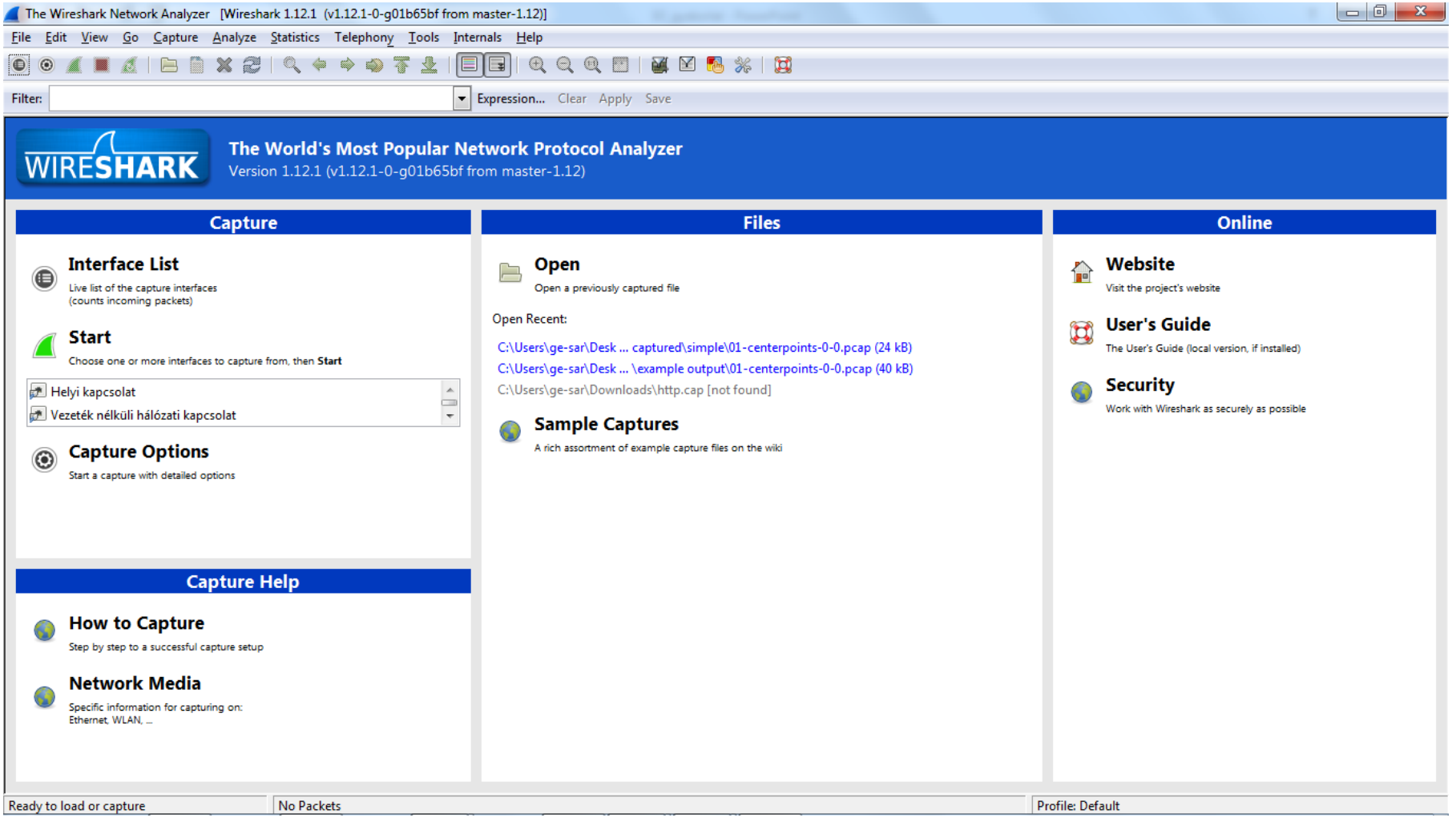
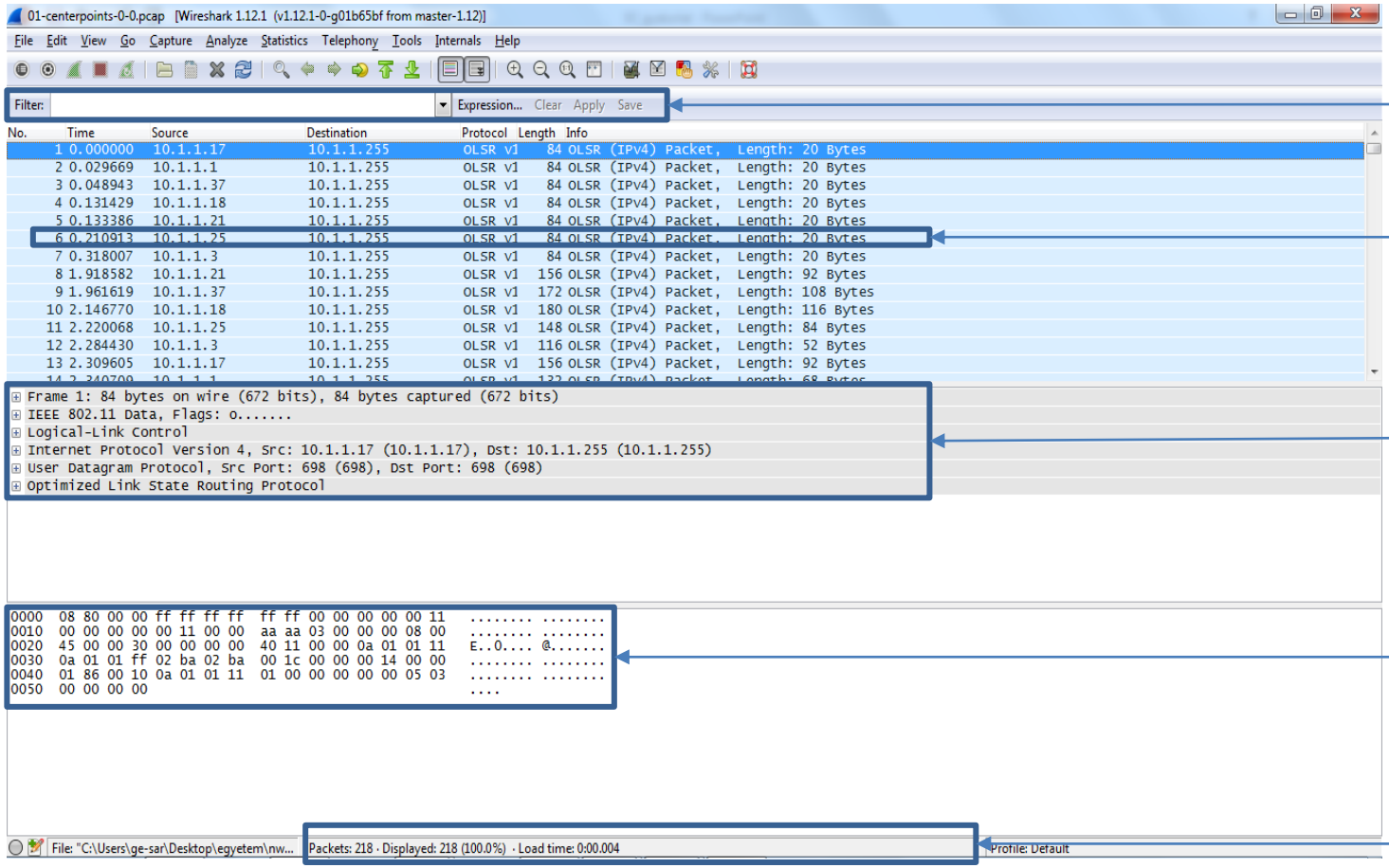
Számítógépes Hálózatok

