

Hilos y Socket

Diego Alexander Murillo Suárez

Fundación universitaria del areandina

Modelos de programación II - IS - 202460-6A - 61

Deyvis Morales

Sábado 10 de agosto del 2024

Introducción

En el siguiente documento se presentará el entendimiento de cómo se usan los hilos y socket, en este caso puntual realizaremos un juego como un pequeño MVP (Minimum viable product) que nos ayudará a entender cómo funciona la arquitectura cliente servidor o también conocida como client-server architecture.

Hilos y Socket

Objetivos

Con este documento lo que se quiere dar a entender cómo se utilizan estas estructuras de datos en el mundo real por medio de la arquitectura cliente servidor, esto con la ayuda del socket y cómo podemos implementarlo en este caso, con un juego en el que se adivinará un número que va a estar dentro de un rango puntual.

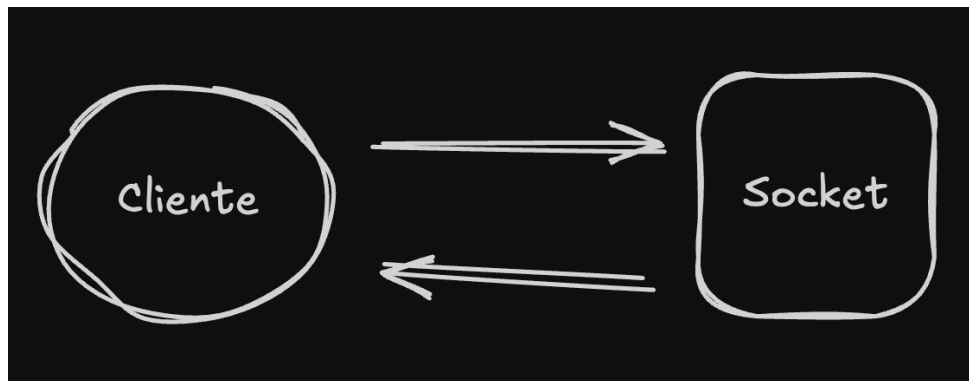
Ejercicio

Construir una aplicación en Python que permite aplicar el tema de hilos y socket, la aplicación consiste en:

- Construir un juego en el que se generan números aleatorios a través de hilos por parte del cliente.
- Establecer una comunicación entre el cliente y el servidor para que el servidor adivine los números generados por el cliente.
- Contabilizar los aciertos y desaciertos cuando finalice la aplicación.
- Cuando el servidor lleve 3 desaciertos seguidos debe salir un mensaje de “Perdiste”.
- Terminar la aplicación cuando el cliente envíe la palabra “terminar”

Solución

Para comenzar, les mostraré a continuación un pequeño dibujo sobre la arquitectura que usaremos en el ejercicio. Donde el cliente le va a enviar la información y el servidor directamente le responderá al cliente.



Para esto necesitamos 2 conexiones aparte que apunten hacia la misma dirección ip y que tengan un puerto diferente. En este caso les mostraré el código del archivo `client_server.py` el cual va a simular el cliente. Como cliente le mandaremos un rango de números y un número que debe adivinar dentro de ese intervalo de números. A continuación el código:

```
1  import socket
2
3  # Crear un socket de cliente TCP/IP
4  client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5
6  # Conectar el socket al servidor
7  server_address = ("localhost", 7536)
8  client_socket.connect(server_address)
9
10 # Enviar números
11 minimun_number = int(input("Ingrese el número mínimo:"))
12 maximun_number = int(input("Ingrese el número máximo: "))
13 number_to_guess = int(input("Ingrese el número a adivinar: "))
14 numbers = f"{minimun_number},{maximun_number},{number_to_guess}"
15 client_socket.sendall(numbers.encode())
16 # Esperar respuesta
17 data = client_socket.recv(1024)
18 print(f"Recibido del servidor: {data.decode()}")
19
20 # Cerrar el socket
21 client_socket.close()
```

En este caso, necesitaremos enviar la info con el método `encode`, esto debido a que el servidor sólo puede recibir bits, y en este caso, todos los tipos de datos están con la unidad de Bytes, siendo 1 Byte = 8 Bits. Si no lo usamos el programa nos dirá que el tipo de dato es diferente de bits y no podrá continuar con el proceso.

