Práctica Autoguiada - Interfase Sockets API

Objetivo: El objetivo es que esta guía sirva para comprender, mediante algunos ejemplos básicos, cómo trabajar con la "API de sockets", desde un lenguaje de alto nivel como Python. Se eligió este lenguaje de programación porque éste, en combinación con su módulo sockets, permite entender la API en cuestión con sus conceptos originales; sin detenerse en los detalles técnicos de implementación que exige un lenguaje más cercano a la máquina como C.

Consignas La idea es que esta guía sirva para comprender, mediante algunos ejemplos básicos, cómo funciona la API de Sockets, tanto desde un programa cliente como servidor, utilizando el stack UDP primero y luego TCP. Se utilizará la versión 3 del lenguaje de programación Python, disponible en múltiples plataformas y sistemas operativos. Si bien no es estrictamente necesario ninguna introducción "fuerte" en el lenguaje, se recomienda leer los primeros capítulos del tutorial en español disponible aquí: https://tutorial.python.org.ar/

Para más documentación y tutoriales, ver: http://python.org.ar/AprendiendoPython

1. Sockets UDP

1.1. Cliente UDP

Escriba el siguiente script, `udp_echo_client.py` :

```
#!/usr/bin/env python3
import socket

HOST = 'localhost'
PORT = 3500
LONG_MENSAJE = 1024
mensaje = 'TyR: Echo sobre UDP'

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(mensaje.encode(), (HOST, PORT))
data = s.recvfrom(LONG_MENSAJE)
s.close()
print ('-->', data[0].decode())
```

1.2. Servidor UDP

Escriba el siguiente script, `udp_echo_server.py` :

```
#!/usr/bin/env python3
import socket

HOST = 'localhost'      # IP o Hostname donde escucha
PORT = 3500             # Puerto de escucha
LONG_MENSAJE = 1024     # Cantidad maxima de bytes a aceptar

# Se crea un socket de tipo Internet (AF_INET) sobre UDP (SOCK_DGRAM)
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind((HOST, PORT))    # Indicamos al socket la direccion IP y Port de escucha
```

```
print ('Socket creado, escuchando en %s:%i' % (HOST, PORT))

while True:
    # Se obtienen los datos desde el sock cliente
    print ('Esperando clientes...')
    data, address = s.recvfrom(LONG_MENSAJE)
    print ('Conexion desde: %s:%i' % (address[0], address[1]))
    # Descomentar la siguiente línea para imprimir por consola el mensaje recibido
    # print ('Recibi: %s' % data.decode())
    # Si el cliente envió algún contenido distinto de un string vacío
    if data:
        s.sendto(data, address)     # Enviamos el echo al cliente
```

1.3. Ejecución y pruebas

1.3.1. Ejecute primero el script `udp_echo_server.py` en una terminal, se verá que imprime por pantalla un mensaje de creación del socket primero y luego de espera por clientes para atender.

1.3.2. Puede validar en otra terminal, mediante el comando `netstat -aunp | grep python`, que el servidor está escuchando por peticiones. De la salida de este comando, puede tomarse el número entero del proceso para verificar que este existe en el árbol de procesos de ps, con el comando `ps -ax | grep <número_de_proceso>`.

1.3.3. En otra terminal, ejecute el cliente `udp_echo_client.py`. Verá que el cliente imprime un mensaje `--> TyR: Echo sobre UDP` y termina. Si se cambia a la terminal del servidor, se verá que se imprimió un mensaje `Conexion desde 127.0.0.1:<numero_puerto_efimero>` y luego el servidor vuelve a quedar a la espera de clientes.

1.4. Validar el tráfico intercambiado con Wireshark

Ahora, vuelva a ejecutar el cliente pero realizando una captura del tráfico UDP puerto 3500 con Wireshark. Observe el tráfico y el comportamiento. Haga una captura 'en vivo' y modifique el script para tomarse pausas de varios segundos con time.sleep() en diferentes partes del código y ver qué sucede.

1.5. Consignas a implementar

1.5.1. Ejecutar el cliente sin haber ejecutado el servidor previamente (puede detener la ejecución con `Ctrl+C`). ¿Qué es lo que sucede? ¿Se puede evitarlo? ¿Cómo? Analizar la situación.

1.5.2. Modificar el script para que en el server se valide la dirección IP del cliente. Ignorar el mensaje recibido si proviene de una dirección IP no reconocida y procesarlo como hasta ahora en caso contrario.

1.5.3. Experimentar variando el *buffer* de recepción a un menor valor que el mensaje que envía el cliente y/o enviar un mensaje más grande. ¿Qué sucede?

1.5.4. Modificar ambos programas de manera tal que el servidor permita interactivamente responder el mensaje ingresando algo por teclado -ver función input()-, el cliente imprima lo imprima por pantalla y de lugar a que éste responda nuevamente, generando una intercambio tipo "chat" entre ambos participantes. Analice el modo de funcionamiento desde el punto de vista del usuario respecto al código.

