deberíamos lanzar un hilo para atenderlo y entrar inmediatamente a la espera de otro cliente.

El código de todo esto puede ser:

```
import socket

server = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
server.bind(("", 8000))
server.listen(1)
```

Ahora nos metemos en un bucle eterno, esperando clientes y atendiéndolos. Para esperar un cliente, la llamada es accept(), que nos devuelve un par (socket_cliente, datos_cliente) del que obtenemos el socket para hablar con el cliente y sus datos (ip, puerto).

```
   # bucle para atender clientes
   while 1:
      # Se espera a un cliente
      socket_cliente, datos_cliente =
server.accept()
      # Se escribe su informacion
      print "conectado "+str(datos_cliente)
```

Para leer los datos del cliente, usamos recv(), pasando como parámetro el número máximo de bytes que queremos leer de una tacada. La lectura se quedará bloqueada hasta que llegue algo del cliente. En cuanto lleguen datos, la llamada nos devolverá esos datos y no esperará hasta el máximo que hemos indicado. El bucle para atender al cliente puede quedar así:

```
      # Bucle indefinido hasta que el cliente envie
"adios"
      seguir = True
      while seguir:
         # Espera por datos
         peticion = socket_cliente.recv(1000)

         # Contestacion a "hola"
         if ("hola"==peticion):
             print str(datos_cliente)+ " envia
hola: contesto"
             socket_cliente.send("pues hola")

         # Contestacion y cierre a "adios"
         if ("adios"==peticion):
             print str(datos_cliente)+ " envia
adios: contesto y desconecto"
             socket_cliente.send("pues adios")
             socket_cliente.close()
             print "desconectado
"+str(datos_cliente)
             seguir = False
```

La variable seguir nos hará permanecer en el bucle hasta que la pongamos a False, cosa que haremos cuando cerremos la conexión con el cliente. recv(1000) lee las peticiones del cliente y con los if comprobamos si lo recibido es "hola" o "adios". Para contestar usamos send(), enviando la cadena de respuesta. Si es "adios", además de contestar, cerramos el socket y ponemos a False la variable seguir, para terminar el bucle.

El cliente

Para el cliente, creamos el socket con la llamada a socket(), igual que en el servidor, pero establecemos la conexión con connect(), indicando el host servidor (localhost en nuestro ejemplo) y el puerto.

```
import socket
...
# Se establece la conexion
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.connect(("localhost", 8000))
```

El resto es más sencillo que en el servidor. Envíamos "hola" con send(), esperamos y escribimos la respuesta con recv(), hacemos una espera de 2 segundos con sleep(), enviamos con send() la cadena "adios", esperamos y escribimos respuesta con recv() y finalmente cerramos el socket con close()

```
import time
...
# Se envia "hola"
s.send("hola")
# Se recibe la respuesta y se escribe en pantalla
datos = s.recv(1000)
print datos
# Espera de 2 segundos
time.sleep(2)
# Se envia "adios"
s.send("adios")
# Se espera respuesta, se escribe en pantalla y se
cierra la
# conexion
datos = s.recv(1000)
print datos
s.close()
```

EJEMPLO DE SOCKETS EN PHP

La extensión Stream también proporciona una forma fácil de crear un socket de servidor con la función _stream_socket_server()_. Esta función también toma como primer argumento una especificación socket, de la misma forma a como se ha hecho con _stream_socketclient().

Para crear un socket de servidor son necesarias las siguientes cosas:

- Enlazar un socket, le dice al sistema operativo que queremos que lleguen paquetes a la red indicada por el socket.
- Comprueba si está disponible una conexión de entrada.
- Acepta la conexión de entrada (con la función stream_socket_accept()).
- Envía alguna información al cliente.
- Cierra la conexión, o deja que el cliente la cierre

Antes de poder escribir en un servidor, primero hay que aceptar la operación el en socket de servidor mediante la función _stream_socket_accept()_. Esta función bloquea el socket hasta que un cliente conecta con el servidor o cuando el tiempo de espera se agota.

Ejemplo:

```
$servidor = stream_socket_server("tcp://127.0.0.1:1339", $errno, $errorMessage);
if ($servidor === false) {
    throw new UnexpectedValueException("No se ha podido enlazar el socket: $errorMessage");
}
for (;;) {
    $cliente = @stream_socket_accept($servidor);
    if ($cliente) {
        stream_copy_to_stream($cliente, $cliente);
        fclose($cliente);
    }
}
```

Para probarlo, abre una ventana en la terminal y escribe:

```
php server.php
```

server.php es el archivo donde has guardado el código. Una vez iniciado, abre una nueva ventana en la terminal y escribe lo siguiente:

```
echo "Hola que tal" | nc 127.0.0.1 1337
// Devuelve Hola que tal
```

Si tienes el puerto 1337 ocupado dará un error Address already in use, puedes probar con cualquier otro, como 9002, por ejemplo.

El proceso que sigue es el siguiente:

Primero enlaza con tcp el socket en la dirección 127.0.0.1 y el puerto 1337. Este es el socket de red, que puede usarse por los clientes para conectar con nuestro servidor. Al igual que ocurría con la función _stream_socketclient(), dos argumentos se pueden pasar por referencia, el código de error y su mensaje. Si ocurre cualquier error, como que el puerto ya esté en uso, podemos ver el mensaje gracias a la condición if $servidor === false.

Cuando se usa la estructura de control for sin ninguna expresión se produce un loop infinito. Esto es necesario para que el servidor siga funcionando hasta que decidamos pararlo.

_$cliente = stream_socketaccept($servidor) bloquea el socket hasta que un cliente conecta con el socket o se agota el tiempo de espera. Se añade el @ para evitar que la función devuelva warnings. _stream_socketaccept() devuelve una conexión o false, por lo que con if ($client) sólo haremos algo si el cliente está conectado.

La función _stream_copy_tostream($cliente, $cliente) copia bytes desde el stream dado en el primer argumento al stream dado en el segundo argumento, por lo que básicamente devuelve todo lo que el cliente envía.

Finalmente se cierra la conexión con fclose($cliente). Esto no siempre es necesario, por ejemplo en una conexión FTP donde se quiere que el cliente la cierre.

Conclusiones

En programación, los sockets son esenciales ya que facilita mucho la comunicación entre cliente y servidor, ahora podemos recibir notificaciones, actualizaciones, estados de conectividad, o información de otros equipos, por eso esta interacción nos ayuda a tener la información en tiempo real sin necesidad de esperar a que el cliente vuelva a solicitar información del servidor, en pocas palabras permite interactuar con el servidor y con otros cliente al instante.

Bibliografías

ChuWiki. (2 de marzo de 2009). Obtenido de: http://chuwiki.chuidiang.org/index.php?title=Sockets_en_Python#:~:text=En%20el%20servidor%2C%20primero%20establecemos,python%2C%20haciendo%20el%20import

%20correspondiente.&text=Llamada%20a%20la%20funci%C3%B3n

%20socket,SOCK_STREAM%20que%20es%20el%20habitua

ECURED. (3 de septiembre de 2019). Obtenido de:

https://www.ecured.cu/Socket

masadelante.com. (2019). Obtenido de:

https://www.masadelante.com/faqs/socket

Turmero, P. (13 de Noviembre de 2015). monografias.com. Obtenido de:

https://www.monografias.com/trabajos106/redes-computadores/redes-computadores2.shtml

https://diego.com.es/sockets-en-php