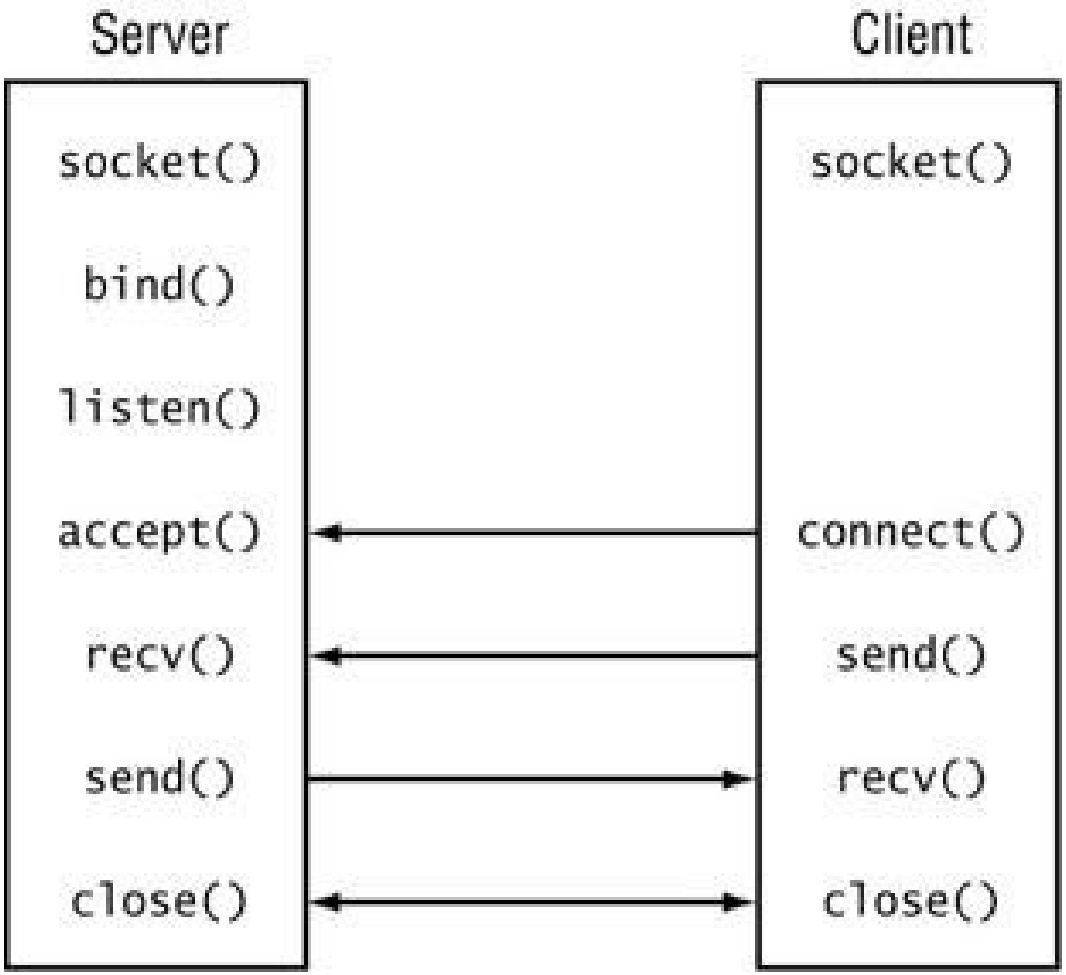
Számítógépes Hálózatok

4. gyakorlat

TCP


Server
Client
socket()
bind()
listen()
accept()
recv()
send()
close()
socket()
connect()
send()
recv()
close()

TCP

- 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()
```

TCP

- send(), sendall()

```
connection.sendall(data)                    #python 2.x
```

```
connection.sendall(data.encode())           #python 3.x
```

- recv()

```
data = connection.recv(16)                  #python 2.x
```

```
data = connection.recv(16).decode()         #python 3.x
```

- close()

```
connection.close()
```

- connect()

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

Struktúraküldése

- Binárissá alakítjuk az adatot

```
import struct
values = (1, ”ab”.encode(), 2.7)          #python2: nem kell encode()
packer = struct.Struct('I 2s f')          #Int, char[2], float
packed_data = packer.pack(*values)
```

- Visszalakítjuk a kapott üzenetet

