Telekommunikációs Hálózatok

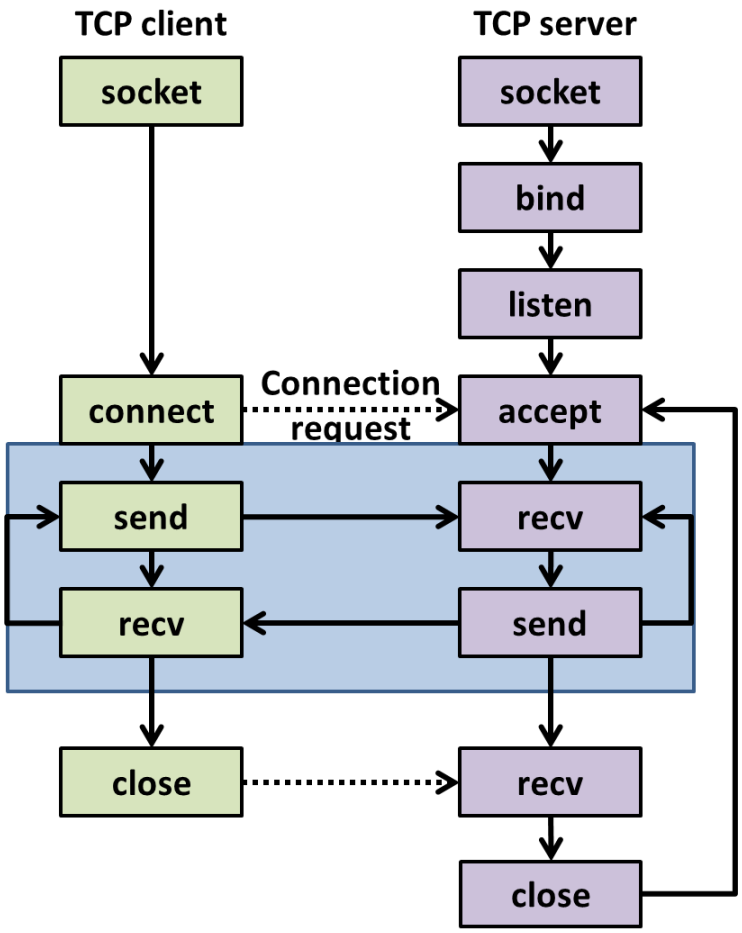
3. gyakorlat

PYTHON SOCKET PROGRAMOZÁS I.

A kommunikációs csatorna kétféle típusa

- Kapcsolat-orientált modell (analógia: telefonbeszélgetés)
  - csomagok megérkeznek jó sorrendben
  - ilyen protokoll a TCP
  - kapcsolódó típus: stream socket
- Kapcsolat-nélküli modell (analógia: postai levelezés)
  - csomagok nem biztos, hogy sorrend helyesen érkeznek, sőt el is veszhetnek
  - előnye a jobb teljesítmény
  - ilyen protokoll a UDP
  - kapcsolódó típus: datagram socket

