Számítógépes Hálózatok

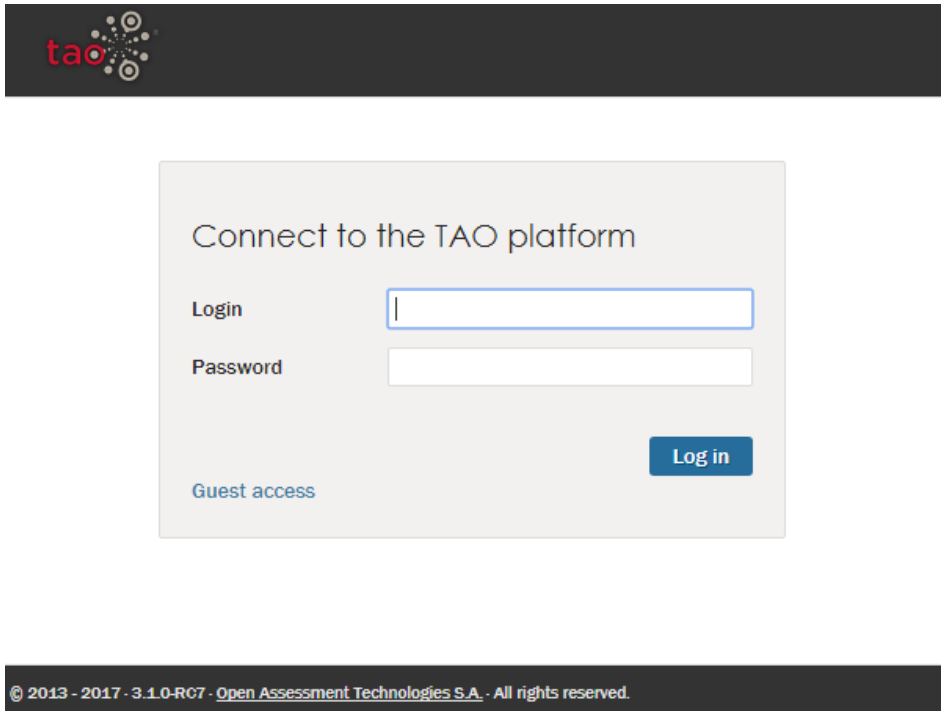
2. gyakorlat

Elérhetőségek

- honlap: http://szalaigj.web.elte.hu/
- email: szalaigindl@inf.elte.hu
- szoba: 2.507 (déli tömb)

Óra elei kisZH

- Elérés:
  - https://oktnb16.inf.elte.hu

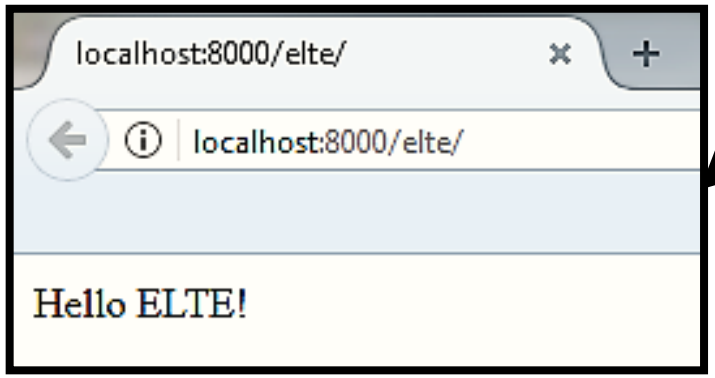


Gyakorlat tematika

- Pyton socket
- Host, port, protokoll
- Bájtsorrendek: little endian, big endian
- Socket TCP kliens-szerver