9. gyakorlat

Wireshark


The Wireshark Network Analyzer [Wireshark 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
WIRESHARK
The World's Most Popular Network Protocol Analyzer
Version 1.12.1 (v1.12.1-0-g01b65bf from master-1.12)
Capture
Interface List
Live list of the capture interfaces
(counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
Helyi kapcsolat
Vezeték nélküli hálózati kapcsolat
Capture Options
Start a capture with detailed options
Capture Help
How to Capture
Step by step to a successful capture setup
Network Media
Specific information for capturing on:
Ethernet, WLAN, ...
Files
Open
Open a previously captured file
Open Recent:
C:\Users\ge-sar\Desk ... captured\simple\01-centerpoints-0-0.pcap (24 kB)
C:\Users\ge-sar\Desk ... \example output\01-centerpoints-0-0.pcap (40 kB)
C:\Users\ge-sar\Downloads\http.cap [not found]
Sample Captures
A rich assortment of example capture files on the wiki
Online
Website
Visit the project's website
User's Guide
The User's Guide (local version, if installed)
Security
Work with Wireshark as securely as possible
Ready to load or capture
No Packets
Profile: Default

Wireshark



Szűrők definiálására alkalmas input eszközök

Csomag összefoglaló nézete

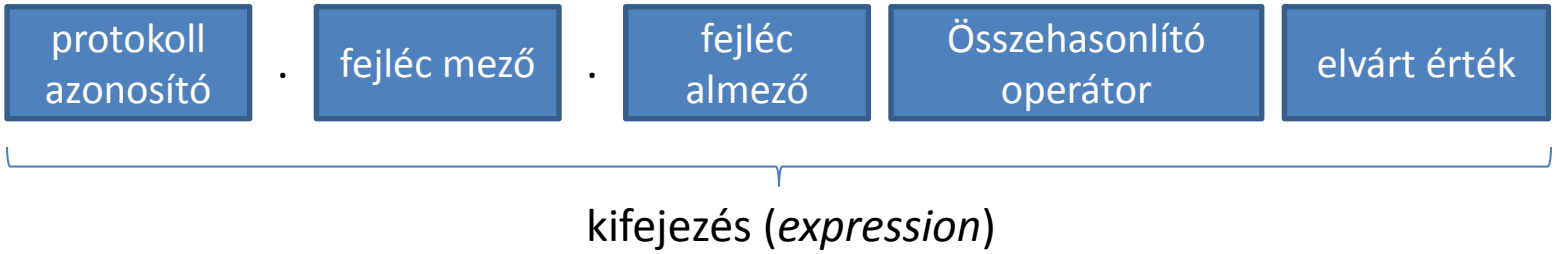
Kiválasztott csomag hierarchikus nézet

Kiválasztott csomag bájt-alapú nézet

Szűrés statisztikái

Wireshark

- Korábban rögzített adatok elemzésére szolgál.
- Szűrés felépítése:



