Számítógépes Hálózatok

5. gyakorlat

BEADANDÓ III. (1 PONT) ISMÉTLÉS

Beadandó – Barkóba

- Készítsünk egy barkóba alkalmazást. A szerver legyen képes kiszolgálni több klienst. A szerver válasszon egy egész számot 1..100 között véletlenszerűen. A kliensek próbálják kitalálni a számot.
- A kliens üzenete egy összehasonlító operátor: <, >, = és egy egész szám, melyek jelentése: kisebb-e, nagyobb-e, mint az egész szám, illetve rákérdez a számra. A kérdésekre a szerver Igen/Nem/Nyertél/Kiestél/ Vége üzenetekkel tud válaszolni. A Nyertél és Kiestél válaszok csak a rákérdezés (=) esetén lehetségesek.
- Ha egy kliens kitalálta a számot, akkor a szerver minden újabb kliens üzenetre az „Vége” üzenetet küldi, amire a kliensek kilépnek. A szerver addig nem választ új számot, amíg minden kliens ki nem lépett.
- Nyertél, Kiestél és Vége üzenet fogadása esetén a kliens bontja a kapcsolatot és terminál. Igen/Nem esetén folytatja a kérdezgetést.
- A kommunikációhoz TCP-t használjunk!
- Folytatás a következő oldalon!

Beadandó – Barkóba

- A kliens logaritmikus keresés segítségével találja ki a gondolt számot. A kliens tudja, hogy milyen intervallumból választott a szerver.
- AZAZ a kliens NE a standard inputról dolgozzon.
- Minden kérdés küldése előtt véletlenszerűen várjon 1-5 mp-et. Ezzel több kliens tesztelése is lehetséges lesz.
- Formai követelmények a következő oldalon!

Beadandó – Barkóba

- Üzenet formátum:
  - Klienstől: bináris formában **egy db karakter, 32 bites egész szám**
    A karakter lehet: <: kisebb-e, >: nagyobb-e, =: egyenlő-e
  - Szervertől: ugyanaz a bináris formátum, de a számnak nincs szerepe (bármi lehet)
    A karakter lehet: I: Igen, N: Nem, K: Kiestél, Y: Nyertél, V: Vége
- Fájlnevek és parancssori argumentumok:
- Szerver: **server.py** <bind_address> <bind_port> # A bindolás során használt pár
- Kliens: **client.py** <server_address> <server_port> # A szerver elérhetősége

- Beadási határidő: **TMS-ben (4 hét múlva)**

CRC, MD5

CRC

- Definiáljuk a G(x) generátor polinomot (G foka r), amelyet a küldő és a vevő egyaránt ismer.
- **Algoritmus:**

1. Legyen G(x) foka r. Fűzzünk r darab 0 bitet a keret alacsony helyi értékű végéhez, így az m+r bitet fog tartalmazni és az $x^rM(x)$ polinomot fogja reprezentálni.

