Telekommunikációs Hálózatok Péntek

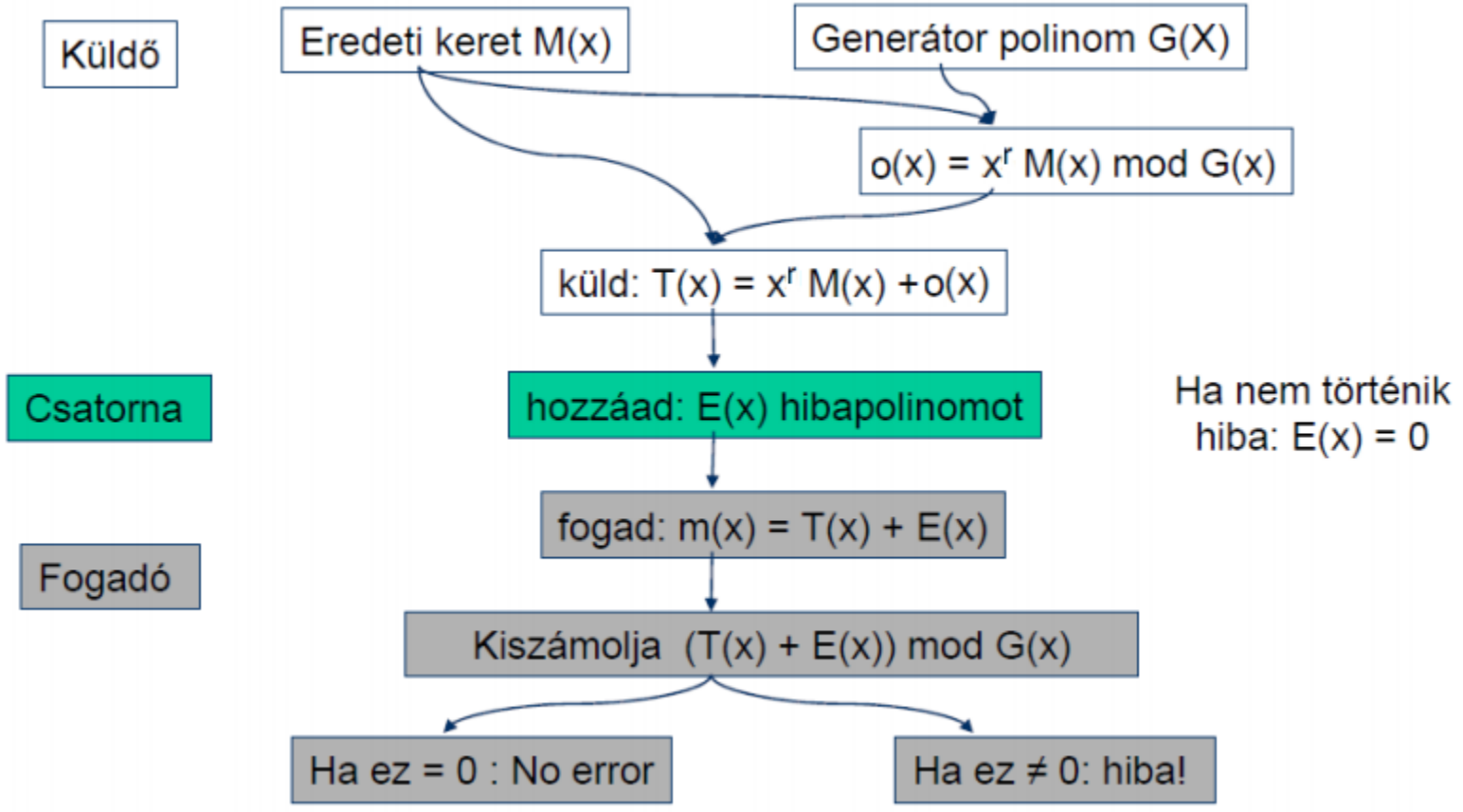
5. gyakorlat

Zárthelyi

- **ZH: 40 perc, 6 kérdés + 1 jegyzetfeltöltés**
- **Az utolsó kérdésnél be kell másolni a ZH feladatok megoldásaihoz tartozó jegyzetet szövegként vagy képként fel kell tölteni, ahol pár mondatban ki van fejtve a megoldás módja az egyes feladatokhoz.**
- **Tehát önmagában a végeredmény nem elegendő!**
- A ZH után folytatjuk a gyakorlatot

