

# Telekommunikációs Hálózatok

## 3. gyakorlat

Egyszerű számítások

# **SZÁMOLÓS FELADATOK**

# Alapfogalmak

- Frekvencia ( $f$ ): elektromágneses hullám másodpercenkénti rezgésszáma. (Mértékegység: Hertz (Hz)=1/s)
- Hullámhossz ( $\lambda$ ): két egymást követő hullámcsúcs (vagy hullámvölgy) közötti távolság
- Fénysebesség ( $c$ ): az elektromágneses hullámok terjedési sebessége vákuumban:  $3 \cdot 10^8$  m/s (Rézben és üvegszáliban ez a sebesség nagyjából a 2/3-adára csökken)
- Összefüggés:  $\lambda \cdot f = c$

# Feladat1

- Az antenna magassága közvetlenül függ a hullámhossztól, a leggyakoribb antenntípusnál a hullámhossz negyede vehető.
- A szokásos antennák hossza 1cm és 5m közé esik.
- Milyen frekvenciatartománynak felel ez meg?

# Feladat2

- Mekkora antenna szükséges a 2.31 GHz és 2.55 GHz közötti frekvencia-tartomány használatához?

# Alapfogalmak

- Tegyük fel, hogy egy modem 2400-szer tud mintát venni másodpercenként
- Egy mintát szimbólumnak hívunk, ami egy vagy több bit is lehet
- Mértékegység: Baud = szimbólumok száma/sec (szimbólum ráta)  
→ ← adatrátát: bit/sec
- $\text{adatrátát} = \text{szimbólum ráta} \times \log_2(\text{szimbólumok száma})$
- Ha a szimbólumnál a 0 volt a 0 bitet, az 1 volt az 1 bitet jelöli (bináris jelátvitel), akkor a szimbólum ráta = adatrátát = 2400 bit/sec
- Ha a szimbólumnál 8 különböző feszültség szintet (0,...,7 V) is megkülönböztetünk, akkor egy szimbólum 3 bitnyi információt fog hordozni (pl. 101 bitek az 5 V-os szintnek felelnek meg). Itt a szimbólum ráta 2400 Baud, de az adatrátát 7200 bit/sec

# Feladat3

- Egy szimbólum átviteléhez szükséges idő  $100\mu s$ . Mekkora a szimbólumráta? Mekkora az adatrátá, ha 2,4,16 szimbólumot használunk?