Kis motiváció…

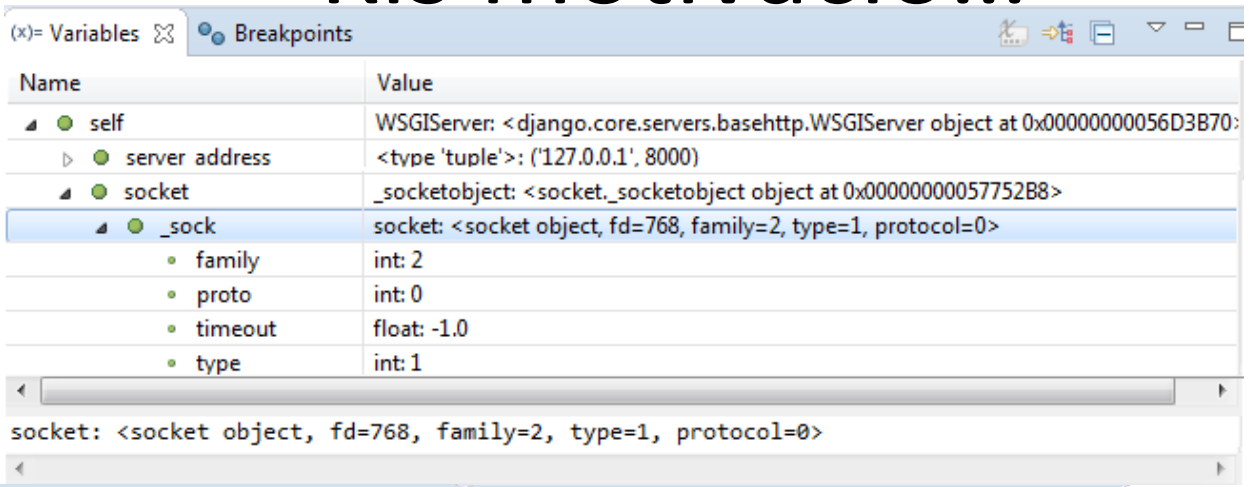
http://localhost:8000/elte/

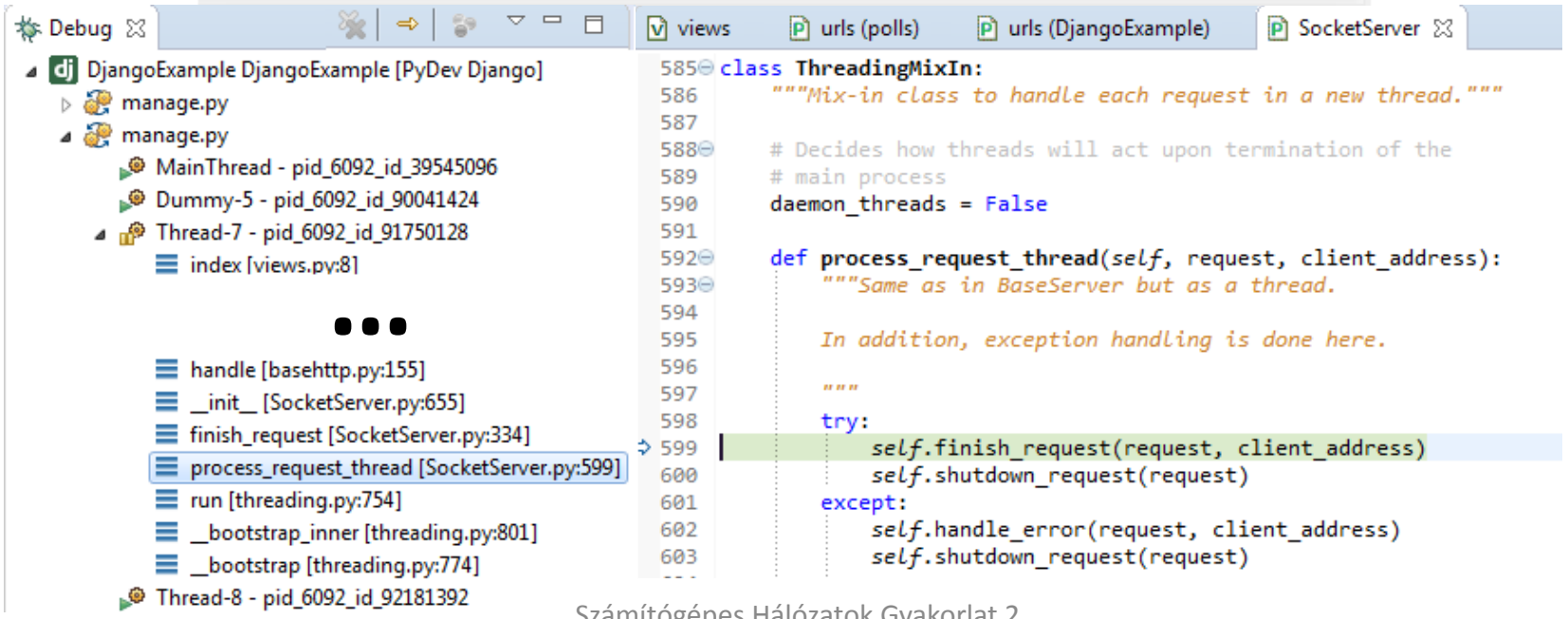


views urls (polls)

```
1  # -*- coding: utf-8 -*-
2  from __future__ import unicode_literals
3
4  from django.shortcuts import render
5
6  from django.http import HttpResponse
7
8  def index(request):
9      return HttpResponse("Hello ELTE!")
```