- Operátorok: or, and, xor, not
- Példa: tcp.flags.ack==1 and tcp.dstport==80

Szűrési feladatok 1 - HTTP

A http_out.pcapng felhasználásával állomány felhasználásával válaszolja meg az alábbi kérdéseket:

1. Milyen oldalakat kértek le a szűrés alapján? Milyen böngészőt használtak hozzá?
2. Hány darab képet érintett a böngészés? (Segítség: *webp.*)
3. Az első képhez tartozó socket-en hány kép kérése történt? Volt-e olyan, amelyet nem töltött le újra, mert megvolt már?
4. Hány olyan erőforrás volt, amelyet nem kellett újra töltenie a böngészőnek? Mely oldalakat érintette ez?
5. Volt-e olyan kérés, amely titkosított kommunikációt takar? (Segítség: *SSL/TLS.*) Kövesse végig az első TCP folyamát. Mit tud kideríteni a kommunikációról?

Szűrési feladatok 2 - DNS

- A dns_out.pcapng felhasználásával állomány felhasználásával válaszolja meg az alábbi kérdéseket:
  1. Hány domén név feloldást kezdeményeztek a szűrés alapján? Mely domén nevek voltak ezek?
  2. Válaszon ki 3 darab különböző domén nevet, és keresse meg a válasz csomagokat hozzájuk? Hány darab válasz van az egyes kérésekre? (Segítség: *ID.*)
  3. Hány olyan névfeloldás volt, amelyre több válasz is érkezett?
  4. Volt-e iteratív lekérdezés a szűrésben? Ha igen, akkor mennyi? Ha nem, akkor mi lehet a magyarázat?

Szűrési feladatok 3 - NEPTUN

- A neptun_out.pcapng felhasználásával állomány felhasználásával válaszolja meg az alábbi kérdéseket:
  1. Milyen oldalakat kértek le a szűrés alapján? Milyen böngészőt használtak hozzá?
  2. Hány darab `SSL/TLS` protokollt használó csomag van? Az elsőn kövesse végig a kommunikációt. Minden működési elvnek megfelelően lezajlott?
  3. Kezdeményezett-e megszakítást a szerver a kommunikáció során?
  4. Kideríthető-e, hogy milyen kommunikáció folyt a szerver és a kliens között? Esetleg megtippelhető-e a használt böngésző típusa?

Socket

Miért is jók a rétegek?

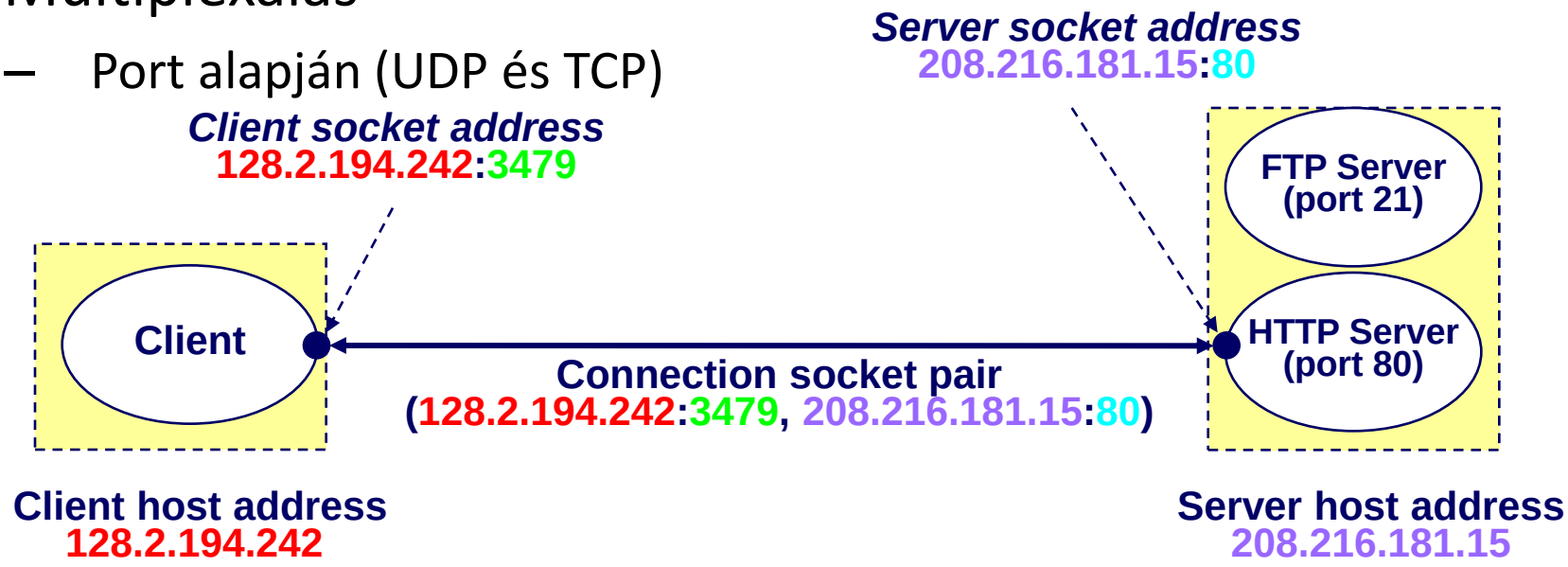
- Ha alkalmazást készítünk, nem akarunk
  - IP csomagok küldésével bajlódni
  - Ethernet keretekkel foglalkozni
  - Implementálni megbízható TCP protokollt
- Az adatunkat rábízzuk az alsóbb rétegre
  - SOCKET: egy API a szállítási réteghez!

