

Telekommunikációs Hálózatok

6. gyakorlat

PYTHON SOCKET - PROXY

Bemutató feladat I.

Készítsünk egy egyszerű TCP alapú proxyt (átjátszó). A proxy a kliensek felé szerverként látszik, azaz a kliensek csatlakozhatnak hozzá. A proxy a csatlakozás után kapcsolatot nyit egy szerver felé (parancssori argumentum), majd minden a kienstől jövő kérést továbbítja a szerver felé és a szervertől jövő válaszokat pedig a kliens felé.

Futtatása pl: `netProxy.py ggombos.web.elte.hu 80`

Ez alkalommal ne készítsünk saját klienset, hanem használjuk egy webböngészőt.

A proxy használja a localhost:10000 -es socket address-t.

Web böngészőbe írjuk be: localhost:10000

Bemutató feladat II. – Tiltsunk le valamilyen tartalmat

A SzamHalo-t tartalmazó URL-ek ne legyenek elérhetőek a proxyn keresztül.

A válasz legyen valamilyen egyszerű HTML üzenet, ami jelzi a blokkolást.

megj: header: „HTTP/1.1 404 Not Found\n\n”

Proxy

- Felhasználásai:
- adat továbbítás
(pl. el akarjuk rejteni a rendes szervert.)
 - folyam vezérlés és kiegyenlítés
(pl. hogy elosztott szerverek egyenletesen kapják a kéréseket.)
 - adat cache-elés
(pl. lehal a szerver, de a proxy ideiglenesen/statikusan kiszolgálja a (le)kéréseket.)
 - stb...
- Veszélyei:
- adatlopás
(pl. a proxy egy nagy cég honlapját játssza át hasonló domain névvel és ugyan úgy használható mint az eredeti, viszont lementi a begépelt jelszavakat)
 - stb...

Számológép Proxy

- Készítsünk a számológéphez egy proxy-t, ami értelmezi a klienstől kapott TCP kéréseket és "szabotálja" azokat.
- Szabotálás: az első operandust megszorozza kettővel, a másodikhoz pedig hozzáad egyet.
- A módosított kérést elküldi a TCP servernek, majd az eredményt visszaküldi a kliensnek.

BEADANDÓ IV. (1 PONT)

Beadandó

netcopy alkalmazás 1 pont

Készíts egy netcopy kliens/szerver alkalmazást, mely egy fájl átvitelét és az átvitt adat ellenőrzését teszi lehetővé CRC vagy MD5 ellenőrzőösszeg segítségével! A feladat során három komponenst/programot kell elkészíteni:

1. Checksum szerver: (fájl azonosító, checksum hossz, checksum, lejárati idő (mp-ben)) négyesek tárolását és lekérdezését teszi lehetővé. A protokoll részletei a következő oldalon.
2. Netcopy kliens: egy parancssori argumentumban megadott fájlt átküld a szervernek. Az átvitel során/végén kiszámol egy md5 checksumot a fájlra, majd ezt feltölti fájl azonosítóval együtt a Checksum szerverre. A lejárati idő 60 mp. A fájl azonosító egy egész szám, amit szintén parancssori argumentumban kell megadni.
3. Netcopy szerver: Vár, hogy egy kliens csatlakozzon. Csatlakozás után fogadja az átvitt bájtokat és azokat elhelyezi a parancssori argumentumban megadott fájlba. A végén lekéri a Checksum szervertől a fájl azonosítóhoz tartozó md5 checksumot és ellenőrzi az átvitt fájl helyességét, melynek eredményét stdoutputra is kiírja. A fájl azonosító itt is parancssori argumentum kell legyen.

