Міністерство освіти і науки України
Національний авіаційний університет
Факультет аеронавігації, електроніки та телекомунікацій
Кафедра телекомунікаційних та радіоелектронних систем

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни «Базові протоколи транспортування інформації»

На тему: «АНАЛІЗ ФУНКЦІОНУВАННЯ СТЕКУ ПРОТОКОЛІВ TCP/IP У МЕРЕЖАХ ІНТЕРНЕТ»

студента групи ТК-305
Іващенка А.В.
Перевірив
Чуприн В.М.

Київ-2020

ЗМІСТ

ВСТУП

Стек (тобто, певним чином упорядкований набір) телекомунікаційних протоколів TCP/IP – це технологічна основа Інтернету. Практично усі пакетні (тобто, не телефонні, що встановлюють безпосередні фізичні з'єднання між абонентами) наземні телекомунікаційні мережі наразі використовують цей стек для передачі будь-яких даних через будь-які різнорідні мережі зв'язку незалежно від того, де саме розташовані користувачі послуг цих мереж, хоч у Нью-Йорку, хоч поруч у ваших власних приміщеннях. З технічної точки зору безпрецедентна привабливість цього стеку пов'язана з його функціональною можливістю забезпечити якісний інформаційний зв'язок між будь-якими комп'ютерами, абсолютно не цікавлячись, де саме ці комп'ютері розташовані і через які саме мережі будуть просуватися дані, котрі належать кінцевим користувачам ресурсів мережі. Топологія цих мереж не має ніякого значення, і кому саме вони належать теж не має значення. З організаційної точки зору таке позбавляє необхідності і користувачам мереж і їхнім адміністраторам

вступати у будь-які бюрократичні і фінансові взаємостосунки з володарями фрагментів мереж, що лежать на шляху між кінцевими пунктами передавання даних. Тобто, користувачі Інтернету фактично не залежать від «бажань» власників та адміністраторів канального та вузлового обладнання. Оскільки «недоговірних» адміністраторів нескладно обійти. Останнім вкрай тяжко за цих умов встановлювати свої «правила гри», займатися монополізацією на ринку надання телекомунікаційних послуг, щось забороняти, на власний розсуд встановлювати ціни на послуги і т.ін. Таку «демократію» людство вже оцінило. Конкуренти цьому стеку щось не проглядаються. Яскравий приклад. Китайська влада вже давно вчиняє всілякого роду перешкоди у наданні послуг Інтернет для пересічних

громадян КНР. Але зацікавлені громадяни, якщо вони дійсно зацікавлені, завжди знаходять шляхи подолання цих перешкод. Таке можливе завдяки функціональним властивостям стеку протоколів TCP/IP. Особливості функціонування стеку протоколів TCP/IP у мережах Інтернет не так просто засвоїти. Багато людей, у принципі, можуть надати відповідь на питання: «Що таке стек протоколів TCP/IP?». Але впевнено пояснити, як він працює, утруднюються. Зокрема, не можуть відобразити часову послідовність перетворень у форматах даних починаючи із рівня прикладних програм і до форматів фізичного рівня, якщо спиратися на семирівневу модель інформаційної взаємодії прикладних систем, а також у зворотному напрямі на приймальній стороні. Можливо також відобразити процес функціонування обладнання за цим стеком відповідно до моделі TCP/IP. Є невелика різниця між цими моделями, що утруднює розуміння роботи даного стеку, особливо з точки зору термінології. Необхідно вкрай обережно і вдумливо користуватися термінологією під час опису телекомунікаційних процесів у пакетних мережах. Чітко розуміти, яка саме модель інформаційної взаємодії використовується для опису процесів, що реалізують протоколи, які є об’єктом розгляду.

ОСНОВНА ЧАСТИНА

Для демонстраці роботи я створю за допомогою мови програмування python простий tcp сервер який буде слухати вхідні повідомлення, а у відповідь відпраляти те саме повідомлення тільки у верхньому регістрі, та проілюструю роботу у вигляді діаграм.