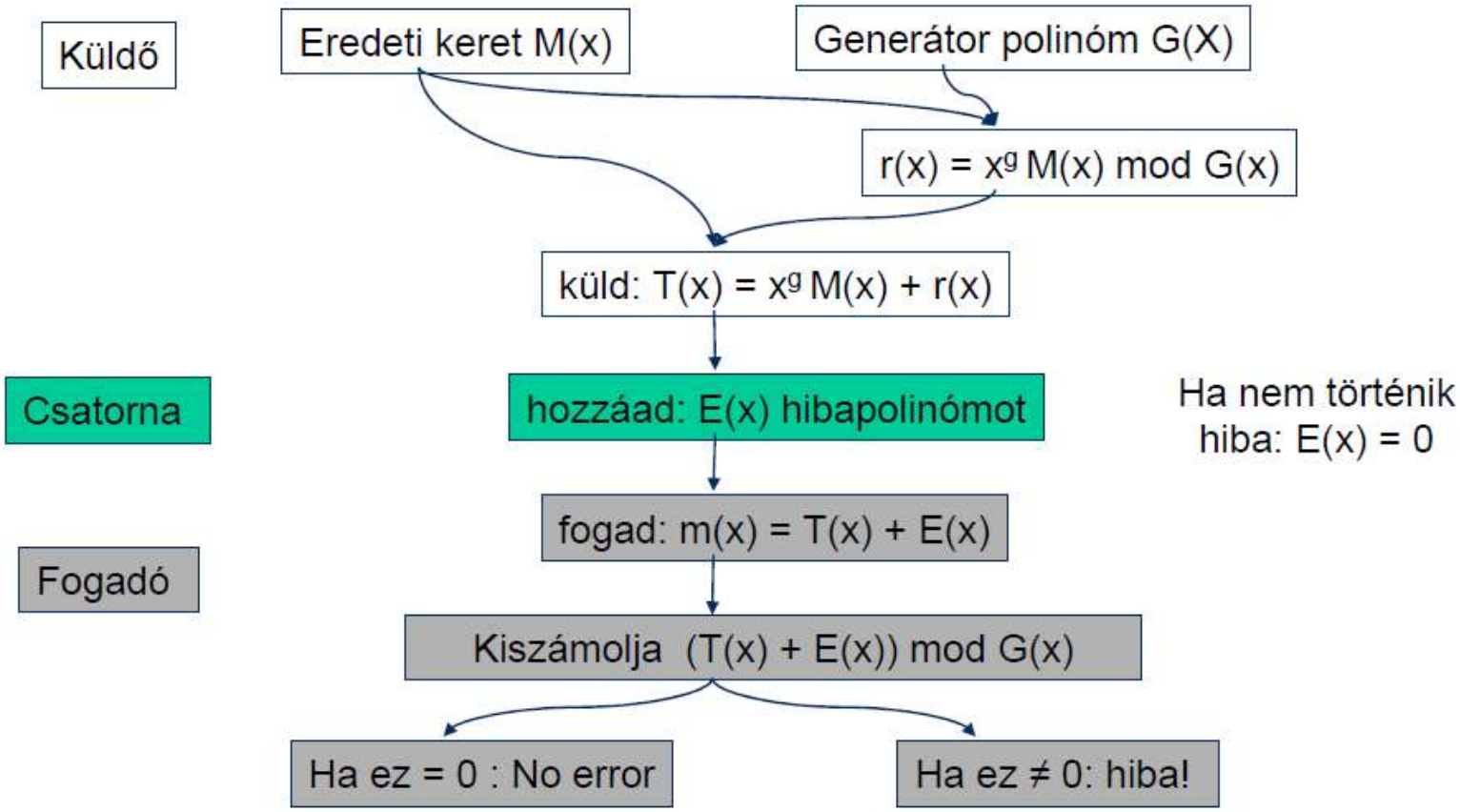
2. Osszuk el az $x^rM(x)$-hez tartozó bitsorozatot a G(x)-hez tartozó bitsorozattal modulo 2

3. Vonjuk ki a maradékot (mely mindig r vagy kevesebb bitet tartalmaz) az $x^rM(x)$-hez tartozó bitsorozatból. Az eredmény az ellenőrző összeggel ellátott, továbbítandó keret. Jelölje a továbbítandó keretnek megfelelő a polinomot T(x).

4. A vevő a T(x) + E(x) polinomnak megfelelő sorozatot kapja, ahol E(x) a hiba polinom. Ezt elosztja G(x) generátor polinommal.