Beadás: **TMS-ben**

Checksum szerver - TCP

- Beszúr üzenet
 - Formátum: szöveges
 - Felépítése: BE|<fájl azon.>|<érvényesség másodpercben>|<checksum hossza bájtszámban>|<checksum bájtjai>
 - A „|” delimiter karakter
 - Példa: BE|1237671|60|12|abcdefabcdef
 - Ez esetben: a fájlazon: 1237671, 60mp az érvényességi idő, 12 bájt a checksum, abcdefabcdef maga a checksum
 - Válasz üzenet: OK
- Kivesz üzenet
 - Formátum: szöveges
 - Felépítése: KI|<fájl azon.>
 - A „|” delimiter karakter
 - Példa: KI|1237671
 - Azaz kérjük az 1237671 fájl azonosítóhoz tartozó checksum-ot
 - Válasz üzenet: <checksum hossza bájtszámban>|<checksum bájtjai>
Péda: 12|abcdefabcdef
 - Ha nincs checksum, akkor ezt küldi: 0|
- Futtatás
 - ./checksum_srv.py <ip> <port>
 - <ip> - pl. localhost a szerver címe bindolásnál
 - <port> - ezen a porton lesz elérhető
 - A szerver végtelen ciklusban fut és egyszerre több klienst is ki tud szolgálni. A kommunikáció TCP, csak a fenti üzeneteket kezeli.
 - Lejárat utáni checksumok törlődnek.

Netcopy kliens – TCP alapú

- Működés:
 - Csatlakozik a szerverhez, aminek a címét portját parancssori argumentumban kapja meg.
 - Fájl bájtjainak sorfolytonos átvitele a szervernek.
 - A Checksum szerverrel az ott leírt módon kommunikál.
 - A fájl átvitele és a checksum elhelyezése után bontja a kapcsolatot és terminál.
- Futtatás:
 - `./netcopy_cli.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <fájl azon> <fájlnév elérési úttal>`
 - <fájl azon>: egész szám
 - <srv_ip> <srv_port>: a netcopy szerver elérhetősége
 - <chsum_srv_ip> <chsum_srv_port>: a Checksum szerver elérhetősége

Netcopy szerver – TCP alapú

- Működés:
 - Bindolja a socketet a parancssori argumentumban megadott címre.
 - Vár egy kliensre.
 - Ha acceptálta, akkor fogadja a fájl bájtjait sorfolytonosan és kiírja a parancssori argumentumban megadott fájlba.
 - Fájlvége jel olvasása esetén lezárja a kapcsolatot és utána ellenőrzi a fájlt a Checksum szerverrel.
 - A Checksum szerverrel az ott leírt módon kommunikál.
 - Hiba esetén a stdout-ra ki kell írni: CSUM CORRUPTED
 - Helyes átvitel esetén az stdout-ra ki kell írni: CSUM OK
 - Fájl fogadása és ellenőrzése után terminál a program.
- Futtatás:
 - `./netcopy_srv.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <fájl azon> <fájlnév elérési úttal>`
 - <fájl azon>: egész szám ua. mint a kliensnél – ez alapján kéri le a szervertől a checksumot
 - <srv_ip> <srv_port>: a netcopy szerver elérhetősége – bindolásnál kell
 - <chsum_srv_ip> <chsum_srv_port>: a Checksum szerver elérhetősége
 - <fájlnév> : ide írja a kapott bájtokat

