

# Telekommunikációs Hálózatok

6. gyakorlat

# Gyakorlás I.

- Készítsünk egy proxy szerveret
  - A TCP számítógép kliensünktől fogadja a kéréseket.
  - Továbbítsa őket az UDP-s számítógép szervernek.
  - A szervertől érkező válaszokat továbbítsa a kliensnek.

# Gyakorlás II.

- Készítsünk egy számológép klienst ami egy UDP szervertől kéri el a TCP-s szerver elérhetőségét!
  - A kliens küldjön egy b"Hello Server" bytearrayet az UDP szervernek.
  - Az UDP szerver válaszoljon a TCP szerver címével, ahova a kliens a számokat és az operátort fogja elküldeni.

# Gyakorlás III.

- Készítsünk proxyt, ahol a kliens egy webböngésző, a szerver pedig egy webszerver.
  - A proxy továbbítsa a böngésző kérését a szervernek.
  - Alapesetben változtatás nélkül továbbítsa a szerver válaszát a böngészőnek.
  - Amennyiben a válasz tartalmazza a “SzamHalo” stringet, akkor a szerver válasza helyett küldjön 404-es hibakódot.

Indítás példa:

```
python3 netProxy.py ggombos.web.elte.hu 9000
```

Böngészőben: localhost:9000

# Gyakorlás IV.

- Tekintsük a következő paritás-technikát:
  - Értelmezzük az  $n$  küldendő adatbitet, mint egy  $k \times l$  bit mátrixot.
  - Minden oszlophoz számoljunk ki egy paritás-bitet (odd parity) és egészítsük ki a mátrixot egy új sorral, mely ezeket a paritás-biteket tartalmazza.
  - Küldjük el az adatokat soronként.
- Példa  $k = 2, l = 3$  esetén:

