Számítógépes Hálózatok

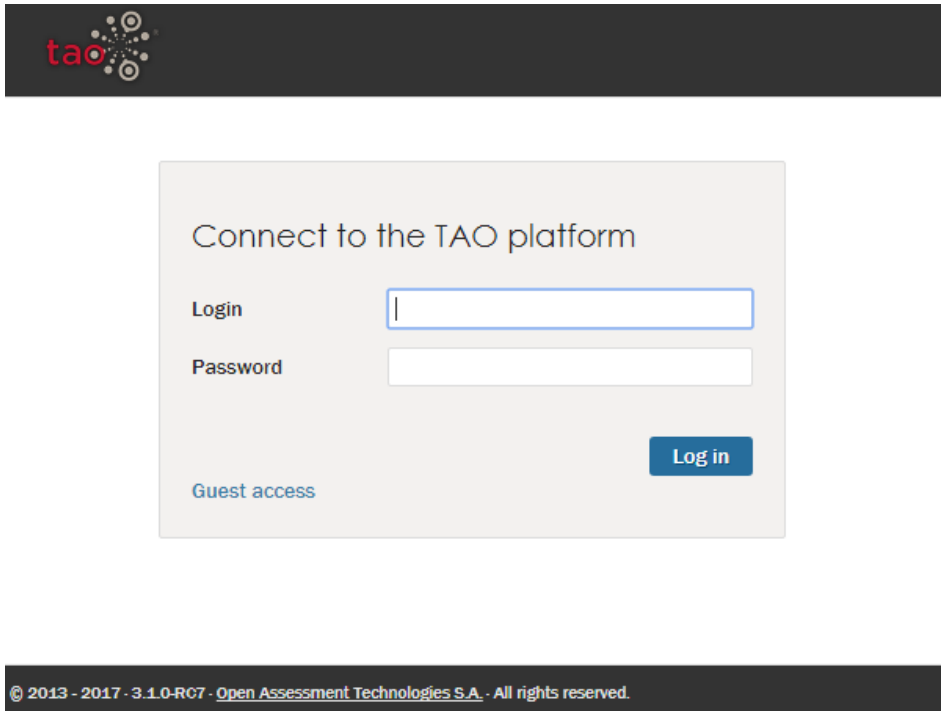
4. gyakorlat

Elérhetőségek

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

Óra eleji kisZH

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



Gyakorlat tematika

- SELECT, chat alkalmazás
- Órai / házi feladat (fizikai réteg)

Select

- Több socketet is szeretnénk egy időben figyelni (a bejövő kapcsolódásokra és a meglevő kapcsolatokból való olvasásra is)
- Probléma: accept és a recv függvények blokkolnak
- Hatékonyabb megoldást szeretnénk, mint a socket timeout használatával egy folyamatos lekérdező ciklus
- → A select fv. segítségével a monitorozás az op. rsz. hálózati rétegében történik

Select

- select.**select**(*rlist, wlist, xlist[, timeout]*)
- Az első három argumentum a „várakozó objektumok” listái:
  - *rlist*: a socketek halmaza, amelyek várakoznak, amíg készek nem lesznek az olvasásra
  - *wlist*: … készek nem lesznek az írásra
  - *xlist*: … egy „kivétel” nem jön
- Az opcionális *timeout* argumentum mp.-ben adja meg az időtúllépési értéket
  - (ha ez nincs megadva → addig blokkol, amíg az egyik socket kész nincs)

Select

- select.**select**(*rlist, wlist, xlist[, timeout]*)
- Visszatér három listával:
  1. visszadja a socketek halmazát, amelyek készek az olvasásra (adat jön)
  2. … készek az írásra (szabad hely van a pufferükben, és lehet írni oda)
  3. … amelyeknél egy „kivétel” jön

Select

- Az „olvasható” socketek három lehetséges esetet reprezentálhatnak:
  - Ha a socket a fő „szerver” socket, amelyiket a kapcsolatok figyelésére használunk → az „olvashatósági” feltétel azt jelenti: kész arra, hogy egy másik bejövő kapcsolatot elfogadjon
  - Ha a socket egy meglévő kapcsolat egy klienstől jövő adattal → az adat a recv() fv. segítségével kiolvasható
  - Ha az előző, de nincs adat → a kliens szétkapcsolt, a kapcsolatot le lehet zárni

Példa hívások select-nél

- setblocking() vagy settimeout()

```
connection.setblocking(0)      # or connection.settimeout(0.0)
connection.setblocking(1)      # or connection.settimeout(None)
```

- select()

```
inputs = [ server ]
outputs = [ ]
timeout=1
readable, writable, exceptional = select.select(inputs, outputs, inputs,timeout)
…
for s in readable:
      if s is server:    #new client connect
            ….
      else:
            ….          #handle client
```

Queue – szálbiztos FIFO konténer

- A Queue python modul egy FIFO implementációt tartalmaz: Queue.**Queue** osztályt, ami megfelelő a többszálúsághoz
- A sor végére a **put()** függvénnyel helyezzük az elemeket
- Az elejéről a **get()** függvénnyel szedjük le,
- vagy a **get_nowait()**-tel,
  - amely nem blokkol, azaz nem vár elérhető elemre,
  - és kivételt jön, ha üres a sor

msvcrt

- Az **msvcrt** modulnak számos olyan függvénye van, amelyek a Windows platformon hasznosak lehetnek:
  - msvcrt.**kbhit**(): *True*-val tér vissza, ha egy billentyűleütés beolvasásra vár
  - msvcrt.**getche**():
    - beolvas egy billentyűleütést,
    - visszatér az eredményül kapott karakterrel,
    - és kiírja a konzolra, ha nyomtatható karakter
    - blokkol, ha nincs billentyűleütés
    - (A *Ctrl-C*-t nem tudja beolvasni)

Feladat 1

- Készítsünk egy TCP chat alkalmazást, amelyen több kliens képes beszélni egymással, egy közös felületen.

