Telekommunikációs Hálózatok

5. gyakorlat

ZH időpontok!!!

- **Okt 28.** zárthelyi (részletekről később)
- **Nov. 4.** ha szükséges, pótzárthelyi, egyébként továbbhaladás (részletekről később)

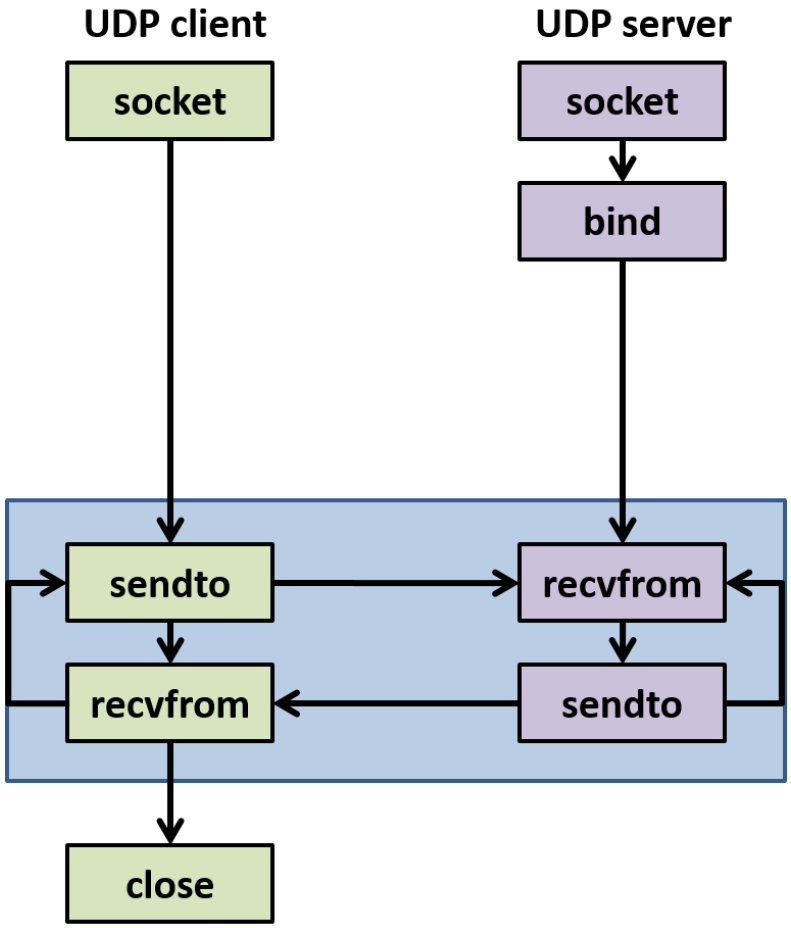
PYTHON SOCKET - UDP

Ismétlés

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**

