MREŽNO PROGRAMIRANJE

Primjena blockchain tehnologije

Ivan Sekovanić

MREŽNO PROGRAMIRANJE

Mrežno programiranje uključuje izradu programskih aplikacija ili procesa koji komuniciraju preko računalnih mreža

Aplikacije si međusobno šalju poruke preko mreže, ali mogu komunicirati i u slučaju da se nalaze na istom uređaju

Tehnologije koje omogućuju mrežnu komunikaciju temelje se na mrežnim socketima

MREŽNO PROGRAMIRANJE

Socket

- Mrežna pristupna točka procesu ili aplikaciji
  - Proces koji želi pristupiti mreži u pravilu će koristiti socket kao točku ulaza i izlaza prema mreži
- Pojam socketa postoji još od početaka ARPANET-a
- Prvi put se spominje 1971. godine
- Većina današnjih implementacija temelji se na specifikaciji objavljenoj 1983. godine (takozvani Berkeley socket-i) kao dio BSD Unix OS-a
- Windows OS koristi Winsock API koji je napravljen po uzoru na BSD socket-e

MREŽNO PROGRAMIRANJE

Socket

- Otvoreni socket uz sebe veže mrežni port na koji prima podatke iz mreže
- Također mu je pridružena IP adresa
- Socketi se smatraju tehnologijom prijenosnog (transportnog) sloja OSI modela
  - Služe kao poveznica transportnog i aplikacijskog sloja
- Temelj su za većinu komunikacijskih protokola aplikacijskog sloja (HTTP, POP3, RPC, itd…)

MREŽNO PROGRAMIRANJE

Postoje 4 vrste socketa:

1. Stream sockets
   - Koriste TCP za međusobnu komunikaciju što omogućuje pouzdanu dostavu podataka na transportnom sloju.
   - Uspostavlja se veza između sudionika komunikacije
   - Podatci se primaju u onom redoslijedu kojim su poslani
   - Ukoliko se dogodi greška prilikom slanja, pošiljatelj je o tome obaviješten

MREŽNO PROGRAMIRANJE

Postoje 4 vrste socketa:

2. **Datagram sockets**
   - **Koriste UDP za međusobnu komunikaciju**
     - **Paket se šalje na odredište bez prisustva mehanizama provjere dostave i retransmisije**
     - **tzv. nepouzdana dostava paketa**
   - **Ne uspostavlja se veza između sudionika komunikacije**

MREŽNO PROGRAMIRANJE

Postoje 4 vrste socketa:

3. Raw sockets
   - Spuštaju se ispod razine TCP-a i UDP-a, do manipulacije samih IP paketa
   - Omogućuju upravljanje paketima bez korištenja npr. TCP-a i njegovih mehanizama
   - Nisu namijenjeni za potrebe većine programera
   - Pogodni za razvoj novih protokola od najniže razine
     - Ukoliko postojeći ne zadovoljavaju potrebe mrežne komunikacije

MREŽNO PROGRAMIRANJE

Postoje 4 vrste socketa:

4. Sequenced packet sockets
   - Slični su Stream socket-ima
   - Podatci se primaju u onom redoslijedu u kojem su poslani.
   - Ne uspostavlja se veza, ali se ipak jamči pouzdana dostava praćenjem paketa
     - Koristi se Sequenced Packet Protocol umjesto TCP-a