Szállítás

- Honnan tudja kinek kell kézbesíteni?
  - Az alsóbb rétegnek szüksége van bizonyos információkra
    - címzés: Hová küldjem?
    - Multiplexálás: Ha megérkezett az adat, akkor melyik processnek továbbítsam???

Cél azonosítása

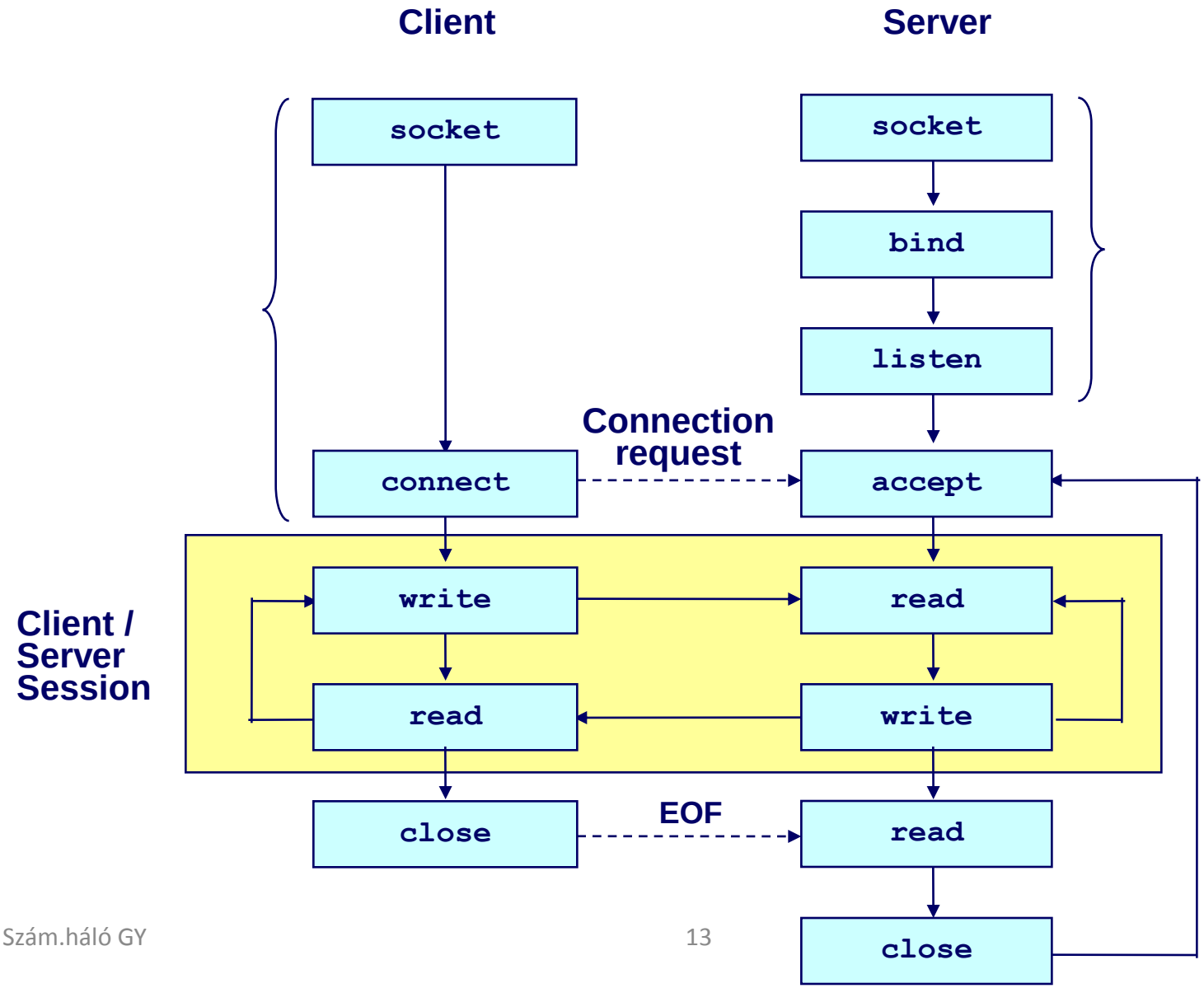
- Címzés
  - IP cím
  - Hostname (IP cím feloldása a DNS segítségével)
- Multiplexálás
  - Port alapján (UDP és TCP)



Socketek

- Socketek használata
  - Socket felkonfigurálása
    - Mi a cél gép? (IP cím, hostname)
    - Mely alkalmazásnak szól az üzenet? (port)
  - Adatküldés
    - Hasonlóan a UNIX fájl írás-olvasáshoz
    - send -- write
    - recv -- read
  - Socket lezárása

Áttekintés


Client
Server
socket
socket
bind
listen
Connection request
connect
accept
Client / Server Session
write
read
read
write
close
EOF
read
close