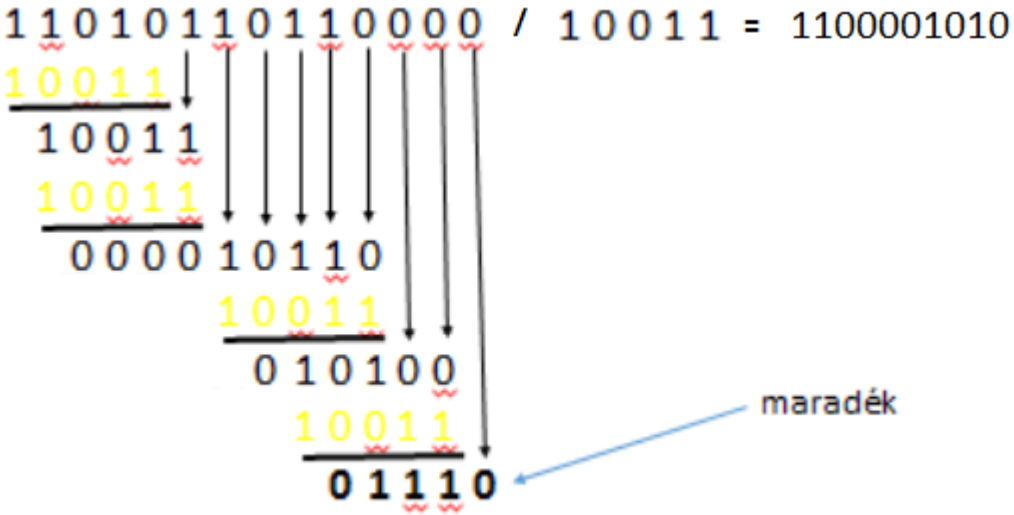
CRC hibajelző kód – emlékeztető

- Forrás: Dr. Lukovszki Tamás fóliái alapján



Példa CRC számításra – emlékeztető

- Keret ($M(x)$): 1101011011
- Generátor ($G(x)$): 10011
- Végezzük el a következő maradékos osztást: $\frac{11010110110000}{10011}$
- (A maradék lesz a CRC ellenőrzőösszeg)



Ellenőrzés:

$$
\begin{array}{r}
1100001010 \\
\cdot \quad 10011 \\
\hline
1100001010 \\
1100001010 \\
1100001010 \\
\hline
11010110111110 \\
+ \quad 01110 \\
\hline
11010110110000
\end{array}
$$

Példa CRC számításra – kiegészítés

- Az előbbi szorzásnál: 1100001010 megfelel a $1 \cdot x^9 + 1 \cdot x^8 + 1 \cdot x^3 + 1 \cdot x$, 10011 megfelel a $1 \cdot x^4 + 1 \cdot x + 1$ polinomnak
- Ha a polinom alakjukban összeszoroznánk:

$$(1 \cdot x^9 + 1 \cdot x^8 + 1 \cdot x^3 + 1 \cdot x) \cdot (1 \cdot x^4 + 1 \cdot x + 1)$$

$$= 1 \cdot x^{13} + 1 \cdot x^{10} + 1 \cdot x^9 + 1 \cdot x^{12} + 1 \cdot x^9 + 1 \cdot x^8 + 1 \cdot x^7 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^5 + 1 \cdot x^2 + 1 \cdot x$$

$$= 1 \cdot x^{13} + 1 \cdot x^{12} + 1 \cdot x^{10} + \underbrace{(1+1)}_{0\ (mod\ 2)} \cdot x^9 + 1 \cdot x^8 + 1 \cdot x^7 + 1 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x$$

- Ez bináris alakban éppen a 11010110111110 lesz

Példa CRC számításra – kiegészítés

- Az előbbi osztásnál: 11010110110000 megfelel a $1 \cdot x^{13} + 1 \cdot x^{12} + 1 \cdot x^{10} + 1 \cdot x^{8} + 1 \cdot x^{7} + 1 \cdot x^{5} + 1 \cdot x^{4}$, 10011 megfelel a $1 \cdot x^{4} + 1 \cdot x + 1$ polinomnak
- Ha a polinom alakjukban végeznénk az osztást, akkor először az osztó legnagyobb fokú tagjával (x^4) leosztanánk az osztandó legnagyobb fokú tagját (x^{13})
- Ez x^9-et eredményezi. „Lekönyveljük”, és összeszorozzuk az osztóval: $x^9 \cdot (1 \cdot x^4 + 1 \cdot x + 1) = x^{13} + x^{10} + x^9$ (Vegyük észre, hogy ez az osztó „eltoltját” fogja mindig eredményezni bináris reprezentációval nézve.)
- Ezt kivonjuk az eredeti osztandóból, amely az $1 \cdot x^{12} + \underbrace{(-1)}_{1\ (mod\ 2)} \cdot x^9 + 1 \cdot x^8 + 1 \cdot x^7 + 1 \cdot x^5 + 1 \cdot x^4$ polinomot eredményezi. Bináris alakban ez a polinom: 1001110110000.