MREŽNO PROGRAMIRANJE

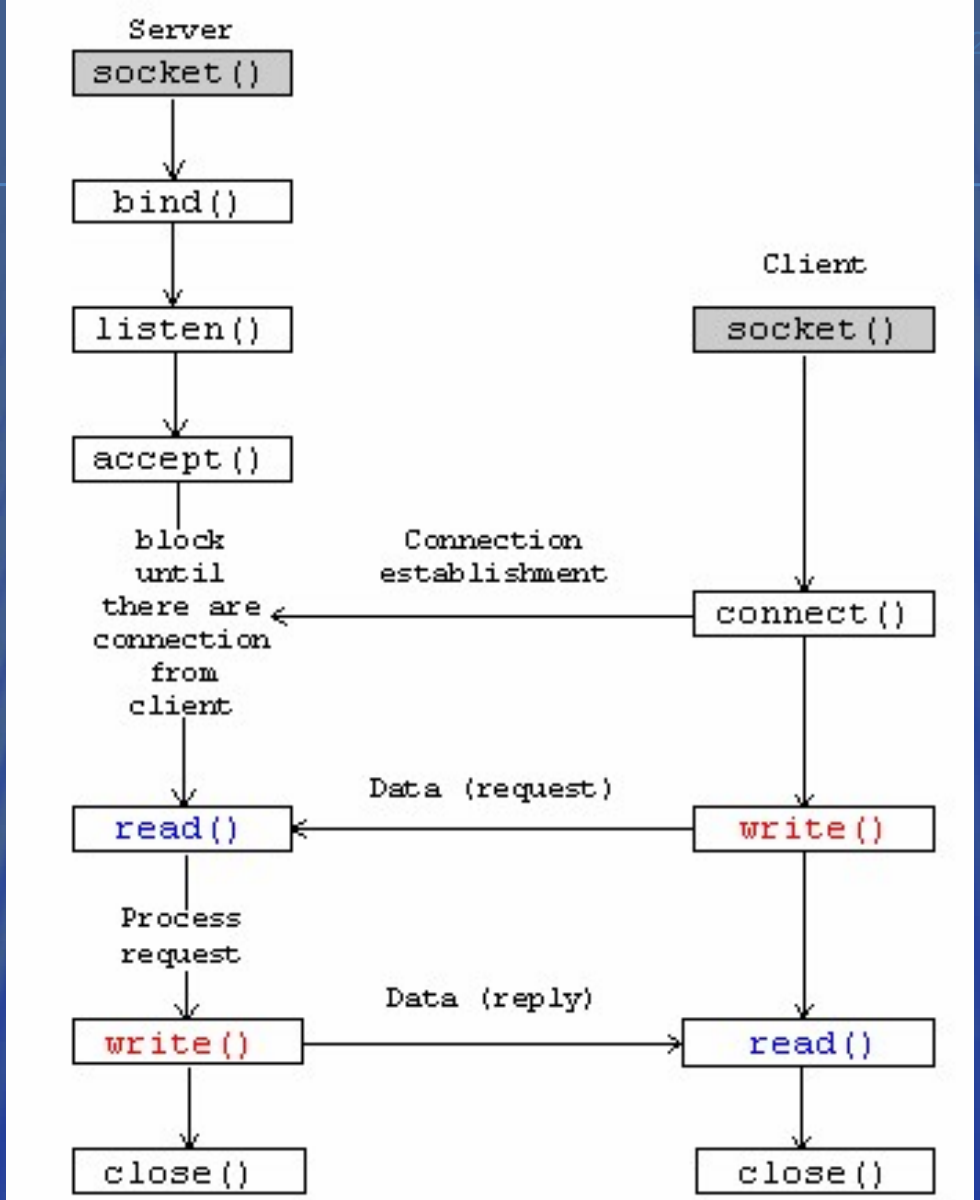
Bitni pozivi funkcija (metoda) kod TCP servera:

- **socket() – stvaramo socket na našem serveru**
- **bind() – prethodno stvorenom socket-u pridružujemo adresu. Adresa se sastoji od mrežnog porta i IP adrese**
- **listen() – server osluškuje dolazne veze od potencijalnog klijenta**
- **accept() – server je blokiran sve dok ne uspostavi vezu s klijentom (prihvaćanje veze)**
- **read() – čitaj podatke sa socket-a**
- **write() – piši podatke na socket**

MREŽNO PROGRAMIRANJE

Bitni pozivi funkcija (metoda) kod TCP klijenta:

- **socket() – stvaramo socket na našem klijentu**
- **connect() – ostvarujemo vezu sa socket-om servera (veza prema serveru). Potrebno je navesti IP adresu i mrežni port servera ili usluge na koju se spajamo**
- **read() – čitaj podatke sa socket-a**
- **write() – piši podatke na socket**