Строримо файл server.py

```python
import socket

# Задаем адрес сервера
HOST = 'localhost'
PORT = 5555

# ------------------------------------Настройки сокета-----------------------------------------------------------

# создание сокета, первый параметр указывает какоя версия IP будет использована если AF_INET значит IPV4...
# если AF_INET6 значит IPV6, вторым параметром мы указывает какой протокол передачи данных мы будет использывать...
# если SOCK_STREAM значит протокол TCP, если  SOCK_DGRAM протокол UDP, все данные взяты и документации python https://docs.python.org/3/library/socket.html#socket-objects
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# занимаем в системе(биндим) адресс и порт для прослушки входящих пакетов(дейтаграмм)
server_socket.bind((HOST, PORT))

# С помощью метода listen мы запустим для данного сокета режим прослушивания
# Метод принимает один аргумент — максимальное количество подключений в очереди
server_socket.listen(1)

# вывод информации в консоль о том что сервер запущен
print('Сервер успешно запущен')

# ----------------------------------------------------------------------------------------------------------------
```

Оскільки сервер запущено локально, І кліент також будет локальною програмою в змінну HOST можна записати localhost, 127.0.0.1 або просто залишити пусту стрічку, PORT потрібно вибирати в діапазоні від в 0 до 65535.(Слід зазначити, що в більшості операційних систем прослуховування портів з номерами 0 - 1023 вимагає особливих привілеїв.)

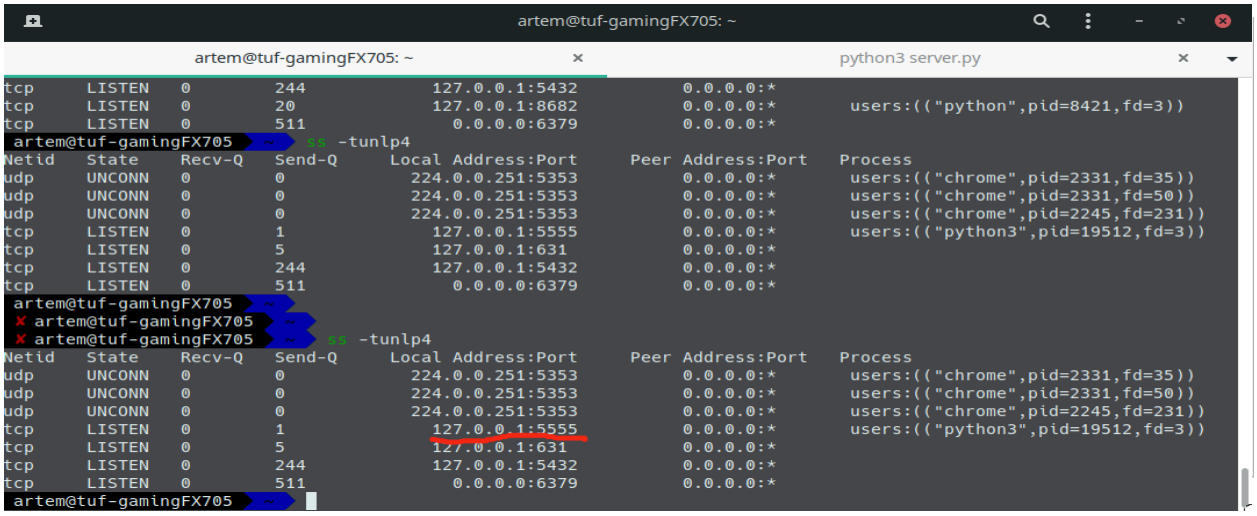
Щоб не писати до кожної стрічки коду пояснення у звіті я написав коментарі прямо у програмі.

```python
# ------------------------------------------------Прослушка запросов-----------------------------------------------
while True:
    # в 2 переменные записываються данные от нового подключения - сокет и адрес отправителя
    connection, address = server_socket.accept()

    # выводим информацию в консоль об новом подключении
    print(f"Новое подключение, адрес - {address[0]}:{address[1]}")

    # Т.к. мы не можем точно знать, что и в каких объемах клиент нам пошлет, то мы будем получать данные от него небольшими порциями.
    # Чтобы получить данные нужно воспользоваться методом recv, который в качестве аргумента принимает количество байт для чтения.
    # Мы будем читать порциями по 1024 байт (или 1 кб)
    data = connection.recv(1024)

    # вывод в консоль того сообщения что отправил на клиент
    print(data)

    # в ответ на входящие данные сервер будет отправлять клиенту ту же самую строку только в верхнем регистре
    connection.send(bytes(data.upper(), encoding='utf-8'))

    connection.close()

# ----------------------------------------------------------------------------------------------------------------
```

Запустимо код.