Emlékeztető (fizikai réteg)

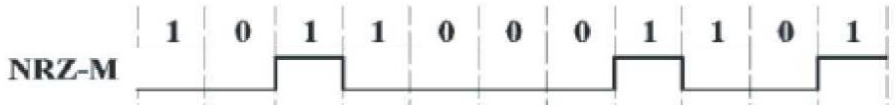
A fogadónak szinkronizálva kell lennie a küldővel. Főprobléma: az óra eltolódás. Megoldások:

1. Egy **explicit órajel** adása:
    - párhuzamos átviteli csatornák használata,
    - szinkronizált adatok,
    - rövid átvitel esetén alkalmas.
2. A fogadó szinkronizálás **kritikus időpontokban:**
    - szinkronizáljunk például egy szimbólum vagy blokk kezdetén,
    - a kritikus időpontokon kívül szabadon futnak az órák,
    - az órajel generátorok rövidtávú stabilitásán nyugszik ez a megközelítés (feltesszük, hogy nem divergálnak el egymástól nagyon gyorsan)
3. **Önütemező jel**:
    - külön órajel szinkronizáció nélkül dekódolható jel,
    - a (kódolt) adatjel tartalmazzon elég információt úgy, hogy a fogadó azonnal tudja, amikor egy bit indul/befejeződik

Bináris kódolások I.

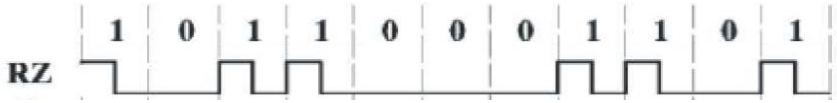
- Non-Return to Zero-Level (NRZ-L)
  - 1: magas feszültség, 0: alacsony (nem önütemező: pl. 0111111110)

- Non-Return to Zero-Mark (NRZ-M)



  - 1: váltás az intervallum elején, 0: nincs váltás (nem önütemező)

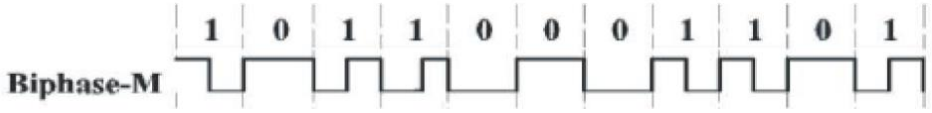
- Return to Zero (RZ)



  - 1: magas feszültség→ alacsony, 0: alacsony (nincs váltás), (nem önü.)

Bináris kódolások II.

- Biphase-Mark



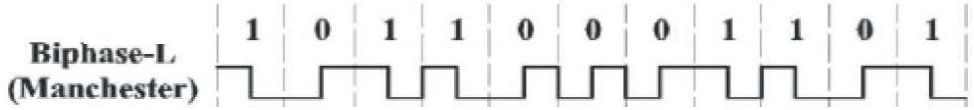
  - minden intervallum elején váltás
  - 1: még egy váltás az intervallum közepén, 0: nincs váltás az intervallum közepén (önütemező)

- Biphase-Space

  - minden intervallum elején váltás
  - minden intervallum elején váltás 1: nincs váltás az intervallum közepén, 0: még egy váltás az intervallum közepén (önütemező)

Bináris kódolások III.

- Manchester (Biphase-L)



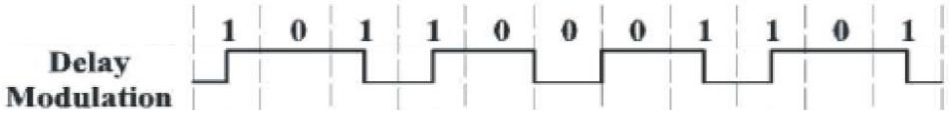
  - 1: magas feszültség → alacsony, 0: alacsony feszültség → magas
  - önütemező: ha egy minta párban megegyeznek a feszültség jelszintek, akkor azt eldobjuk (a digitális szignál folyamatosan ismétlődik)
  - dekódolás: ha a mintapárban magas feszültségről alacsonyra vált, akkor 1, ha alacsonyról magasra, akkor 0
- Mit lát a fogadó oldal, ha egy hosszú szünet (0-s jelszint) után az 10 bitsorozatot próbáljuk átvinni Manchester kódolással?
- Hogyan orvosolható ez a probléma?

Bináris kódolások IV.

- Differential Manchester

  - minden intervallum közepén váltás
  - 1: nincs váltás az intervallum elején, 0: váltás az intervallum elején (önütemező)

- Delay Modulation (Miller)



  - 1: váltás az intervallum közepén, 0: váltás az intervallum végén, ha 0 következik, nincs váltás, ha 1 következik (önütemező)

Órai / házi feladat

- Készíts olyan server-kliens socket alkalmazást, ahol a kliens elküld egy bitsorozatot (max 16) és egy kódolást (’NRZ-L’,’RZ’,’Manchester’,’DiffManchester’), amire a szerver kiszámolja a kódolt jelerőséget! A jelerőségek hármasok: az intervallum eleje, közepe és vége.
- pl:
  - kliens küldi: („NRZ-L”, 1100)
  - server válasza: (1,1,1),(1,1,1),(0,0,0),(0,0,0)
- help:
  - x00-k eltüntetése egy string végéről:

```
szoveg.rstrip('\x00')
```

VÉGE
KÖSZÖNÖM A FIGYELMET!