Server
socket()
bind()
listen()
accept()
block until there are connection from client
Client
socket()
Connection establishment
connect()
Data (request)
read()
write()
Process request
Data (reply)
write()
read()
close()
close()

PYTHON SOCKETS

Python posjeduje modul za socket-e koji omogućuje poziv metoda definiranih BSD specifikacijom

Posjeduje sve metode koje su inače dostupne za rad sa socket-ima u programskom jeziku C

Najvažnije metode za rad sa socket-ima u Python-u:

socket(), bind(), listen(), accept(), connect(), connect_ex(), send(), recv(), close()

PYTHON SOCKETS – TCP SERVER

Primjer TCP servera koji vraća klijentu podatke koje je primio – echo server

```
import socket  #uvoz modula socket

HOST = '127.0.0.1'    # varijabla s pridruženom IP adresom (localhost)
PORT = 65432          # varijabla s pridruženim mrežnim portom
```

- **Dozvoljeni portovi su od 1 do 65535, ali je preporučljivo izbjegavati raspon 1-1023**

PYTHON SOCKETS – TCP SERVER

TCP echo server:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        conn, addr = s.accept()
        with conn:
                print('Connected by', addr)
                while True:
                        data = conn.recv(1024)
                        if not data:
                                break
                        conn.sendall(data)
```

PYTHON SOCKETS – TCP SERVER

TCP echo server:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

- pozivamo metodu socket iz modula socket – stvaramo socket i definiramo ga kao objekt s za buduće referenciranje
  - Ukoliko koristimo with statement, nije nužno pozivati metodu close() prilikom zatvaranja socket-a
- socket.AF_INET – određuje tzv. address family
  - AF_INET koristimo za IPv4 dok AF_INET6 koristimo za IPv6 adrese

PYTHON SOCKETS – TCP SERVER

TCP echo server:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

- socket.SOCK_STREAM – određuje vrstu socketa
  - U ovom slučaju to je Stream Socket koji koristi TCP za komunikaciju
  - Za UDP komunikaciju koristimo SOCK_DGRAM
  - Primjeri ostalih vrsta: SOCK_RAW; SOCK_SEQPACKET

PYTHON SOCKETS – TCP SERVER

TCP echo server:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        conn, addr = s.accept()
```

- s.bind() – socketu pridružuje prethodno definiranu IP adresu i broj mrežnog porta
  - varijabla HOST ne mora sadržavati IP adresu, može i ime host-a. Može i sadržavati prazan string – u tom slučaju prihvaća veze s bilo kojeg mrežnog sučelja
- s.listen() – server počinje s osluškivanjem dolaznih veza

PYTHON SOCKETS – TCP SERVER

TCP echo server:

```
conn, addr = s.accept()
```

- **conn, addr = s.accept() – blokira server sve dok ne dobije dolaznu vezu**
- **stvara se novi objekt koji predstavlja vezu. U ovom primjeru objekt je pohranjen u varijablu conn**
- **u drugu varijablu (addr iz ovog primjera) se sprema adresa klijenta (host i port) koji se je spojio na server**

PYTHON SOCKETS – TCP SERVER

```
with conn:
        print('Connected by', addr)
        while True:
                data = conn.recv(1024)
                if not data:
                        break
                conn.sendall(data)
```

- print ispisuje adresu spojenog klijenta
- koristimo beskonačnu while petlju za primanje podataka i njihovo slanje nazad klijentu
- if not data - izvršava se ukoliko server primi prazan byte string ---> b″

PYTHON SOCKETS – TCP SERVER

TCP echo server:

```
while True:
        data = conn.recv(1024)
        if not data:
                break
        conn.sendall(data)
```

- conn.recv() – prima podatke i pohranjuje ih u varijablu
  - 1024 - broj bajtova koji se odjednom mogu primiti
- conn.sendall(data) – šalje sve što se nalazi u varijabli data
  - sendall() ustraje dok ne pošalje sve što joj je predano na slanje ili se javi greška