Чудово, у консолі ми бачимо повідомлення про успішний старт серверу, щоб переконатися що сервер прослуховує 5555 port, я запущу системну команду **ss** щоб це перевірити

(це інструмент, який використовується для виведення мережевий статистики **ss** дає більш докладні відомості про TCP-підключення і про станах з'єднань, ніж більшість інших інструментів. Зокрема, ss може виводити дані про таких сутності, як PACKET, TCP, UDP, DCCP, RAW, і сокети домена Unix.)



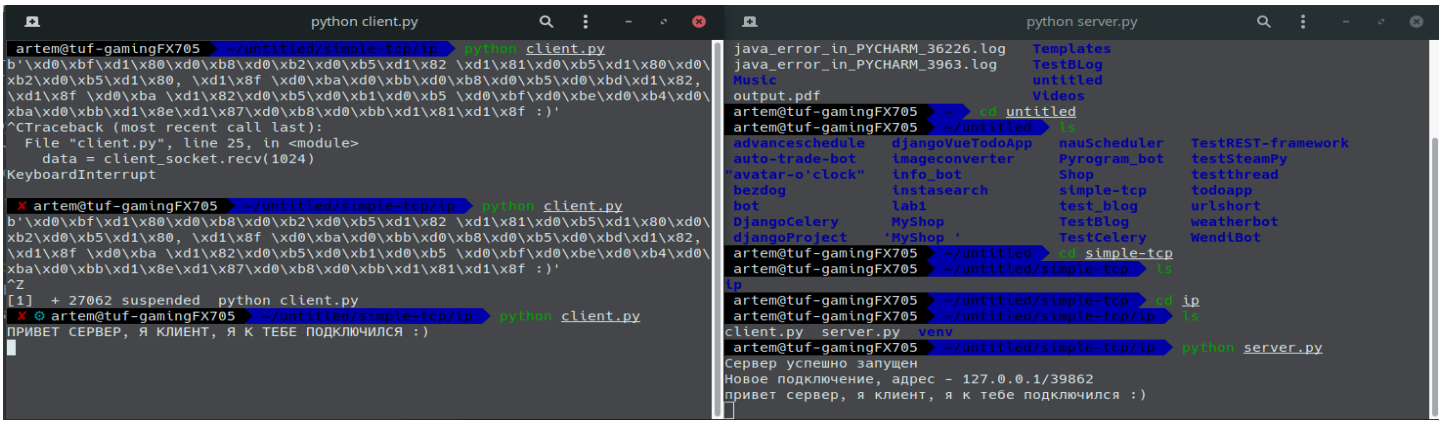
Як можна побачити все працює, чудово, тепер створимо клієнта який будет відправляти якісь дані на наш сервер.

Створимо client.py

```python
import socket

# адрес сервера
CONNECT_HOST = 'localhost'

# порт сервера
CONNECT_PORT = 5555

# создание КЛИЕНТСОГО сокета, первый параметр указывает какоя версия IP будет использована если AF_INET значит IPV4...
# если AF_INET6 значит IPV6, вторым параметром мы указывает какой протокол передачи данных мы будет использывать...
# если SOCK_STREAM значит протокол TCP, если  SOCK_DGRAM протокол UDP, все данные взяты и документации python https://docs.python.org/3/library/socket.html#socket-objec
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# устанавливаем соедение с сервером
client_socket.connect((CONNECT_HOST, CONNECT_PORT))

# отправляем серверу новое сообщение
client_socket.send(bytes('привет сервер, я клиент, я к тебе подключился :)', encoding='utf-8'))

while True:

    # Т.к. мы не можем точно знать, что и в каких объемах сервер нам пошлет, то мы будем получать данные от него небольшими порциями.
    # Чтобы получить данные нужно воспользоваться методом recv, который в качестве аргумента принимает количество байт для чтения.
    # Мы будем читать порциями по 1024 байт (или 1 кб)
    data = client_socket.recv(1024)
    if data:
        # выведем в консоль сообщение которое пришло нам от сервера
        print(data.decode('utf-8'))
```

Код схожий на попередній, за виключенням того що тут ми підключаємося до серверу.