TCP

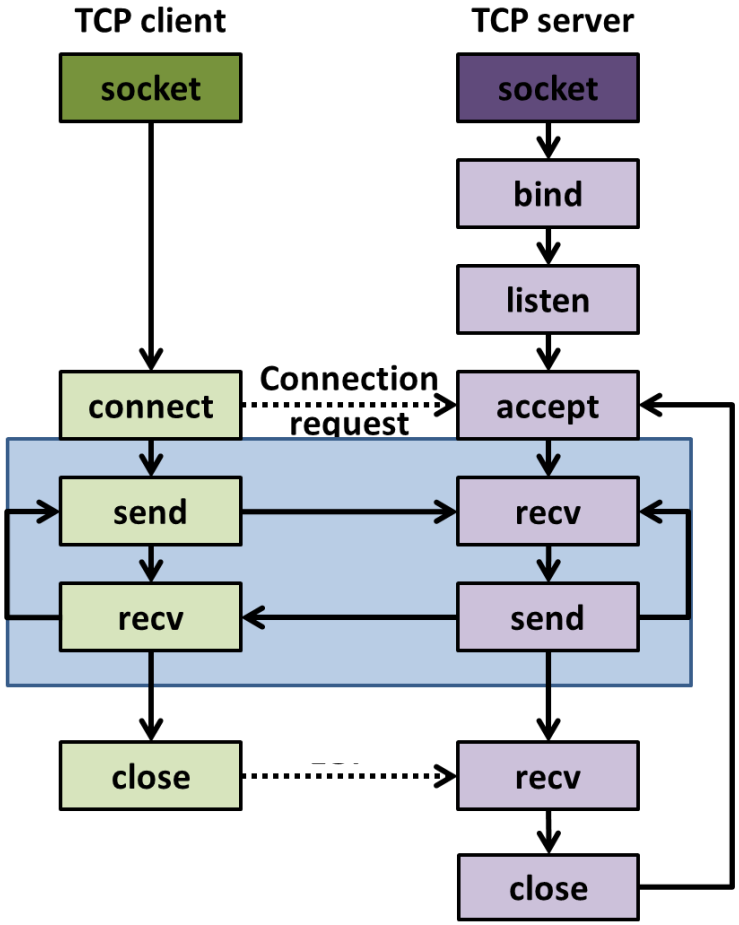

TCP client
TCP server
socket
socket
bind
listen
connect
Connection request
accept
send
recv
recv
send
close
recv
close

Socket leíró beállítása

- socket.**socket**([*family*
  [, *type*
  [, *proto*]]])
- *family* : socket.AF_INET → IPv4 (AF_INET6 → IPv6)
- *type* : socket.SOCK_STREAM → TCP
- *proto* : 0

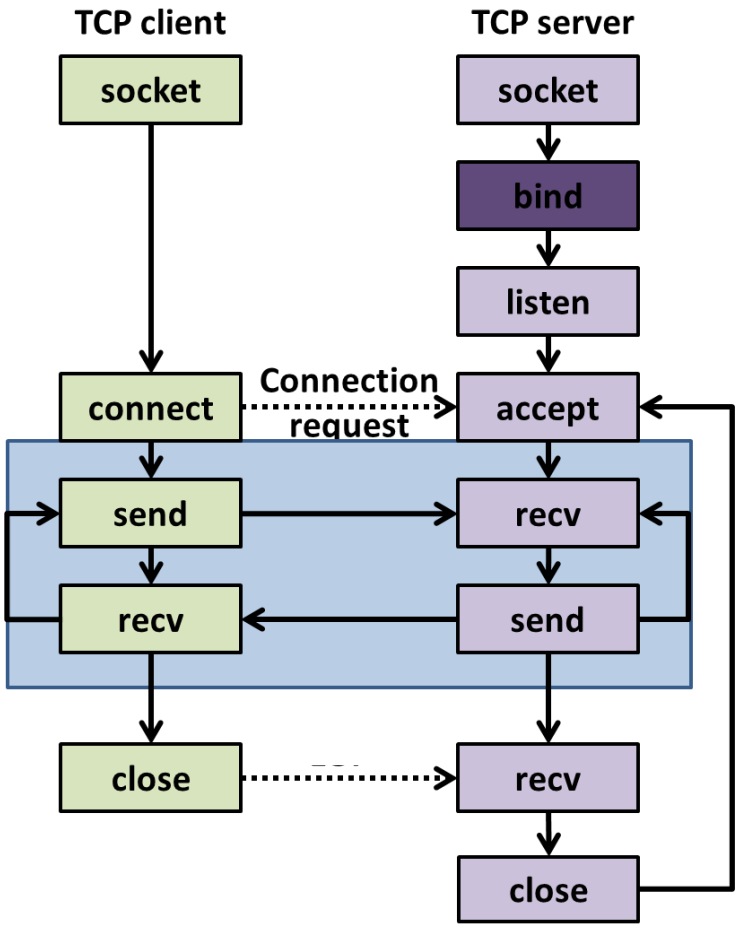
(alapértelmezett protokoll lesz)