UDP

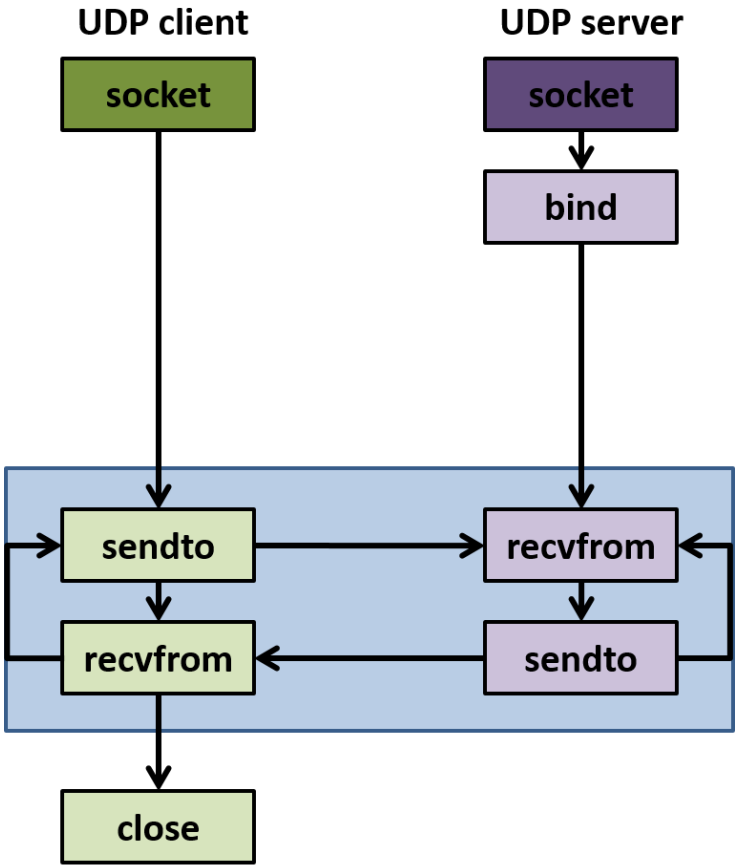

UDP client
UDP server
socket
socket
bind
sendto
recvfrom
recvfrom
sendto
close

Socket leíró beállítása

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

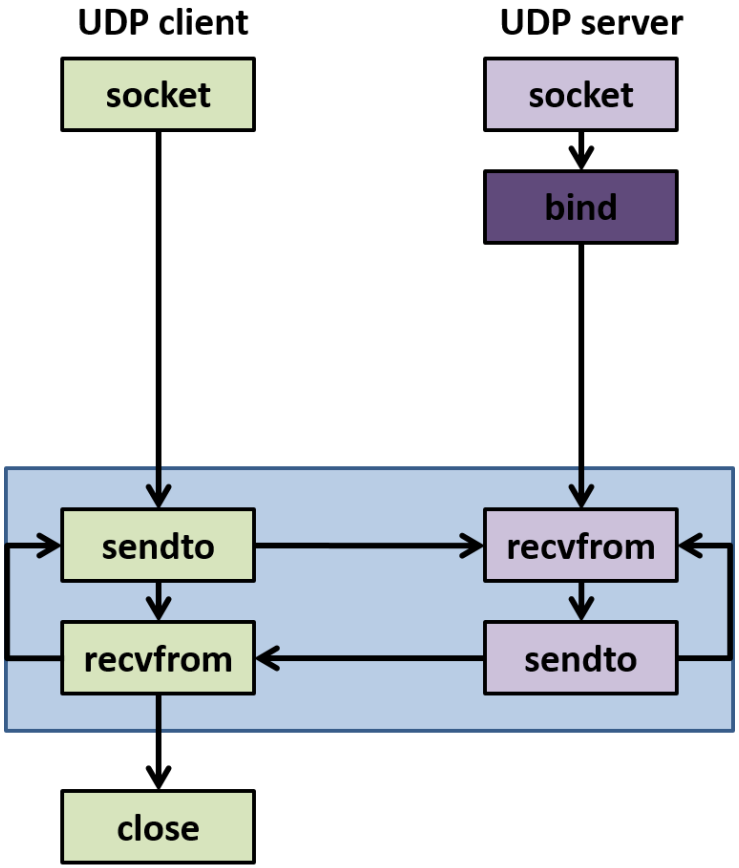
(alapértelmezett protokoll lesz)

