PKS - Otvorenie a pracovanie zo soketom

(Príprava na Z2)

1. **Príprava**
   Rozdeľte sa do dvojíc, jeden bude mať na starosti naprogramovanie serveru a drúhí bude mať na starosti naprogramovanie klienta. Pripojte sa na eduroam.

2. **Základné posielanie a prijatie správy**
   Vyskúšajte si základné poslanie a prijatie správy. Nezabudnite si správne nastaviť IP adresy:

   Klient:

```python
import socket

UDP_IP = "A.B.C.D" # server IP address
UDP_PORT = 50601
MESSAGE = b"Hello, World!"

print("UDP target IP: %s" % UDP_IP)
print("UDP target port: %s" % UDP_PORT)
print("message: %s" % MESSAGE)

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
sock.close()
```

   Server:

```python
import socket

UDP_IP = "A.B.C.D" # server IP address
UDP_PORT = 50601

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
print("received message: %s" % data)
sock.close()
```

(zdroj: https://wiki.python.org/moin/UdpCommunication)
Odchyťte komunikáciu pomocou Wiresharku. Čo znamená riadok sock.bind() ?

3. **Prerobíme si kód na viac objektovo-orientovaný prístup:**

Klient

```python
import  socket

CLIENT_IP = "127.0.0.1" # client host ip  A.B.C.D
CLIENT_PORT = 50602 # client port for recieving communication
```

```python
SERVER_IP = "127.0.0.1" # Server host ip (public IP) A.B.C.D
SERVER_PORT = 50601

class Client:

    def __init__(self, ip, port, server_ip, server_port) -> None:
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # UDP socket creation
        self.server_ip = server_ip
        self.server_port = server_port


    def receive(self):
        data = None
        data, self.server = self.sock.recvfrom(1024) # buffer size is 1024 bytes
        return data #1

    def send_message(self, message):
        self.sock.sendto(bytes(message,encoding="utf-8"),(self.server_ip,self.server_port))


    def quit(self):
        self.sock.close() # correctly closing socket
        print("Client closed..")


if __name__=="__main__":
    client  = Client(CLIENT_IP, CLIENT_PORT, SERVER_IP, SERVER_PORT)
    data = "empty"
    print("Input your message: ") #1
    client.send_message(input()) # 1
    data = client.receive() # 1
    if data != None: # 1
        print(data) # 1
    else: # 1
```

```
        print("Message has not been received") #1
    client.quit()
```

Server

```
import  socket

SERVER_IP = "127.0.0.1" # Server host ip (public IP) A.B.C.D
SERVER_PORT = 50601 # Server port for recieving communication


class Server:

    def __init__(self, ip, port) -> None:
        self.sock = 
socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # UDP socket 
creation
        self.sock.bind((ip, port)) #needs to be tuple 
(string,int)


    def receive(self):
        data = None
        while data == None:
            data, self.client= self.sock.recvfrom(1024) # 
buffer size is 1024 bytes
        print("Received message: %s" % data)
        return data # 1
        return str(data,encoding="utf-8")

    def send_response(self):
        self.sock.sendto(b"Message received... closing 
connection",self.client)


    def quit(self):
        self.sock.close() # correctly closing socket
        print("Server closed..")
```

```
if __name__=="__main__":
    server  = Server(SERVER_IP, SERVER_PORT)
    data = "empty"
    data  = server.receive() # 1
    if data != None: # 1
        server.send_response() # 1
    else: # 1
       print("Message has not been received") #1
    server.quit()
```

Odchyťte komunikáciu pomocou Wiresharku. Ako sa zmenilo správanie programu?