1 – Socket leíró beállítása

- **Mind a kliens, mind a szerver oldalon**
  - *int socket(int domain, int type, int protocol);*
- *domain*
  - AF_INET -- IPv4 (AF_INET6 -- IPv6)
- *type*
  - SOCK_STREAM -- TCP
  - SOCK_DGRAM -- UDP
- *protocol*
  - 0
- TCP példa:
  - *int sock = socket(AF_INET, SOCK_STREAM, 0);*

2 - Bindolás

- **Csak a SZERVERnél kell elvégezni!!!**
  - *int bind(int sock, const struct sockaddr *my_addr, socklen_t addrlen);*
- *sock*
  - A fájl leíró, amit a socket() parancs visszaadott
- *my_addr*
  - struct sockaddr_in használatos IPv4 esetén, amit castolunk (struct sockaddr*)-ra
- addrlen : A my_addr mérete (sizeof valami)

```
struct sockaddr_in {
    short            sin_family;   // e.g. AF_INET
    unsigned short   sin_port;     // e.g. htons(3490)
    struct in_addr   sin_addr;     // see struct in_addr, below
    char             sin_zero[8];  // zero this if you want to
};
struct in_addr {
    unsigned long s_addr;  // load with inet_aton()
};
```

Példa kód | eddig egy szerver:

```
struct sockaddr_in saddr;
int sock;
unsigned short port = 80;

if ( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    // Ha hiba történt
    perror("Error creating socket");
    ...
}

memset(&saddr, '\0', sizeof(saddr));        // kinullázza a struktúrát
saddr.sin_family = AF_INET;                 // ua. mint a socket()-nél
saddr.sin_addr.s_addr = htonl(INADDR_ANY); // helyi cím, amin figyel
saddr.sin_port = htons(port);               // a port, amin figyel

if ( bind(sock, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) {
    // Ha hiba
    perror("Error binding\n");
    ...
}
```

Mi az a htonl() és htons()?

- Bájt sorrend (byte order)
  - A hálózati bájt sorrend big-endian
  - Host esetén bármi lehet: big- vagy little-endian
    - x86 - little-endian
    - SPARC - big-endian
- Konverzió a sorrendek között:
  - *htons(), htonl()*: host -> hálózati short/long
  - *ntohs(), ntohl()*: hálózati -> host short/long
- Mi az, amit konvertálni KELL?
  - címek
  - portok

Példa

00000000 00000000 00000100 00000001

Address	Big-Endian representation of 1025	Little-Endian representation of 1025
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

3 (Szerver) - Listen

- **Eztán a szerver mindent tud ahhoz, hogy figyelje a socketet**
  - *int listen(int sock, int backlog);*
- *sock*
  - Socket leíró, amit a socket() adott vissza
- *backlog*
  - ennyi kapcsolódási igény várakozhat a sorban

- Példa:
  - *listen(sock, 5);*

4 (Szerver) - Accept

- **A szerver elfogadhatja a kezdeményezett kapcsolatokat**
  - *int accept(int sock, struct sockaddr *addr, socklen_t *addrlen)*
- *sock*
  - Mint korábban
- *addr*
  - pointer egy kliens címzési struktúrára (struct sockaddr_in *). Ezt castoljuk (struct sockaddr *)-ra.
  - Ebbe kerülnek a kapcsolódó kliens adatai(cím, port…)
- *addrlen*
  - Pointer az addr struktúra méretét tartalmazó objektumra. Az értékének meg kell egyezni a sizeof(*addr)-vel!!!
- Pl:
  - *int isock=accept(sock, (struct sockaddr_in *) &caddr, &clen);*

Rakjuk össze a szervert

```
int sock, clen, isock;
unsigned short port = 80;
if ( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
   ...
}

memset(&saddr, '\0', sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
saddr.sin_port = htons(port);

if ( bind(sock, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) {
   ...
}

if (listen(sockfd, 5) < 0) {// operációs rendszer utasítása a socket
    figyelésére…
   ...
}

clen = sizeof(caddr);
// egy bejövő kapcsolat elfogadása:
if ( (isock = accept(sock, (struct sockaddr *) &caddr, &clen)) < 0) {
     perror("Error accepting\n");
     ...
}
```

Mi a helyzet a klienssel?

- A kliensnél nincsen bind(), listen() és accept()
- **Ehelyett konnektálnia kell!**
  - *int connect(int sock, const struct sockaddr *saddr, socklen_t addrlen);*
- Pl.
  - *connect(sock, (struct sockaddr *) &saddr, sizeof(saddr));*

Domain Name System (DNS)

- Küldjünk adatot a www.valami.org-ra?
  - Megoldás a DNS: Hostname és IP összerendelések adatbázisa (Azért ennél több!!!)

```
struct hostent {
    char  *h_name;       // hivatalos hostname
    char  **h_aliases;   // alternatív nevek vektora
    int   h_addrtype;    // címzési típus, pl. AF_INET
    int   h_length;      // cím hossza bájtokban, pl. IPv4 esetén 4 bájt
    char  **h_addr_list; // Címek vektora
    char  *h_addr;       // első(dleges) cím, lényegében a h_addr_list[0]
};
```

- hostname -> IP cím
  - *struct hostent *gethostbyname(const char *name);*
- IP cím -> hostname
  - *struct hostent *gethostbyaddr(const char *addr, int len, int type);*

