Számítógépes Hálózatok

4. gyakorlat

Socket programozás, Select

TÖBBKLIENSES KAPCSOLAT.

Visszatekintés

- Múlt héten létrehoztunk egy egyszerű szerver-kliens kapcsolatot.
- 1 kliens el tudott küldeni adatot és az 1 szerver fogadta.
- Próbáljuk ki több klienssel!
- Tapasztalt probléma:
  - A szerver nem írja ki a második kliens csatlakozást (egyből).
  - Ha az első kliennsel küldjük el az adatot: nincs probléma VISZONT
  - Ha a második felcsatlakozott klienssel elküldjük az adatot, nem érkezik egyből válasz, csak miután elküldtük az első klienssel.
  - A szerver csak az első klienstől kapott üzenet után írja ki a többi szöveget.
- Oka:
  - a recv() blokkolja a szerver futását, és a szerver végig várakozik az első kliensre.

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
- Egy lehetséges megoldás lenne különböző szálak használata, de drága a szálak közti kapcsolgatás (környezetváltás, context switch)
- →A select fv. segítségével a monitorozás az op. rsz. hálózati rétegében történik
- →értesíti a programot, amikor valami olvasható a socket-ról, vagy amikor készen áll az írásra

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. visszaadja 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

Socket beállítása

- 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
- Pl.: s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

Select

- setblocking() or settimeout()

```
connection.setblocking(0)      # or connection.settimeout(1.0)
```