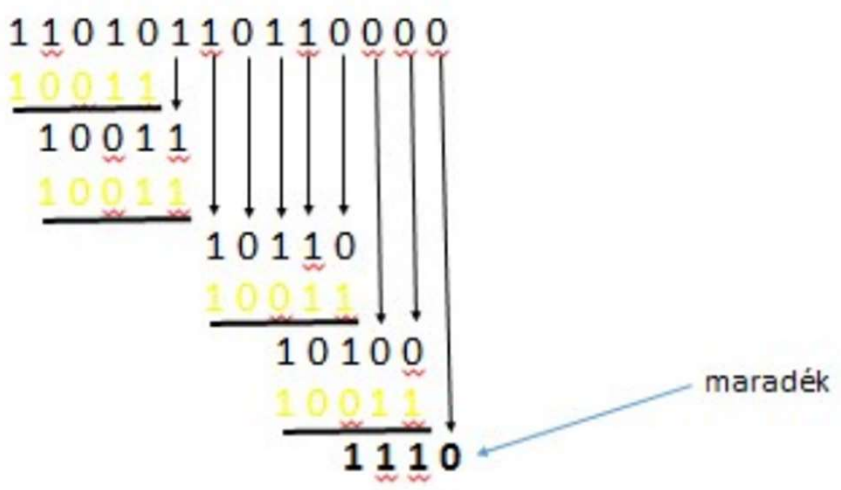
- Ha az osztási maradék, amit R(x) jelöl, nem nulla, akkor hiba történt

CRC


Küldő
Eredeti keret M(x)
Generátor polinóm G(X)
r(x) = x^g M(x) mod G(x)
küld: T(x) = x^g M(x) + r(x)
Csatorna
hozzáad: E(x) hibapolinómot
Ha nem történik hiba: E(x) = 0
fogad: m(x) = T(x) + E(x)
Fogadó
Kiszámolja (T(x) + E(x)) mod G(x)
Ha ez = 0 : No error
Ha ez ≠ 0: hiba!

CRC példa

- Keret: 1101011011
- Generátor: 10011
- A továbbítandó üzenet: 11010110111110
- Osztás binárisan: xor-ozgatunk



CRC példa

- Az osztásban 11010110110000 az $x^{13}+x^{12}+x^{10}+x^8+x^7+x^5+x^4$ polinomot reprezentálja. 10011 pedig az x^4+x+1 polinomot.
- Ebből (11010110110000) * (1110) = 11010110111110 ami az elküldendő keretünk lesz.
- (11010110111110) mod 10011 = 0
- Amennyiben hozzáadtunk volna egy E(x) hibapolinomot (pl. $E(x) = x^2+x$ =110), akkor a maradék nem nulla (a példában 11) lenne, így tudnánk, hogy meghibásodott a keret.

CRC, MD5 pythonban

- CRC

```
import binascii, zlib

test_string = "Fekete retek rettenetes".encode('utf-8')

print(hex(binascii.crc32(bytearray(test_string))))
print(hex(zlib.crc32(test_string)))
```

- MD5

```
import hashlib

test_string = "Fekete retek rettenetes".encode('utf-8')

m = hashlib.md5()
m.update(test_string)
print(m.hexdigest())
```

FÁJLÁTVITEL (SZÖVEG ÉS KÉP)

Fájl átvitel

- fájl bináris megnyitása

```
with open („input.txt”, „rb”) as f:
        …
```

- read(x) – x bytes

```
        …
        f.read(128)    #128 byte-ot fog beolvasni
```

„When size is omitted or negative, the entire contents of the file will be read and returned; it’s your problem if the file is twice as large as your machine’s memory. „ - python.org

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, lejárat (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 paracssori 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 - PROXY

Feladat

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 klienstől jövő kérést továbbítja a szerver felé és a szervertől jövő válaszokat pedig a kliens felé.

