UNIVERSIDAD IBEROAMERICANA

CIUDAD DE MÉXICO

Práctica Sockets

Fundamentos de Redes Digitales
Asignatura

Primavera 2022
Periodo

Edgar Ortiz Loyola Rivera Melo
Profesor

Jocelyne González Hernández
Sandra Lorena Quijada León
Alumna(s)

Objetivo

- Localizar los comandos adecuados para poder obtener el resultado esperado.
- Interpretar los datos obtenidos en Wireshark.
- Aprender a utilizar Python al desarrollar códigos funcionales.
- Entender cada una de las líneas de código realizadas.
- Especificar la funcionalidad de los comandos utilizados.
- Entender la importancia de los threads.

Resumen

En esta práctica pudimos ver la importancia de implementar threads en nuestros programas, aprendimos a utilizar Python y observar que tan distinto es en comparación con el lenguaje en C que ya conocemos. Como en las anteriores prácticas / actividades analizar el funcionamiento tanto Wireshark como en la terminal es muy importante, en Wireshark entiendes que es lo que se va realizando detalladamente, así que apoyándonos de esta herramienta pudimos aprender mucho más de nuestro código.

Desarrollo

El uso de threads en los programas es muy utilizado porque presenta ciertas ventajas como:

1. Aprovechar la capacidad de almacenamiento: Gracias a los threads podemos ejecutar varias tareas al mismo tiempo entonces mejoras la capacidad del ordenador y aceleras la ejecución de las tareas.
2. Mejora la capacidad de respuesta: Utilizando threads el programa puede ir respondiendo mientras se estén realizando otras tareas en segundo plano.
3. Beneficia a modular el código: Divide la funcionalidad del programa en tareas más pequeñas, lo cual ayuda también en el mantenimiento del código.

Es muy común ver que se utilizan los threads en lenguajes de programación de alto nivel como es en nuestro caso con Python. Es importante mencionar que, aunque los threads comparten la memoria, cada hilo es capaz de ejecutarse de manera independiente. Del mismo modo, mencionar los desafíos que conlleva utilizar threads tiene que ver con las condiciones de carrera o bloqueo que se pueden presentar ocasionando fallas en el programa, por lo que es importante saber implementarlos de manera correcta.

Desarrollo detallado de la práctica

Lo que se ve en pantalla

Para poder obtener los objetivos de la práctica fue necesario hacer dos programas en Python, uno para el cliente y otro para el servidor que funcionaran al mismo tiempo para poder ver cómo es que se va recibiendo y dando respuesta a los mensajes. Para poder ejecutarlo, primero es necesario correr la parte del servidor y después la del cliente.

Lo que el programa tenía que mostrar era, en primer lugar, un menú en donde aparecen todas las opciones que tiene el cliente para elegir dependiendo de la acción que quiera realizar, nuestras opciones de menú incluían mostrar la hora del servidor, contar el número de S en una cadena o mostrar el

contenido del directorio hogar. Se está ejecutando mediante un ciclo que le va preguntando al usua rio si desea seguir realizando más acciones o si quiere salir del programa, del mismo modo, si la opción que ingresaste no está dentro del parámetro muestra opción no válida.

Aquí se puede observar la parte del menú funcionando dependiendo a la selección del cliente:

```
cpb02-07@cpb0207-iMac:~/REDES/TCPPractica1/CG$ python3 CLIENT1.py
Elige una opcion
1. Mostrar la hora del servidor
2. Contar el número de 'S' en una cadena
3. Contenido del directorio hogar
Escoge una opción: 5

Opción no válida

 ¿Desea continuar? S / N: N
```

```
cpb02-07@cpb0207-iMac:~/REDES/TCPPractica1/CG$ python3 CLIENT1.py
Elige una opcion
1. Mostrar la hora del servidor
2. Contar el número de 'S' en una cadena
3. Contenido del directorio hogar
Escoge una opción: 1
La hora es:[20:26:37]

 ¿Desea continuar? S / N: S
Elige una opcion
1. Mostrar la hora del servidor
2. Contar el número de 'S' en una cadena
3. Contenido del directorio hogar
Escoge una opción: 2
Input lower sentence:sandra
El número de letras 'S' en la cadena es: 1

 ¿Desea continuar? S / N: S
Elige una opcion
1. Mostrar la hora del servidor
2. Contar el número de 'S' en una cadena
3. Contenido del directorio hogar
Escoge una opción: 3
El contenido del directorio hogar:['cpb02-07']

 ¿Desea continuar? S / N: N
```