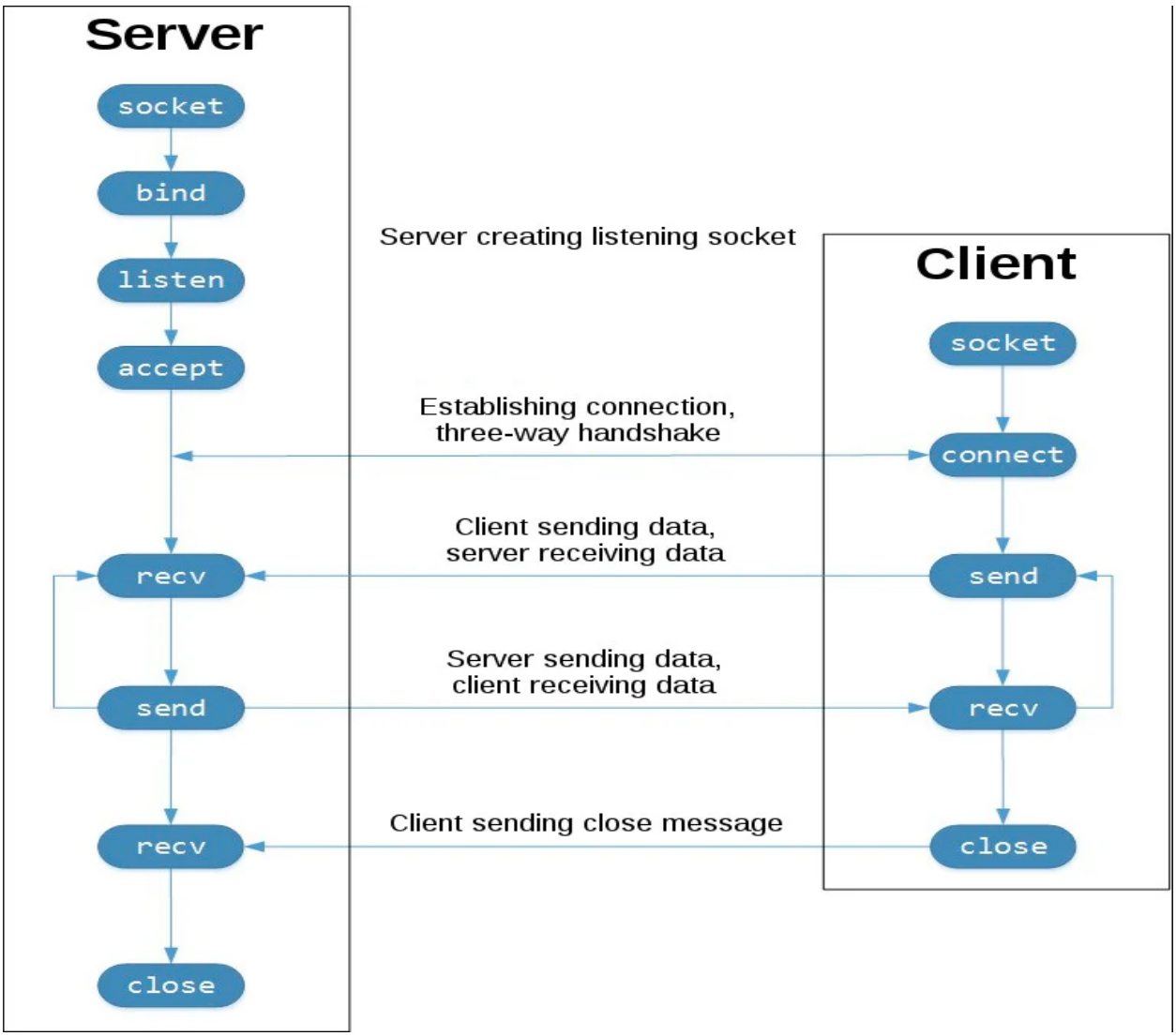
Запустимо код.