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



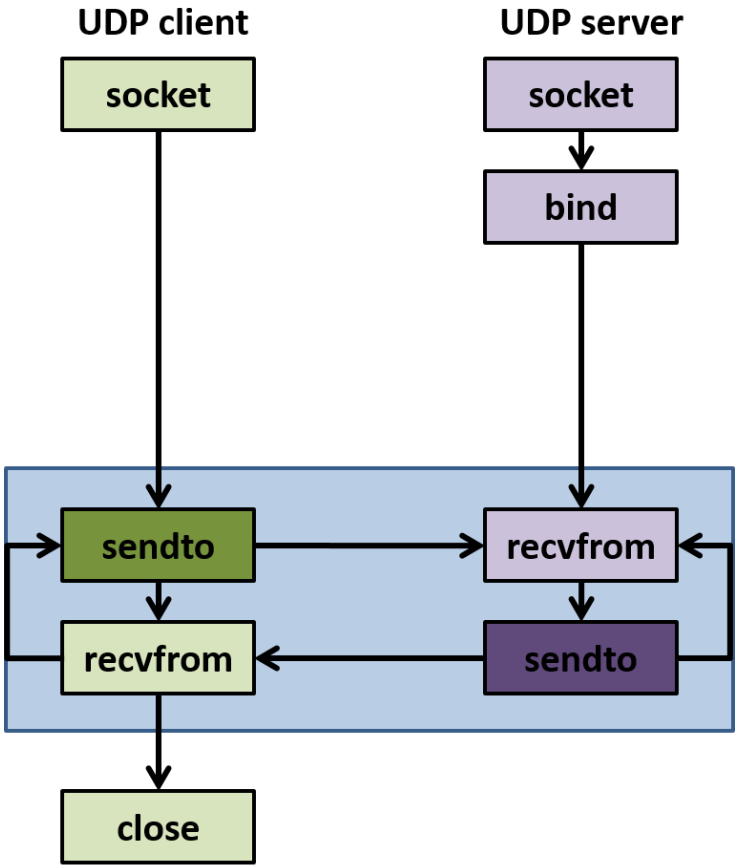
Bindolás – ismétlés

- 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



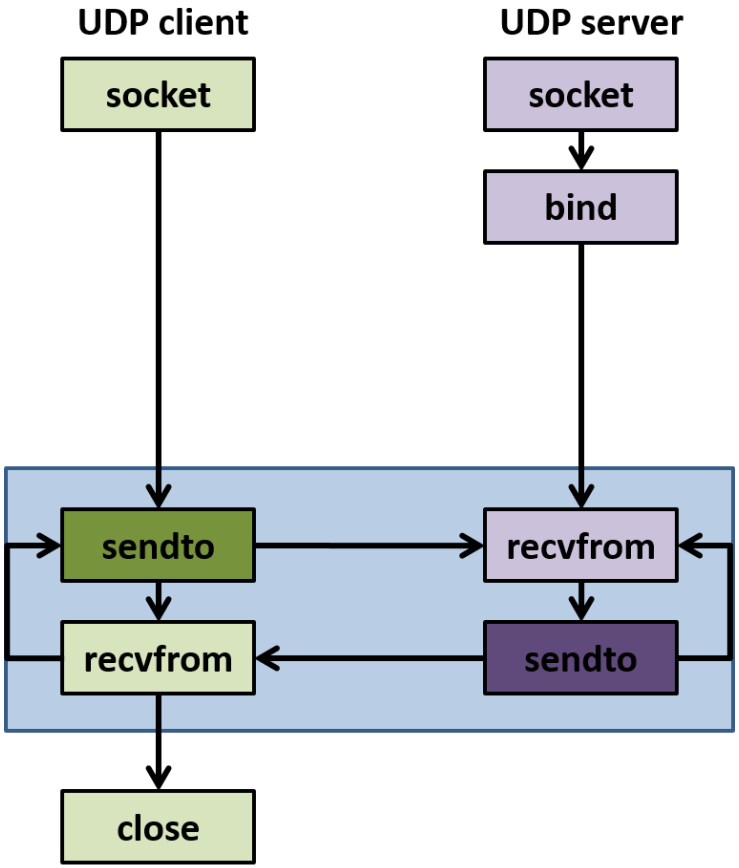
sendto

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



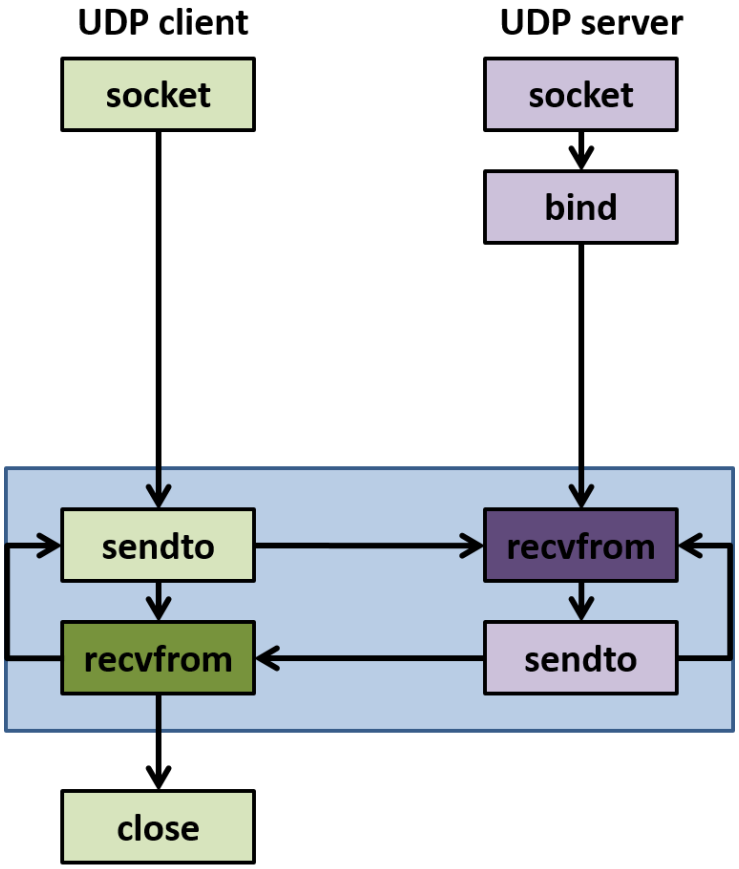
sendto

- **Fontos, hogy egy UDP üzenetnek bele kell férni egy egyszerű csomagba (ez IPv4 esetén kb. 65 KB-ot jelent)**
- 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



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 (*bytes*, *address*) tuple, ahol a fogadott adat bytes 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 1

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

Nézzük meg a megoldást!

Alhálózati maszk

- Az alhálózat egy logikai felosztása egy IP hálózatnak. Az IP cím ezért két részből áll: hálózatszámból és hoszt azonosítóból.
- A szétválasztás a 32 bites alhálózati maszk segítségével történik, amellyel bitenkénti ÉS-t alkalmazva az IP címre megkapjuk a hálózat-, komplementerével pedig a hoszt azonosítót.
- Ez arra jó, hogy meg tudjuk határozni, hogy a címzett állomás a helyi alhálózaton van-e, vagy sem.
- Az utóbbi esetben az alapértelmezett router felé továbbítják a csomagot (default gateway).

Alhálózati maszk

- CIDR jelölés: kompakt reprezentációja egy IP címnek és a hozzátartozó hálózatszámnak
- → IP cím + ’/’ + decimális szám.
- Pl.: 135.46.57.14/24 esetben 135.46.57.14 az IP cím,
- 255.255.255.0 a hálózati maszk (24 db. 1-es bit az elejétől),