PYTHON SOCKETS – TCP KLIJENT

```
import socket

HOST = '127.0.0.1' # The server's hostname or IP address
PORT = 65432 # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))
        s.sendall(b'Hello, world')
        data = s.recv(1024)

print('Received', data)
```

- **uvozimo modul socket te definiramo adresu servera na koji se spajamo.**

PYTHON SOCKETS – TCP KLIJENT

TCP echo klijent:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((HOST, PORT))
        s.sendall(b'Hello, world')
        data = s.recv(1024)
print('Received', data)
```

- stvaramo socket na isti način kao na serveru, te metodom .connect() određujemo na koju adresu (servera) se socket spaja
- s.sendall() – šalje navedeni string u obliku byte objekta
- na kraju podatci se primaju i ispisuju

PYTHON SOCKETS – UDP SERVER

Primjer UDP servera koji vraća klijentu podatke koje je primio – echo server

```
import socket  #uvoz modula socket

HOST = '127.0.0.1'    # varijabla s pridruženom IP adresom (localhost)
PORT = 65432          # varijabla s pridruženim mrežnim portom
```

- **Započinjemo s istim linijama koda kao kod TCP servera**

PYTHON SOCKETS – UDP SERVER

UDP echo server:

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.bind((HOST, PORT))
        while True:
                data, address = s.recvfrom(1024)
                print(data,"\n", address)

                s.sendto(data, address)
```

- Potrebno je odraditi bind-anje socketa na adresu
  - Nemamo listen() i accept() kao kod TCP servera

PYTHON SOCKETS – UDP SERVER

UDP echo server:

```
while True:
        data, address = s.recvfrom(1024)
        print(data,"\n", address)

        s.sendto(data, address)
```

- s.recvfrom() – vraća objekt s podatcima i adresu (klijenta) s koje su podatci stigli
- adresa nam treba u metodi s.sendto() jer ne postoji uspostavljena veza kao u slučaju TCP-a, pa za svako slanje treba navesti kuda (na koju adresu) podatci odlaze

PYTHON SOCKETS – UDP KLIJENT

UDP echo klijent:

```
import socket

HOST = '127.0.0.1' # The server's hostname or IP address
PORT = 65432 # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.sendto(b'Hello, world', (HOST,PORT))
    data = s.recv(1024)

print('Received', data)
```

PYTHON SOCKETS – UDP KLIJENT

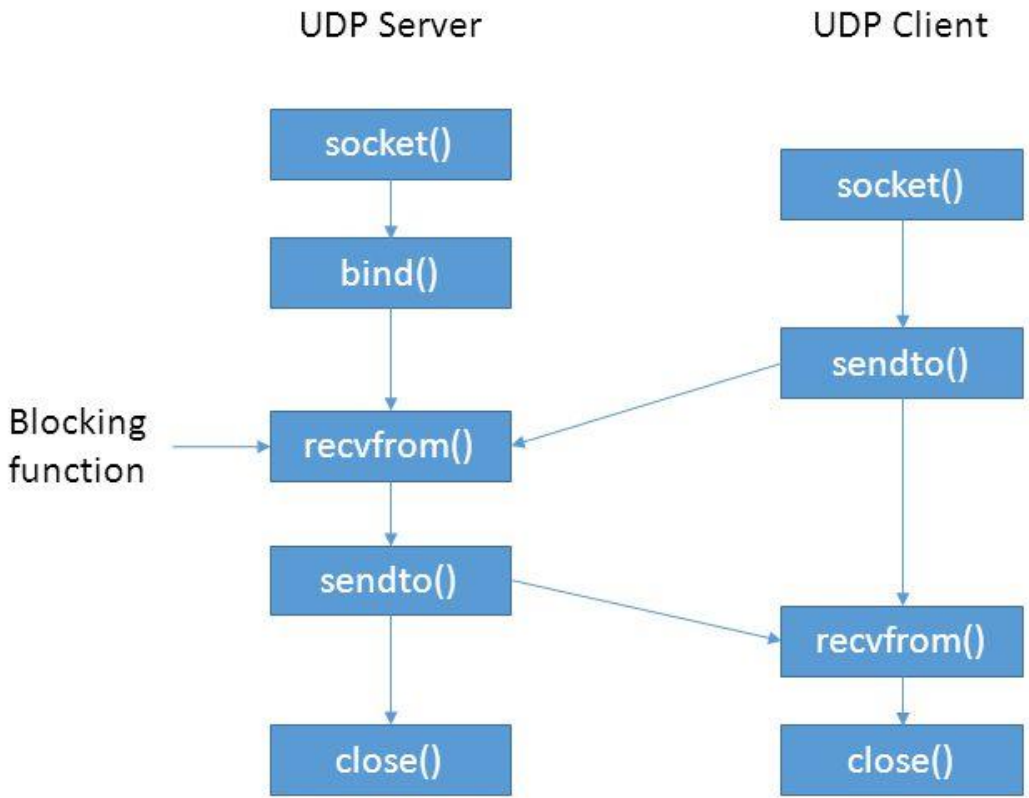
UDP echo klijent:

```
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
    s.sendto(b'Hello, world', (HOST,PORT))
    data = s.recv(1024)

print('Received', repr(data))
```