PYTHON SOCKET - UDP

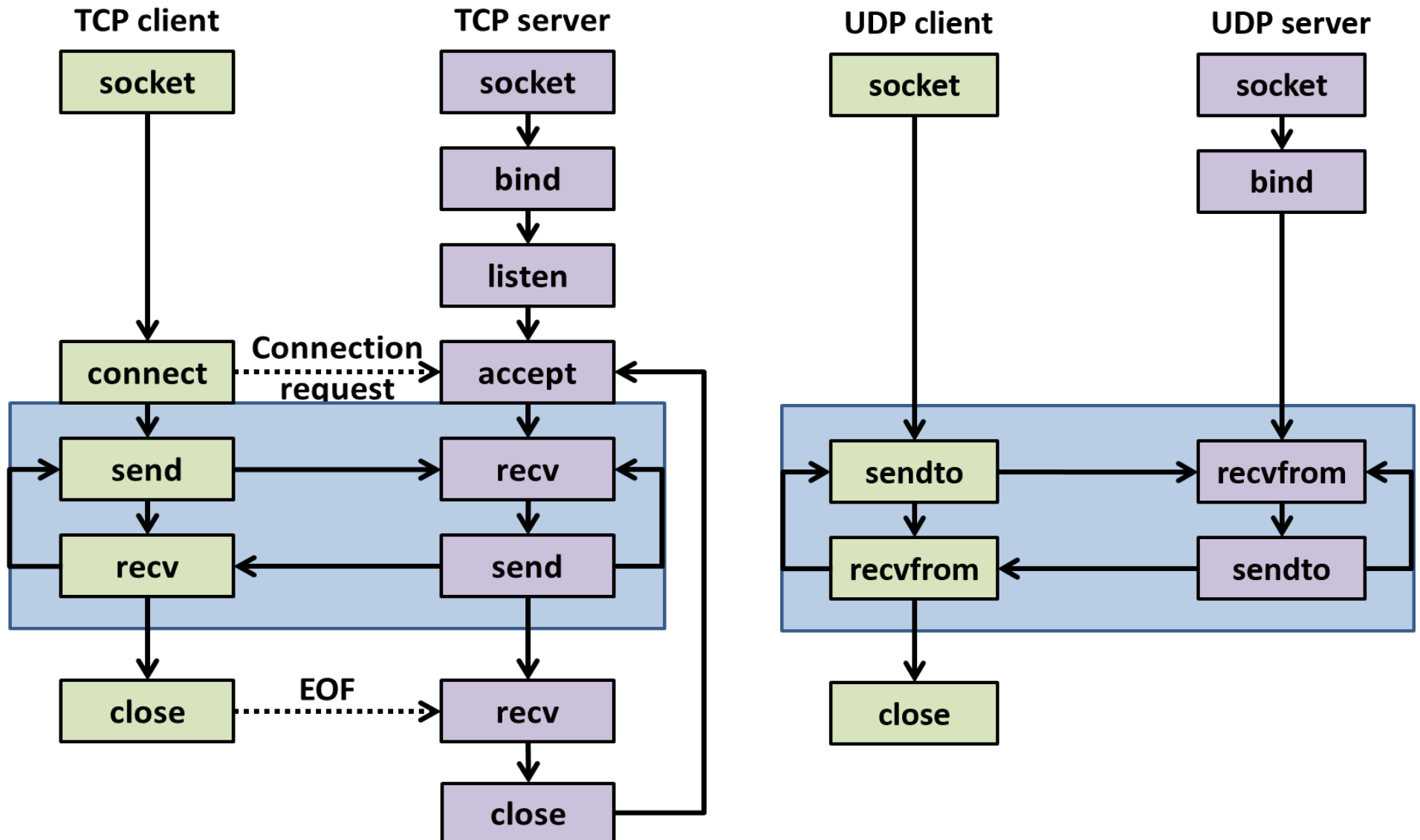
A kommunikációs csatorna kétféle típusa

- **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

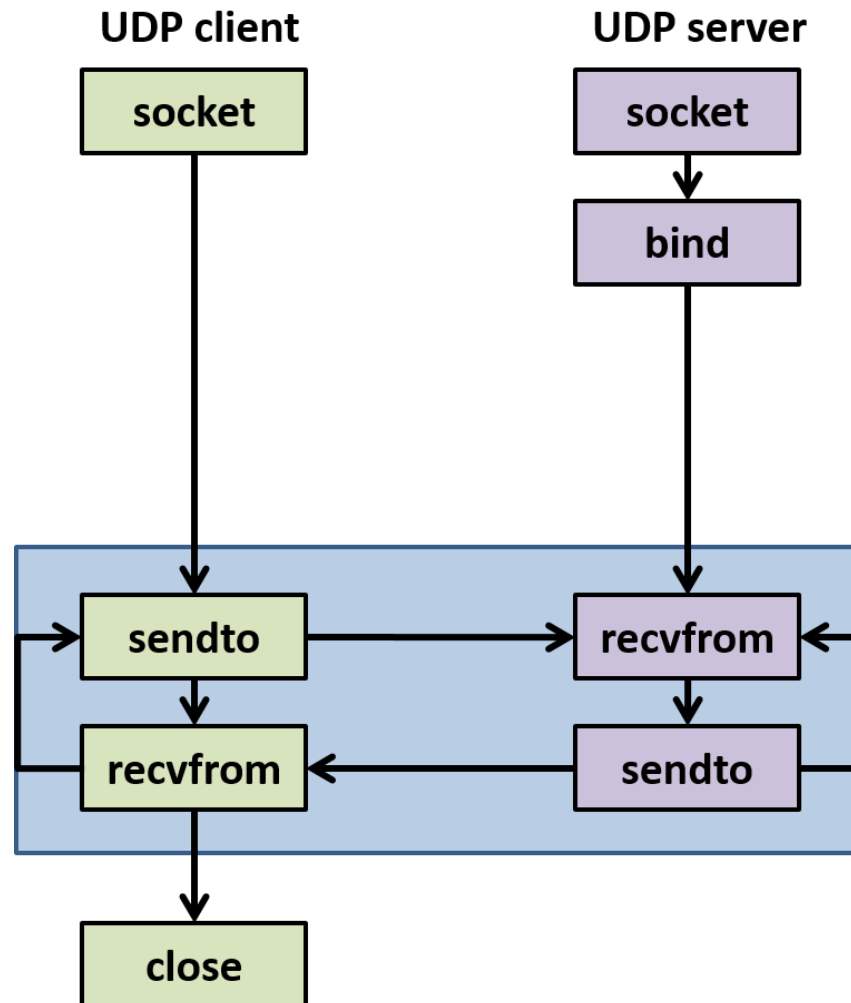
TCP

VS