- így 135.46.57.0 a hálózat azonosító.

Alhálózati maszk – példa

	10000111	00101110	00111001	00001110	**135.46.57.14**
AND	11111111	11111111	11111111	00000000	**255.255.255.0**
	10000111	00101110	00111001	00000000	**135.46.57.0**

135.46.57.14/24 → 135.46.57.0

Számolós feladat 1

- Hány cím érhető el a következő alhálózati maszkokkal? Adjuk meg a minimális és maximális címet is!
- 188.100.22.12/32
- 188.100.22.12/20
- 188.100.22.12/10

Számolós feladat 1 megoldása

188.100.22.12/32

	10111100	01100100	00010110	00001100	**188.100.22.12**
AND	11111111	11111111	11111111	11111111	**255.255.255.255**
	10111100	01100100	00010110	00001100	**188.100.22.12**

egy darab a 188.100.22.12

Számolós feladat 1 megoldása

188.100.22.12/20

	10111100	01100100	00010110	00001100	**188.100.22.12**
AND	11111111	11111111	11110000	00000000	**255.255.240.0**
	10111100	01100100	00010000	00000000	**188.100.16.0**

$2^{32-20} = 2^{12} = 4096$ darab lenne, de valójában ebből még kettőt le kell vonni, mert speciális jelentéssel bírnak:

- csupa 0: az alhálózat hálózati címe (magára az alhálózatra vonatkozik)
- csupa 1-es: broadcast a helyi hálózaton

Min.	10111100	01100100	00010000	00000001	**188.100.16.1**
Max.	10111100	01100100	00011111	11111110	**188.100.31.254**

Számolós feladat 1 megoldása

188.100.22.12/10

	10111100	01100100	00010110	00001100	**188.100.22.12**
AND	11111111	11000000	00000000	00000000	**255.192.0.0**
	10111100	01000000	00000000	00000000	**188.64.0.0**