Kis motiváció…





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 ⇒ 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 ⇒ nevekre hivatkozunk helyettük (pl. www.elte.hu), amelyekhez tartozó IP címekre történő leképezéseket egy domain name server tud elvégezni
  - Spec. jelentéssel bír a localhost (127.0.0.1)
  - Port számokról később

Pyton socket, host név feloldás

- Socket csomag használata

```
import socket
```

- gethostname()

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

- gethostbyname()

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

- gethostbyname_ex()

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

- gethostbyaddr()

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

Feladat1

- Írassuk ki a gethostname() fv. eredményét!
- Következő hostnevekre futtassuk meg a gethostbyname() és a gethostbyname_ex() fv-eket:
  - 'homer', 'www.python.org', 'inf.elte.hu',
- Kérdezzük le a gethostbyaddr() fv segítségével a következő IP címek hostneveit:
  - '157.181.161.79', '185.43.207.92'

A végpontok azonosítása, címzése hálózaton II.

- Két socket kommunikációjához ⇒ ismerni kell egymás azonosítóit
- Két részből állhat: IP címből és port számból
  - Az IP címek már voltak
  - 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 ⇒ az OS keres egy szabad portot

Port számok és protokollok

- getservbyname()

```
import urlparse
parsed_url = urlparse.urlparse(url)
port = socket.getservbyname(parsed_url.scheme)
```

- getservbyport()

```
print urlparse.urlunparse(
      (socket.getservbyport(port), 'example.com', '/', '', '', '')
 )
```

- getprotobyname()

```
print socket.getprotobyname('icmp')
```

Feladat2.a.

- Kérdezzük le a portszámot a következő URL-ekhez:
  - 'http://www.example.com',
  - 'https://www.example.com',
  - 'ftp://example.com',
  - 'gopher://gopher.example.com',
  - 'smtp://mail.example.com',
  - 'imap://mail.example.com',
  - 'imaps://mail.example.com',
  - 'pop3://pop.example.com',
  - 'pop3s://pop.example.com',

Feladat2.b.

- Készítsük el a következő portokhoz az urleket:
  - 80, 443, 21, 70, 25, 143, 993, 110, 995
- Kérdezzük le 1..100-ig a fenntartott portokat!
- Kérdezzük le a szállítói réteghez tartozó konstanst a következő protokolokhoz:
  - 'icmp', 'tcp', 'udp'

Szerver adatok lekérdezése

- socket.**getaddrinfo**(*host*, *port*[, *family*[, *socktype*[, *proto*[, *flags*]]]])
- A bemenetként kapott *host* és *port* argumentumokat fordítja le egy tuple*-re úgy, hogy az összes szükséges infó meglegyen ahhoz, hogy kapcsolódjunk hozzá:

(family, socktype, proto, canonname, sockaddr)

- A *family*, *socktype* és *proto* arg.-okkal szűkíteni lehet a visszatérő címlistát
- A *flags*-nél különböző AI_* konstansokat lehet megadni

* rendezett n-es, ~ csak olvasható lista

Szerver adatok lekérdezése

- getaddrinfo()

```
for response in socket.getaddrinfo('www.python.org', 'http'):
    family, socktype, proto, canonname, sockaddr = response
```

```
for response in socket.getaddrinfo('www.python.org', 'http',
                           socket.AF_INET,      # family
                           socket.SOCK_STREAM,  # socktype
                           socket.IPPROTO_TCP,  # protocol
                           socket.AI_CANONNAME, # flags
                           ):
    family, socktype, proto, canonname, sockaddr = response
```

Feladat3

- Kérdezzük le a 'www.python.org' 'http' szerverének az információit!

- Kérdezzük le a 'www.inf.elte.hu' oldal 'http' servernevét, ahol meg van adva a név flag (AI_CANNONNAME)