4. **Vytvoríme komunikáciu, až kým ju klient neukončí**

Zmeňte kód nasledovne:

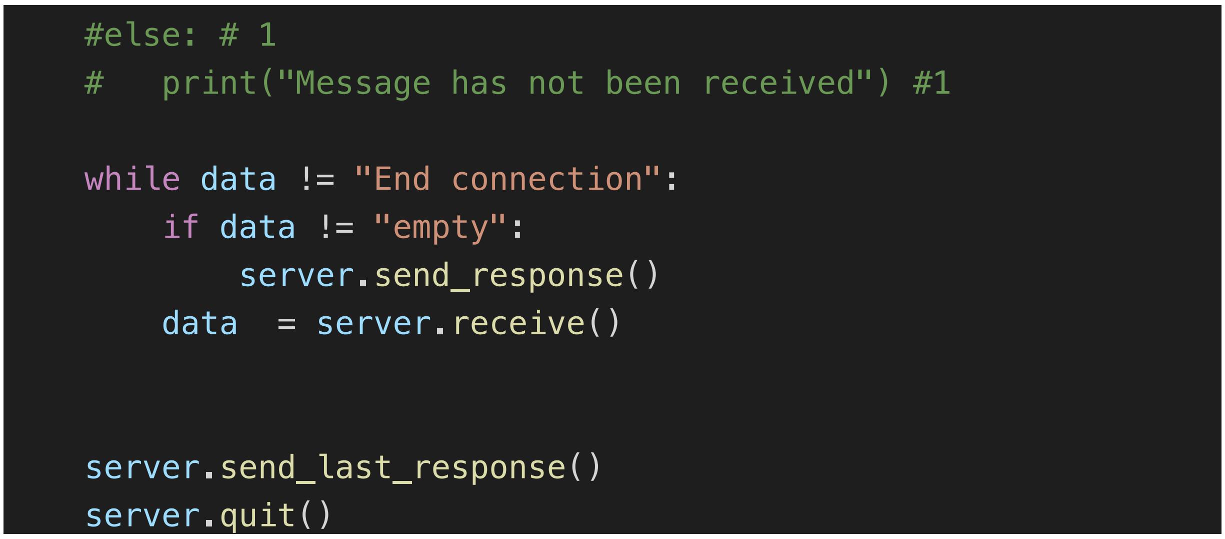
Klient

```
import  socket

CLIENT_IP = "127.0.0.1" # client host ip  A.B.C.D
CLIENT_PORT = 50602 # client port for recieving communication

SERVER_IP = "127.0.0.1" # Server host ip (public IP) A.B.C.D
SERVER_PORT = 50601

class Client:

    def __init__(self, ip, port, server_ip, server_port) -> 
None:
        self.sock = socket.socket(socket.AF_INET, 
socket.SOCK_DGRAM) # UDP socket creation
        self.server_ip = server_ip
        self.server_port = server_port


    def receive(self):
        data = None
        data, self.server = self.sock.recvfrom(1024) # buffer 
size is 1024 bytes
        # return data #1
        return str(data,encoding="utf-8")

    def send_message(self, message):
        self.sock.sendto(bytes(message,encoding="utf-
8"),(self.server_ip,self.server_port))


    def quit(self):
        self.sock.close() # correctly closing socket
        print("Client closed..")


if __name__=="__main__":
    client  = Client(CLIENT_IP, CLIENT_PORT, SERVER_IP,
SERVER_PORT)
    data = "empty"
    #print("Input your message: ") #1
    #client.send_message(input()) # 1
    #data = client.receive() # 1
    #if data != None: # 1
    #    print(data) # 1
    #else: # 1
    #    print("Message has not been received") #1

    while data != "End connection message recieved... closing
connection":
        print("Input your message: ")
        client.send_message(input())
        data = client.receive()
        print(data)

    client.quit()
```