UDP

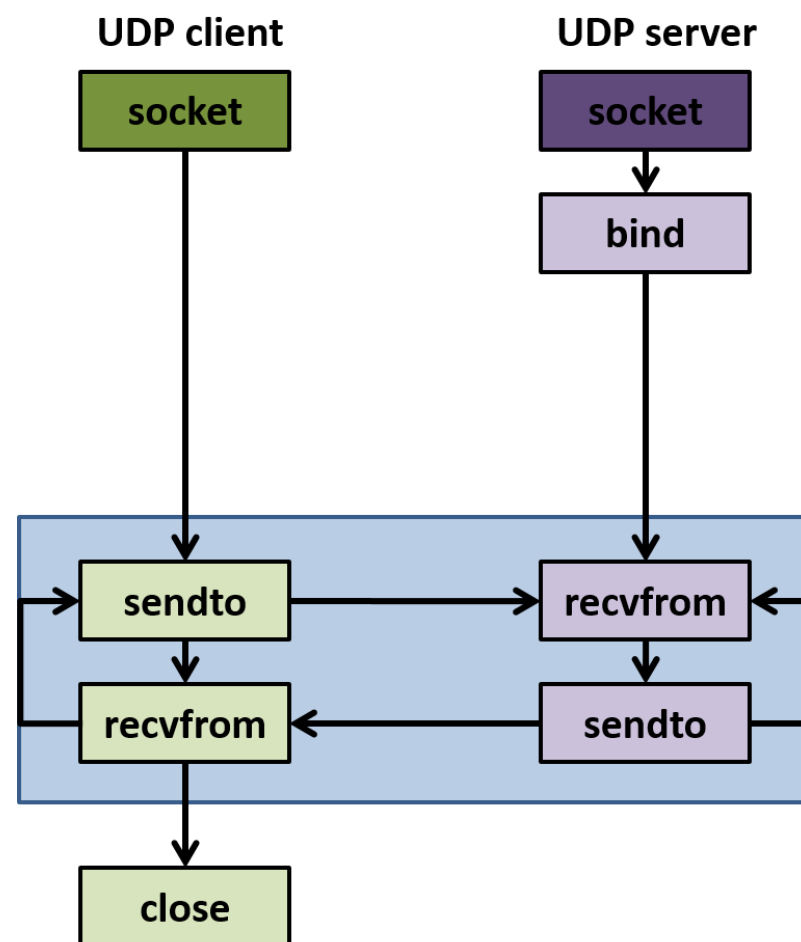


UDP



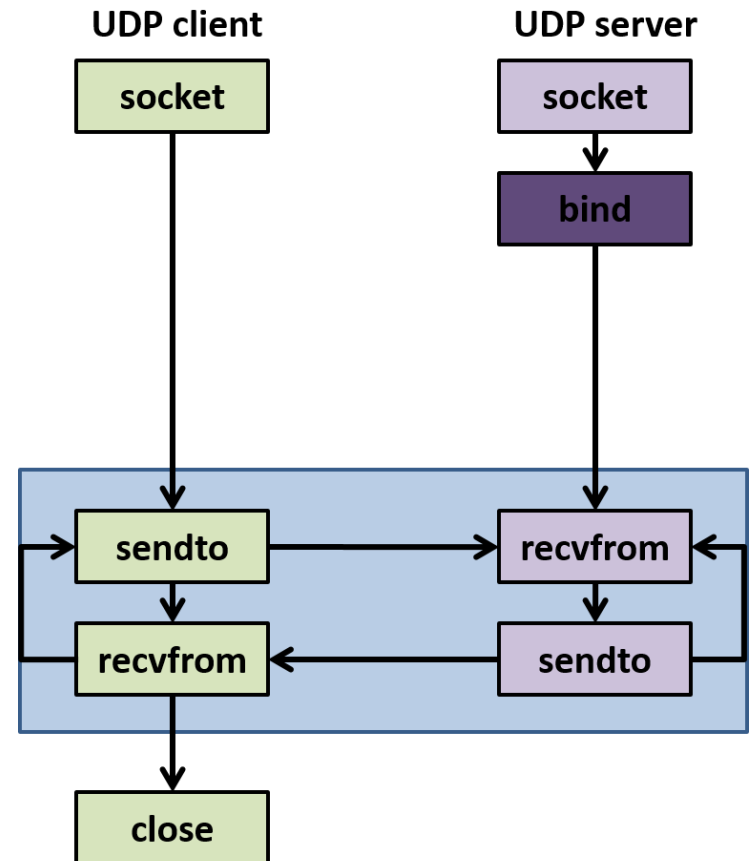
Socket leíró beállítása

- `socket.socket([family, type, proto])`
- *family* : `socket.AF_INET` → IPv4
(`AF_INET6` → IPv6)
- ***type* : `socket.SOCK_DGRAM` → UDP**
- *proto* : 0
(alapértelmezett protokoll lesz)
- visszatérési érték: egy socket objektum, amelynek a metódusai a különböző socket rendszer hívásokat implementálják