Bájtsorrendek

00000000 00000000 00000100 00000001

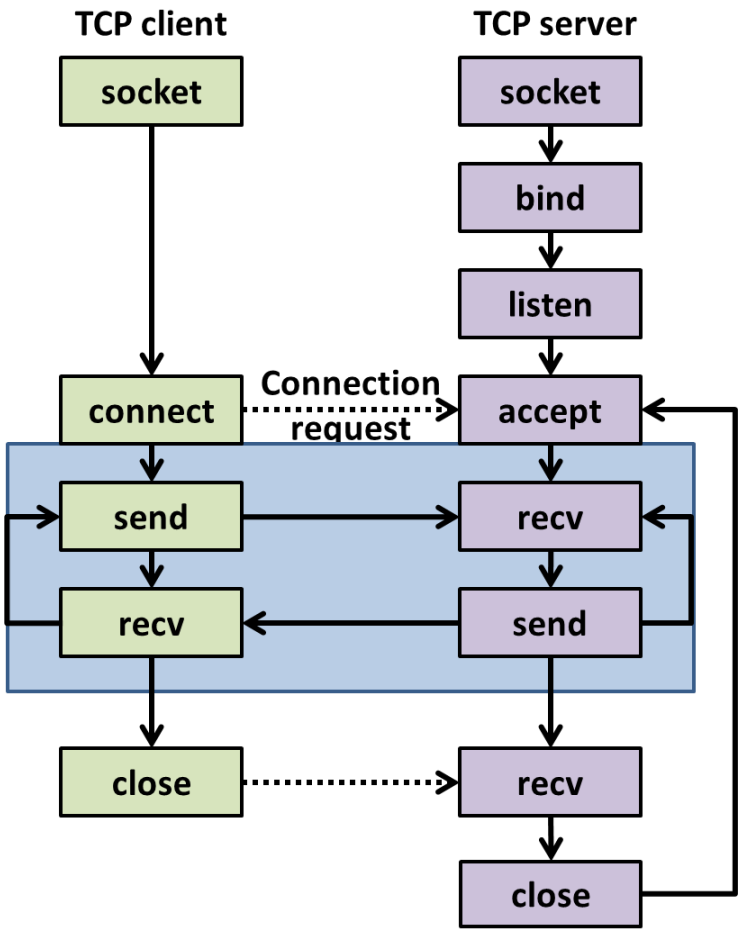
Address	Big-Endian representation of 1025	Little-Endian representation of 1025
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

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

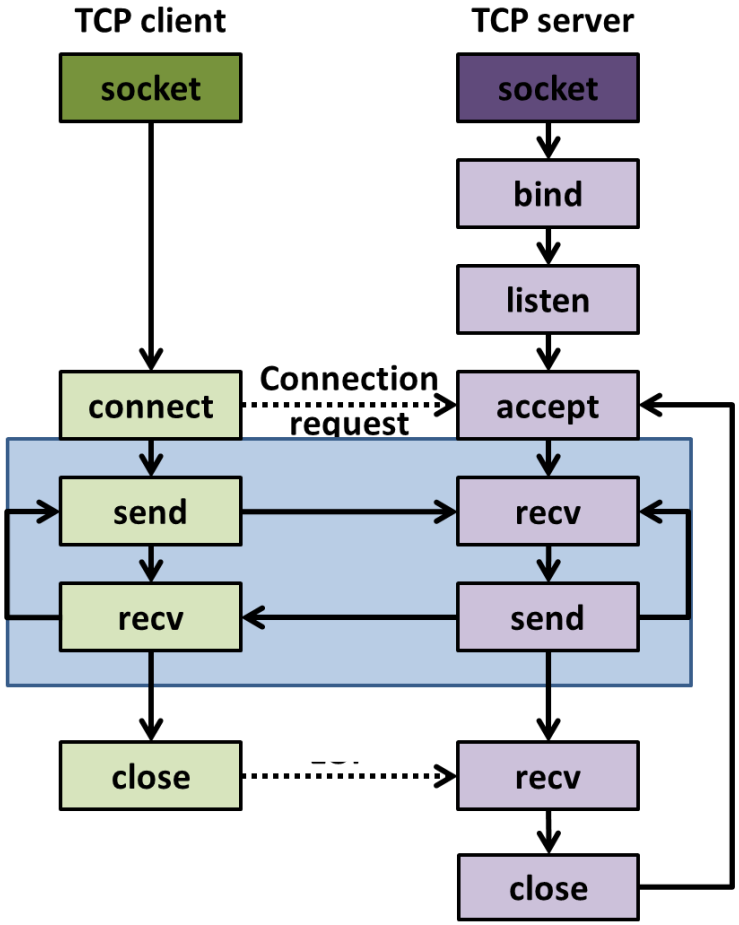

TCP client
TCP server
socket
socket
bind
listen
connect
Connection request
accept
send
recv
recv
send
close
recv
close