Server

```
import  socket

SERVER_IP = "127.0.0.1" # Server host ip (public IP) A.B.C.D
SERVER_PORT = 50601 # Server port for recieving communication
```

```
class Server:

    def __init__(self, ip, port) -> None:
        self.sock =
socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # UDP socket
creation
        self.sock.bind((ip, port)) #needs to be tuple
(string,int)


    def receive(self):
        data = None
        while data == None:
            data, self.client= self.sock.recvfrom(1024) #
buffer size is 1024 bytes
        print("Received message: %s" % data)
        #return data # 1
        return str(data,encoding="utf-8")

    def send_response(self):
        #self.sock.sendto(b"Message received... closing
connection",self.client)
        self.sock.sendto(b"Message received...",self.client)

    def send_last_response(self):
        self.sock.sendto(b"End connection message recieved...
closing connection",self.client)

    def quit(self):
        self.sock.close() # correctly closing socket
        print("Server closed..")


if __name__=="__main__":
    server  = Server(SERVER_IP, SERVER_PORT)
    data = "empty"
    #data  = server.receive() # 1
    #if data != None: # 1
    #    server.send_response() # 1
```



```
    #else: # 1
    #   print("Message has not been received") #1

    while data != "End connection":
        if data != "empty":
            server.send_response()
        data  = server.receive()


    server.send_last_response()
    server.quit()
```

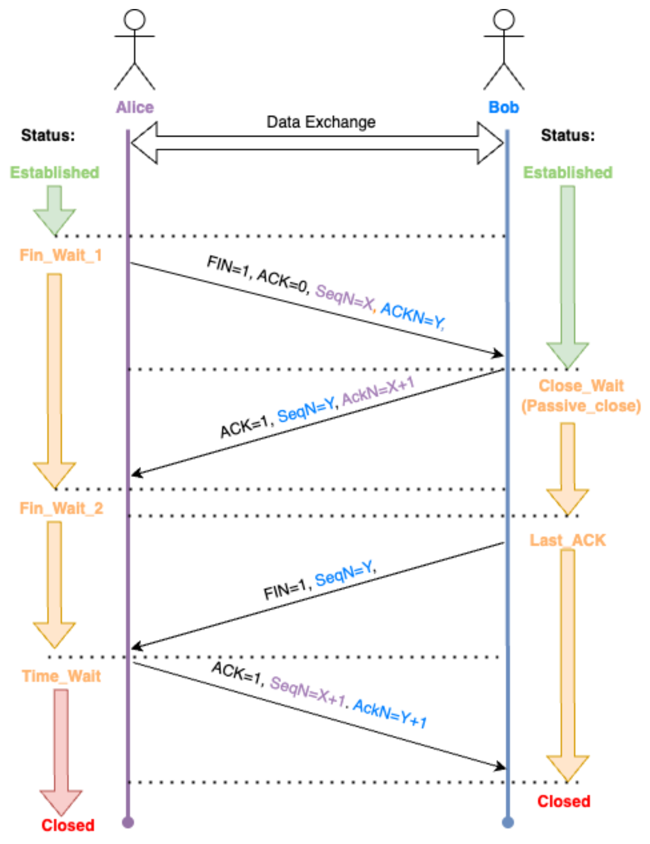
Odchyťte komunikáciu pomocou Wiresharku. Ako sa tento krát zmenilo správanie programu?

5. Vytvorte metódu na strane servera a aj klienta, ktorá pred tým, ako začne komunikácia, nadviaže spojenie pomocou (pseudo)three way handshake.
   Nepoužívajte flagy, stačí použiť textový formát, teda poslať správu s obsahom napr. ,,Syn=1,Ack=1“

   Odchyťte túto komunikáciu pomocou Wiresharku.

6. Vytvorte metódu na strane servera a aj klienta, ktorá ukončí spojenie pomocou (pseudo)Four way handshake.
   Nepoužívajte flagy, stačí použiť textový formát, teda poslať správu s obsahom napr. ,,Fin=1,Ack=1“

   Odchyťte túto komunikáciu pomocou Wiresharku.


Alice
Bob
Data Exchange
Status:
Status:
Established
Established
Fin_Wait_1
FIN=1, ACK=0, SeqN=X, ACKN=Y,
Close_Wait
(Passive_close)
ACK=1, SeqN=Y, AckN=X+1
Fin_Wait_2
Last_ACK
FIN=1, SeqN=Y,
Time_Wait
ACK=1, SeqN=X+1, AckN=Y+1
Closed
Closed