```
import struct
unpacker = struct.Struct('I 2s f')
unpacked_data = unpacker.unpack(data)
```

- megj.: integer 1 – 4 byte, stringként 1 byte, azaz hatékonyabb stringként átküldeni.

Feladat - Számológép

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.

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.accept()
            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!

- Alakítsuk át a kliens működését úgy, hogy ne csak egy kérést küldjön a szervernek, hanem csatalakozás után 5 kérés-válasz üzenetváltás történjen, minden kérés előtt 2 mp várakozással (time.sleep(2))! A kapcsolatot csak a legvégén bontsa a kliens!

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 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

- 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!

Beadandó

- 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-en**

CRC

- Definiáljuk a G(x) generátor polinomot (G foka r), amelyet a küldő és a vevő egyaránt ismer.
- **Algoritmus:**

1. Legyen G(x) foka r. Fűzzünk r darab 0 bitet a keret alacsony helyi értékű végéhez, így az m+r bitet fog tartalmazni és az $x^rM(x)$ polinomot fogja reprezentálni.

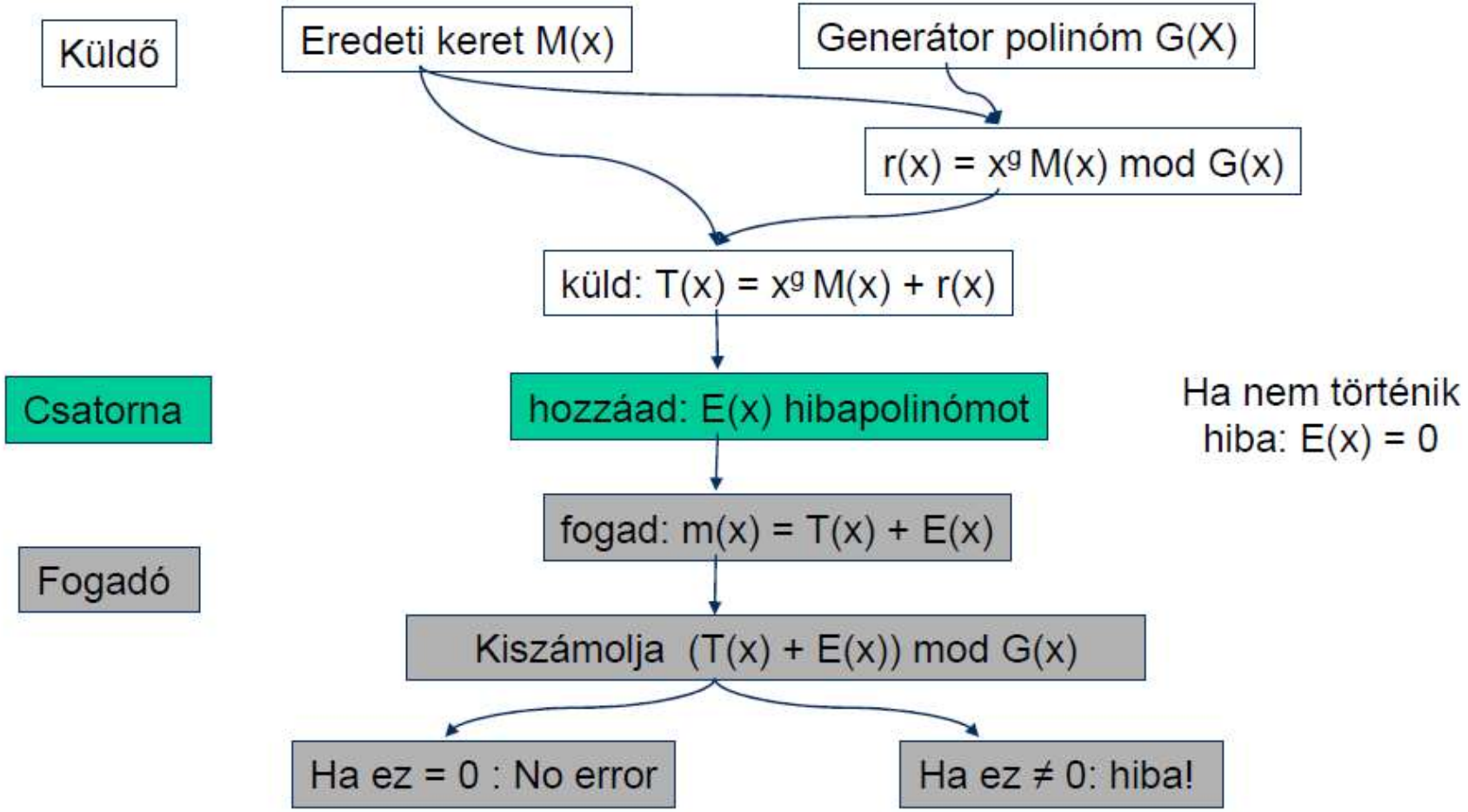
2. Osszuk el az $x^rM(x)$-hez tartozó bitsorozatot a G(x)-hez tartozó bitsorozattal modulo 2

3. Vonjuk ki a maradékot (mely mindig r vagy kevesebb bitet tartalmaz) az $x^rM(x)$-hez tartozó bitsorozatból. Az eredmény az ellenőrző összeggel ellátott, továbbítandó keret. Jelölje a továbbítandó keretnek megfelelő a polinomot T(x).

4. A vevő a T(x) + E(x) polinomnak megfelelő sorozatot kapja, ahol E(x) a hiba polinom. Ezt elosztja G(x) generátor polinommal.

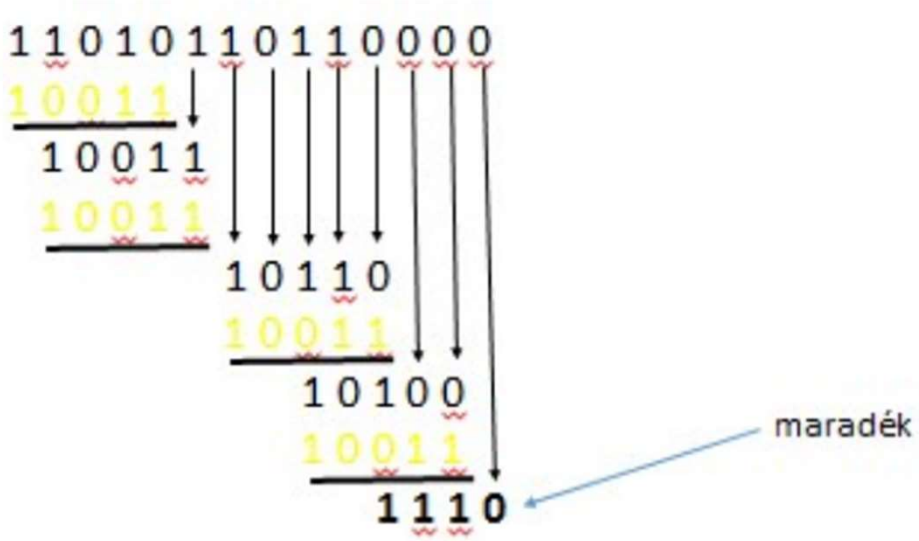
- Ha az osztási maradék, amit R(x) jelöl, nem nulla, akkor hiba történt

CRC


Küldő
Eredeti keret M(x)
Generátor polinóm G(X)
r(x) = x^g M(x) mod G(x)
küld: T(x) = x^g M(x) + r(x)
Csatorna
hozzáad: E(x) hibapolinómot
Ha nem történik hiba: E(x) = 0
fogad: m(x) = T(x) + E(x)
Fogadó
Kiszámolja (T(x) + E(x)) mod G(x)
Ha ez = 0 : No error
Ha ez ≠ 0: hiba!

CRC példa

- Keret: 1101011011
- Generátor: 10011
- A továbbítandó üzenet: 11010110111110
- Osztás binárisan: xor-ozgatunk



CRC példa

- Az osztásban 11010110110000 az $x^{13}+x^{12}+x^{10}+x^8+x^7+x^5+x^4$ polinomot reprezentálja. 10011 pedig az x^4+x+1 polinomot.
- Ebből (11010110110000) * (1110) = 11010110111110 ami az elküldendő keretünk lesz.
- (11010110111110) mod 10011 = 0
- Amennyiben hozzáadtunk volna egy E(x) hibapolinomot (pl. $E(x) = x^2+x$ =110), akkor a maradék nem nulla (a példában 11) lenne, így tudnánk, hogy meghibásodott a keret.

CRC, MD5 pythonban

- CRC