Socket leíró beállítása

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

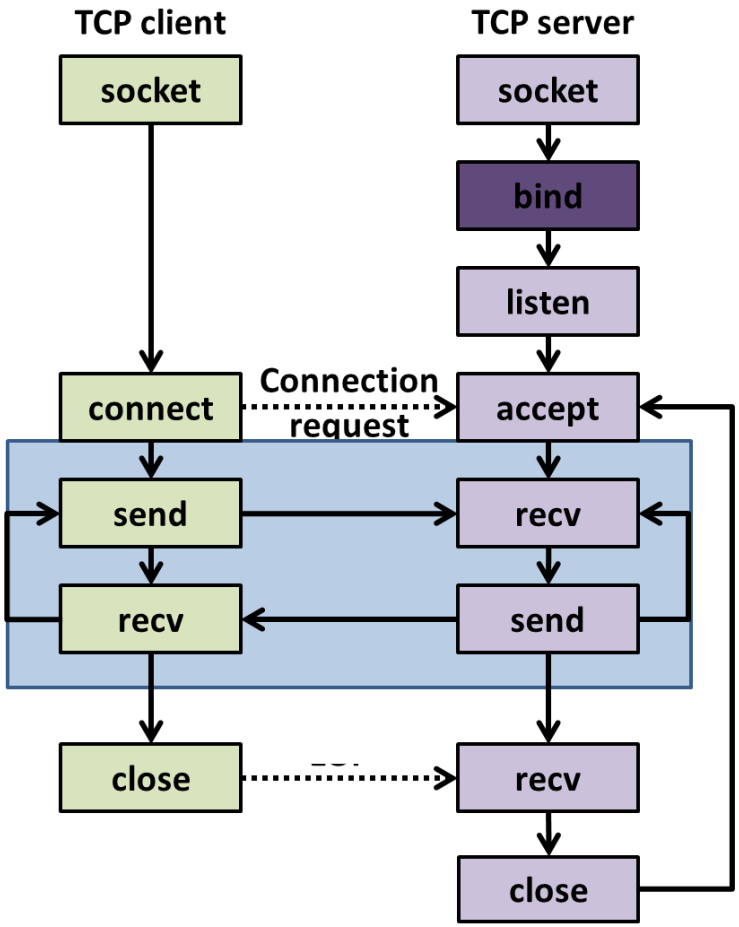
(alapértelmezett protokoll lesz)