Eso sería en general todo lo que el cliente iría viendo en su pantalla al ejecutar el programa.

Especificaciones de los códigos

Servidor

Comenzamos con la parte del servidor, en donde se utilizan ciertas bibliotecas necesarias como: import socket (permite la comunicación de red), import threading (ejecuta muchos hilos simultáneamente), import os (permite el acceso al sistema de archivos y directorios), from datetime import datetime (Se importa la clase datetime desde la biblioteca datetime para trabajar con fechas y horas)

Para o que nos sirve el comando print_lock = threading.Lock() es importante ya que protege la salida de muchos hilos.

Iniciamos con nuestro bucle infinito while TRUE para que todo el tiempo se estén mostrando las opciones del menú hasta que el usuario se quiera salir del programa, opcion = connectionSocket.recv(1024) va a recibir los datos enviados por el cliente a través del socket y los almacena en la variable "opcion". Para poder trabajar con los datos necesitamos decodificarlos, es por eso que se utiliza op = opcion.decode() .

Una vez realizado esto, utilizando if o elif pusimos las opciones de nuestro menú con los comandos necesarios para que realizara lo especificado.

Para la opción 1 se usó hora = datetime.now().strftime("[%H:%M:%S]") sirve para tener la hora actual, del mismo modo le decimos en que formato es en el que lo queremos mostrar en este caso 24 horas. Teniendo esto enviamos un mensaje en donde le decimos cuál es la hora enviando la cadena al cliente a través de un socket mediante el comando connectionSocket.send(message.encode()) .

En cuanto a nuestra opción 2 recibimos los datos del cliente a través del socket y los almacena en la variable "sentence", mediante el código sentence = connectionSocket.recv(1024), del mismo modo decodificamos los datos para poder manipularlos Aquí convertimos la cadena en mayúsculas por medio del comando mayuscula = msg.upper(), contamos el número de veces que aparece la letra “S” en la cadena utilizando count_s = mayuscula.count("S") y mostramos con un mensaje para el usuario el número de “S” que tuvo la cadena ingresada, por último con connectionSocket.send(message.encode()) enviamos la cadena al cliente.

Para la opción 3 definimos la ruta del directorio del cual queremos obtener el contenido en nuestro caso con dir_path = "/home" , obtenemos la lista de todos los archivos y subdirectorios del directorio especificado con dir_list = os.listdir(dir_path), le mostramos al usuario un mensaje que muestre la lista del contenido del directorio hogar, utilizamos connectionSocket.send(message.encode()) para enviárselo al cliente.

Creamos la condición por si la opción que ingresan no es la válida y cerramos la conexión con el socket.

En el main, se utilizó serverPort = 12001 que es un puerto en el cual el servidor va a estar escuchando las conexiones entrantes, serverSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) utiliza el protocolo de internet (IP) y el de transporte (TCP) para las conexiones entrantes después de crear nuestro socket necesitamos colocar serverSocket.bind(("192.168.127.215", serverPort)) que lo que va a hacer es asociar una dirección IP con un puerto (la dirección IP de la máquina y el puerto previamente definido), serverSocket.listen(5) va a escuchar las conexiones entrantes en el socket, el 5 indica que puede escuchar hasta 5 conexiones pendientes, se genera un while TRUE para hacer un bucle infinito que nos permita siempre estar escuchando las conexiones, a continuación se muestra un mensaje por parte del servidor, el comando print_lock.acquire() permite sincronizar la salida en la consola si hay varios subprocesos ejecutándose al mismo tiempo.

