IPN

Instituto Politécnico Nacional

UPIIZ

Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas

Ingeniería en:

Sistemas Computacionales

Materia:

Sistemas Distribuidos

Docente:

Héctor Alejandro Acuña Cid

Actividad:

“Práctica 01. Creación de sockets”

Alumno:

Alejandro Tamayo Castro

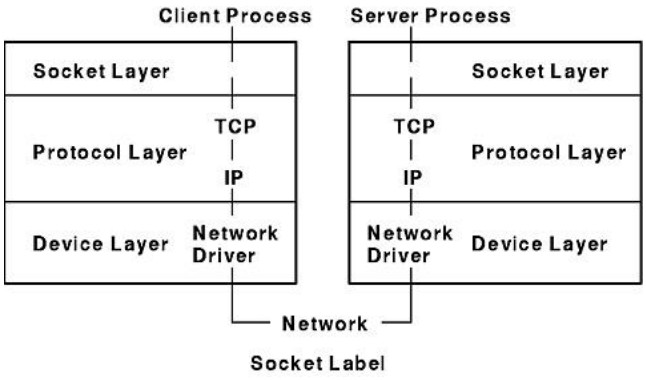
Grupo:

7CM1

Zacatecas, Zac. septiembre 14 de 2023

Introducción:

Los sockets se tratan de canales de comunicación entre procesos, permitiendo así el intercambio de datos locales o entre redes. Un único socket es un punto dinal de un canal de comunicación bidireccional. Estos cuentan con tres capas: **capa de socket**: proporciona interfaz entre las subrutinas y las capas inferiores, **capa de protocolo**: contiene los módulos de protocolo utilizados para la comunicación, y **capa de dispositivo:** contiene los controladores de dispositivo que controlan los dispositivos de red.



El modelo cliente-servidor consta de dos sockets, el del servidor que es el punto final de una via de acceso de comunicación bidireccional, se mantenga a la escucha de otro socket, en este caso, el socket del cliente, que es el otro extremo de la vida de comunicación, este se comunica con el servidor.

Descripción del Código:

Se implementan dos scripts de python, el socket del cliente y el socket del servidor, en el socket del servidor se importan los modulos socket y sys, el primero proporciona los componentes necesarios para trabajar con los sockets, el segundo provee el acceso a diferentes partes del sistema, para realizar tareas de configuracion y entorno del sistema al momento de la ejecucion.

En la línea 3, se inicializa el socket y se guarda en la variable S, para referenciar en otras pert4es del código, la clase socket.socket() requiere de 4 parámetros, 2 de ellos opcionales, ya que se configuran por defecto, de acuerdo a la documentación de del módulo, el primer parámetro que recibirá es la familia, en este caso AF_INET que se refiere al protocolo de internet a usar (IPV4 en este), el segundo es el tipo de socket, en el script se usa SOCK_STREAM que se refiere al protocolo TCP. En la linea 4 se guardan dos parámetros, la dirección del servidor, localhost pues se trabaja en la misma máquina, y el puerto en el que escuchará a los otros sockets, o a los clientes.En la linea 6 se unen los datos del servidor con el socket instanciado en la linea 3 para finalmente en la linea 7 comenzar con la escucha. Se inicia un ciclo While infinito, en el que se imprime el mensaje de espera y da paso a la impresión de errores del sistema al momento de ejecución, si el socket escucha una conexión y es aceptada, en la linea 10 se hace uso de dos variables: connection y datos_cliente, para guardar el resultado del método accept() que de acuerdo a la documentación devuelve un par de datos (conn, address), donde conn es un objeto tipo socket usado para enviar y recibir datos, y address que es la direccion del socket del cliente. En la linea 11 se comienza con la estructura *try… finally* que se asegura de reazliar todo el programa antes de terminar su ejecución, en la linea 14 comienza otro ciclo While infinito, en este caso se usa para guardar los datos que se reciben del cliente mediante el método recv() usando la coneccion recibida en la linea 10. En la linea 16 se imprime el mensaje recibido, para posteriormente en la linea 17 se pregunte si existe el mensaje recibido se proceda a devolver el mensaje tal cual a llegado, esto se hace mediante la linea 20 y el empleo del método sendall(), uniendo el mensaje mediante data. En la linea 18 se observa una instrucción comentada, esta se puede utilizar para

mandar un mensaje escrito desde el servidor. Si el mensaje de la linea 17 no existe, entonces se imprime que no existen mas datos y se imprimen los datos del cliente para finalizar con el segundo While infinito. Si ya terminó la ejecución completa de *try… finally* entonces se cierra la conexión con el cliente y se mantiene el ciclo infinto de escucha de conexiones.

```
1  import socket
2  import sys
3  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4  datos_servidor = ('localhost', 4000)
5  print (sys.stderr, 'Empezando a levantar %s puerto %s' % datos_servidor)
6  s.bind(datos_servidor)
7  s.listen(1)
8  while True:
9      print (sys.stderr, 'Esperando para realizar conexión')
10     connection, datos_cliente = s.accept()
11     try:
12         print (sys.stderr, 'conexión desde', datos_cliente)
13
14         while True:
15             data = connection.recv(19)
16             print (sys.stderr, 'recibido "%s"' % data.decode())
17             if data:
18                 print (sys.stderr, 'devolviendo mensaje al cliente')
19                 #data = input().encode()
20                 connection.sendall(data)
21             else:
22                 print (sys.stderr, 'no hay mas datos', datos_cliente)
23             break
24     finally:
25         connection.close()
```