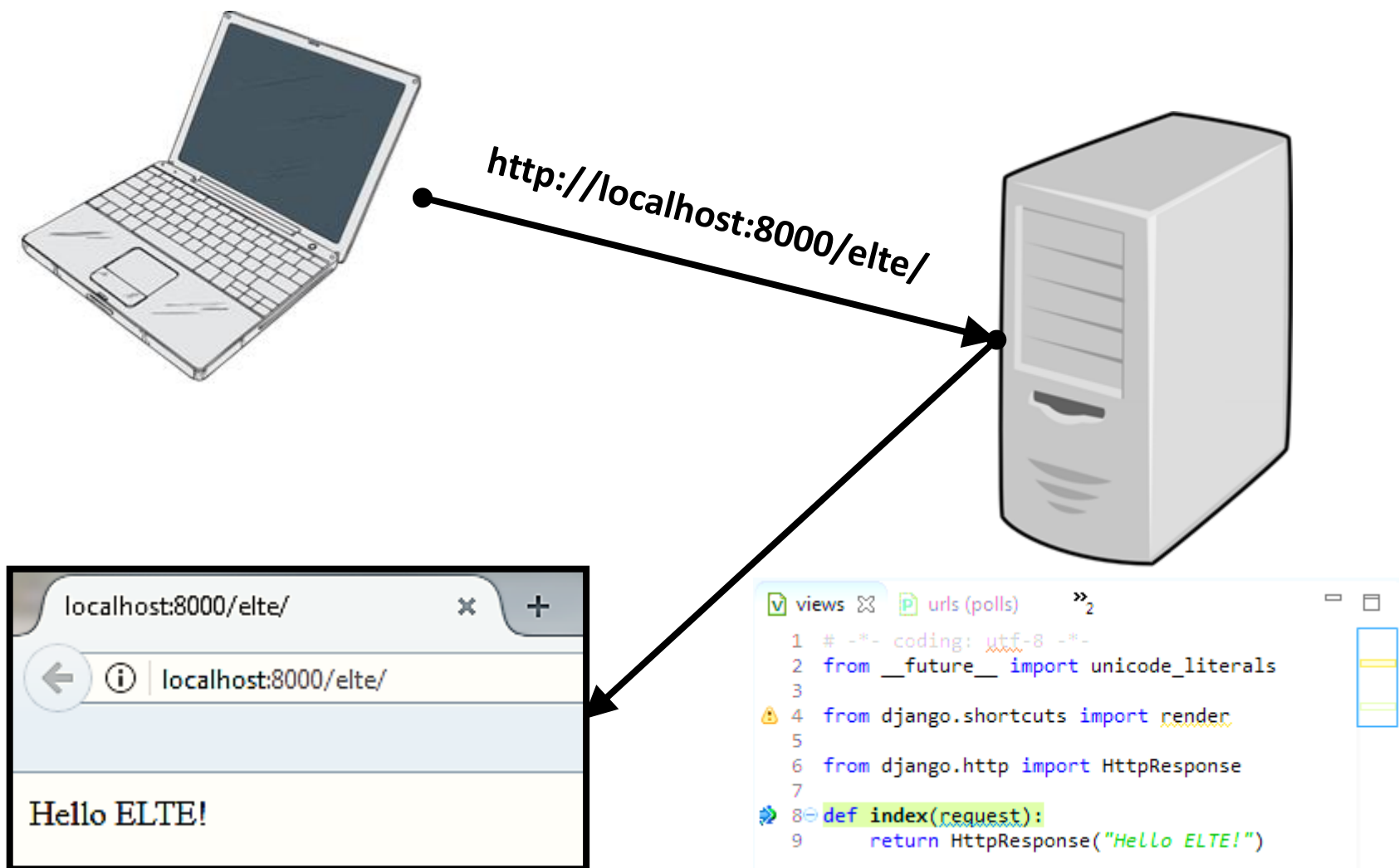
# Feladat4

- Egy küldő egy üvegszál kábelben egy fényszignált küld  $PS$  teljesítménnyel. Tegyük fel, hogy a fogadónál ennek a szignálnak legalább  $PS/1000$  teljesítménnyel kell megérkezni ahhoz, hogy fel tudja ismerni. A kábelben a szignál teljesítményének csökkenése kilométerenként 8%. Milyen hosszú lehet a kábel?



# **PYTHON SOCKET PROGRAMOZÁS I.**

# Kis motiváció...



# Kis motiváció...

The screenshot shows a Python IDE with a debugger window at the top and a code editor at the bottom.

**Debugger Window:**

- Variables:** A table showing the current state of variables.

Name	Value
self	WSGIServer: <django.core.servers.basehttp.WSGIServer object at 0x00000000056D3B70>
server address	<type 'tuple': ('127.0.0.1', 8000)>
socket	_socketobject: <socket._socketobject object at 0x00000000057752B8>
_sock	socket: <socket object, fd=768, family=2, type=1, protocol=0>
family	int: 2
proto	int: 0
timeout	float: -1.0
type	int: 1
- Breakpoints:** Empty.
- Console:** Displays the output of the current step: `socket: <socket object, fd=768, family=2, type=1, protocol=0>`

**Code Editor:**

- Left Panel (Debugger View):** Shows the call stack. The current frame is `process_request_thread [SocketServer.py:599]`. Other frames include `run [threading.py:754]`, `__bootstrap_inner [threading.py:801]`, `__bootstrap [threading.py:774]`, `index [views.py:8]`, `Thread-7 - pid_6092_id_91750128`, `Dummy-5 - pid_6092_id_90041424`, `MainThread - pid_6092_id_39545096`, `manage.py`, and `DjangoExample DjangoExample [PyDev Django]`.
- Right Panel (Code View):** Shows the source code of the `ThreadingMixIn` class. The current line is 599, which is highlighted in the debugger. The code is as follows:

```
585 class ThreadingMixIn:
586     """Mix-in class to handle each request in a new thread."""
587
588     # Decides how threads will act upon termination of the
589     # main process
590     daemon_threads = False
591
592     def process_request_thread(self, request, client_address):
593         """Same as in BaseServer but as a thread.
594
595         In addition, exception handling is done here.
596
597         """
598         try:
599             self.finish_request(request, client_address)
600             self.shutdown_request(request)
601         except:
602             self.handle_error(request, client_address)
603             self.shutdown_request(request)
```

# Socket programozás

- Mi az a *socket*?
- A socket-ekre **kommunikációs végpontokként** gondolhatunk különböző programok közti adatcsere esetén (amelyek lehetnek ugyanazon a gépen, vagy különböző gépeken). Ezeket különböző operációs rendszerek támogatják, mint például: Unix/Linux, Windows, Mac stb.
- Egy program a saját socket-én keresztül ír és olvas egy másik program socket-jére/socket-jéről ⇒ az **adatátvitelt a socket-ek intézik** egymás között.
- Ezeket jól lehet használni a **kliens-szerver** modellnél, ahol a kliens valamilyen szolgáltatást igényel, amelyet a szerver szolgál ki. (Például a kliens egy weboldalt igényelhet HTTP protokollon keresztül egy webszervertől.)