1.5.5. Implementar un mensaje en el protocolo de mensajes que provoque que el servidor termine su ejecución.

1.5.6. Implementar un mensaje en el protocolo de mensajes que provoque la ejecución de un comando en el servidor con os.system() y éste devuelva el código de salida (*exit status*) al cliente.

2. Sockets TCP

2.1. Cliente TCP

Escriba el siguiente script, tcp_echo_client.py :

```
#!/usr/bin/env python3
import socket

HOST = '127.0.0.1'       # IP del servidor
PORT = 3500              # Puerto del servidor
MSG = 'Hola Mundo!'
LONG_MENSAJE = 1024      # Cantidad maxima de bytes a leer del stream

# Se crea un socket de tipo Internet (AF_INET) sobre TCP (SOCK_STREAM)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Nos conectamos al host y puerto del servidor
s.connect((HOST, PORT))
# Enviamos un string
s.sendall(MSG.encode())
# Leemos la respuesta del socket y lo cargamos en la variable 'data'
data = s.recv(LONG_MENSAJE)
# Cerramos el socket contra el server
s.close()
print ('Recibido: ', data.decode())
```

2.2. Servidor TCP

Escriba el siguiente script, tcp_echo_server.py :

```
#!/usr/bin/env python3
import socket

HOST = '127.0.0.1'        # IP o Hostname donde escucha
PORT = 3500               # Puerto de escucha
maxpedidos = 5            # Cantidad de pedidos acumulados (backlog)
LONG_MENSAJE = 1024       # Cantidad maxima de bytes a aceptar

# Se crea un socket de tipo Internet (AF_INET) sobre TCP (SOCK_STREAM)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Seteo opcion REUSEADDR del sol_socket
# Ver https://hea-www.harvard.edu/~fine/Tech/addrinuse.html
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
s.bind((HOST, PORT))      # Indicamos al socket la direccion IP y puerto
s.listen(maxpedidos)      # Indicamos el nro max de pedidos que aceptara
print ('Socket creado, escuchando en %s:%i' % (HOST, PORT))

while True:
    client, address = s.accept()    # Acepta las conexiones
    # Se obtienen los datos desde el socket cliente
    data = client.recv(LONG_MENSAJE)
    print ('Conexion desde: %s:%i' % (address[0], address[1]))
    # Descomentar la siguiente línea para imprimir por consola el mensaje recibido
    # print ('Recibi: %s' % data.decode())
    # Si el cliente envió algún contenido distinto de un string vacío
    if data:
        client.send(data)         # Enviamos el echo al cliente
    client.close()                # Cerramos el sock cliente
```

2.3. Ejecución y pruebas

2.3.1. Ejecute primero el script `tcp_echo_server.py` en una terminal, se verá que imprime por pantalla un mensaje de creación del socket primero y luego de espera por clientes para atender.

2.3.2. Puede validar en otra terminal, mediante el comando `netstat -atnp | grep python`, que el servidor está escuchando por peticiones. De la salida de este comando, puede tomarse el número entero del proceso para verificar que este existe en el árbol de procesos de ps, con el comando `ps -ax | grep <número_de_proceso>`.

2.3.3. En otra terminal, ejecute el cliente `tcp_echo_client.py`. Verá que el cliente imprime un mensaje `Recibido: Hola Mundo!` y termina. Si se cambia a la terminal del servidor, se verá que se imprimió un mensaje `Conexion desde 127.0.0.1:<numero_puerto_efimero>` y luego el servidor vuelve a quedar a la espera de clientes.

2.4. Validar el tráfico intercambiado con Wireshark

Ahora, vuelva a ejecutar el cliente pero realizando una captura del tráfico TCP puerto 3500 con Wireshark. Observe el tráfico y el comportamiento. Haga una captura 'en vivo' y modifique el script para tomarse pausas de varios segundos con time.sleep() en diferentes partes del código y ver qué sucede.

2.5. Consignas a implementar

2.5.1. Establecer la cola de `s.listen()` en 1 y ejecutar dos clientes a la vez. ¿Qué sucede?

2.5.2. Experimentar reduciendo la cantidad de bytes enviados desde el cliente a tamaños menores que el tamaño mismo del mensaje, utilizando `send()` en lugar de `sendall()`. ¿Qué se debe hacer para enviar todo el mensaje en tal caso? ¿Se podría describir cómo funciona `send()` a diferencia de `sendall()`?

2.5.3. Experimentar reduciendo la cantidad de bytes recibidos en el servidor con `recv()`, ¿Qué se debe hacer para recibir todo lo que el cliente desea enviar?

2.5.4. ¿Qué se debe hacer para enviar un mensaje de texto y una imagen por el mismo socket TCP? Analizar alternativas y proponer soluciones.