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



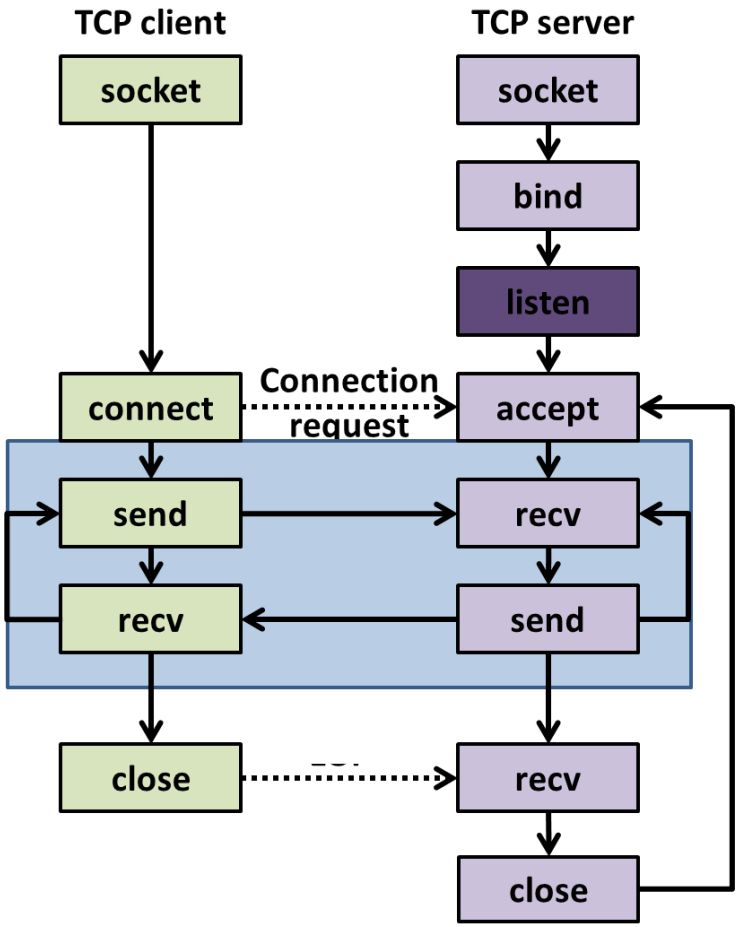
Bindolás

- 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



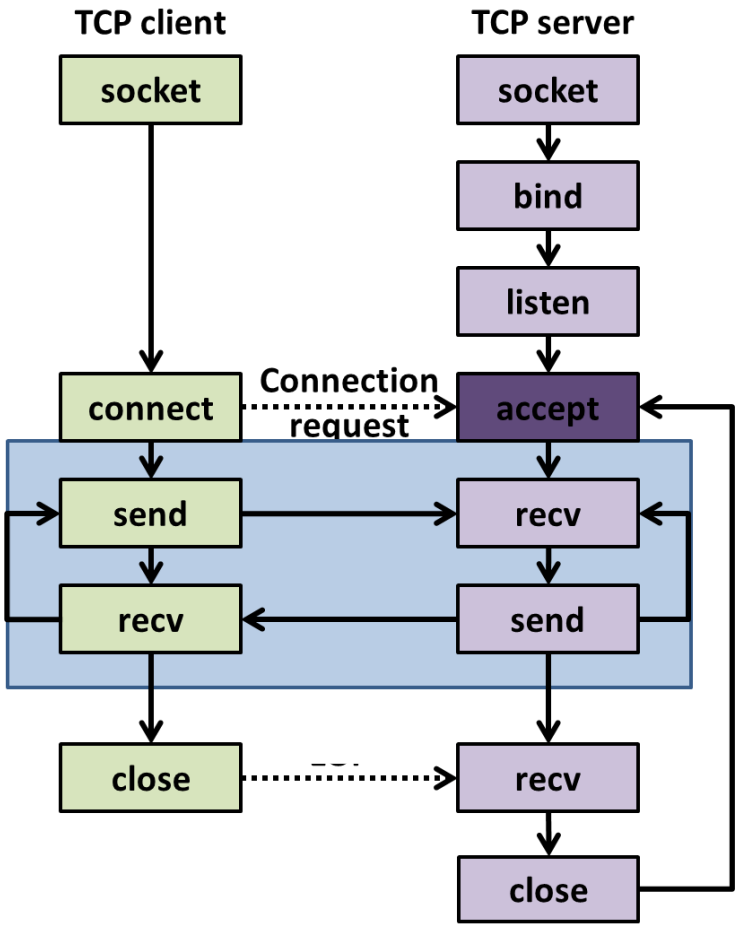
Listen

- 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()

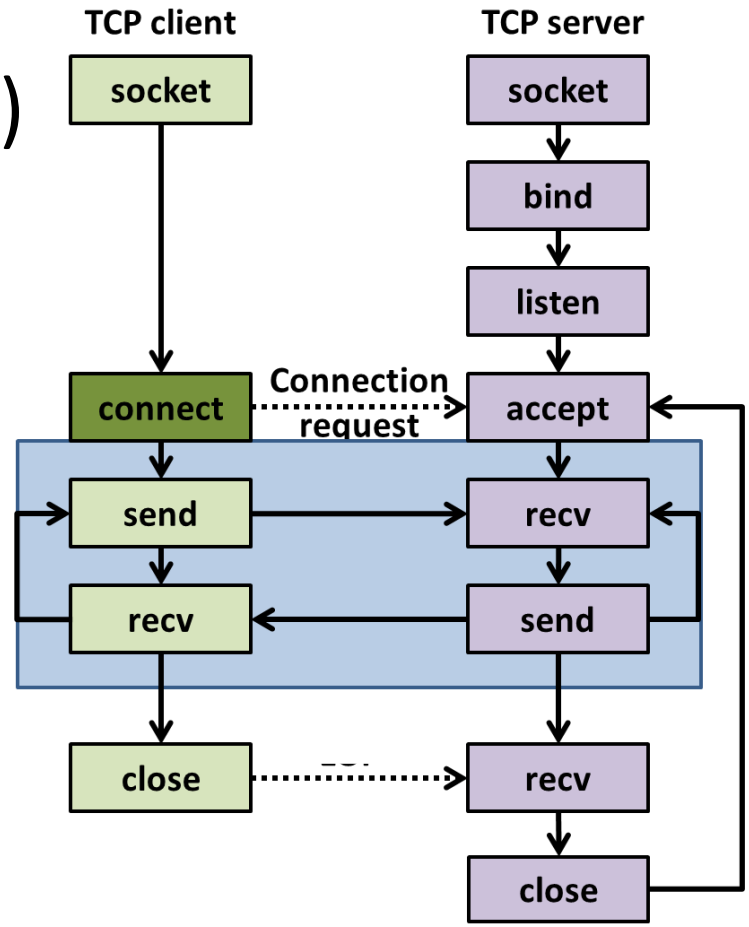
```
sock.listen(1)
```

- accept()