Példa CRC számításra – kiegészítés

- Ennek a polinomnak a legnagyobb fokú tagját osztjuk majd tovább az osztó legnagyobb fokú tagjával az előző lépésekhez hasonlóan, amely x^8-at eredményezi. „Lekönyveljük”, és összeszorozzuk az osztóval:

$$x^8 \cdot (1 \cdot x^4 + 1 \cdot x + 1) = x^{12} + x^9 + x^8$$

- Ezt kivonjuk az eredeti osztandóból, amely az $1 \cdot x^7 + 1 \cdot x^5 + 1 \cdot x^4$ polinomot eredményezi. Bináris alakban ez a polinom: 10110000. Ezt folytatjuk.
- A végén a „lekönyvelt” tagokat összeadjuk, amely az eredményt adja, továbbá a megmaradt, az osztónál kisebb fokú polinom a maradékot.

Feladat1

- Adva a $G(x) = x^4 + x^3 + x + 1$ generátor polinom.
- Számoljuk ki a $1100\ 1010\ 1110\ 1100$ bemenethez a 4-bit CRC ellenőrzőösszeget!
- A fenti üzenet az átvitel során sérül, a vevő adatkapcsolati rétege az $1100\ 1010\ 1101\ 1010\ 0100$ bitsorozatot kapja. Történt-e olyan hiba az átvitel során, amit a generátor polinommal fel lehet ismerni? Ha nem, akkor ennek mi lehet az oka?

Feladat1 megoldása

- Mivel a generátor polinom foka 4, ezért négy 0-t írunk a bemenet végéhez. A $G(x)$ bináris alakban: 11011 lesz, tehát a $\frac{1100\ 1010\ 1110\ 1100\ 0000}{11011}$ maradékos osztást kell elvégeznünk:

```
1100101011011000000   /   11011
-----------------------
1100101011011000000
11011
-----------------------
0010010111011000000
 11011
-----------------------
 1001111011000000
 11011
-----------------------
 100011011000000
 11011
-----------------------
 10101011000000
 11011
-----------------------
 1110011000000
 11011
-----------------------
 011111000000
 11011
-----------------------
 0100000000
 11011
-----------------------
 10110000
 11011
-----------------------
 1101000
 11011
-----------------------
 000100   →   0100 a CRC ellenőrzőösszeg
```

Feladat1 megoldása

- Az előbbi számításnál az jött ki, hogy 1100 1010 1110 1100 0100 lenne az a bitsorozat, amelyet a vevő kapna. Ha ebből kivonjuk az alfeladatban megadott sorozatot, az alábbi eredmény jön ki:

$$
\begin{array}{rl}
 & 11001010111011000100 \\
- & 11001010110110100100 \\
\hline
 & 00000000001101100000
\end{array}
$$

- Tehát a két bitsorozat pontosan a generátor polinom többszörösével tér egymástól, amely tehát a hiba polinom. Ezt pedig nem lehet felismerni.

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())
```

Előző órai Feladat2

- Készítsünk egy szerver-kliens alkalmazást, ahol a kliens elküld 2 számot és egy operátort (négy alapművelet közül) a szervernek, amely kiszámolja és visszaküldi az eredményt. A kliens üzenete legyen struktúra.

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

- 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)
```