Algunos de los comandos de vital importancia son client_thread = threading.Thread(target=threaded, args=(connectionSocket, addr)) y client_thread.start() porque nos permiten crear subprocesos.

Por último, cuando el bucle infinito se termina, el socket del servidor se cierra para liberar los recursos utilizados por el servidor.

Cliente

Se importan las bibliotecas socket y threading previamente ya mencionado su funcionamiento en el programa, con serverName = "192.168.127.215" y serverPort = 12001 definimos la dirección IP y el puerto, del mismo modo es necesario crear un socket utilizando el protocolo de internet (IP) y el de transporte (TCP) para conectarnos con el servidor, utilizando clientSocket.connect((serverName, serverPort)) nos conectamos al servidor usando la IP y puerto definidos anteriormente.

En el main, realizamos igual un ciclo infinito en donde se van a estar mostrando las opciones del menú hasta que el usuario decida salir, cuando el usuario selecciona una opción se envía al servidor a través del socket del cliente y dependiendo de la opción seleccionada se realizan las operaciones necesarias para que funcione correctamente: si la opción es 1, el cliente recibe la hora del servidor y la muestra en la consola, si es 2 el cliente ingresa una cadena y envía esa cadena al servidor. El servidor devuelve el número de veces que aparece la letra "S" en la cadena y el cliente lo muestra en la consola. Por el contario si es 3, el servidor envía el contenido del directorio hogar del servidor y el cliente lo muestra en la consola y si no entra entre las opciones permitidas, se muestra un mensaje de error.

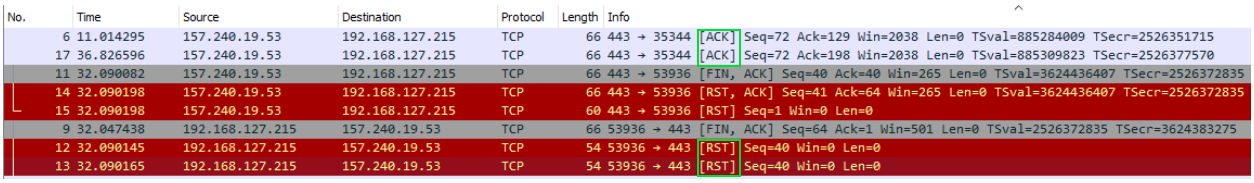
Tenemos que recordar que se va a ir estando mostrando la opción para ver si el usuario desea continuar con las solicitudes o se quiere salir, si el usuario ingresa la letra N, lo que sucede es que el bucle infinito se rompe y el programa termina.

Proceso en Wireshark

Como ya se mencionó, lo que vimos en la terminal es muy importante ya que se observó que el resultado sí fue el esperado, pero Wireshark a lo que nos ayuda es a entender más a fondo cómo es que se hacen dichas solicitudes, nos indica cuales son los puertos de entrada y de salida, si hubo algún error estableciendo la conexión entre muchos más factores que nos permiten entender mejor la práctica.

Primero abrimos Wireshark y le dimos click en el botón de la aleta para empezar con la capturación de datos, continuamos ejecutando el servidor y una vez que nos dio el mensaje de que está listo para recibir TCP ejecutamos el del cliente, seleccionamos la opción de nuestro menú haciendo todas las acciones que quiera el usuario hasta que decida colocar la N para salir del bucle y terminar el programa. Una vez que se colocó la letra N, continuamos dándole click al botón rojo para detener la captura de los datos.