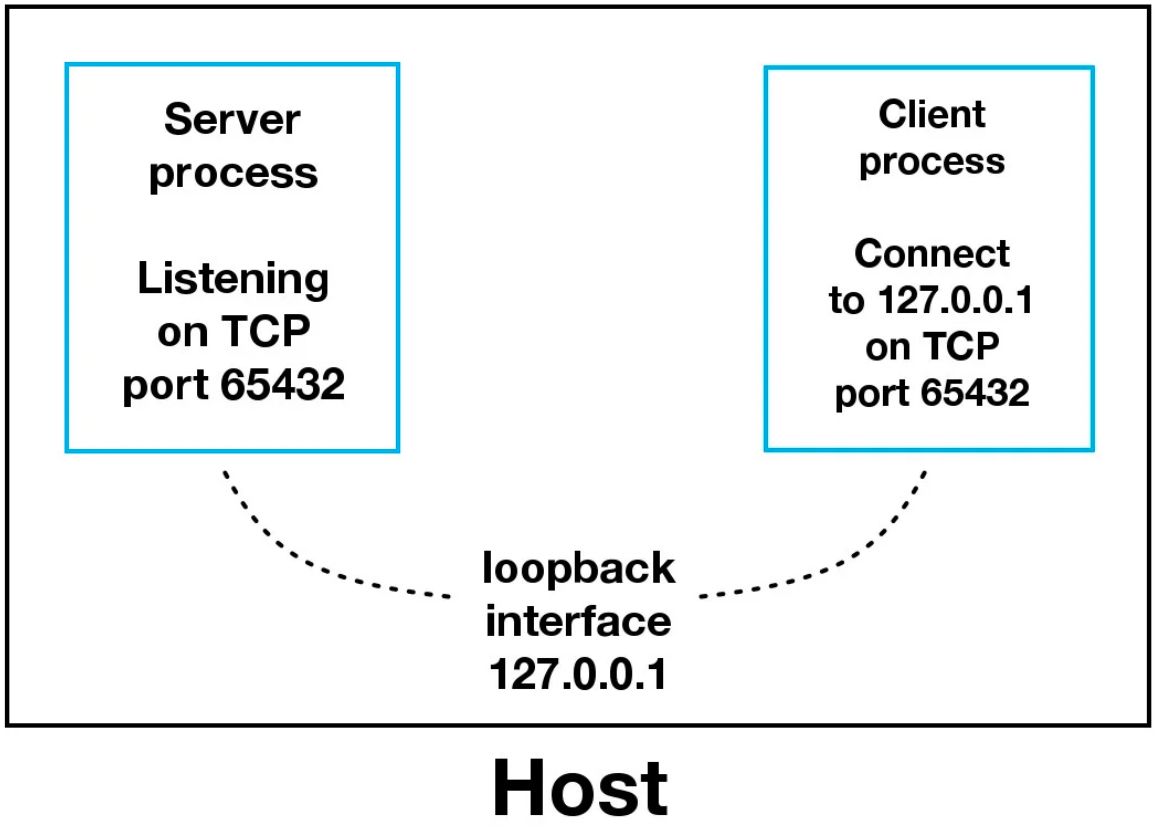
Зліва консоль клієнта, справа консоль сервера.

Як можна побачити коли ми запустили программу client.py на сервері ми отримали від клієнта його адресу та його повідомлення, у відповідь на це наш сервер перевів стрічку у верхній регістр та відправив його назад клієнту.