# A végpontok azonosítása, címzése hálózaton I.

- Két socket kommunikációjához  $\Rightarrow$  ismerni kell egymás azonosítóit
- Két részből állhat: IP címből és port számból
  - Az **IP címek** (IPv4 protokoll) 4 bájtos egész számok (pl. 157.181.152.1), amelyek (egyértelműen) azonosítják a gépeket, amelyek csatlakoznak az Internethez
  - Nehéz lenne megjegyezni ezeket  $\Rightarrow$  nevekre hivatkozunk helyettük (pl. [www.elte.hu](http://www.elte.hu)), amelyekhez tartozó IP címekre történő leképezéseket egy domain name server tud elvégezni
  - Spec. jelentéssel bír a localhost (127.0.0.1)

# A végpontok azonosítása, címzése hálózaton II.

- Két socket kommunikációjához  $\Rightarrow$  ismerni kell egymás azonosítóit
- Két részből állhat: IP címből és port számból
  - Bizonyos protokollokhoz tartoznak fix portszámok, konstansok (szállítási protokollok)!
  - A **port számok** 2 bájtos egész számok, amelyek a gépen belül futó alkalmazásoknak az azonosításában segítenek
  - A portok közül vannak, amelyek foglaltak (pl. ftp a 21-es port, ssh a 22-es, http 80-as...)
  - A 0-s portnak spec. jelentése van  $\Rightarrow$  az OS keres egy szabad portot

# Python socket, host név feloldás

- Socket csomag használata

```
import socket
```

- `gethostname()`

```
hostname = socket.gethostname()
```

- `gethostbyname()`

```
hostip = socket.gethostbyname('www.example.org')
```

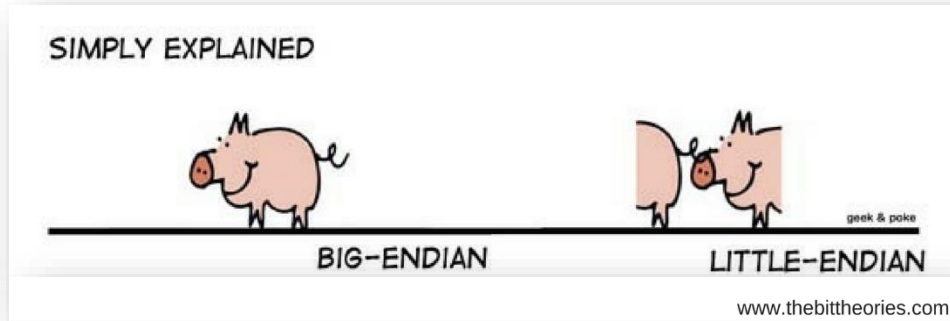
- `gethostbyname_ex()`

```
hostname, aliases, addresses = socket.gethostbyname_ex(host)
```

- `gethostbyaddr()`

```
hostname, aliases, addrs = socket.gethostbyaddr('157.181.161.79')
```

# Bájtsorrendek



00000000 00000000 00000100 00000001

Address	Big-Endian representation of 1025	Little-Endian representation of 1025
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

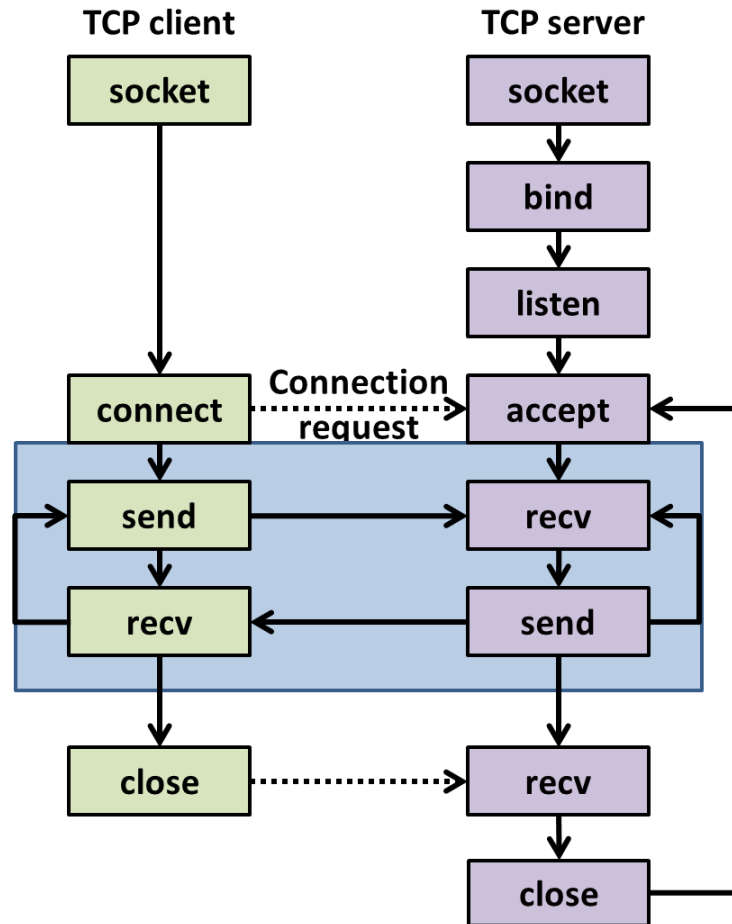
- A hálózati bájtsorrend big-endian (a magasabb helyi értéket tartalmazó bájtsor van elől)
- Hoszt esetén bármi lehet: big- vagy little-endian
- Konverzió a sorrendek között:
  - 16 és 32 bites pozitív számok kódolása
  - `socket.htons()`, `socket.htonl()` – host to network short / long
  - `socket.ntohs()`, `socket.ntohl()` – network to host short / long