$2^{32-10} - 2 = 2^{22} - 2 = 4194302$ darab

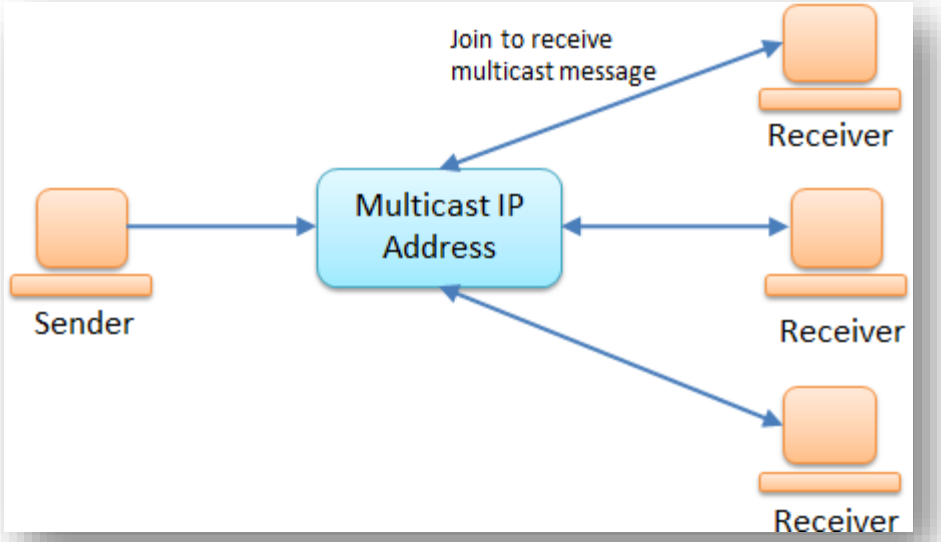
Min.	10111100	01000000	00000000	00000001	**188.64.0.1**
Max.	10111100	01111111	11111111	11111110	**188.127.255.254**

Socket beállítása – emlékeztető

- socket.**setsockopt**(*level*, *optname*, *value*): az adott socket opciót állítja be
- Általunk használt *level* értékek az alábbiak lesznek:
  - socket.**IPPROTO_IP**: jelzi, hogy IP szintű beállítás
  - socket.**SOL_SOCKET**: jelzi, hogy socket API szintű beállítás
- Az *optname* a beállítandó paraméter neve, pl.:
  - socket.**SO_REUSEADDR**: a kapcsolat bontása után a port újrahasznosítása
- A *value* lehet sztring vagy egész szám:
  - Az előbbi esetén biztosítani kell a hívónak, hogy a megfelelő biteket tartalmazza (a struct segítségével)
  - A socket.**SO_REUSEADDR** esetén ha 0, akkor lesz hamis a „tulajdonság”, egyébként igaz

Multicast

- A pont-pont összeköttetés sokféle kommunikációs igényt ki tud szolgálni



- Ugyanazt az infót külön-külön elküldeni a társaknak nem optimális az erőforrás kihasználtság szempontjából
- A multicast egy időben több végpontnak is tudja szállítani az üzenetet → jobb hatékonyság

Multicast

- A multicast üzenetek küldésénél **UDP-t** használunk
  - (a TCP végpontok közötti kommunikációs csatornát igényel)
- Egy IPv4 címtartomány van lefoglalva a multicast forgalomra
  - (224.0.0.0-230.255.255.255)
- Ezeket a címeket speciálisan kezelik a routerek és switchek

Multicast üzenet küldése

- Ha a multicast üzenet küldője választ is vár, nem fogja tudni, hogy hány db. válasz lesz
- → időtúllépési értéket állítunk be, hogy elkerüljük a blokkolást a válaszra történő határozatlan idejű várakozás miatt:

```
sock.settimeout(0.2) # 0.2 sec.
```

Multicast üzenet küldése

- Továbbá életidő (Time To Live (TTL)) értéket is szükséges beállítani a csomagon:
  - A TTL kontrollálja, hogy hány db. hálózat kaphatja meg a csomagot
    - „Hop count”: a routerek csökkentik az értékét, ha 0 lesz → eldobják a csomagot
  - A **setsockopt** függvény segítségével majd a socket.**IP_MULTICAST_TTL**-t kell beállítani