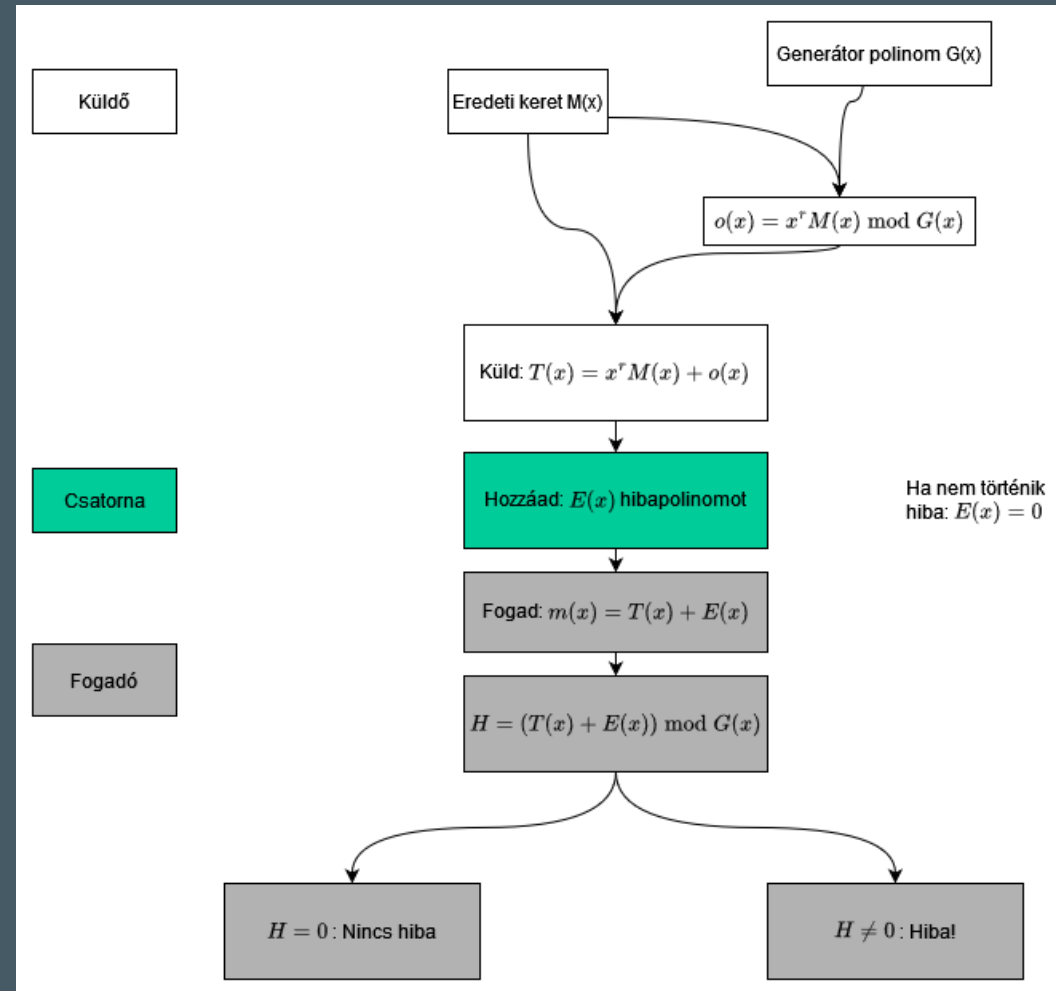
1	0	1
<hr/>		
0	1	1
<hr/>		
0	0	1

# Gyakorlás IV.

- Hogy viselkedik ez a módszer egyszerű bit-hibák és löketszerű (burst) bit-hibák esetén, ha  $k = 3$ ,  $l = 4$ ? Milyen hosszú lehet egy bitsorozat, melynek minden bitje hibás, hogy a hibázást meg tudjuk állapítani? (Löketszerű: egymás utáni bitek hibásan jönnek át)
- Egészítsük ki a mátrixot egy új oszloppal is, amely minden sorhoz paritás-bitet tartalmaz (két dimenziós paritás technika). Hogyan használható ez a módszer 1-bithiba javítására? Mi a helyzet több bithibával és löketszerű-hibákkal?

# CRC hibajelző kód

Forrás: Dr. Lukovszki Tamás diái alapján



# Példa CRC számításra

- Keret ( $M(x)$ ): 1101011011
- Generátor ( $G(x)$ ): 10011
- Végezzük el a következő maradékos osztást:  $\frac{1101011011_2}{10011_2}$
- A maradék lesz a CRC ellenőrző összeg

The diagram illustrates the long division of the dividend 1101011011 by the generator 10011. The dividend is extended with four zeros to 11010110000. The division proceeds as follows:

1 1 0 1 0 1 1 0 1 1 0 0 0 0 / 1 0 0 1 1 = 1 1 0 0 0 0 1 0 1 0												
					↓	↓	↓	↓	↓	↓	↓	
1	0	0	1	1								
<hr/>												
1	0	0	1	1								
<hr/>												
0	0	0	0		1	0	1	1	0			
					↓	↓	↓	↓	↓	↓		
					1	0	0	1	1			
					<hr/>							
					0	1	0	1	0	0		
					↓	↓	↓	↓	↓	↓		
					1	0	0	1	1			
					<hr/>							
					0	1	1	1	0			

A blue arrow points from the label "Maradék" to the final remainder 01110.



# Gyakorlás V.

- Adott a  $G(x) = x^4 + x^3 + x + 1$  generátor polinom.
- Számoljuk ki a  $1100101011101100_2$  bemenethez a 4-bites CRC ellenőrzőösszeget!
- A fenti üzenet az átvitel során sérül, a vevő adatkapcsolati rétege az  $11001010110110100100_2$  bitsorozatot kapja.
- Történt-e olyan hiba az átvitel során, amit a generátor polinommal fel lehet ismerni? Ha nem, akkor ennek mi lehet az oka?

# CRC és hashelés pythonban

- CRC

```
import binascii, zlib
test_string= "You must cut down the mightiest tree in the forest with a herring".encode('utf-8')
print(hex(binascii.crc32(bytearray(test_string))))
print(hex(zlib.crc32(test_string)))
```

- MD5

```
import hashlib
test_string= "You must cut down the mightiest tree in the forest with a herring".encode('utf-8')
m = hashlib.md5()
m.update(test_string)
print(m.hexdigest())
```

- SHA1/SHA256

```
import hashlib
test_string= "You must cut down the mightiest tree in the forest with a herring".encode('utf-8')
m = hashlib.sha1()    #or hashlib.sha256
m.update(test_string)
print(m.hexdigest())
```

# IV. Beadandó

## *Netcopy*

- Leírás

# IV. Beadandó

- 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:
  - Checksum szerver: (fájl azonosító, checksum hossz, checksum, lejárat (mp-ben)) négyesek tárolását és lekérdezését teszi lehetővé.
  - 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 lejárat idő 60 mp. A fájl azonosító egy egész szám, amit szintén parancssori argumentumban kell megadni.
  - 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.

# 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

# Checksum szerver - TCP

- Kivesz üzenet (folyt.)
  - Példa: KI|1237671
  - Azaz kérjük az 1237671 fájl azonosítóhoz tartozó checksum-ot
- Válasz üzenet: |
  - Péda: 12|abcdefabcdef
- Ha nincs checksum, akkor ezt küldi: 0|
- 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, de elég, ha csak a következő kérésnél ellenőrzi.
- Futtatás:

```
python3 checksum_srv.py <ip> <port>
```

- <ip> - pl. localhost a szerver címe bindolásnál
- <port> - ezen a porton lesz elérhető

# Netcopy kliens – TCP

- 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:

```
python3 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

- Működés:
  - Bindolja a socketet a parancssori argumentumban megadott címre.
  - Vár egy kliensre.
  - Ha acceptálta, akkor fogadja a fájl bájtjait sorfolytonosan és kiírja a parancssori 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.



# Netcopy szerver – TCP

- Futtatás:

```
python3 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

**Leadás:** A program leadása a TMS rendszeren .zip formátumban, amiben egy [checksum\\_srv.py](#), egy [netcopy\\_cli.py](#) és egy [netcopy\\_srv.py](#) szerepeljen!

**Határidő:** TMS rendszerben