# A kommunikációs csatorna kétféle típusa

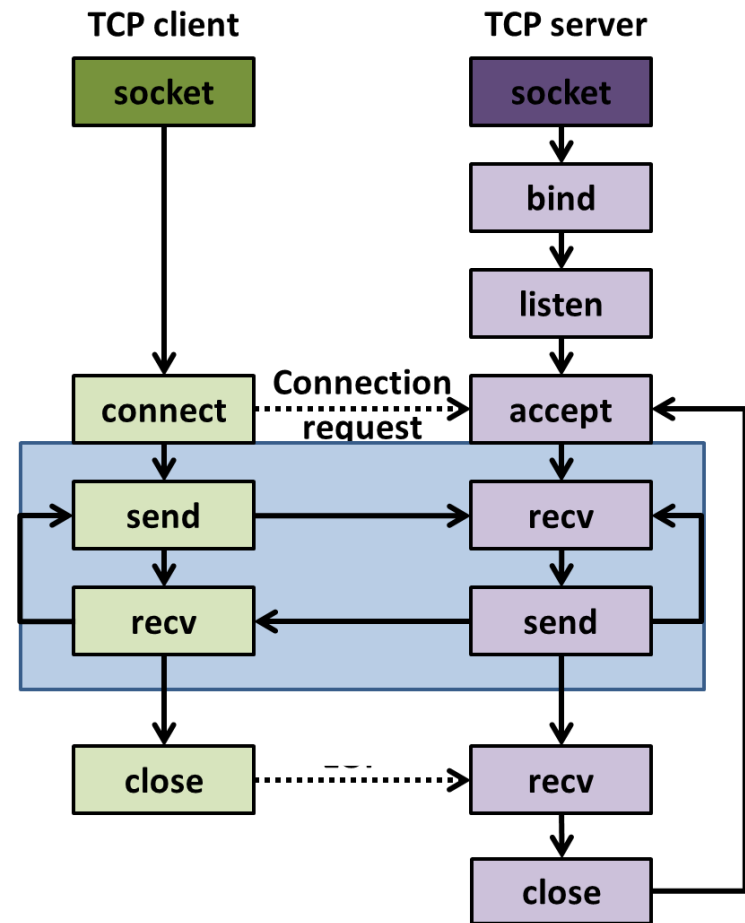
- Kapcsolat-orientált modell (analógia: telefonbeszélgetés)
  - csomagok megérkeznek jó sorrendben
  - ilyen protokoll a TCP
  - kapcsolódó típus: stream socket
- Kapcsolat-nélküli modell (analógia: postai levelezés)
  - csomagok nem biztos, hogy sorrend helyesen érkeznek, sőt el is veszhetnek
  - előnye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