Egy kliens példa

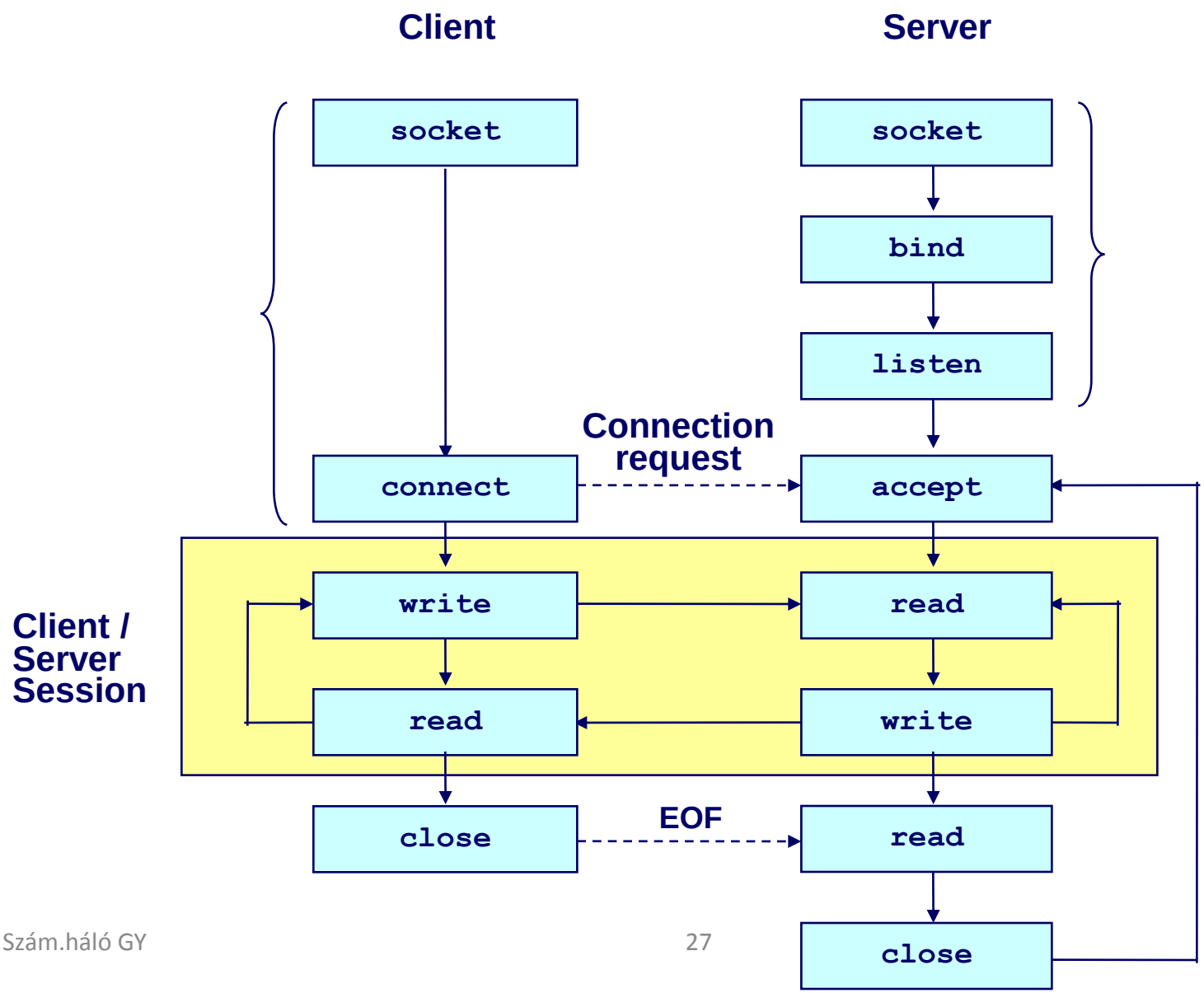
```
struct sockaddr_in saddr;
struct hostent *h;
int sock, connfd;
unsigned short port = 80;
if ( (sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
     ...
}

if ( (h = gethostbyname("www.valami.org")) == NULL) { // Lookup the hostname
    perror("Unknown host\n");
}

memset(&saddr, '\0', sizeof(saddr));          // zero structure out
saddr.sin_family = AF_INET;                   // match the socket() call
memcpy((char *) &saddr.sin_addr.s_addr, h->h_addr_list[0], h->h_length); // copy the
     address
saddr.sin_port = htons(port);                 // specify port to connect to

if ( (connfd = connect(sock, (struct sockaddr *) &saddr, sizeof(saddr)) < 0) { //
     connect!
    perror("Cannot connect\n");
}
```

Ezzel csatlakoztunk

- A szerver elfogadta a kapcsolatot, és a kliens konektált.
- Adat küldése és fogadása
  - *ssize_t read(int fd, void *buf, size_t len);*
  - *ssize_t write(int fd, const void *buf, size_t len);*
- Példa:
  - *read(connsockfd, buffer, sizeof(buffer));*
  - *write(connsockfd, "hey\n", strlen("hey\n"));*

TCP Szegmentálás

- A TCP nem garantálja, hogy az adatokat olyan darabokban továbbítja, ahogy mi azt elküldjük!
  - Meg kell nézni, hogy mit kaptunk a read() végén
    - Az egyik fél elküldi a “Hello\n” sztringet
    - A másik 2 üzenetet kap “He”, “llo\n”
    - Ergo 1 write, 2 read művelet ebben a példában
  - Abban az esetben ha nem egyben kapjuk meg az üzenetet használjunk buffert a read()-hez