Pl: netProxy.py ggombos.web.elte.hu 80

Web browserbe írjuk be: localhost:10000

Feladat – Tiltsunk le valamilyen tartalmat

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

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

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

Számológép Proxy

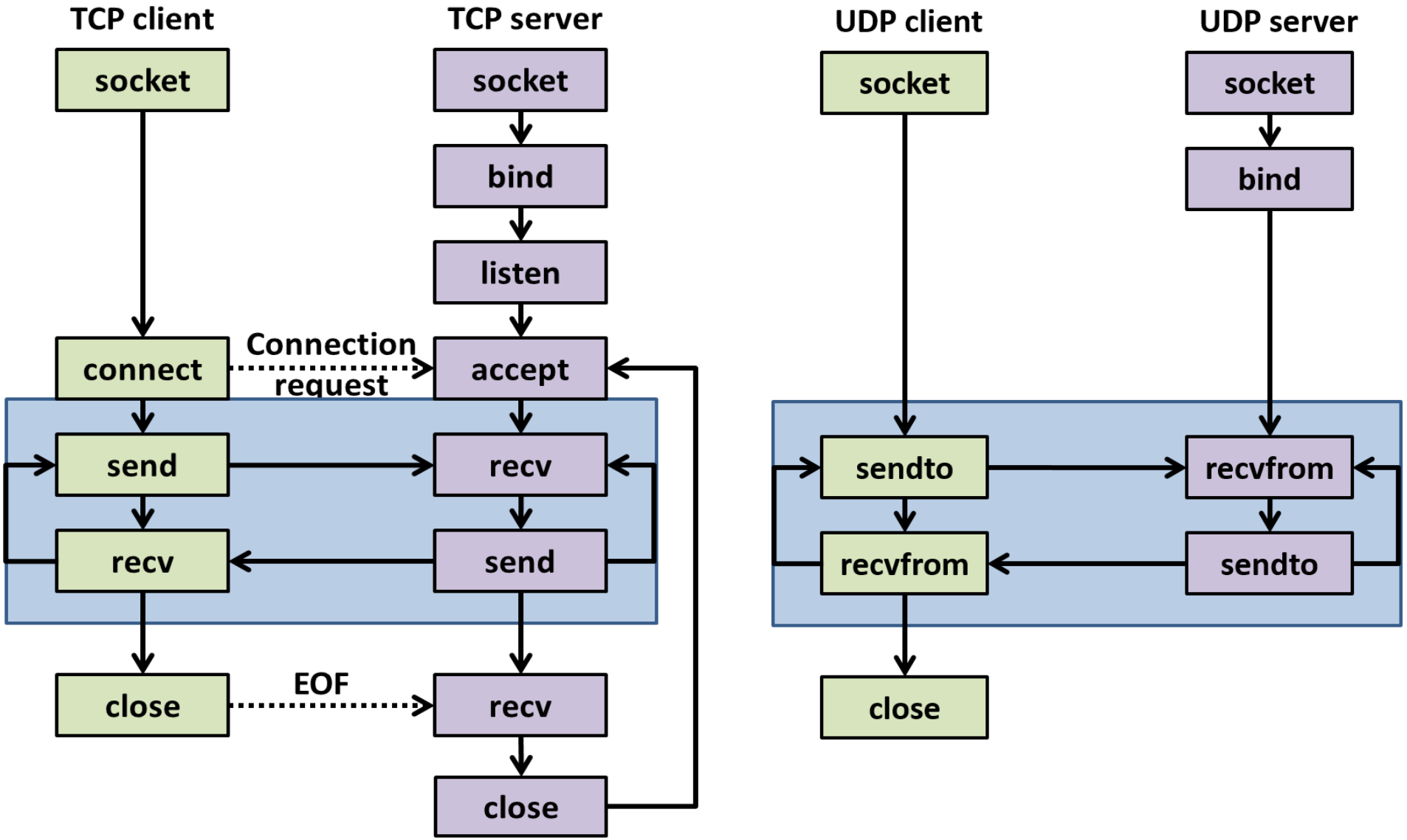
- Készítsünk a számológéphez egy proxy-t, ami értelmezi a klienstől kapott TCP kéréseket és "szabotálja" azokat.
- Szabotálás: az első operandust megszorozza kettővel, a másodikhoz pedig hozzáad egyet.
- A módosított kérést elküldi a TCP servernek, majd az eredményt visszaküldi a kliensnek.