- visszatérési érték: egy socket objektum, amelynek a metódusai a különböző socket rendszer hívásokat implementálják



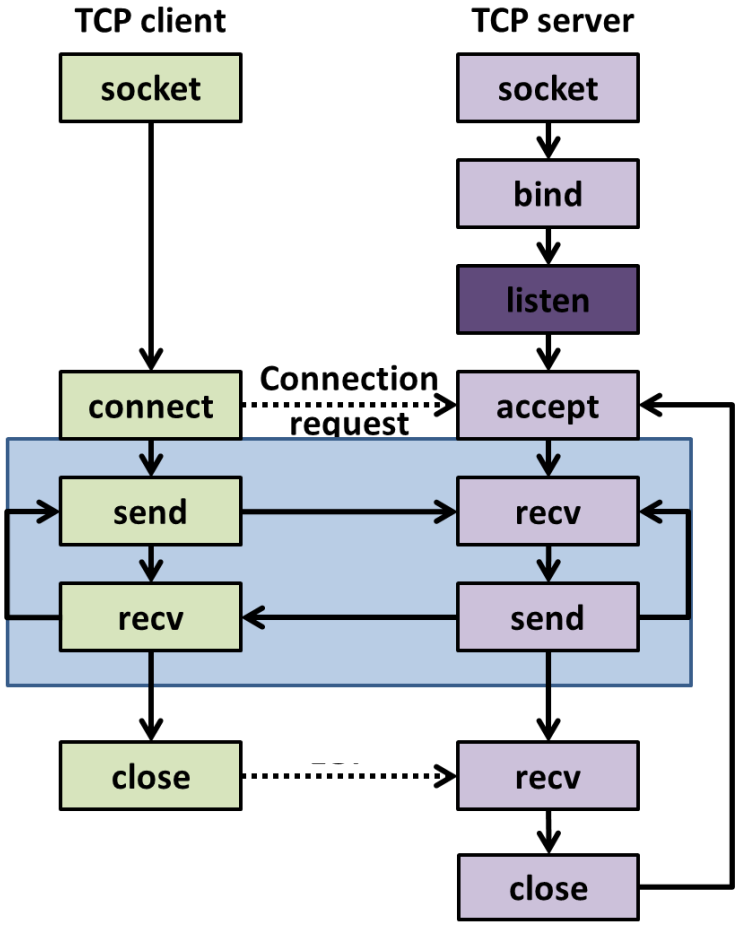
Bindolás

- socket.socket.**bind**(*address*)
- A socket objektum metódusa
- *address* : egy tuple, amelynek az első eleme egy hosztnév vagy IP cím (sztring reprezentációval), második eleme a portszám