# TCP



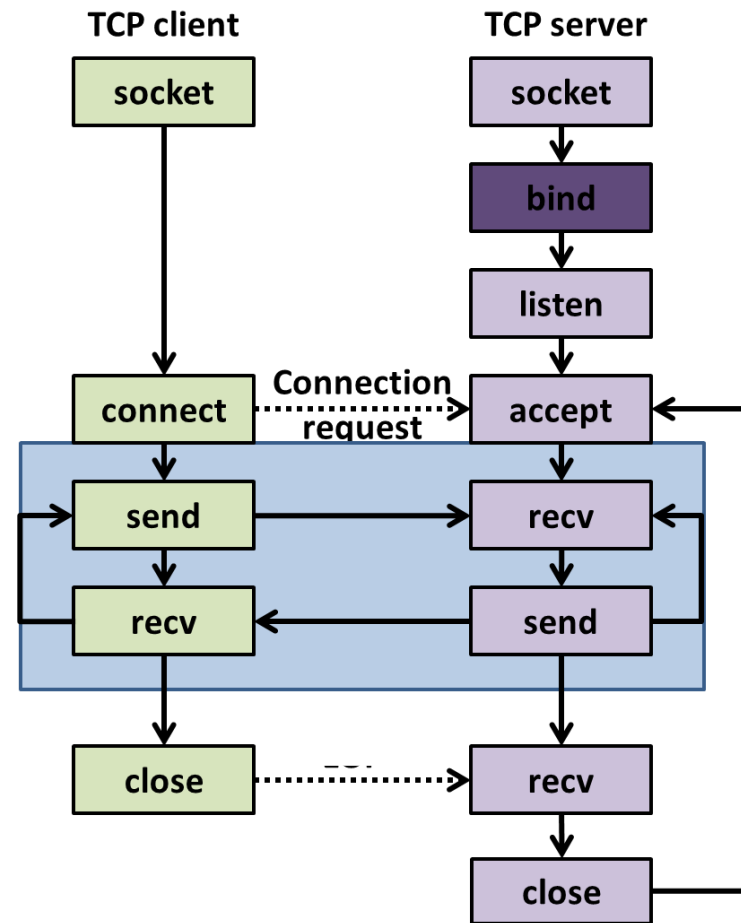
# Socket leíró beállítása

- `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



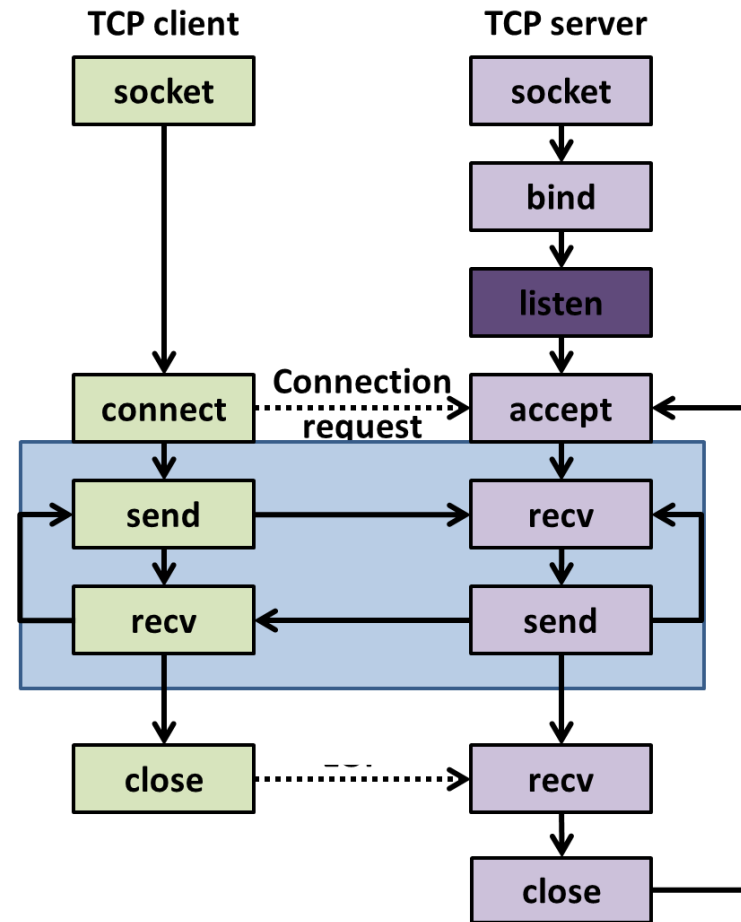
# Bindolás

- `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



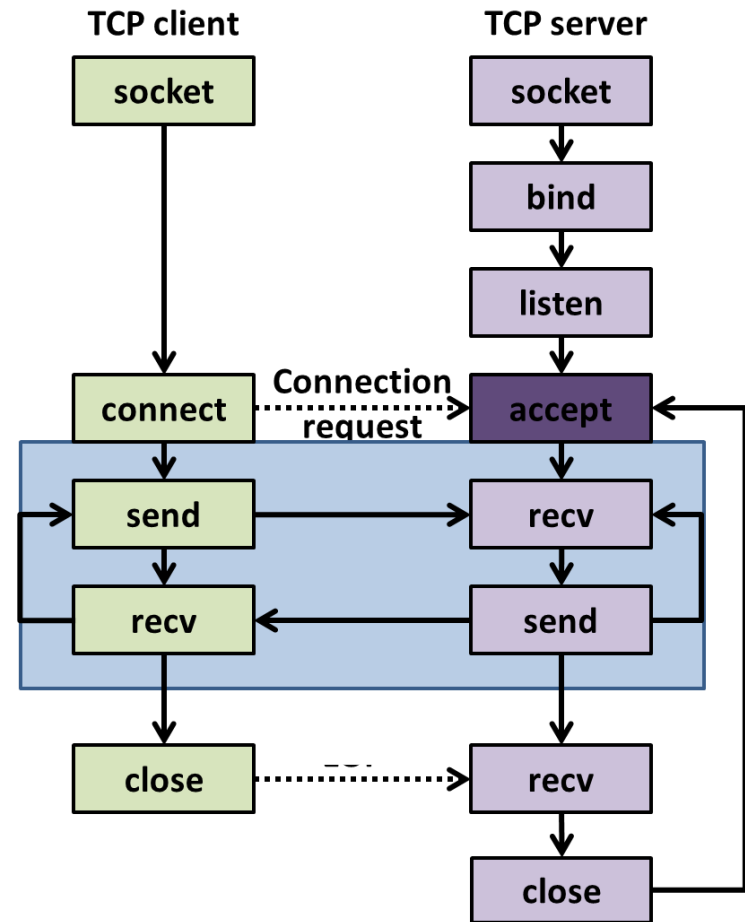
# Listen

- `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()`

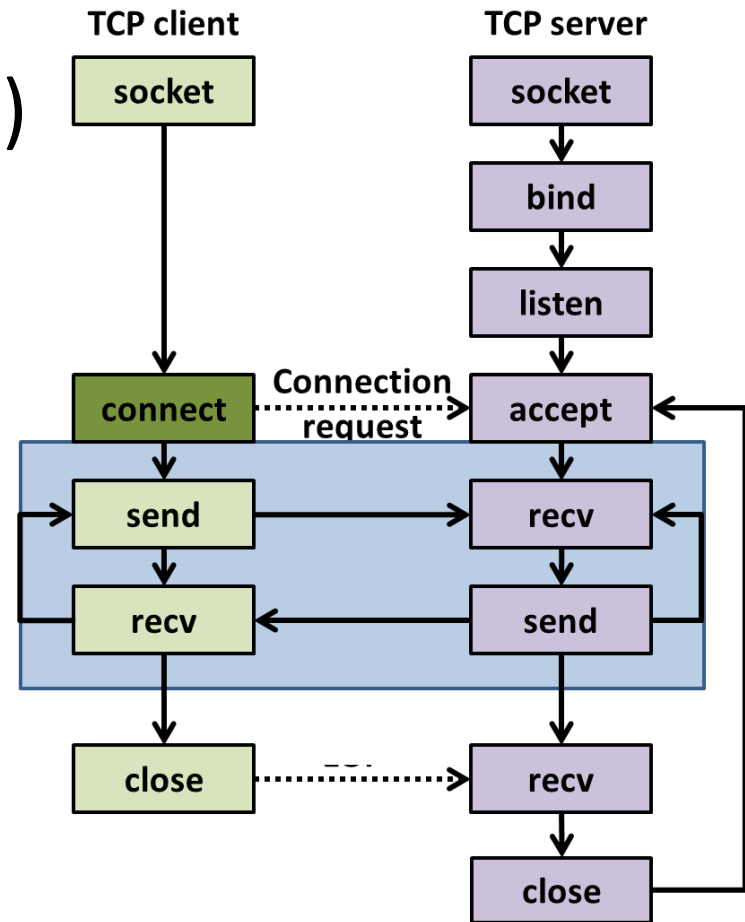
