Telekommunikációs Hálózatok

4. gyakorlat

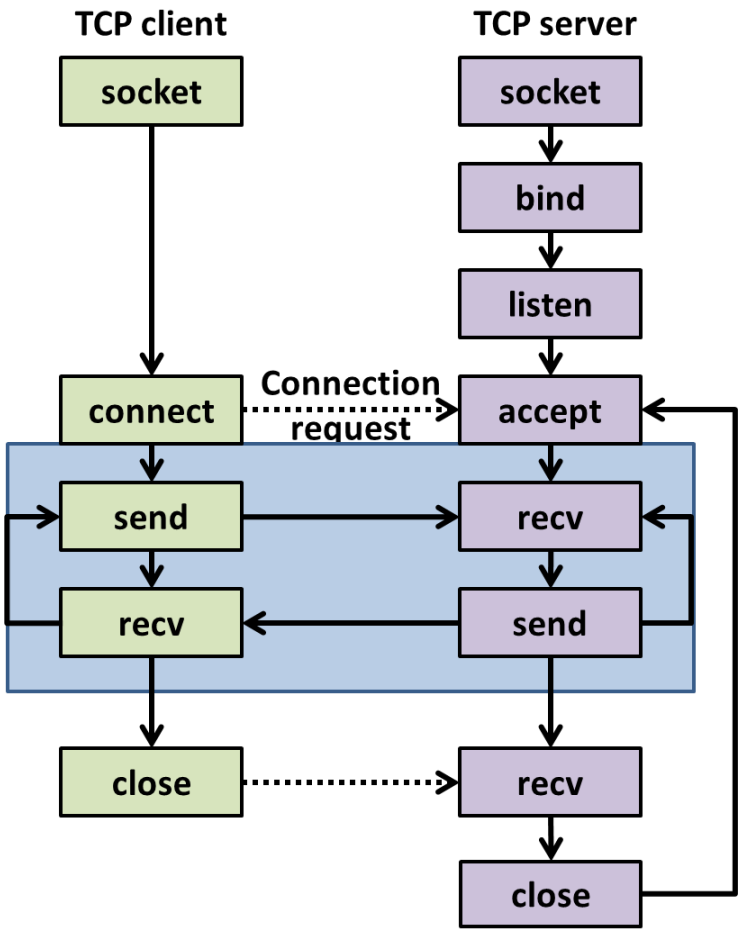
Ismétlés

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

Ismétlés

TCP



Ismétlés

Példa hívások TCP-nél I.

- socket()

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

- bind()

```
server_address = ('localhost', 10000)
sock.bind(server_address)
```

- listen()

```
sock.listen(1)
```

- accept()

```
connection, client_address = sock.accept()
```

Példa hívások TCP-nél II.

- connect

```
server_address = ('localhost', 10000)
sock.connect(server_address)
```

- send(), sendall()

```
connection.sendall(data)
```

- recv()

```
data = connection.recv(16)
```

- close()

```
connection.close()
```

Socket timeout

- setblocking() vagy settimeout()

```
sock.setblocking(0)          # or sock.setblocking(False)
                             # or sock.settimeout(0.0)
                             # or sock.settimeout(1.0)
sock.setblocking(1)          # or sock.setblocking(True)
                             # or sock.settimeout(None)
```

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

- 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-ről, 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

Példa hívások select-nél

- 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
```

Feladat1

- Készítsünk egy TCP alkalmazást, amelyen több kliens képes egyszerre üzenetet küldeni a szervernek, amely minden üzenetre csak annyit ír vissza, hogy „OK”. (Használjuk a select függvényt!)
- Nézzük meg a megoldást!

Feladat2

- Alakítsuk át úgy a számológép szervert, hogy egyszerre több klienssel is képes legyen kommunikálni! Ezt a select függvény segítségével tegye!
- Alakítsuk át a kliens működését úgy, hogy ne csak egy kérést küldjön a szervernek, hanem csatlakozás után 5 kérés-válasz üzenetváltás történjen, minden kérés előtt 2 mp várakozással (time.sleep(2))! A kapcsolatot csak a legvégén bontsa a kliens!

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)

Feladat3

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

Barkóba

HÁZI FELADAT III.

Feladat

- 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.
- Folytatás a következő oldalon!

Feladat

- Ha egy kliens kitalálta a számot, akkor a szerver minden újabb kliens üzenetre a „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!
- 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.
- Folytatás a következő oldalon!

Feladat

- 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

Leadás: A program leadása a TMS rendszeren .zip formátumban, amiben egy client.py és egy server.py szerepeljen!
Határidő: TMS rendszerben

VÉGE
KÖSZÖNÖM A FIGYELMET!