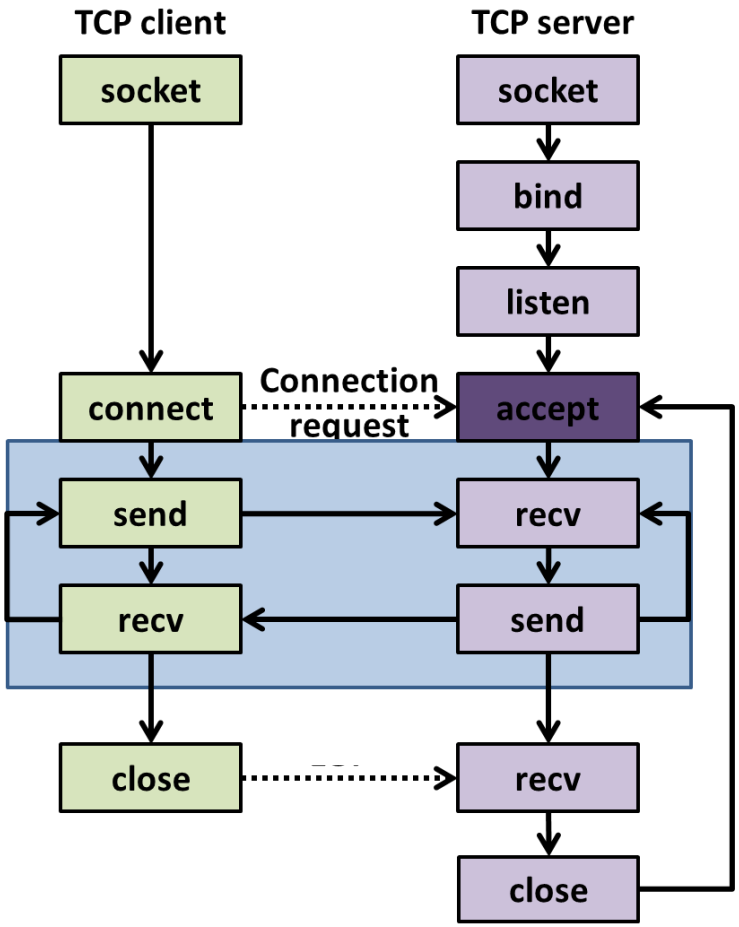
Listen

- socket.socket.**listen**(*backlog*)
- A socket objektum metódusa
- *backlog* : egy egész szám, ennyi kapcsolódási igény várakozhat a sorban



Accept

- socket.socket.**accept**()
- A socket objektum metódusa
- A szerver elfogadhatja a kezdeményezett kapcsolatokat
- visszatérési érték: egy tuple,
  - amelynek az első eleme egy **új socket objektum** a kapcsolaton keresztüli adatküldésre és fogadásra
  - második eleme a kapcsolat túlsó végén lévő cím



Példa hívások TCP-nél I.

- socket()

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- bind()

```
server_address = ('localhost', 10000)
sock.bind(server_address)
```

- listen()