PYTHON SOCKET - UDP

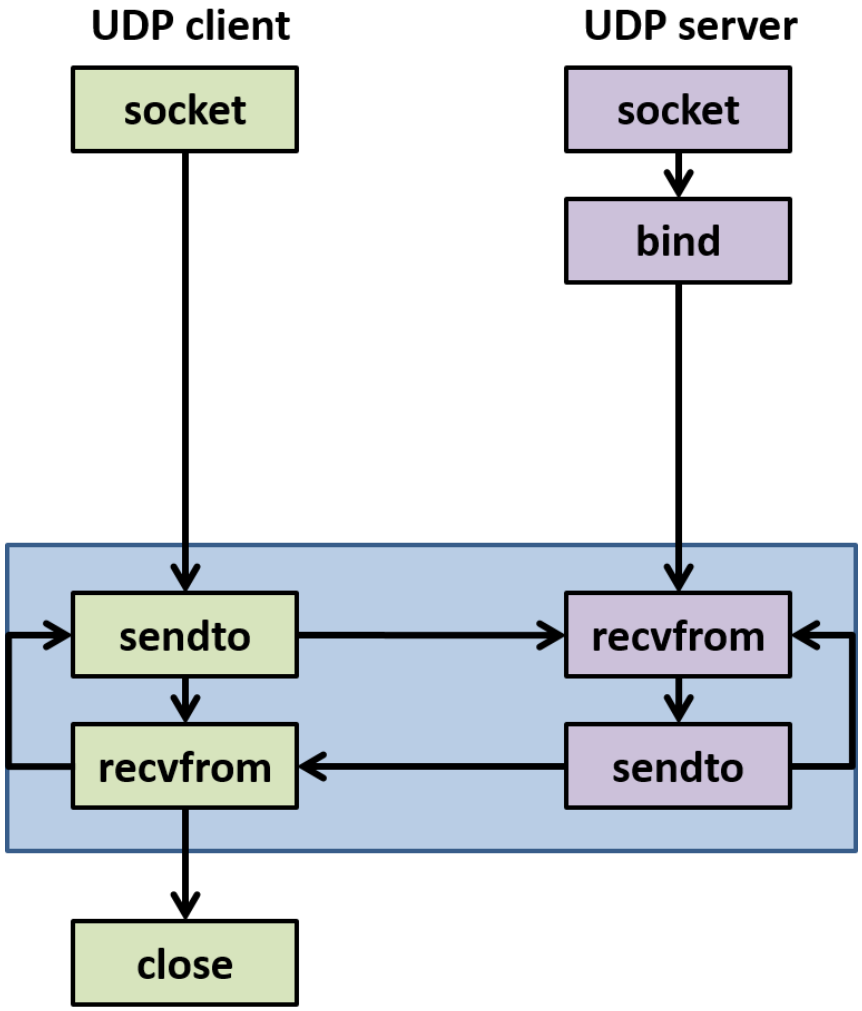
A kommunikációs csatorna kétféle típusa

- 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


TCP client
TCP server
socket
socket
bind
listen
connect
Connection request
accept
send
recv
recv
send
close
EOF
recv
close
UDP client
UDP server
socket
socket
bind
sendto
recvfrom
recvfrom
sendto
close