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

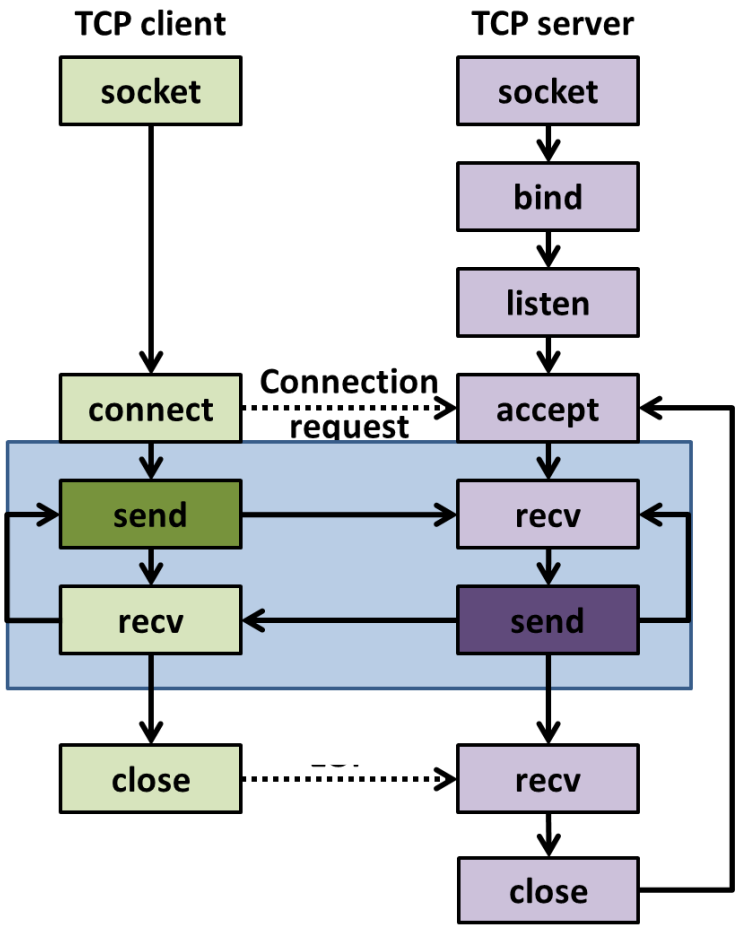
Connect

- 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



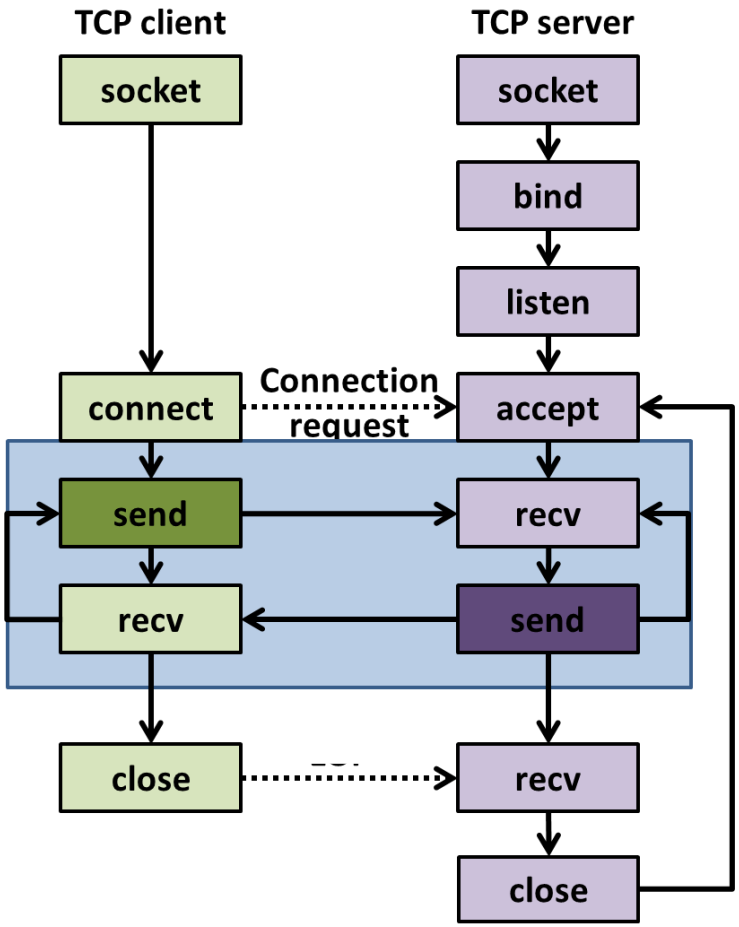
Send

- socket.socket.**send**(*string* *[, flags]*)
- A socket objektum metódusa
- Adatküldés (*string*) a socketnek
- *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