- s.sendto() – šalje byte string na adresu određenu parom HOST i PORT
- Ukoliko nam ne treba adresa pošiljatelja, možemo koristiti s.recv() umjesto s.recvfrom()

UDP Socket Programming Overview



BLOKIRAJUĆI I NE BLOKIRAJUĆI POZIVI

- Razmotrimo naš TCP server koji podržava spajanje samo jednog klijenta
- Izvođenje programa na serveru je blokirano sve dok se na njega ne spoji klijent i dok klijent ne počne slati podatke
- Pozivatelj funkcije je blokiran dok ne dobije rezultat ili dok poziv ne rezultira greškom ili ne istekne možebitni timeout

BLOKIRAJUĆI I NE BLOKIRAJUĆI POZIVI

- Kako riješiti problem blokade cijelog servera dok ne dobije odgovor od klijenata
- Pozivi funkcija se mogu postaviti u ne blokirajući mod
  - Program odmah nastavlja s izvođenjem iako funkcija nije vratila rezultat
  - Potrebno je program izraditi na način da može nastaviti iako nije odmah dobio rezultat (podatci još možda nisu spremni, problemi s mrežom…), te da stalno provjerava da li je rezultat stigao

BLOKIRAJUĆI I NE BLOKIRAJUĆI POZIVI

Blokirajući pozivi metoda

- **Zaustavljaju izvođenje aplikacije dok ne dobiju odgovor**
- **Npr. metode accept(), connect(), send(), recv() blokiraju izvođenje vaše aplikacije (programa)**
  - **blokiraju zbog toga što čekaju odgovor ili neke podatke iz mreže**
- **Metode je moguće postaviti i u ne blokirajući mod, ali to je potrebno uzeti u obzir prilikom dizajna aplikacije jer će se program nastaviti izvoditi prije nego je dobio potrebne podatke**

BLOKIRAJUĆI I NE BLOKIRAJUĆI POZIVI

- Blokirajući i ne blokirajući pozivi vs. sinkrona i asinkrona komunikacija
  - oba koncepta su vrlo slična
  - kod ne blokirajućeg poziva kao rezultat se vraća što god je u tom trenutku dostupno, te pozivatelj obično treba pokušati ponovni poziv da bi dobio ostatak podataka
  - Kod asinkrone komunikacije poziv je obično vezan uz neku povratnu funkciju ili uz stvaranje eventa koji signalizira programu da je odgovor postao dostupan

BLOKIRAJUĆI I NE BLOKIRAJUĆI POZIVI

- Postavljanje socketa u ne blokirajući mod:
  - nad socketom se poziva metoda .setblocking(False)

```
nb_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
nb_socket.bind((host, port))
nb_socket.listen()
print('listening on', (host, port))
nb_socket.setblocking(False)
```