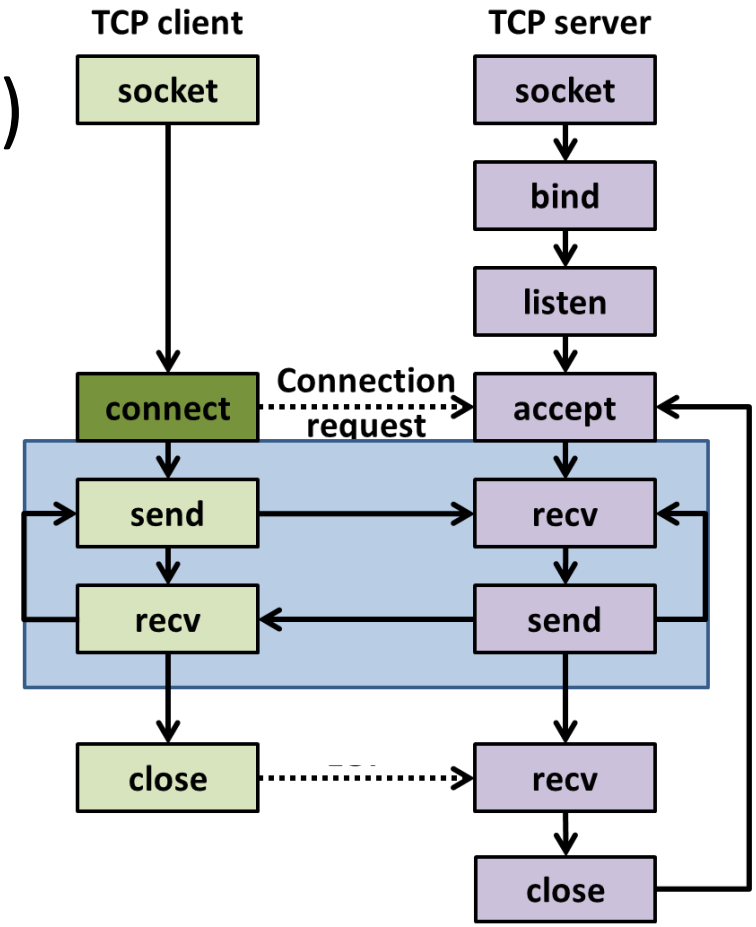
```
sock.listen(1)
```

- accept()

```
connection, client_address = sock.accept()
```

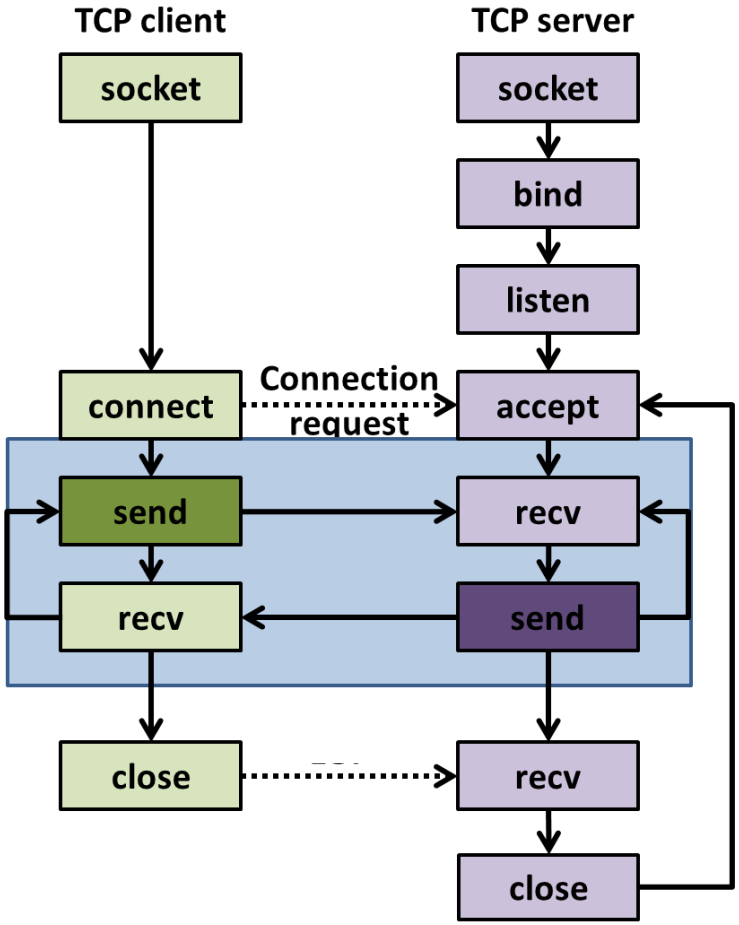
Connect

- socket.socket.**connect**(*address*)
- A socket objektum metódusa
- Kapcsolódás megkezdése egy távoli sockethez az *address* címen (ezt például kezdeményezheti egy kliens)
- Az *address* típusát ld. a **bind** függvénynél