- 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
            client, client_addr = s.accpet()
            inputs.append(client)
      else:
            ….          #handle client
```

Feladat – Számológép II.

- Alakítsuk át úgy a számológép szervert, hogy egyszerre több klienssel is képes legyen kommunikálni! Ezt a select függvény segítségével tegye!

Feladat - Chatszoba

- Készítsünk egy chat alkalmazást, amelyen a chat szerverhez csatlakozott kliensek képes beszélni egymással!

- 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.

BEADANDÓ III. (1 PONT)

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)**

Gyakorlás a Barkóba beadandóhoz

ÓRAI FELADAT

Feladat – Számológép III.

- Alakítsuk át a kliens működését úgy, hogy ne csak egy kérést küldjön a szervernek, hanem csatalakozás után 10 kérés-válasz üzenetváltás történjen.
- Alakítsuk át hogy ne standard inputról, hanem random értékeket küldjön (random.randint(1,100)) structban!
- Minden kérés előtt 2 mp várakozással (time.sleep(2))!
- A szerver ugyan abban a struct formátumba küldje vissza az értéket, ahol az első szám lesz az eredmény, a többi adat pedig nem lényeges.
- A kapcsolatot csak a legvégén bontsa a kliens!

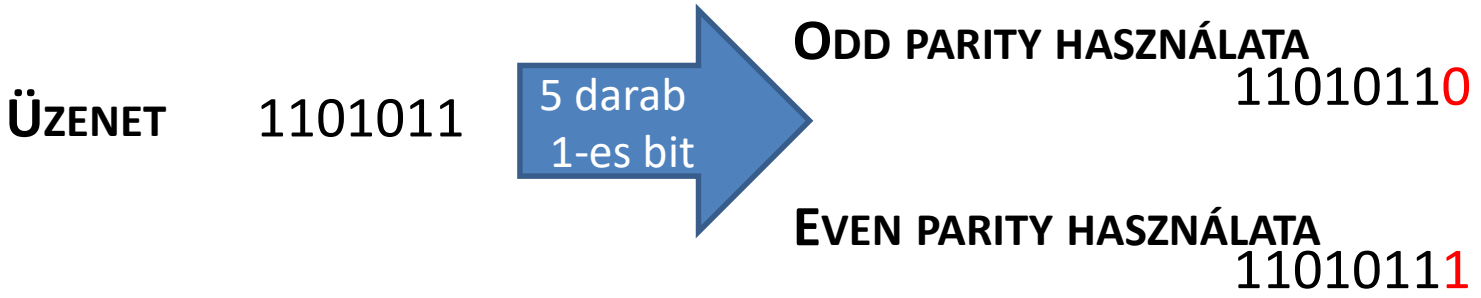
REDUNDANCIA, KÓDOLÁS

Redundancia

- Redundancia nélkül:
  - 2^m lehetséges üzenet írható le *m* biten
  - Ekkor minden hiba egy új helyes üzenetet eredményez → a hiba felismerése lehetetlen
- Emiatt egy keret felépítése:
  - *m* adat bit (üzenet bit)
  - *r* redundáns/ellenőrző bit (üzenetből számolt, új információt nem hordoz)
  - A teljes küldendő keret (kódszó) hossza: *n = m+r*.

Paritás bit használata

- A paritásbitet úgy választjuk meg, hogy ha a kódszóban levő 1-ek száma
  - **Odd parity** – páratlan, akkor 0 befűzése; egyébként 1-es befűzése
  - **Even parity** – páros, akkor 0 befűzése; egyébként 1-es befűzése



Hiba felügyelet Hamming távolsággal

- Hamming távolság: két azonos hosszúságú bitszóban a különböző bitek száma.
- Kiterjesztése azonos hosszúságú bitszavak S halmazára:

$$d(S) \coloneqq \min_{x,y \in S \wedge x \neq y} d(x,y)$$

  - (S halmazt hívják kódkönyvnek vagy egyszerűen kódnak is.)
- *d* bit **hiba felismeréséhez** a megengedett (helyes) keretek halmazában legalább *d+1* Hamming távolság szükséges.
- *d* bit **hiba javításához** a megengedett (helyes) keretek halmazában legalább *2d+1* Hamming távolság szükséges
- Egy $S \subseteq \{0,1\}^n$ **kód rátája** $R_S = \frac{\log_2|S|}{n}$.
  - (a hatékonyságot karakterizálja)
- Egy $S \subseteq \{0,1\}^n$ **kód távolsága** $\delta_S = \frac{d(S)}{n}$.
  - (a hibakezelési lehetőségeket karakterizálja)

Feladat

- Adott S kódkönyv: $S = [1000010,0011011,1011010,0011101]$
- Adjuk meg S Hamming távolságát ($d(S)$)!
- Adjuk meg S kód rátáját (R_S) és távolságát (δ_S)!
- Mit mondhatunk S hibafelismerő és javító képességéről? Igazoljuk az állításunkat!

Megoldás

	1000010	0011011	1011010	0011101
1000010	0	4	2	6
0011011	4	0	2	2
1011010	2	2	0	4
0011101	6	2	4	0

$\rightarrow d(S) = 2$

- $R_S = \frac{\log_2 |S|}{n} = \frac{\log_2 4}{7} = 0.2857$ és $\delta_S = \frac{2}{7} = 0.2857$
- Max. 1 bithiba ismerhető fel, de 0 javítható (mivel a $d(S) = 2$)

Feladat

Egyetlen paritásbit által nyújtottnál nagyobb biztonságot akarunk elérni, így olyan hibaészlelő sémát alkalmazunk, amelyben két paritásbit van: az egyik a páros, a másik a páratlan bitek ellenőrzésére.

- Mekkora e kód Hamming-távolsága?
- Mennyi egyszerű és milyen hosszú burst-ös hibát képes kezelni?

Megoldás

- A kód Hamming-távolsága 2, mivel a páros pozíciókban lévő paritás bit független a páratlan pozíciókban levőtől, külön-külön pedig könnyen látszik, hogy a H-táv 2 → 1 hibát tudunk jelezni.
- A burst-ös hibánál 3 hosszúságúnál még épp tudjuk jelezni, ha baj van, mivel így vagy a páros vagy a páratlan pozíciókra csak 1 hiba fog esni, azt pedig jelezni fogja a megfelelő paritás bit.

VÉGE