RAD S VIŠE SOCKETA

- Kako provjeriti da li socket ima spremne podatke za čitanje ili pisanje?
- Postoje različiti koncepti i moduli pomoću kojih je moguće ostvariti konkurentno izvođenje programa u python-u
  - koristit ćemo poziv select() iz modula selectors kako bi pratili da li socket ima podatke spremne za čitanje ili pisanje
  - Python komunicira s operativnim sustavom kako bi dobio informaciju o stanju mrežnih buffera
  - ne koristimo stvarnu konkurentnost (višedretvenost, višejezgrenost)

RAD S VIŠE SOCKETA - SERVER

- Primjer servera koji podržava spajanje više dolaznih veza

```
import selectors
sel = selectors.DefaultSelector()
# ...
nb_socket  = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
nb_socket.bind((host, port))
nb_socket.listen()
```

- uvoz modula selectors te stvaranje selector objekta kojeg vežemo za varijablu po izboru (sel)

RAD S VIŠE SOCKETA - SERVER

- Primjer servera koji podržava spajanje više dolaznih veza

```
...
print('listening on', (host, port))
nb_socket.setblocking(False)
sel.register(nb_socket , selectors.EVENT_READ, data=None)
```

  - stvoreni socket postavljamo u ne blokirajući mod
  - selektoru dajemo informaciju da socket treba motriti
    - U ovom slučaju želimo da selektor reagira samo ako postoje podatci za čitanje iz socketa (EVENT_READ)
    - u varijablu data pohranjujemo podatke po potrebi

RAD S VIŠE SOCKETA - SERVER

- Petlja koja provjerava događaje (event loop)

```
while True:
        events = sel.select(timeout=None)
        for key, mask in events:
                if key.data is None:
                        accept_wrapper(key.fileobj)
                else:
                        service_connection(key, mask)
```

- sel.select() prati da li na socketu postoje događaji (EVENT_READ ili EVENT_WRITE ako je postavljen)
- sel.select(timeout=None) blokira izvođenje dok se na socketu nešto ne dogodi

RAD S VIŠE SOCKETA - SERVER

- select() za svaki socket vraća tuple koji se sastoji od tuple-a key i maske koja daje informaciju o događaju koji je zabilježen pomoću select-a (čitanje, pisanje ili oboje)
- Tuple key sadrži objekt koji predstavlja socket i podatke spremljene u varijabli data (sockobj, data)
- U ovom primjeru, ukoliko je data postavljen u None, znamo da je događaj došao od socketa za slušanje, pa aktiviramo funkciju za prihvaćanje nove veze – u suprotnom, znamo da je događaj došao od već uspostavljene veze pa pozivamo funkciju koja obrađuje podatke

RAD S VIŠE SOCKETA - SERVER

- Funkcija koja obrađuje zahtjeve za novom vezom.
- Poziv accept() prihvaća novu vezu te u varijablu conn vraća socket objekt koji predstavlja tu vezu. U varijablu addr sprema tuple s adresom klijenta (host i port)

```
def accept_wrapper(sock):
        conn, addr = sock.accept()
        print('accepted connection from', addr)
        conn.setblocking(False)
        data = types.SimpleNamespace(addr=addr, inb=b'', outb=b'')
        events = selectors.EVENT_READ | selectors.EVENT_WRITE
        sel.register(conn, events, data=data)
```

RAD S VIŠE SOCKETA - SERVER

```
conn.setblocking(False)
data = types.SimpleNamespace(addr=addr, inb=b'', outb=b'')
events = selectors.EVENT_READ | selectors.EVENT_WRITE
sel.register(conn, events, data=data)
```

- socket objekt se postavlja u ne blokirajuće stanje
- socket se prijavljuje selektoru za praćenje. Pri tom želimo da se prate READ i WRITE događaji
- Također uz socket pohranjujemo data varijablu u kojoj vodimo evidenciju o adresi klijenta spojenog na socket, te zatim imamo varijable inb i outb u koje pohranjujemo možebitne podatke na čekanju (služi nam kao programski buffer)