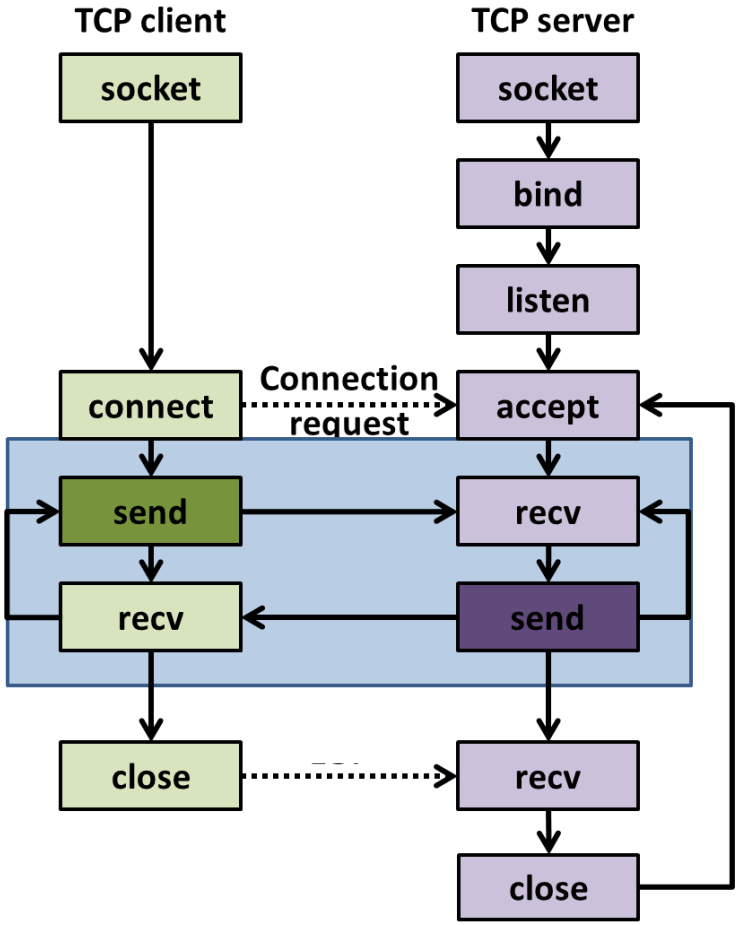
Send

- socket.socket.**send**(*bytes* *[, flags]*)
- A socket objektum metódusa
- Adatküldés (*bytes*) a socketnek
- *bytes* → *string*: b'vmi'.decode('utf-8')
- *string* → *bytes*: 'vmi'.encode()



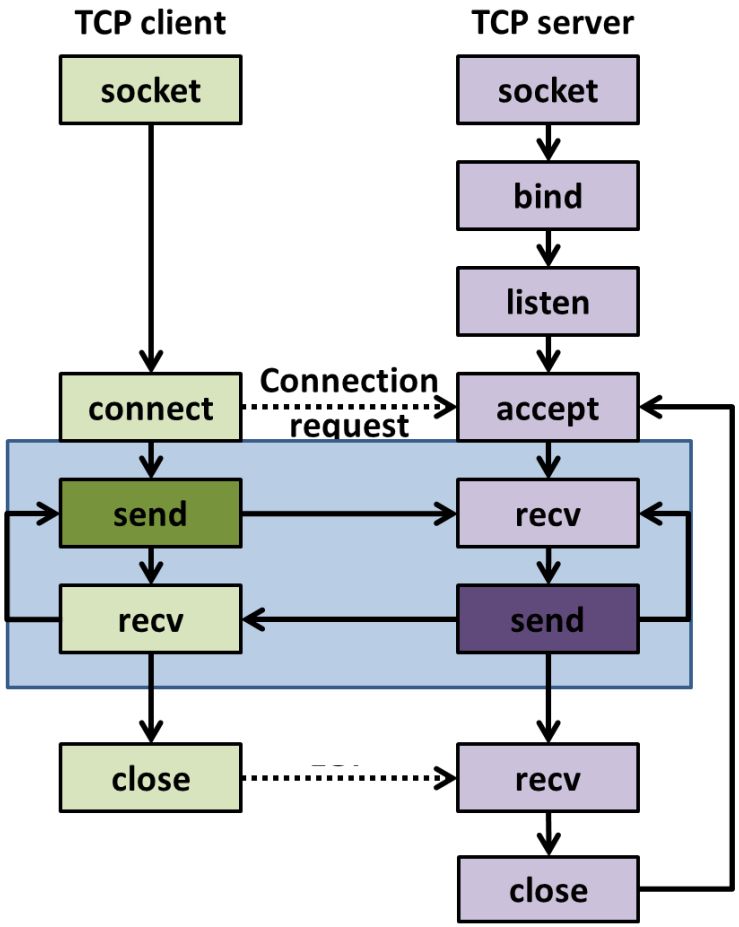
Send

- socket.socket.**send**(*bytes* *[, flags]*)
- *flags* : 0 (nincs flag meghatározva)
- A socketnek előtte már csatlakozni kellett a távoli sockethez!
- visszatérési érték: az átküldött bájtok száma
  - az alkalmazásnak kell ellenőrizni, hogy minden adat átment-e
  - ha csak egy része ment át: újra kell küldeni a maradékot