```
sock.listen(1)
```

- `accept()`

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

# Connect

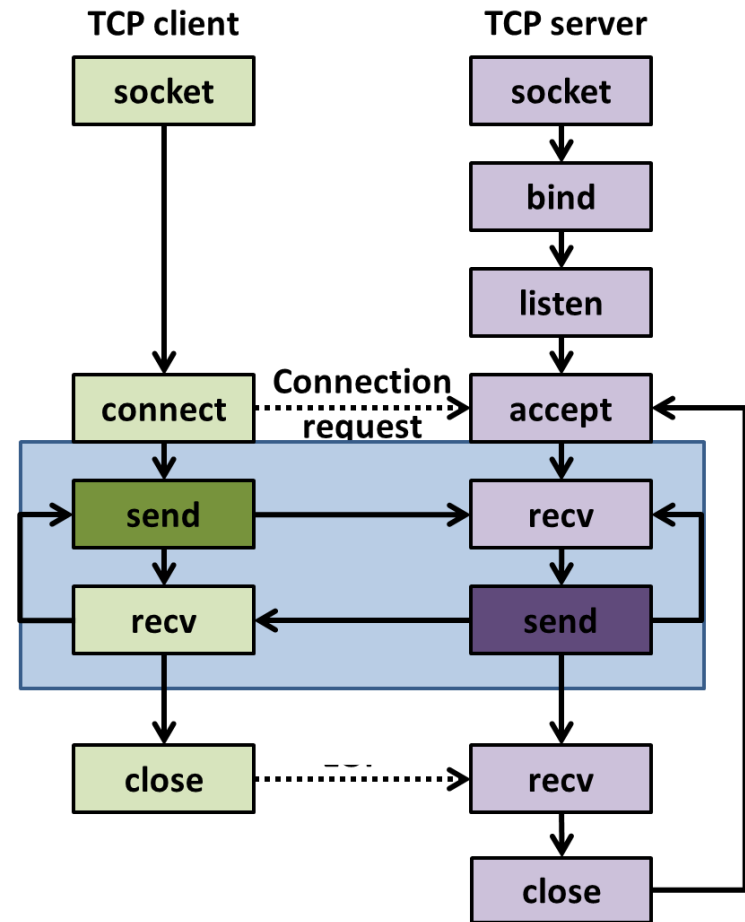
- `socket.socket.connect(address)`
- A socket objektum metódusa
- Kapcsolódás megkezdése egy távoli sockethez az *address* címen (ezt például kezdeményezheti egy kliens)
- Az *address* típusát ld. a **bind** függvénynél