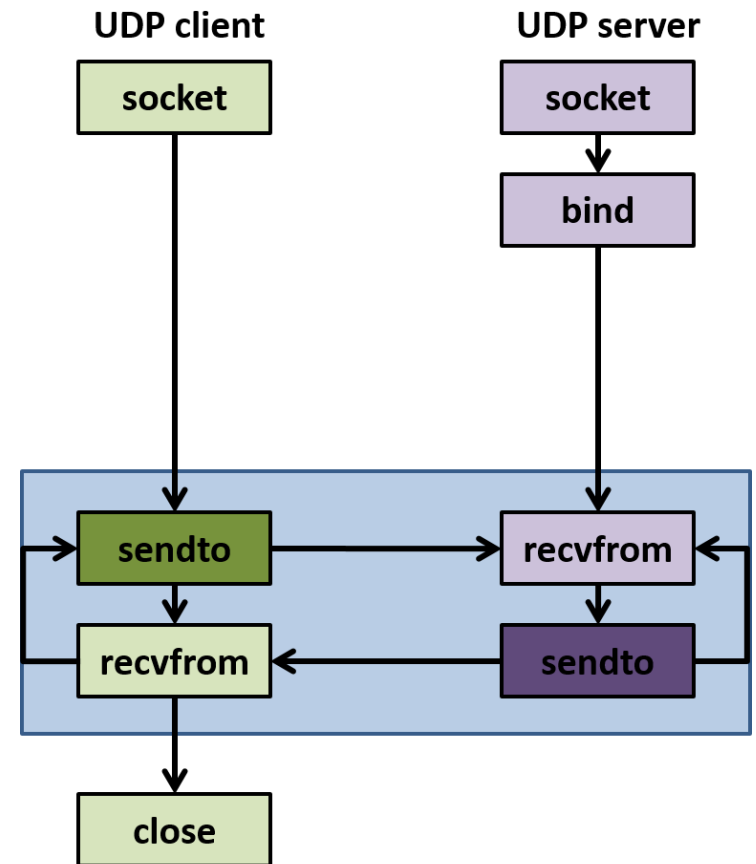
Bindolás – ismétlés

- `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



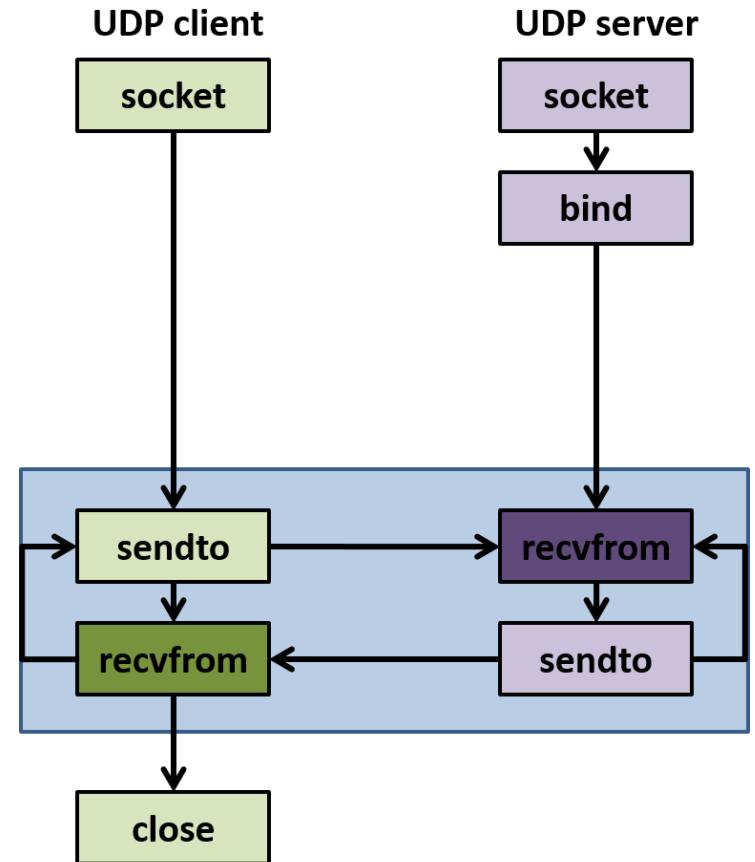
sendto

- `socket.socket.sendto(string, address)`
- `socket.socket.sendto(string, flags, address)`
- A socket objektum metódusai
- Adatküldés (*string*) a socketnek
- *flags* : 0 (nincs flag meghatározva)
- **A socketnek előtte nem kell csatlakozni a távoli sockethez, mivel azt az *address* meghatározza**



recvfrom

- `socket.socket.recvfrom(bufsize
[, flags])`
- A socket objektum metódusa
- Üzenet fogadása
- *bufsize* : a max. adatmennyiség, amelyet egyszerre fogadni fog
- *flags* : 0 (nincs flag meghatározva)
- **visszatérési érték: egy (*string*, *address*) tuple, ahol a fogadott adat sztring reprezentációja és az adatküldő socket címe szerepel**



UDP

- `socket`

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

- `recvfrom()`

```
data, address = sock.recvfrom(4096)
```

- `sendto()`

```
sent = sock.sendto(data, address)
```

Feladat: Hello UDP felett

Készítsünk egy kliens-szerver alkalmazást, amely UDP protokollt használ. A kliens küldje a „Hello Server” üzenetet a szervernek, amely válaszolja a „Hello Kliens” üzenetet.

Feladat - Számológép UDP felett

Készítsünk egy szerver-kliens alkalmazást, ahol a kliens elküld 2 számot és egy operátort a szervernek, amely kiszámolja és visszaküldi az eredményt. A kliens üzenete legyen struktúra. Használjunk UDP protokollt!

Szorgalmi feladat otthonra – fájltvitel UDP felett

Fájltvitel megvalósítása úgy, hogy a fájl letöltése UDP felett legyen megoldva. Készüljünk fel arra, hogy az átvitel során csomagvesztés, vagy sorrend csere is történhet! Az UDP szerver portját szabadon definiálhatjuk!

A hibakezeléshez egy javaslat:

Max. 1000 bájtanként UDP csomagokban elkezdjük átküldeni a fájl tartalmát. Minden csomag egy pár bájtos fejléccel indul, amiben jelezzük, hogy az utolsó darab-e, amit átküldtünk, továbbá egy másik mező jelzi a byteoffset-et a fájl elejétől. Működés:

- Ha a kliens kapott egy adatcsomagot, akkor egy nyugtacsomagot küld vissza.
- A nyugtacsomag fogadása után a szerver, küldi a következő adatcsomagot.
- Ha nem jön nyugta, akkor T idő után újraküldi a korábbi adatcsomagot. (pl. $T=200\text{ms}$)
- Ha nyugta veszik el, akkor a vevő az offset alapján el tudja dönteni, hogy egy új adatcsomag, vagy egy korábbi duplikátuma érkezett-e.
- Ha az utolsó csomag is megérkezett, akkor a kliens nyugtázza azt is és lezárja a fájlba írást. A szerver az utolsó nyugta után befejezi az átvitelt.

VÉGE
KÖSZÖNÖM A FIGYELMET!