A continuación mostraremos las diferentes salidas o outputs que puede tener el cliente al ejecutarse:

```

Recibido del servidor: Adios.
❌ (MPII) habi@Mackbook-Habi Socket % python client_socket.py
Traceback (most recent call last):
  File "/Users/habi/Universidad/MDII/Socket/client_socket.py", line 8, in <module>
    client_socket.connect(server_address)
ConnectionRefusedError: [Errno 61] Connection refused

```

La imagen de arriba, indica que la conexión fue rechazada, esto debido a que el socket no está preñado, en caso de que sí, pueden haber dos caminos, poner los 3 números que se necesitan, o por el contrario poner en todos los números 0 para cerrar la conexión. A continuación ambos pasos:

```

(MPII) habi@Mackbook-Habi Socket % python client_socket.py
Ingrese el número mínimo:1
Ingrese el número máximo: 10
Ingrese el número a adivinar: 6
Recibido del servidor: Mensaje recibido. Intentos con exito: 0,intentos fallidos:1

```

En este caso menciona el cliente los 3 números que el cliente deseaba, sin embargo al final, se menciona que el servidor falló y se le suma un punto en los intentos fallidos, ahora revisemos en caso de que se coloque el número 0 en cada variable que se necesite:

```

Ingrese el número mínimo:0
Ingrese el número máximo: 0
Ingrese el número a adivinar: 0
Recibido del servidor: Adios!

```

Ahora, necesitamos el servidor o socket que recoja esos números que nos dio el cliente y que por medio de la librería random nos dé un número que el servidor adivine. Todo esto, independientemente si es el número que el cliente quería que adivinemos o no. Es por eso que necesitamos crear otro archivo aparte llamado server_socket.py que primeramente recoja los números y luego empiece a adivinar el número:

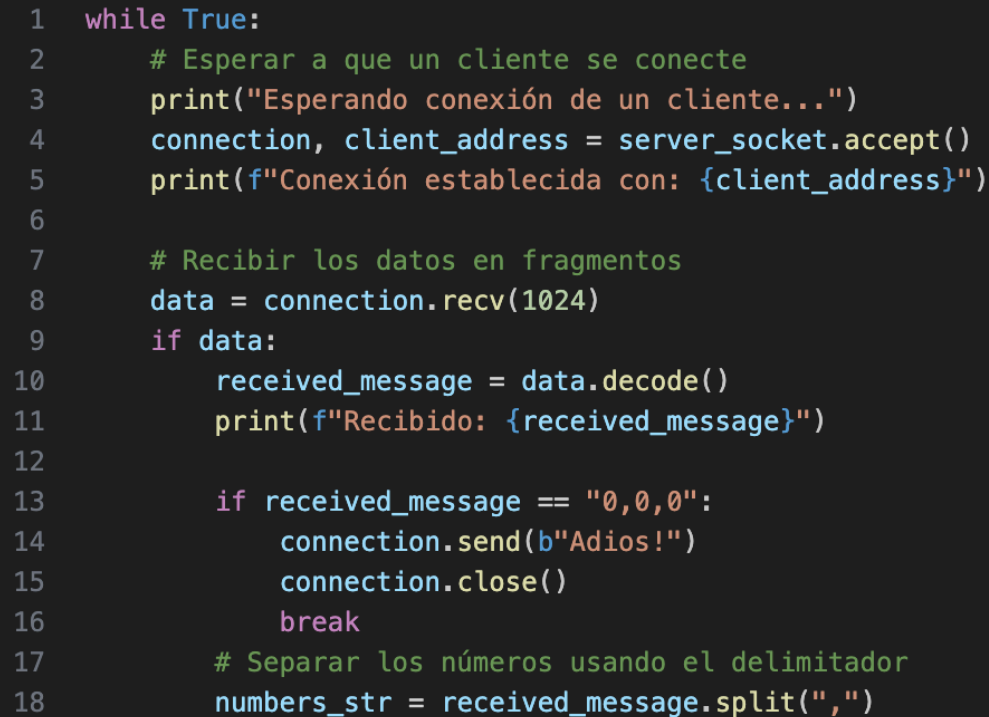


```
1  import socket
2  from utils import Utils
3  # Crear un socket de servidor TCP/IP
4  server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5
6  # Vincular el socket a la dirección y puerto
7  server_address = ("localhost", 7536)
8  server_socket.bind(server_address)
9
10 # Escuchar conexiones entrantes
11 server_socket.listen(5)
12 print(f"Servidor escuchando en {server_address}")
13
14
15 shifts_successes_quantity = 0
16 shifts_failures_quantity = 0
```