На діаграмі нижче можна побачити послідовність викликів API сокетів і потік даних для TCP :

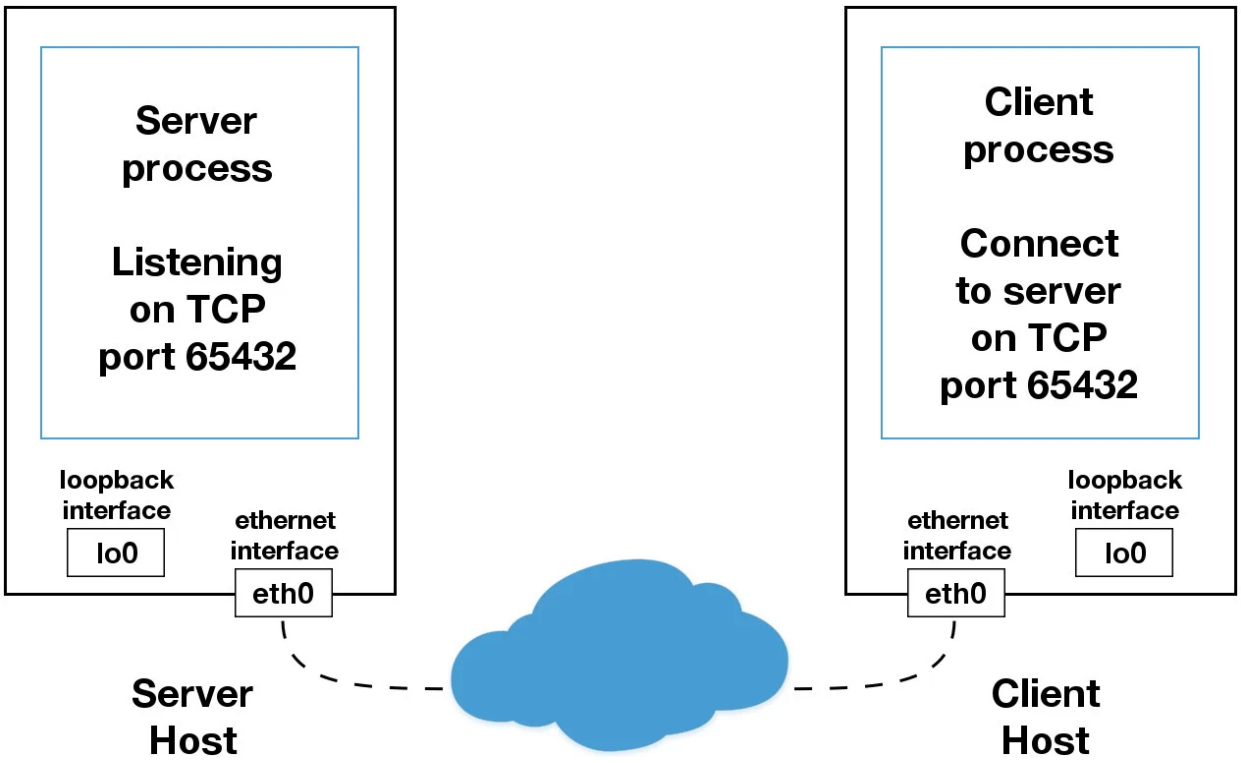


Давайте докладніше розглянемо, як клієнт і сервер спілкувалися один з одним:



При використанні інтерфейсу зворотного зв'язку (IPv4-адрес 127.0.0.1 або IPv6-адреса :: 1) дані ніколи не покидають хост і не стосуються зовнішньої мережі. На наведеній вище діаграмі інтерфейс зворотнього петлі міститься всередині хоста. Це відображає внутрішню природу інтерфейсу зворотного зв'язку, а також те, що з'єднання і дані, які передають його, є локальними для хоста.

Якщо я б використовував IP-адреса, відмінний від 127.0.0.1 або :: 1в своїх додатках, він, ймовірно, прив'язаний до інтерфейсу Ethernet, який підключений до зовнішньої мережі. Це шлюз до інших хостів за вашого localhost(діаграмма нижче):



ВИСНОВКИ

В ході виконання даної роботи мною було побудовано простий сервер та клієнт які обмінюються між собою даними по tcp протоколу для того щоб показати як працює стек TCP/IP для передачі даних в локальній мережі. Роботу програми було проілюстровано на діаграмах.