Telekommunikációs Hálózatok

4. gyakorlat

TCP (ismétlés)

TCP (ismétlés)

- socket()

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

- bind()

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

- listen()

```
server.listen(1)
```

- accept()

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

TCP (ismétlés)

- send(), sendall()

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

- recv()

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

- close()

```
connection.close()
```

- connect()

```
server_address = (‚localhost’, 10000)
client.connect(server_address)
```

Socket timeout

- setblocking() vagy settimeout()
- Amennyiben nem végezhető el a művelet várjunk amíg elvégezhető nem lesz:

```
sock.setblocking(1)     # or sock.setblocking(True)
                        # or sock.settimeout(None)
```

- Egy idő után (settimeout nem 0 értékkel) vagy azonnal dobjunk kivételt, ha nem végezhető el:

```
sock.setblocking(0)     # or sock.setblocking(False)
                        # or sock.settimeout(0.0)
                        # or sock.settimeout(1.0)
```

Socket konfigurálása

- socket.setsockopt(level, optname, value): az adott socket opciót állítja be
- Általunk használt **level** értékek az alábbiak lesznek:
  - socket.**IPPROTO_IP**: IP szintű beállítás
  - socket.**SOL_SOCKET**: socket API szintű beállítás
- Az **optname** a beállítandó paraméter neve, pl.:
  - socket.**SO_REUSEADDR**: a kapcsolat bontása után a port újrahasznosítása
- A paraméterül adható szimbolikus konstansok teljes listája

Socket konfigurálása

- A **value** lehet bytestring vagy egész szám:
  - Az előbbi esetén biztosítania kell a hívónak, hogy a megfelelő biteket tartalmazza (pl. a struct segítségével)
  - A socket.**SO_REUSEADDR** esetén ha 0, akkor lesz hamis a „tulajdonság”, egyébként igaz

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

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 connects
        ...
    else:
        ...             #handle client
```

Gyakorlás I.

- Készítsünk egy TCP alkalmazást, amelyen több kliens képes egyszerre üzenetet küldeni a szervernek.
- A szerver minden üzenetre `b"OK"` bytestringgel válaszoljon.
- Használjuk a select függvényt!

Gyakorlás II.

- 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 is:
  - Ne csak egy kérést küldjön a szervernek, hanem csatlakozás után 5 kérés-válasz üzenetváltás történjen.
  - Minden kérés előtt várjon 2 másodpercet (time.sleep(2))!
  - A kapcsolatot csak az 5. válasz megérkezése után bontsa!

Chat alkalmazás

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

III. Beadandó

Barkóba

- Leírás

III. Beadandó

- 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 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!

III. Beadandó

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

Logaritmikus keresés (bináris keresés) emlékeztető

Leadás

Paraméterezés:

```
python3 server.py <bind_address> <bind_port> # A bindolás során használandó pár
python3 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