UDP

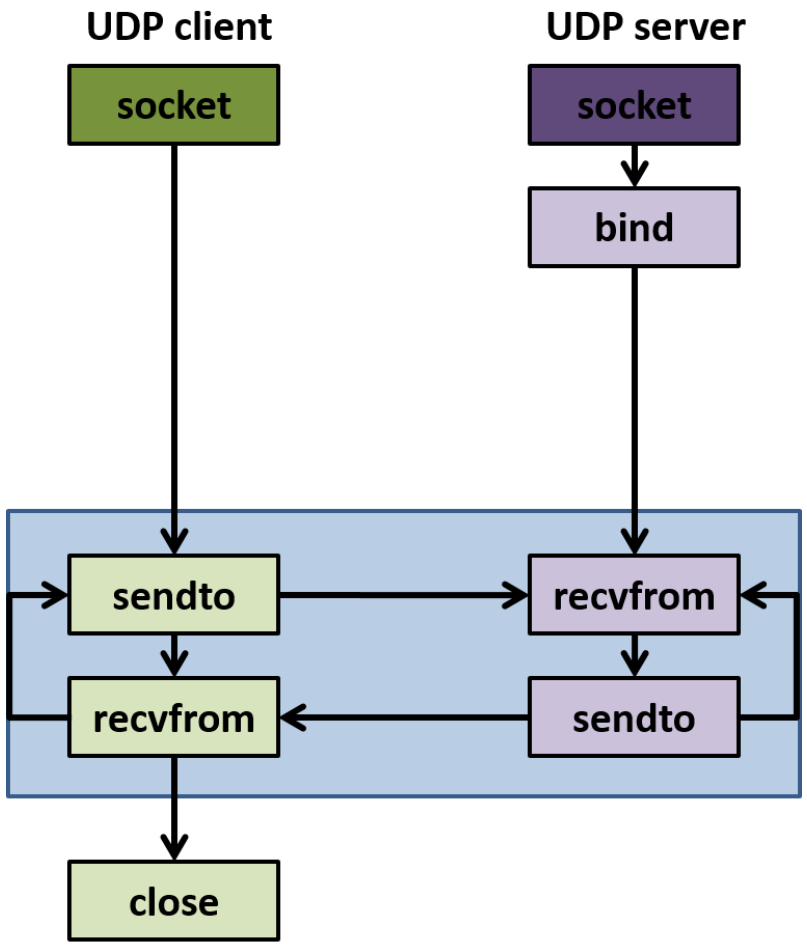

UDP client
UDP server
socket
socket
bind
sendto
recvfrom
recvfrom
sendto
close

Socket leíró beállítása

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

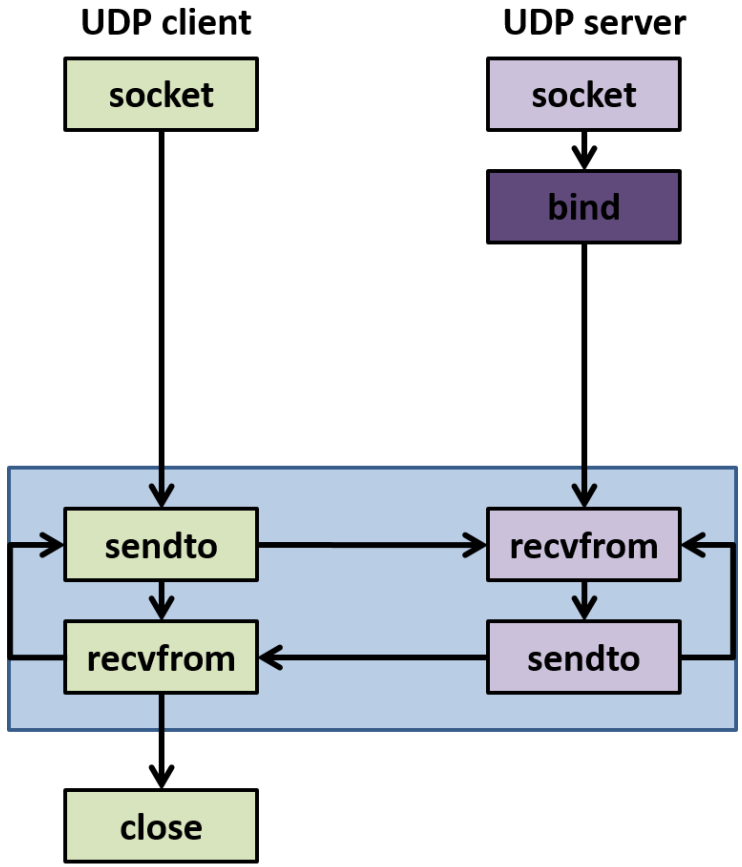
(alapértelmezett protokoll lesz)