Con los datos obtenidos como salen muchos, es necesario hacer un filtrado en la barra que aparece colocando tcp ya que de esta manera es mucho más fácil analizarlos. Lo que podemos ver en la información son dos tipos de paquetes que es muy importante saber lo que significan ya que te van a decir de lo que se encargan los cuales son: TST (Transmition Start) que para lo que nos va a servir es para iniciar la transmisión que se utiliza para generar un paquete TST al dispositivo destino para iniciar la conexión y establecer los parámetros iniciales de la transmisión, el siguiente paquete es ACK (Acknowledgement) el cual se utiliza para confirmar que llegó el paquete enviado desde el origen al destino, cuando el dispositivo de destino recibe un paquete de datos, envía un paquete ACK de vuelta al dispositivo de origen para confirmar que el paquete se recibió correctamente.



No.	Time	Source	Destination	Protocol	Length	Info
6	11.014295	157.240.19.53	192.168.127.215	TCP	66	443 → 35344 [ACK] Seq=72 Ack=129 Win=2038 Len=0 TSval=885284009 TSecr=2526351715
17	36.826596	157.240.19.53	192.168.127.215	TCP	66	443 → 35344 [ACK] Seq=72 Ack=198 Win=2038 Len=0 TSval=885309823 TSecr=2526377570
11	32.090082	157.240.19.53	192.168.127.215	TCP	66	443 → 53936 [FIN, ACK] Seq=40 Ack=40 Win=265 Len=0 TSval=3624436407 TSecr=2526372835
14	32.090198	157.240.19.53	192.168.127.215	TCP	66	443 → 53936 [RST, ACK] Seq=41 Ack=64 Win=265 Len=0 TSval=3624436407 TSecr=2526372835
15	32.090198	157.240.19.53	192.168.127.215	TCP	60	443 → 53936 [RST] Seq=1 Win=0 Len=0
9	32.047438	192.168.127.215	157.240.19.53	TCP	66	53936 → 443 [FIN, ACK] Seq=64 Ack=1 Win=501 Len=0 TSval=2526372835 TSecr=3624383275
12	32.090145	192.168.127.215	157.240.19.53	TCP	54	53936 → 443 [RST] Seq=40 Win=0 Len=0
13	32.090165	192.168.127.215	157.240.19.53	TCP	54	53936 → 443 [RST] Seq=40 Win=0 Len=0

Conclusiones

- Logramos comprender la manera en que se van enviando las solicitudes de los mensajes y cómo es que se van contestando para poder determinar si se estableció la conexión y pudo entender el mensaje.
- Fue interesante ver cómo el cliente y el servidor se conectan, ya que como siempre nada más eres el cliente no alcanzas a apreciar que es lo que está haciendo el servidor para que efectivamente te proporcione la respuesta que quieres obtener.
- Logramos entender cómo es que se debe de programar en Python, vimos ciertas coincidencias y algunas diferencias como el printf y el else if que en Python se llaman de otra manera.
- Entendimos todos los comandos descritos anteriormente, ya que gracias a ese nuevo conocimiento logramos hacer un programa que siguiera las especificaciones dadas.
- A medida que fuimos avanzando en la práctica se nos hacía mucho más fácil ver la comunicación que iba existiendo entre el cliente y el servidor.
- De manera práctica vimos la importancia de la utilización de threads en el programa.
- Realizamos el programa solicitado con éxito aprendiendo demasiado por cada una de las etapas que tuvimos que pasar para conseguirlo.

Referencias de consulta

1. *Threads y Procesos. (2023). CódigoFacilito.* https://codigofacilito.com/articulos/threads-procesos
2. *Juan (Código Pitón. (2022, November 3). Cómo Usar Hilos (Threads) en Python - Código Pitón. Código Pitón.* https://www.codigopiton.com/como-usar-hilos-o-threads-en-python/
3. *Socket Programming with Multi threading in Python. (2017, September 30). GeeksforGeeks; GeeksforGeeks.* https://www.geeksforgeeks.org/socket-programming-multi-threading-python/
4. *Socket Programming in Python. (2017, June 20). GeeksforGeeks; GeeksforGeeks.* https://www.geeksforgeeks.org/socket-programming-python/