- 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
```

Feladat3

- 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!

Feladat4

- 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!

Házi feladat – 1 pont

- 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!
- Folytatás a következő oldalon!

Házi feladat – 1 pont

- 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!

Házi feladat – 1 pont

- 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: **2020.11.01. 23:59**

Házi feladat
netcopy alkalmazás 1 pont

Készíts egy netcopy kliens/szerver alkalmazást, mely egy fájl átvitelét és az átvitt adat ellenőrzését teszi lehetővé CRC vagy MD5 ellenőrzőösszeg segítségével! A feladat során három komponenst/programot kell elkészíteni:

1. Checksum szerver: (fájl azonosító, checksum hossz, checksum, lejárat (mp-ben)) négyesek tárolását és lekérdezését teszi lehetővé. A protokoll részletei a következő oldalon.
2. Netcopy kliens: egy parancssori argumentumban megadott fájlt átküld a szervernek. Az átvitel során/végén kiszámol egy md5 checksumot a fájlra, majd ezt feltölti fájl azonosítóval együtt a Checksum szerverre. A lejárati idő 60 mp. A fájl azonosító egy egész szám, amit szintén parancssori argumentumban kell megadni.
3. Netcopy szerver: Vár, hogy egy kliens csatlakozzon. Csatlakozás után fogadja az átvitt bájtokat és azokat elhelyezi a parancssori argumentumban megadott fájlba. A végén lekéri a Checksum szervertől a fájl azonosítóhoz tartozó md5 checksumot és ellenőrzi az átvitt fájl helyességét, melynek eredményét stdoutputra is kiírja. A fájl azonosító itt is parancssori argumentum kell legyen.

Beadás: BE-AD rendszeren keresztül, DE még nincs meghirdetve

Checksum szerver - TCP

- Beszúr üzenet
  - Formátum: szöveges
  - Felépítése: BE|<fájl azon.>|<érvényesség másodpercben>|<checksum hossza bájtszámban>|<checksum bájtjai>
  - A „|” delimiter karakter
  - Példa: BE|1237671|60|12|abcdefabcdef
    - Ez esetben: a fájlazon: 1237671, 60mp az érvényességi idő, 12 bájt a checksum, abcdefabcdef maga a checksum
  - Válasz üzenet: OK
- Kivesz üzenet
  - Formátum: szöveges
  - Felépítése: KI|<fájl azon.>
  - A „|” delimiter karakter
  - Példa: KI|1237671
    - Azaz kérjük az 1237671 fájl azonosítóhoz tartozó checksum-ot
  - Válasz üzenet: <checksum hossza bájtszámban>|<checksum bájtjai>
    Péda: 12|abcdefabcdef
  - Ha nincs checksum, akkor ezt küldi: 0|
- Futtatás
  - .\checksum_srv.py <ip> <port>
    - <ip> - pl. localhost a szerver címe bindolásnál
    - <port> - ezen a porton lesz elérhető
  - A szerver végtelen ciklusban fut és egyszerre több klienst is ki tud szolgálni. A kommunikáció TCP, csak a fenti üzeneteket kezeli.
  - Lejárat utáni checksumok törlődnek.

Netcopy kliens – TCP alapú

- Működés:
  - Csatlakozik a szerverhez, aminek a címét portját parancssori argumentumban kapja meg.
  - Fájl bájtjainak sorfolytonos átvitele a szervernek.
  - A Checksum szerverrel az ott leírt módon kommunikál.
  - A fájl átvitele és a checksum elhelyezése után bontja a kapcsolatot és terminál.
- Futtatás:
  - .\netcopy_cli.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <fájl azon> <fájlnév elérési úttal>
    - <fájl azon>: egész szám
    - <srv_ip> <srv_port>: a netcopy szerver elérhetősége
    - <chsum_srv_ip> <chsum_srv_port>: a Checksum szerver elérhetősége

Netcopy szerver – TCP alapú

- Működés:
  - Bindolja a socketet a parancssori argumentumban megadott címre.
  - Vár egy kliensre.
  - Ha acceptálta, akkor fogadja a fájl bájtjait sorfolytonosan és kiírja a paracssori argumentumban megadott fájlba.
  - Fájlvége jel olvasása esetén lezárja a kapcsolatot és utána ellenőrzi a fájlt a Checksum szerverrel.
  - A Checksum szerverrel az ott leírt módon kommunikál.
  - Hiba esetén a stdout-ra ki kell írni: CSUM CORRUPTED
  - Helyes átvitel esetén az stdout-ra ki kell írni: CSUM OK
  - Fájl fogadása és ellenőrzése után terminál a program.
- Futtatás:
  - .\netcopy_srv.py <srv_ip> <srv_port> <chsum_srv_ip> <chsum_srv_port> <fájl azon> <fájlnév elérési úttal>
    - <fájl azon>: egész szám ua. mint a kliensnél – ez alapján kéri le a szervertől a checksumot
    - <srv_ip> <srv_port>: a netcopy szerver elérhetősége – bindolásnál kell
    - <chsum_srv_ip> <chsum_srv_port>: a Checksum szerver elérhetősége
    - <fájlnév> : ide írja a kapott bájtokat

VÉGE
KÖSZÖNÖM A FIGYELMET!