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



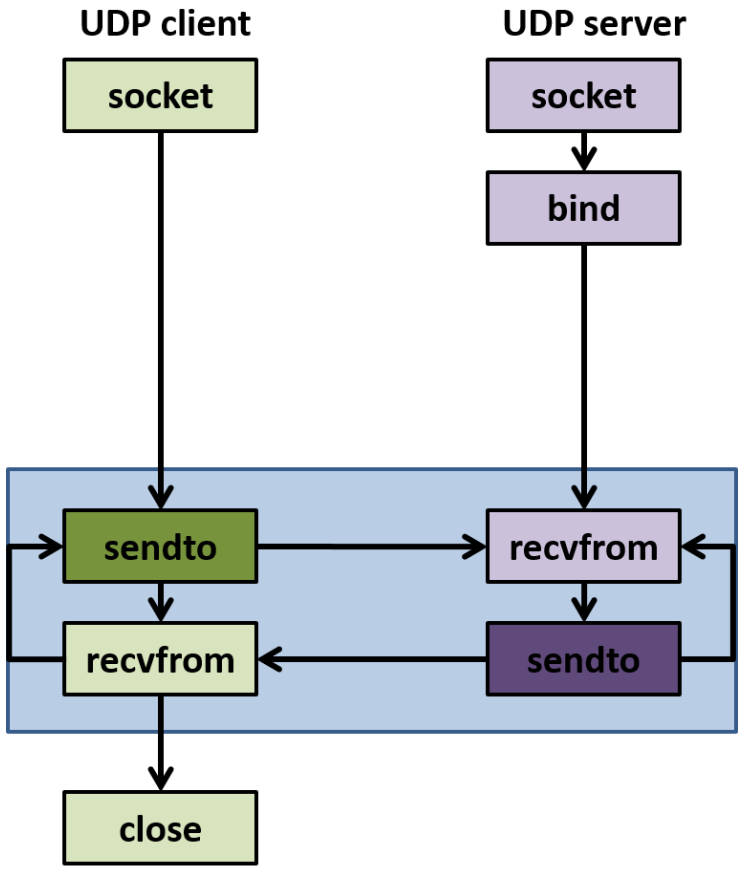
Bindolás – ismétlés

- 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



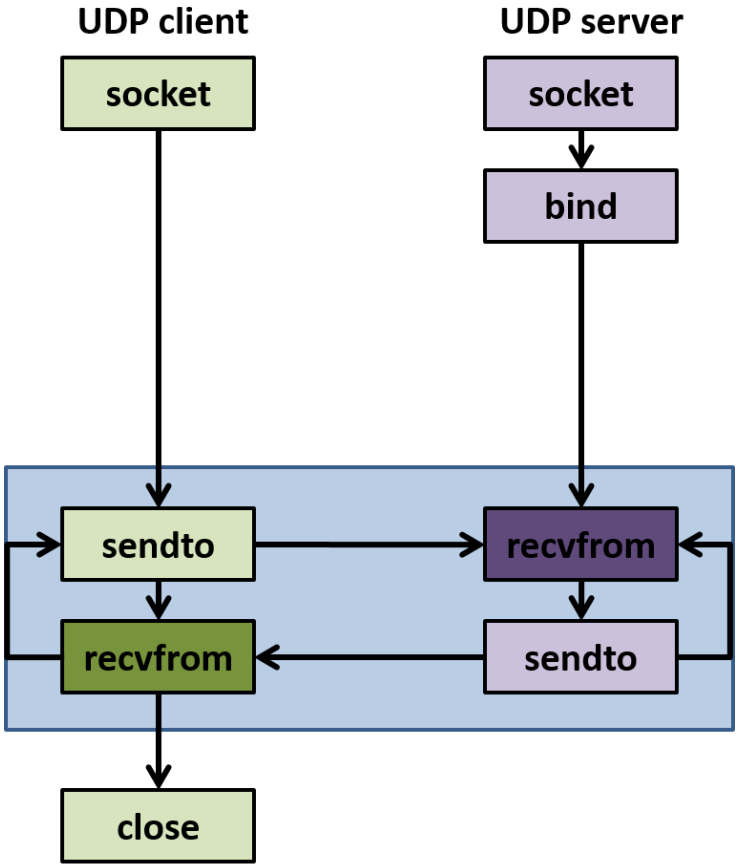
sendto

- 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 – fájlátvitel UDP felett

Fájlátvitel 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 bájtonként 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=200ms)
- 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!