Para el socket del cliente se configura de la misma manera, pues se emplea el protocolo IPV4 y TCP, se guardan los datos del servidor y posteriormente se imprimen para control de la conexión, en la linea 6 se observa el primer cambio, pues a diferencia del servidor, en este script se comienza la conexión con el servidor, en este punto no existe un While infinito, solo un *try… finally* que como se menciono anteriormente se trata de una estructura para asegurar la ejecución completa del programa antes de terminarlo, en la linea 8 se guarda un mensaje encriptado o codificado, para control visual se imprime el mensaje a enviar en la linea 9, mientras que en la linea 10 se envia completamente el mensaje con el método sendall(), se inicializa la variable cantidad_recibida en 0, esto en la linea 11 y en la linea 12 se

cuenta la longitud del mensaje que se envio, esto para que la linea 13 mediante el While, se pueda recibir el mensaje que el servidor envía, si la cantidad recibida es menor que la esperada, o en otras palabras, si el mensaje aun no se a recibido completamene. En la linea 16 se imprime el mensaje que se recibio y para terminar en la linea 18 se declara el final del try, en este punto se culmina la conexión y se termina la ejecución del script.

```
1  import socket
2  import sys
3  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4  datos_servidor = ('localhost', 4000)
5  print (sys.stderr, 'conectando a %s puerto %s' % datos_servidor)
6  s.connect(datos_servidor)
7  try:
8      mensaje = 'Aquí va el mensaje'.encode()
9      print (sys.stderr, 'enviando "%s"' % mensaje.decode())
10     s.sendall(mensaje) # enviando el mensaje
11     cantidad_recibida = 0
12     cantidad_esperada = len(mensaje) # recibiendo la respuesta
13     while cantidad_recibida < cantidad_esperada:
14         data = s.recv(19)
15         cantidad_recibida += len(data)
16     print (sys.stderr, 'recibiendo "%s"' %data.decode())
17
18 finally:
19     print (sys.stderr, 'cerrando socket')
20     s.close()
```

Si se desea enviar un mensaje personalizado al momento de iniciar el cliente, se puede modificar la linea 8 del mensaje, añadiendo a la linea el método imput para capturar entrada del teclado de la siguiente manera:

```
1  mensaje = input('Dame un mensaje: ').encode()
```

Resultados:

El resultado de ejecutar los scripts es el siguiente, al momento de iniciar el script del servidor se crea una tarea infinita que comienza con la escucha de peticiones, de ahí que imprima el mensaje esperando para realizar la conexión, despues de iniciar un script de cliente, el servidor detecta el intento de conexión y la acepta, imprime el mensaje “conexión desde ()”, y se imprimen los datos del cliente que se esta conectando, en este punto el puerto que pide la conexión es el 52243, que pertenece al cliente y se conecta al puerto 4000 del servidor, una vez que recibe el mensaje es impreso en pantalla y se devuelve al cliente, finalizado el envio del mensaje nuevamente vuelve a escuchar peticiones, por lo que el script del servidor nunca se termina automaticamente.

```
trator/Descargas/servidor.py
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> Empezando a levantar localhost puerto 4000
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> Esperando para realizar conexión
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> conexión desde ('127.0.0.1', 52243)
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> recibido "Aquí va el mensaje"
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> devolviendo mensaje al cliente
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> Esperando para realizar conexión
```

En el script del cliente, el que tiene un mensaje predefinido a enviar, al momento de iniciarlo se comienza con la conexión al servidor en su puerto 4000, una vez establecida la conexión se comienza con el envio del mensaje, para posteriormente recibirlo y cerrar la conexión, este script se termina de ejecutar completamente, lo que lo diferencia del script del servidor.

```
strator/Descargas/cliente.py
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> conectando a localhost puerto 4000
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> enviando "Aquí va el mensaje"
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> recibiendo "Aquí va el mensaje"
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> cerrando socket
```

Para enviar un mensaje personalizado solo cambiando la linea especificada, se agrega un paso para ingresar el mensaje para despues ejecutarse tal cual el script anterior.

```
istrator/Descargas/cliente02.py
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> conectando a localhost puerto 4000
Dame un mensaje: YSYA
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> enviando "YSYA"
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> recibiendo "YSYA"
<_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'> cerrando socket
```

Referencias:

IBM documentation. (s. f.-b). https://www.ibm.com/docs/es/aix/7.3?topic=concepts-sockets

Socket — low-level networking interface. (s. f.). Python documentation. https://docs.python.org/3/library/socket.html

SYS — System-specific Parameters and Functions. (s. f.). Python documentation. https://docs.python.org/3/library/sys.html