Como pueden ver acá en principio le diremos la dirección y puerto y la cantidad de direcciones que se pueden conectar, en este caso, 5, adicionalmente, pondremos la cantidad de intentos fallidos y exitosos que tenga el servidor a la hora de adivinar. Por el momento será de 0.

Ahora, usaremos un bucle infinito que a menos de que la respuesta del cliente no sea 0,0,0 entonces no parará de ejecutarse el bucle. En este caso, es lo que un socket hace, esperar la respuesta en vivo de un cliente, esto podría ser costoso a nivel laboral, es por eso que se deben buscar otras alternativas distintas al utilizar un socket cuando ya estás dentro de una empresa y tu

fin no sea crear temas como lo pueden ser videojuegos o chats o cosas que se deban hacer en vivo.



```
1  while True:
2      # Esperar a que un cliente se conecte
3      print("Esperando conexión de un cliente...")
4      connection, client_address = server_socket.accept()
5      print(f"Conexión establecida con: {client_address}")
6
7      # Recibir los datos en fragmentos
8      data = connection.recv(1024)
9      if data:
10         received_message = data.decode()
11         print(f"Recibido: {received_message}")
12
13         if received_message == "0,0,0":
14             connection.send(b"Adios!")
15             connection.close()
16             break
17         # Separar los números usando el delimitador
18         numbers_str = received_message.split(",")
```

Lo que hace el código anterior es que el cliente se conecte a la dirección del servidor, luego de eso, dividirá los números, todo esto con el fin de que no queden juntos con una cadena, sino que por el contrario, cada uno tenga su espacio en la memoria.

```

1  if len(numbers_str) >= 3:
2      minimun_number = int(numbers_str[0])
3      maximun_number = int(numbers_str[1])
4      number_to_guess = int(numbers_str[2])
5      print(
6          f"Números recibidos: Mínimo={minimun_number}, Máximo={maximun_number}, Adivinar={number_to_guess}"
7      )
8
9      if Utils.guess_number(minimun_number, maximun_number, number_to_guess):
10         shifts_successes_quantity += 1
11     else:
12         shifts_failures_quantity += 1

```

Ahora, lo que haremos es que efectivamente, una vez dividido el string de números, debemos formatear cada uno a entero para que en este caso, utilizaremos la función de la clase Utils, la cual la creamos en un archivo aparte, y tiene el método `guess_number`, la cuál es la función que nos ayudará a adivinar el número por parte del servidor. A continuación la clase Utils con el método `guess_number`:

```

1  import random
2  class Utils:
3
4      def guess_number(minimun, maximun, number_to_guess) -> bool:
5          successes_counter = 0
6
7          while successes_counter < 3:
8              random_number = random.randint(minimun, maximun)
9
10             if random_number == number_to_guess:
11                 print(f"El número {random_number} es el correcto")
12                 return True
13             elif random_number < number_to_guess:
14                 print(
15                     f"El número {random_number} es menor que el número que se quiere adivinar"
16                 )
17                 minimun = random_number + 1
18             elif random_number > number_to_guess:
19                 print(
20                     f"El número {random_number} es mayor que el número que se quiere adivinar"
21                 )
22                 maximun = random_number - 1
23
24             successes_counter += 1
25             print(f"Llevas {successes_counter} desacierto(s)")
26
27     print(f"Perdiste, el número a adivinar era {number_to_guess}")
28     return False

```


Este método nos ayudará a crear el contador de intentos exitosos y fallidos a la hora de adivinar el número.