```
import binascii, zlib

test_string = "Fekete retek rettenetes".encode('utf-8')

print(hex(binascii.crc32(bytearray(test_string))))
print(hex(zlib.crc32(test_string)))
```

- MD5

```
import hashlib

test_string = "Fekete retek rettenetes".encode('utf-8')

m = hashlib.md5()
m.update(test_string)
print(m.hexdigest())
```

SHA-2 pythonban

- SHA-224,SHA-256,SHA-384,SHA-512
- Mindegyik ugyanúgy működik, csak a hossz tér el.
- Az interface azonos az MD5-ével.

```
import hashlib

test_string = "Fekete retek rettenetes".encode('utf-8')

m = hashlib.sha256()
m.update(test_string)
print(m.hexdigest())
```

Fájl átvitel

- fájl bináris megnyitása

```
with open („input.txt”, „rb”) as f:
        …
```

- read(x) – x bytes

```
        …
        f.read(128)    #128 byte-ot fog beolvasni
```

„When size is omitted or negative, the entire contents of the file will be read and returned; it’s your problem if the file is twice as large as your machine’s memory. „ - python.org

PYTHON SOCKET - PROXY

Feladat

Készítsünk egy egyszerű TCP alapú proxyt (átjátszó). A proxy a kliensek felé szerverként látszik, azaz a kliensek csatlakozhatnak hozzá. A proxy a csatlakozás után kapcsolatot nyit egy szerver felé (parancssori argumentum), majd minden a klienstől jövő kérést továbbítja a szerver felé és a szervertől jövő válaszokat pedig a kliens felé.

Pl: netProxy.py ggombos.web.elte.hu 80

Web browserbe írjuk be: localhost:10000

Feladat – Tiltsunk le valamilyen tartalmat

A SzamHalo-t tartalmazó URL-ek ne legyenek elérhetők a proxyn keresztül.

A válasz legyen valamilyen egyszerű HTML üzenet, ami jelzi a blokkolást.

megj: header: „HTTP/1.1 404 Not Found\n\n”

Proxy

- Készítsünk a számológéphez egy proxy-t, ami értelmezi a klienstől kapott TCP kéréseket, az első operandust megszorozza kettővel, a másodikhoz pedig hozzáad egyet. A módosított kérést elküldi a TCP servernek, majd az eredményt visszaküldi a kliensnek.

VÉGE