Client
Server
socket
socket
bind
listen
Connection request
connect
accept
Client / Server Session
write
read
read
write
close
EOF
read
close

Socket lezárása

- Sose felejts el lezárni a socketet!!! Olyan fontos, mint a fájloknál!!!
  - *int close(int sock);*

- Eztán a szerver új kapcsolatot fogadhat el

Socket I/O: select()

```
int select(int maxfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);

FD_CLR(int fd, fd_set *fds);   /* clear the bit for fd in fds */
FD_ISSET(int fd, fd_set *fds); /* is the bit for fd in fds? */
FD_SET(int fd, fd_set *fds);   /* turn on the bit for fd in fds */
FD_ZERO(fd_set *fds);          /* clear all bits in fds */
```

- **maxfds**: tesztelendő leírók (descriptors) száma
  - (0, 1, ... maxfds-1) leírókat kell tesztelni
- **readfds**: leírók halmaza, melyet figyelünk, hogy érkezik-e adat
  - visszaadja a leírók halmazát, melyek készek az olvasásra (ahol adat van jelen)
  - Ha az input érték **NULL**, ez a feltétel nem érdekel
- **writefds**: leírók halmaza, melyet figyelünk, hogy írható-e
  - visszaadja a leírók halmazát amelyek készek az írásra
- **exceptfds**: leírók halmaza, melyet figyelünk, hogy exception érkezik-e
  - visszaadja a leírók halmazát amelyeken kivétel érkezik

Socket I/O: select()

```
int select(int maxfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);

struct timeval {
      long tv_sec;        /* seconds /
      long tv_usec;             /* microseconds */
}
```

- **timeout**
  - ha **NULL**, várakozzunk addig amíg valamelyik leíró I/O-ra kész
  - különben várakozzunk a **timeout**-ban megadott ideig
    - Ha egyáltalán nem akarunk várni, hozzunk létre egy timeout structure-t, melyben a timer értéke 0

- Több információhoz: man page

Socket I/O: select()

```
int fd, n=0;                          /* original socket */
int newfd[10];                        /* new socket descriptors */
while(1) {
        fd_set readfds;
        FD_ZERO(&readfds); FD_SET(fd, &readfds);

        /* Now use FD_SET to initialize other newfd's
           that have already been returned by accept() */

        select(maxfd+1, &readfds, 0, 0, 0);
        if(FD_ISSET(fd, &readfds)) {
                newfd[n++] = accept(fd, ...);
        }
        /* do the following for each descriptor newfd[i], i=0,…,n-1*/
        if(FD_ISSET(newfd[i], &readfds)) {
                read(newfd[i], buf, sizeof(buf));
                /* process data */
        }
}
```

- Ezután a web-szerver képes többb kapcsolatot kezelni...

Socket alapok

- Segédanyag (magyar):
  - http://ggombos.web.elte.hu/oktatas/SzamHalo/socket/beej_hun.htm
- Windows socket(C++)
  - Dokumentáció:
    - http://msdn.microsoft.com/en-us/library/windows/desktop/ms741394(v=vs.85).aspx
- Linux socket (C, C++)
  - Dokumentáció:
    - http://www.linuxhowtos.org/C_C++/socket.htm
- Fejlesztő környezet: CodeBlocks

CodeBlocks Windows specifikus beállítása

- Fordító:
  - GNU GCC Compiler (Project →Build options)
- Linker beállítása a SOCKET könyvtárgyűjteményhez.
  - (Project →Buildoptions →Linkersettings libws2_32.a hozzáadása a könyvtárakhoz)
  - C:\Program Files\CodeBlocks\MinGW\lib\ libws2_32.a
- Minden C++ állomány, amely SOCKET-et is használ, egy speciális sorral kell kezdődjön:
  - #define_WIN32_WINNT 0x501
- Kapcsolódó include állományok:
  - winsock2.h, ws2tcpip.h

Socket Windows specifikus hívása

- A folyamatnak kérelmeznie kell a WinsockDLL használatát. Fontosabb kapcsolódó hívások:
  - WSAStartup
    - Ez a hívás kezdeményezi a használatát WinsockDLL egy folyamat számára. (*igényelt verzió*, *a SOCKET implementáció részleteihez tartozó mutató*)
  - WSAGetLastError
    - A legutóbb meghiúsult *SOCKET* hívás hiba státuszát adja vissza.
  - WSACleanup
    - A Winsock 2 DLL használatának befejezését jelzi. (ws2_32.dll)

Socket Windows specifikus hívása

```
intiResult= WSAStartup(MAKEWORD(2, 2), &wsaData);

if(iResult!=0) {
   cout<< "WSAStartupfailed: "<<iResult<<endl;
   return 1;
}else{
   cout<<" >> Winsock DLL is ready to use."<<endl;
}
```