Multicast üzenet fogadása

- A fogadó oldalon szükség van arra, hogy a socket-et hozzáadjuk a multicast csoporthoz:
  - A **setsockopt** segítségével az **IP_ADD_MEMBERSHIP** opciót kell beállítani
  - A socket.**inet_aton**(*ip_string*): az IPv4 cím sztring reprezentációjából készít 32-bitbe csomagolt bináris formátumot
  - Meg lehet adni azt is, hogy a fogadó milyen hálózati interfészen figyeljen, esetünkben most az összesen figyelni fog: socket.**INADDR_ANY**

Multicast üzenet fogadása

- socket.**INADDR_ANY** a bind hívásnál is lehet használni
  - Ott az '' (üres) sztring reprezentálja → a socket az összes lokális interfészhez kötve lesz
- Nem mindenhol tudunk kötni egy multicast címre
  - Nem minden platform támogatja, a Windows az egyik ilyen
  - Ilyenkor: „socket.error: [Errno 10049] The requested address is not valid in its context” hiba jön
  - Kénytelenek vagyunk ebben az esetben az **INADDR_ANY**-t használni, viszont az fontos, hogy a portnak **a szerver által használt portot adjuk meg**
  - (localhost-tal nem működne, mert akkor a multicast hálózatot nem tudjuk elérni)

Példa hívások multicast-nál

- setsockopt() (sender)

```
ttl = struct.pack('b', 1) # '\x01'
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)
```

- socket hozzávétele a multicast grouphoz (recv)

```
multicast_group = '224.3.29.71'
group = socket.inet_aton(multicast_group)                    # '\xe0\x03\x1dG'
mreq = struct.pack('4sL', group, socket.INADDR_ANY)          # '\xe0\x03\x1dG\x00\x00\x00\x00'
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
```

Feladat 2

- Készítsünk egy multicast fogadó és küldő alkalmazást!
- Csak a saját gépünkön fusson a küldő és a fogadó is (TTL értéke 1)
- Nézzük meg a megoldást!

Udp video streaming példa

A Fájlok / Gyak05 könyvtárból töltsük le az alábbi fájlokat:

videograbber.py, video.mp4

cv2 telepítés:

```
pip install opencv-python
```

Forrás: https://docs.opencv.org

Udp video streaming példa

```
cv2.namedWindow(winname)
```

- **winname** – Name of the window in the window caption that may be used as a window identifier.
- The function **namedWindow** creates a window that can be used as a placeholder for images and trackbars. Created windows are referred to by their names.

Forrás: https://docs.opencv.org

Udp video streaming példa

```
cv2.imdecode(buf, flags)
```

- **buf** – Input array or vector of bytes.
- **flags** – Flags specifying the color type of a loaded image (1: Return a 3-channel color image.)
- The function reads an image from the specified buffer in the memory.

Forrás: https://docs.opencv.org

Udp video streaming példa

```
cv2.imshow(winname, mat)
```

- **winname** – Name of the window.
- **mat** – Image to be shown.
- The function **imshow** displays an image in the specified window.

Forrás: https://docs.opencv.org

Udp video streaming példa

```
cv2.imencode(ext, img[, params])
```

- **ext** – File extension that defines the output format.
- **img** – Image to be written.
- **params** – Format-specific parameters.
- The function compresses the image and stores it in the memory buffer that is resized to fit the result.

Forrás: https://docs.opencv.org

Udp video streaming példa

```
cv2.VideoCapture(filename)
```

- **filename** – name of the opened video file
- VideoCapture constructor.

```
cv2.VideoCapture.read()
```

- This is the most convenient method for reading video files or capturing data from decode and return the just grabbed frame.

Feladat 3 - 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!

Feladat 4 - Képküldés UDP felett

Küldjünk át egy képet UDP segítségével.

- 200 byte-onként küldjünk
- Ha vége a filenak akkor küldjünk üres stringet
- Minden kapott üzenetre OK legyen a válasz

Feladat 5: Chat UDP-vel

- Készítsünk egy chat alkalmazást, amelynél egy chat szerveren keresztül tudnak a kliensek beszélni egymással!
- A kliensek először csak elküldik a nevüket a szervernek
- A szerver szerepe, hogy a kliensektől jövő üzenetet minden más kliensnek továbbítja névvel együtt: [<név>] <üzenet> ; pl. [Józsi] Kék az ég!
- A kliensek a szervertől jövő üzeneteket kiírják a képernyőre.

VÉGE
KÖSZÖNÖM A FIGYELMET!