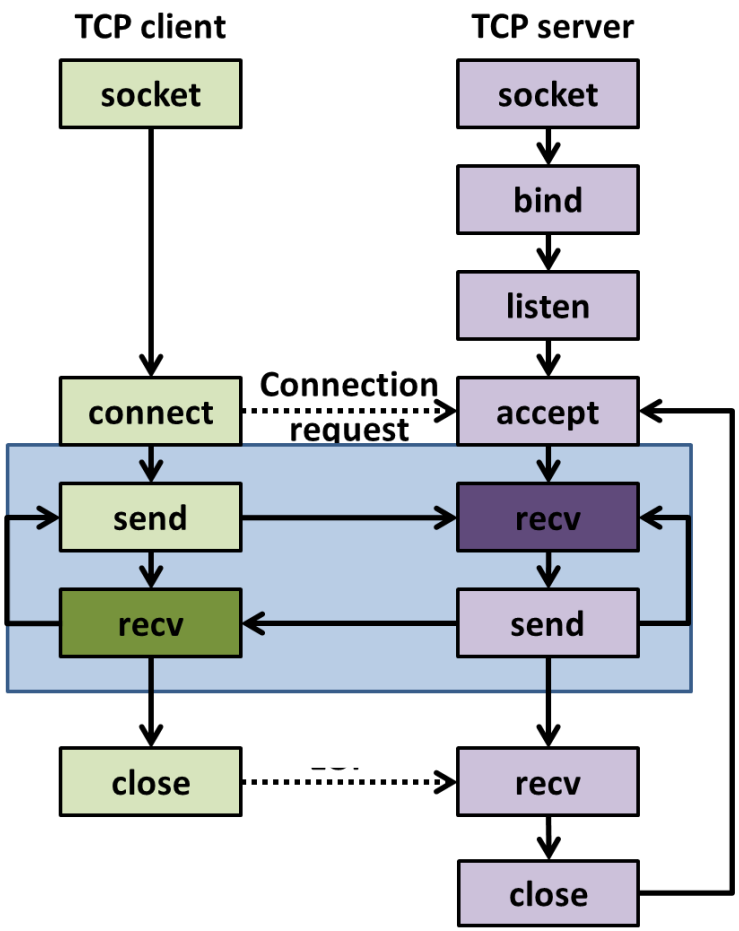
Sendall

- socket.socket.**sendall**(*string* *[, flags]*)
- A socket objektum metódusa
- Az előzőhöz hasonló
- A különbség: addig küldi az adatot a *string*-ből, 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



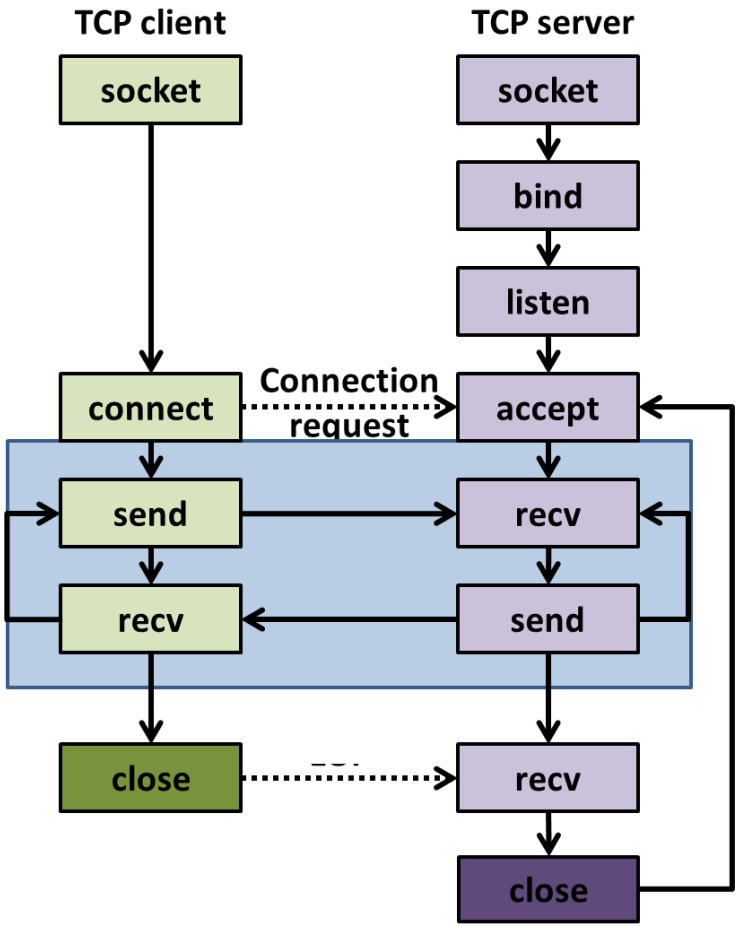
Recv

- 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 sztring 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()
```

Feladat4

- 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.)

VÉGE
KÖSZÖNÖM A FIGYELMET!