RAD S VIŠE SOCKETA - SERVER

```
def service_connection(key, mask):
        sock = key.fileobj
        data = key.data
        if mask & selectors.EVENT_READ:
                recv_data = sock.recv(1024) # vraća primljene podatke
                if recv_data:
                        data.outb += recv_data
                else:
                        print('closing connection to', data.addr)
                        sel.unregister(sock)
                        sock.close()
        if mask & selectors.EVENT_WRITE:
                if data.outb:
                        print('echoing', repr(data.outb), 'to', data.addr)
                        sent = sock.send(data.outb) # vraća broj poslanih bajtova
                        data.outb = data.outb[sent:]
```

RAD S VIŠE SOCKETA - SERVER

```
if mask & selectors.EVENT_READ:
                recv_data = sock.recv(1024) # Should be ready to read
                if recv_data:
                        data.outb += recv_data
                else:
                        print('closing connection to', data.addr)
                        sel.unregister(sock)
                        sock.close()
```

- ukoliko se na socketu jave podatci za čitanje, ulazi se u ovaj if koji preuzima podatke pomoći sock.recv() i dodaje ih u varijablu data.outb
- Ukoliko sock.recv() vrati prazan niz, ide se u prekid veze i odjavu socketa sa selektora

RAD S VIŠE SOCKETA - SERVER

```
if mask & selectors.EVENT_WRITE:
        if data.outb:
                print('echoing', repr(data.outb), 'to', data.addr)
                sent = sock.send(data.outb) # vraća broj poslanih bajtova
                data.outb = data.outb[sent:]
```

- ovaj if je zadužen za pisanje u socket
- dok god ima podataka u varijabli data.outb, podatci se pomoću naredbe sock.send() šalju prema klijentu

RAD S VIŠE SOCKETA - KLIJENT

```
def start_connections(host, port, num_conns):
        server_addr = (host, port)
        for i in range(0, num_conns):
                connid = i + 1
                print('starting connection', connid, 'to', server_addr)
                sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                sock.setblocking(False)
                sock.connect_ex(server_addr)
                events = selectors.EVENT_READ | selectors.EVENT_WRITE
                dataVAR = types.SimpleNamespace(connid=connid,
                        msg_total=sum(len(m) for m in messages),
                        recv_total=0,
                        messages=list(messages),
                        outb=b'')
                sel.register(sock, events, data=dataVAR)
```

RAD S VIŠE SOCKETA - KLIJENT

```
def start_connections(host, port, num_conns):
        .....
        .....
        while True:
                events = sel.select(timeout=None)
                for key, mask in events:
                        if key.data is None:
                                pass
                        else:
                                service_connection(key, mask)
```

RAD S VIŠE SOCKETA - KLIJENT

```
def service_connection(key, mask):
        sock = key.fileobj
        dataSC = key.data
        if mask & selectors.EVENT_READ:
                recv_data = sock.recv(1024)  # Should be ready to read
                if recv_data:
                        print('received', repr(recv_data), 'from connection',
                        dataSC.connid)
                        dataSC.recv_total += len(recv_data)
                if not recv_data or dataSC.recv_total == dataSC.msg_total:
                        print('closing connection', dataSC.connid)
                        sel.unregister(sock)
                        sock.close()
```

RAD S VIŠE SOCKETA - KLIJENT

```
def service_connection(key, mask):
        ....
        ....
        if mask & selectors.EVENT_WRITE:
                if not dataSC.outb and dataSC.messages:
                        dataSC.outb = dataSC.messages.pop(0)
                if dataSC.outb:
                        print('sending', repr(dataSC.outb), 'to connection',
                        dataSC.connid)
                        sent = sock.send(dataSC.outb)
                        dataSC.outb = dataSC.outb[sent:]
```

ZATVARANJE TCP VEZE

- Veza se može prekinuti s jedne strane
- U windowsima, ukoliko je veza prekinuta s jedne strane, a na drugoj strani netko pokuša čitati pomoću socket.recv() dogoditi će se greška na recv() pozivu
- Signalizirati drugom sudioniku da će doći do prekida veze kako bi on prestao sa čitanjem

VIŠE NA LINKU...

https://realpython.com/python-sockets/