Echo Client/Server

Introducción

Los sockets se usan casi en cualquier parte, pero son una de las tecnologías peor comprendidas.

De las diferentes formas de IPC (Inter Process Communication), los sockets son de lejos la más popular. En una plataforma dada, probablemente hay otras formas de IPC más rápidas, pero para comunicaciones inter-plataforma, los sockets son casi la única elección.

Las redes permiten que los programas de computadora se comuniquen entre sí, incluso si se ejecutan en diferentes máquinas. Los programas de redes funcionan de **igual a igual** (el mismo programa se ejecuta en diferentes máquinas), o más comúnmente, cliente / servidor (los programas cliente envían solicitudes a un servidor).

Se inventaron en Berkeley como parte de la variante BSD de UNIX. Se extendieron muy rápidamente junto con Internet. Y con razón, la combinación de los sockets con INET hace que la comunicación entre máquinas cualesquiera sea increiblemente sencilla (al menos comparada con otros esquemas).

Un socket queda determinado por tres parámetros.

- El ***dominio***, esto es, la familia de protocolos que se utilizará para la comunicación (X.25, IPv4, IPv6, AppleTalk,...)
- El ***tipo*** que especifica si es orientado a conexión (STREAM) o no (DGRAM) y si la lectura/escritura es bloqueante o no.
- El ***protocolo*** especifica un protocolo particular para ser utilizado con el socket. Normalmente existe un protocolo único para soportar un tipo de socket particular dentro de un dominio.

El módulo de socket de bajo nivel de Python (en el que todas las redes de nivel superior de Python- los módulos de trabajo están basados) soporta direcciones IPv4 e IPv6. También admite los protocolos de red más utilizados, incluido UDP, un protocolo no orientado a conexión ligero pero poco confiable donde los datos se envían como paquetes discretos (datagramas) pero sin garantía de que llegará, y TCP (Protocolo de Control de Transmisión), una conexión confiable y protocolo orientado a la transmisión. Con TCP, se puede enviar y recibir cualquier cantidad de datos: el socket es responsable de dividir los datos en fragmentos que son lo suficientemente pequeño como para enviar y para reconstruir los datos en el otro extremo.

Los protocolos FTP y HTTP están construidos sobre TCP, y las aplicaciones cliente/servidor normalmente usan TCP porque necesitan ser orientados a conexión y proporciona fiabilidad.

Sólo se van a tratar los sockets INET, pero éstos representan el 99% de los sockets que se usan. Y sólo se hablará de los STREAM sockets - a menos que realmente sepa lo que estás haciendo (en cuyo caso este documento no le será útil), conseguirá un comportamiento mejor y más rendimiento de un STREAM socket que de cualquier otro.

En este práctica crearemos una aplicación cliente / servidor básica: un servidor que espera solicitudes, y un cliente que envía solicitudes al servidor y lea la respuesta del servidor. Para que esto funcione, los clientes deben saber dónde conectarse al servidor, es decir, la IP del servidor dirección y número de puerto. Además, tanto el cliente como el servidor deben enviar y recibir datos utilizando un protocolo acordado utilizando formatos de datos que ambos entender.

Enunciado

En este práctica crearemos una aplicación cliente / servidor básica: un servidor que espera solicitudes, y un cliente que envía solicitudes al servidor y lea la respuesta del servidor. Para que esto funcione, los clientes deben saber dónde conectarse al servidor, es decir, la IP del servidor dirección y número de puerto. Además, tanto el cliente como el servidor deben enviar y recibir datos utilizando un protocolo acordado utilizando formatos de datos que ambos entender.

El servidor: Recibe mensajes del cliente y los envía de vuelta.
El cliente: Envía mensajes al servidor y los recibe de vuelta.

Salida del servidor

```
$ python ./socket_echo_server.py

starting up on localhost port 10000
waiting for a connection
connection from ('127.0.0.1', 52186)
received "This is the mess"
sending data back to the client
received "age.  It will be"
sending data back to the client
received " repeated."
sending data back to the client
received ""
no more data from ('127.0.0.1', 52186)
waiting for a connection
```

Salida del cliente

```
$ python socket_echo_client.py

connecting to localhost port 10000
sending "This is the message.  It will be repeated."
received "This is the mess"
received "age.  It will be"
received " repeated."
closing socket
```

Entrega

La entrega del ejercicio comprenderá:

- Los ficheros python relativos al cliente y al servidor
- Memoria explicativa del código entregado en los ficheros anteriores