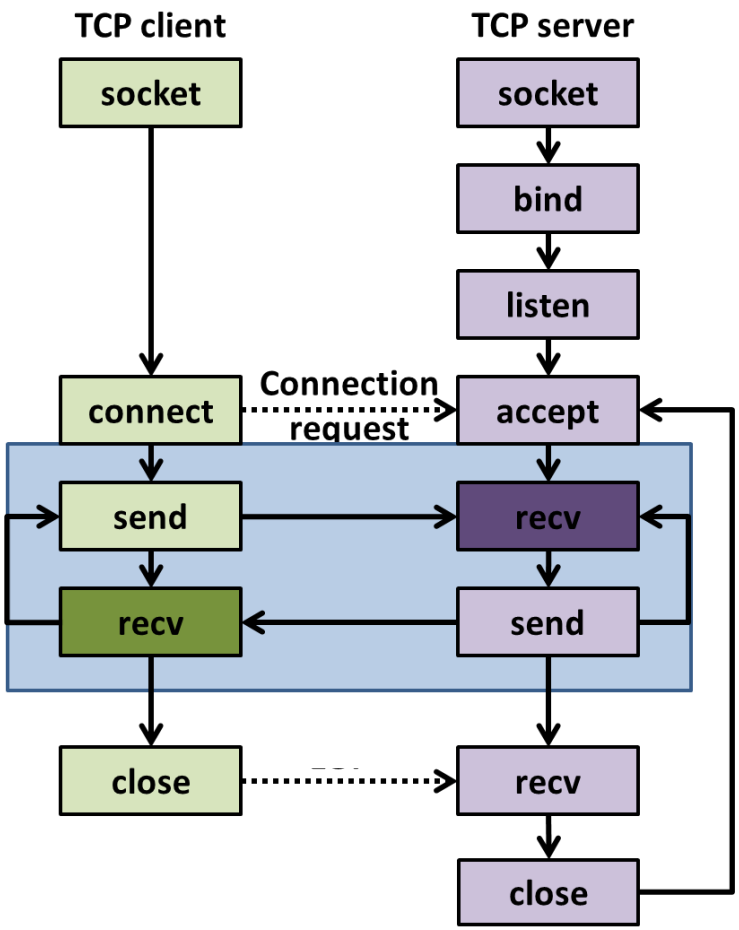
Sendall

- socket.socket.**sendall**(
  bytes
  [, flags])
- A socket objektum metódusa
- Az előzőhöz hasonló
- A különbség: addig küldi az adatot a *bytes*-ból, ameddig az összes át nem ment, vagy hiba nem történt (ebben az esetben már nem lehet kideríteni, hogy mennyi adat ment át)
- visszatérési érték: None, ha sikeres volt