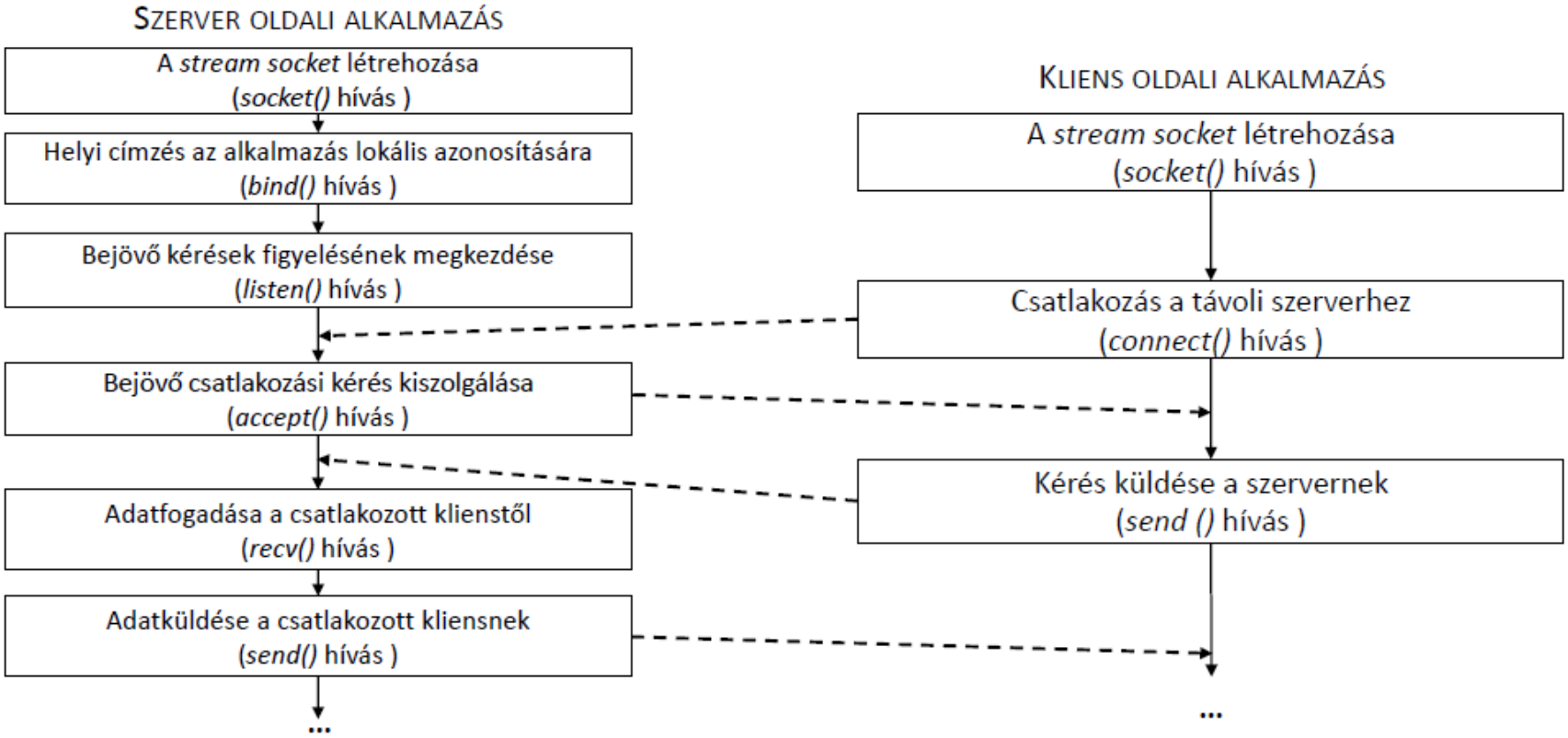
Socket Windows specifikus hívása

```
if(WSACleanup() != 0) {
        dwError=WSAGetLastError();
        if(dwError==WSANOTINITIALISED) {
                cout<<"A successful WSAStartup call must  occur before
                        using this function."<<endl;
                return 1;
        }elseif(dwError==WSAENETDOWN) {
                cout<<"The networksubsystem has failed."<<endl;
                return 1;
        }else{
                cout<<"Function failed with error:„
                        <<dwError<<endl;
                return 1;
        }
}
```

Linux könyvtárak

- netdb.h
  - Hálózati adatbázis műveleteket definícióit tartalmazza. (Például a `hostent` struktúra.)
- sys/types.h
  - Adattípusok definícióit tartalmazza. (Például a `size_t` típus.)
- sys/socket.h
  - A főbb `SOCKET` fejléceket tartalmazza. (Például a `sockaddr` struktúra.)
- arpa/inet.h
  - Internet műveletek definícióit tartalmazza. (Például a `htonl`, `ntohl` vagy az `inet_addr` konverziós függvények.)

TCP


SZERVER OLDALI ALKALMAZÁS
A *stream socket* létrehozása (*socket()* hívás)
Helyi címzés az alkalmazás lokális azonosítására (*bind()* hívás)
Bejövő kérések figyelésének megkezdése (*listen()* hívás)
Bejövő csatlakozási kérés kiszolgálása (*accept()* hívás)
Adatfogadása a csatlakozott klienstől (*recv()* hívás)
Adatküldése a csatlakozott kliensnek (*send()* hívás)
...
KLIENS OLDALI ALKALMAZÁS
A *stream socket* létrehozása (*socket()* hívás)
Csatlakozás a távoli szerverhez (*connect()* hívás)
Kérés küldése a szervernek (*send ()* hívás)
...

Feladat

1. Nézzük meg a honlapon található server működését!
2. Írjunk egy kliens programot ami képes a szerverrel kommunikálni!

Vége