Después de ejecutarse la función y hacer los incrementos respectivos, lo que haremos será imprimir estos valores tanto para el cliente como para el servidor por medio del siguiente código:

```

1  # Imprimir los contadores en el servidor
2      print(f"La cantidad de turnos exitosos es {shifts_successes_quantity}")
3      print(f"La cantidad de turnos fallidos es {shifts_failures_quantity}")
4
5      # Responder al cliente
6      message_to_send_to_client = f"Mensaje recibido. Intentos con éxito: {shifts_successes_quantity},
7      intentos fallidos:{shifts_failures_quantity}"
8      connection.sendall(message_to_send_to_client.encode())

```

Ahora miremos cómo se muestra en la consola antes de que el socket tenga una conexión:

```

(MPIL) habi@Mackbook-Habi Socket % python server_socket.py
Servidor escuchando en ('localhost', 7536)
Esperando conexión de un cliente...

```

Luego de que se haga la conexión con el cliente, recibirá los números que el cliente puso:

```

Recibido: 1,10,4
Números recibidos: Mínimo=1, Máximo=10, Adivinar=4
El número 4 es el correcto
La cantidad de turnos exitosos es 1
La cantidad de turnos fallidos es 0
Esperando conexión de un cliente...

```

En este caso, el servidor tuvo mucha suerte y adivino el número 4 a la primera, como pueden ver se suma un intento en la cantidad de turnos que fueron exitosos. Ahora veamos y pongamos un rango más difícil:

```

Conexión establecida con: ('127.0.0.1', 50537)
Recibido: 1,100,47
Números recibidos: Mínimo=1, Máximo=100, Adivinar=47
El número 62 es mayor que el número que se quiere adivinar
Llevas 1 desacierto(s)
El número 9 es menor que el número que se quiere adivinar
Llevas 2 desacierto(s)
El número 15 es menor que el número que se quiere adivinar
Llevas 3 desacierto(s)
Perdiste, el número a adivinar era 47
La cantidad de turnos exitosos es 1
La cantidad de turnos fallidos es 1
Esperando conexión de un cliente...

```

En este caso puntual como podemos ver, al llegar a 3 desaciertos se menciona que perdió debido a que no adivino el número en esa cantidad de intentos, por lo que se le suma un punto a la cantidad de turnos fallidos. Finalmente, si el usuario ingresa todos los números en 0, se cerrará el socket y se terminará el proceso, a continuación el ejemplo:

```

Conexión establecida con: ('127.0.0.1', 50547)
Recibido: 0,0,0
(MPII) habi@Mackbook-Habi Socket % 

```

Si bien, no se muestra nada de lado del servidor, se enviará un mensaje hacia el cliente mencionando “Adios!”

Discusión y conclusiones

Este ejercicio nos mostró de una manera pequeña y simple cómo funciona la arquitectura cliente servidor, sistemas modernos como la jam de spotify para escuchar música con tus amigos, usar documentos de google para realizar documentos en vivo o videojuegos online, todos estos sistemas mencionados anteriormente utilizan sockets para su correcto funcionamiento y tienen sus propios servidores. Sin embargo, si no se tiene una buena gestión o una buena lógica del negocio, por medio del socket pueden ocurrir ataques, es por eso que pueden haber hackers o cheaters dentro de los juegos en línea. O si en dado caso, intentan hackear un documento de google, lo único que podrán alterar es ese documento en específico, pero no tendrá mucho impacto con los demás servicios que puede tener Google.

Referencias

Python Organization. *Random - How To - Sockets programming*. Última actualización de la documentación el 14 de Septiembre del 2024 de <https://docs.python.org/es/3/howto/sockets.html>.

Python Organization. *Random - Generate pseudo-random numbers*. Última actualización de la documentación el 14 de Septiembre del 2024 de <https://docs.python.org/es/3/library/random.html>

Roberto Gomez. *Los sockets de unix*. <https://cryptomex.org/SlidesProg/Sockets.pdf>

Steve Chipman. *How does google docs work?*. Artículo creado el 8 de octubre del 2022 de [How does google docs work?](#).