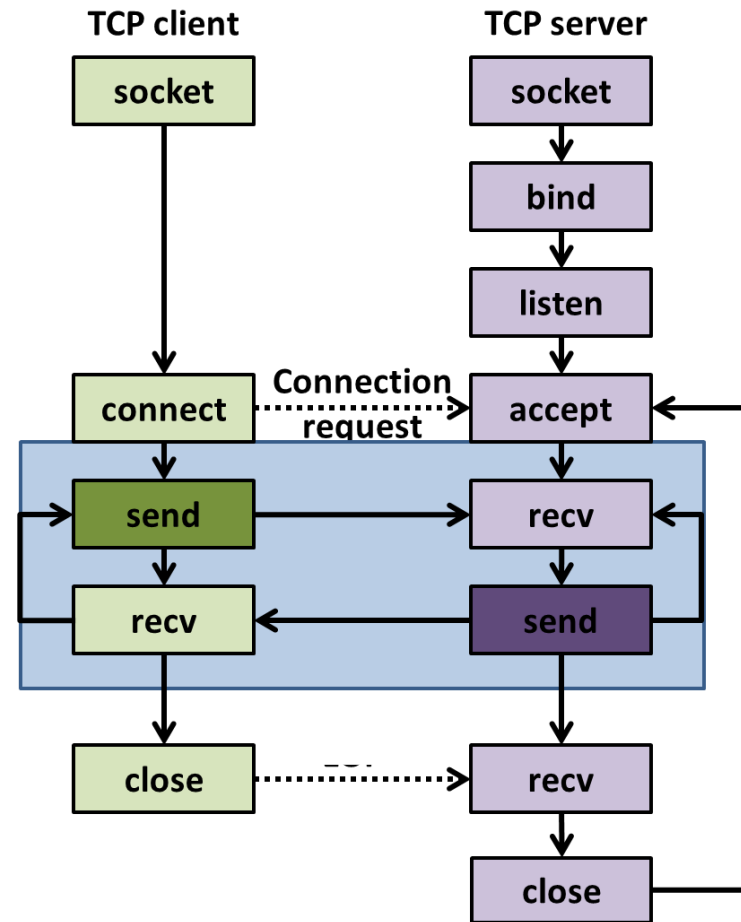
# Send

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



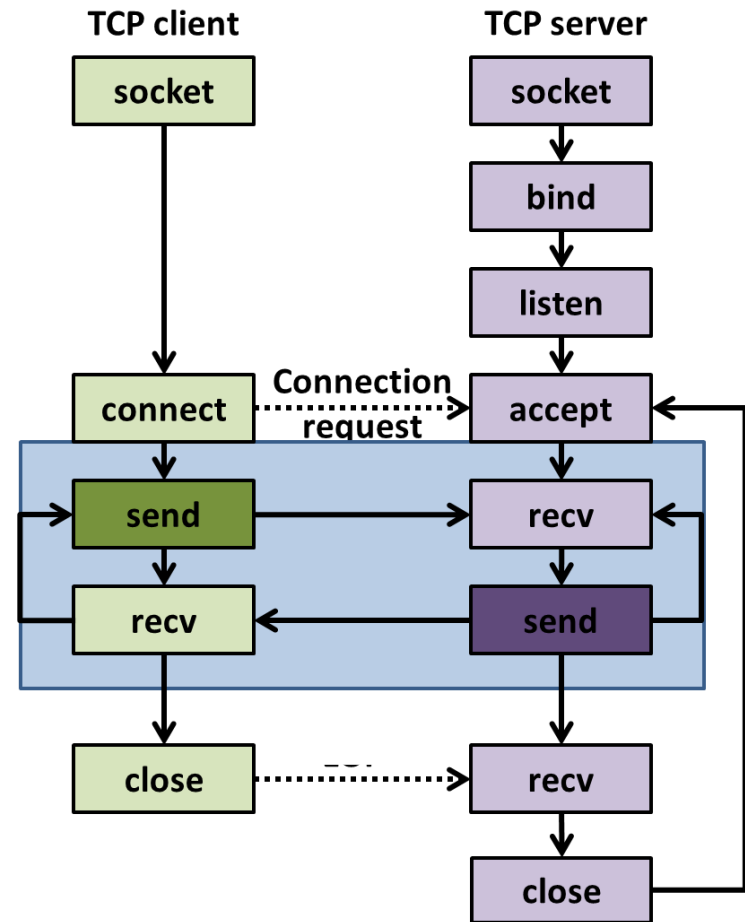
# Send

- `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



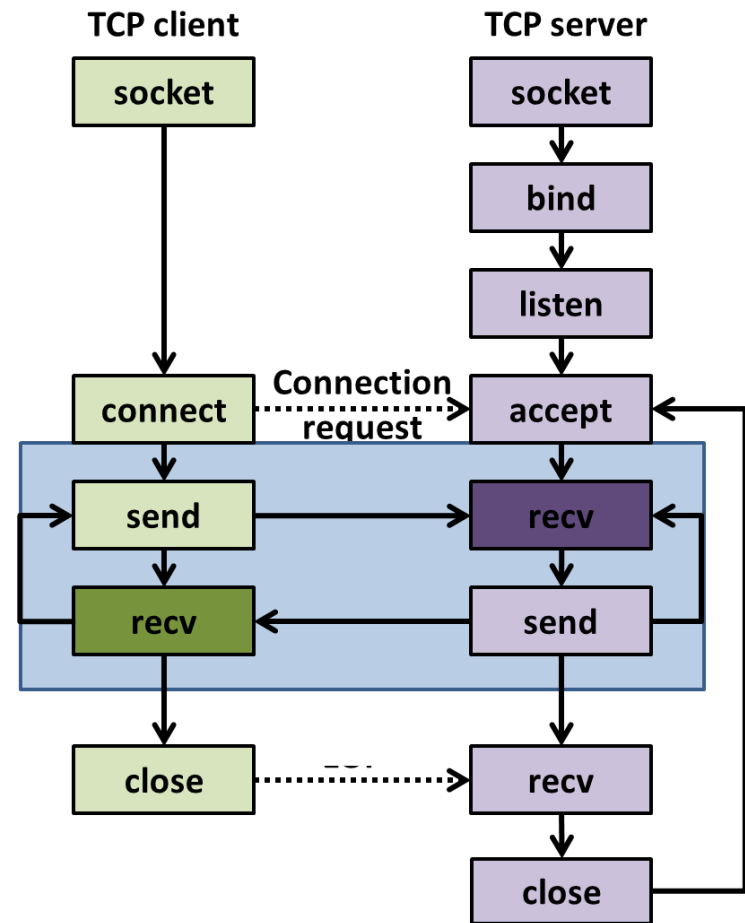
# Sendall

- `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



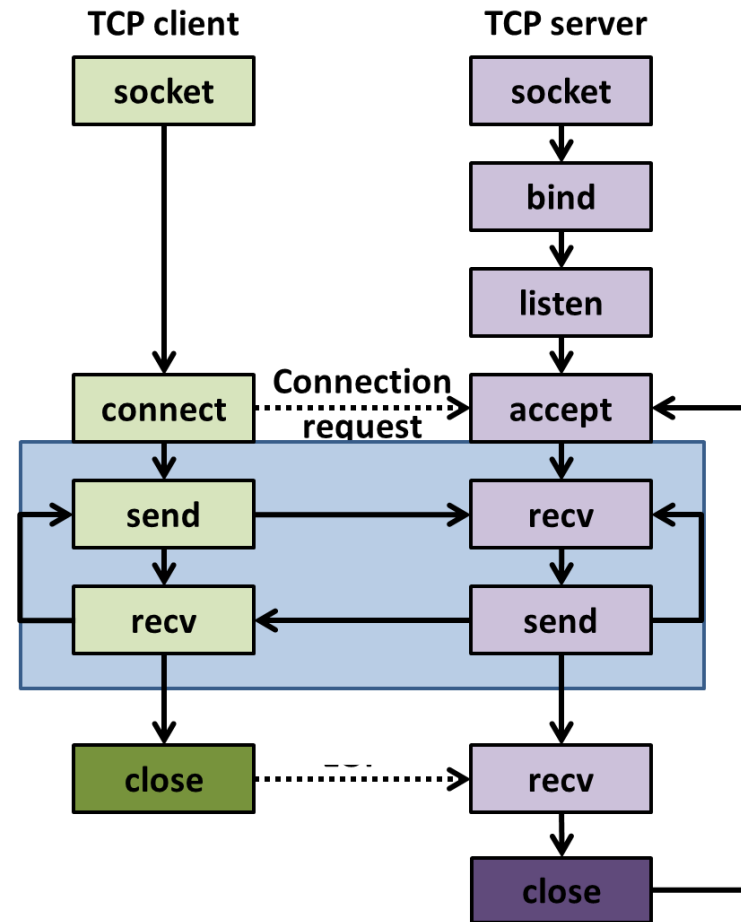
# Recv

- `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()
```

# Feladat5

- 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 tetszőlegesen kapjunk az oprendszerből!  
(`sys.argv[1]`)
- (A `netstat -a` (windows) vagy `netstat -tulnp` (linux) parancsokkal megnézhetjük a használt portokat.)

# A struct modul

- A `struct.pack(fmt, v1, v2, ...)`: egy sztringgel tér vissza, amely az adott *fmt* formátumnak megfelelően csomagolja be a bemenetként kapott értéke(ke)t (binárissá alakítja)
  - pl. formátumoknál a 'b' (előjeles) char C-típust (1-bájtos egész),
  - a '4sL' 4 méretű char tömböt és egy (előjel nélküli) long C-típust (4-bájtos egész) jelöl



# Struktúráküldése

- Binárisá alakítjuk az adatot
  - A Struct konstruktorában a formátumot adjuk meg, - hasonlóan az előbbihez, - amely alapján írja/olvassa a bináris adatot
  - A \*-operátor az alábbi esetben úgy fog viselkedni, mintha ','-vel elválasztva felsoroltuk volna a *values* elemeit

```
import struct
values = (1, 'ab', 2.7)
packer = struct.Struct('l 2s f')          #Int, char[2], float
packed_data = packer.pack(*values)
```

- Visszaalakítjuk a kapott üzenetet

```
import struct
unpacker = struct.Struct('l 2s f')
unpacked_data = unpacker.unpack(data)
```

- megj.: integer 1 – 4 byte, sztringként 1 byte, azaz hatékonyabb sztringként átküldeni.

# Feladat6

- 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.

**VÉGE**  
**KÖSZÖNÖM A FIGYELMET!**