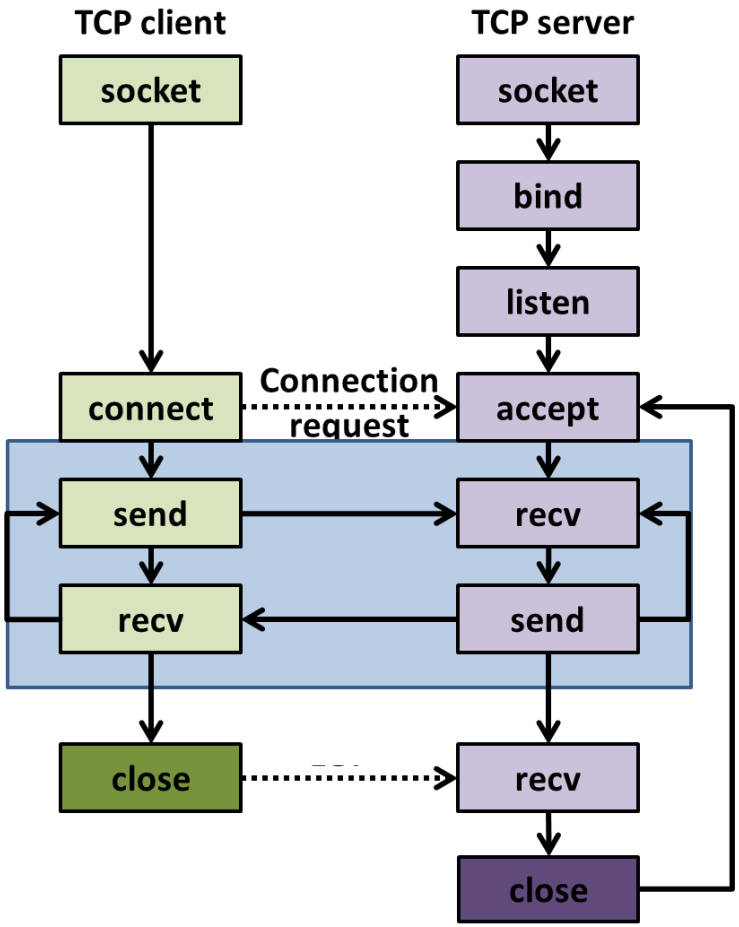
Recv

- socket.socket.**recv**(*bufsize* *[, flags]*)
- A socket objektum metódusa
- Üzenet fogadása
- *bufsize* : a max. adatmennyiség, amelyet egyszerre fogadni fog
- *flags* : 0 (nincs flag meghatározva)
- visszatérési érték: a fogadott adat *bytes* reprezentációja



Close

- socket.socket.**close**()
- A socket objektum metódusa
- A socket lezárása:
  - az összes további művelet a socket objektumon el fog bukni
  - a túlsó végpont nem fog több adatot kapni
  - ez el fogja engedni a kapcsolathoz tartozó erőforrásokat, **de** nem feltétlen zárja le azonnal (ha erre szükség van, akkor érdemes **shutdown** hívást a **close** elé tenni)



Példa hívások TCP-nél II.

- connect

```
server_address = ('localhost', 10000)
sock.connect(server_address)
```

- send(), sendall()

```
connection.sendall(data)
```

- recv()

```
data = connection.recv(16)
```

- close()

```
connection.close()
```

Feladat1

- Készítsünk egy egyszerű kliens-server alkalmazást, ahol a kliens elküld egy 'Hello server' üzenetet, és a szerver pedig válaszol neki egy 'Hello kliens' üzenettel!
- Változtassuk meg hogy ne az előre megadott portot adjuk, hanem bemenetként kapjuk, vagy egy tetszőlegeset kapjunk az oprendszertől! (sys.argv[1])
- (A `netstat –a` (windows) vagy `netstat –tulpn` (linux) parancsokkal megnézhetjük a használt portokat.)

Listen – sok kliens

- Készítsünk egy olyan alkalmazást, ahol a szerver oldalon:

```
sock.listen(1)
```

- A klienshez tartozó szkriptben 3 db. kliens socket-et hozunk létre és mindegyikkel a szerverhez próbálunk csatlakozni egymásután
- Figyeljük meg mi történik!
- (Windows-nál a végtelen ciklusban futó szervert sima „Ctrl+C”-vel nem tudjuk kilőni parancssorban, hanem „Ctrl+Break” billentyűkombinációval lehet. A „Break” billentyű helye laptoponként eltérhet: pl. Ctrl+Fn+Pause, Ctrl+Fn+B stb.)

Feladat2

- Készítsünk egy szerver-kliens alkalmazást, ahol a kliens elküld 2 számot és egy operátort (négy alapművelet közül) a szervernek, amely kiszámolja és visszaküldi az eredményt. A kliens üzenete legyen struktúra.

Protokoll

HÁZI FELADAT II.

Feladat

Két részfeladatból fog állni:

1. A paraméterben kapott bináris fájlokat kell beolvasni és kiíratni az első soruk tartalmát a standard outputra! (A különböző fájlok sorainak formátuma hallgatónként változó lesz.)
2. Ki kell írni a standard outputra különböző értékeket bináris formátumban (azaz a pack eredményét)! (Az értékek hallgatónként változók lesznek.) A string hosszát a szöveg mögött lévő szám jelzi! Használandó struct paraméterek: f, i, c, ?, Xs (ahol a X a string hossza, pl: 3s)

Leadás: A program leadása a TMS rendszeren .zip formátumban, amiben egy client.py szerepeljen!

Határidő: TMS rendszerben

VÉGE
KÖSZÖNÖM A FIGYELMET!