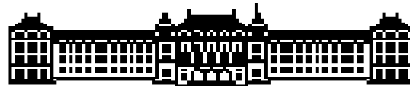
Socketek programozása

Moldován István

Sonkoly Balázs



BME TMIT

Áttekintés

- Socketek általánosan
- Fontosabb rendszerhívások és struktúrák
- Socket programozási technikák
- Példák

Socketek

- cél: IPC (Inter-Process Communication)
- akár hálózaton keresztül
- első implementációk
  - 1983: 4.2BSD Unix, socket API (C)
  - 1989: UC Berkeley - free release of OS and networking library (BSD / Berkeley sockets)
  - POSIX sockets (kisebb módosítások)
  - 1991: Winsock
- socket:
  - API
  - kommunikációs végpont

Socketek

- Az operációs rendszer biztosítja
- Kommunikációt biztosít:
  - Processzek között
  - Különböző platformok között
- Socket típusok:
  - UNIX (IPC gépen belül)
  - INET (IPC távoli gépek között)
  - Infravörös Socketek

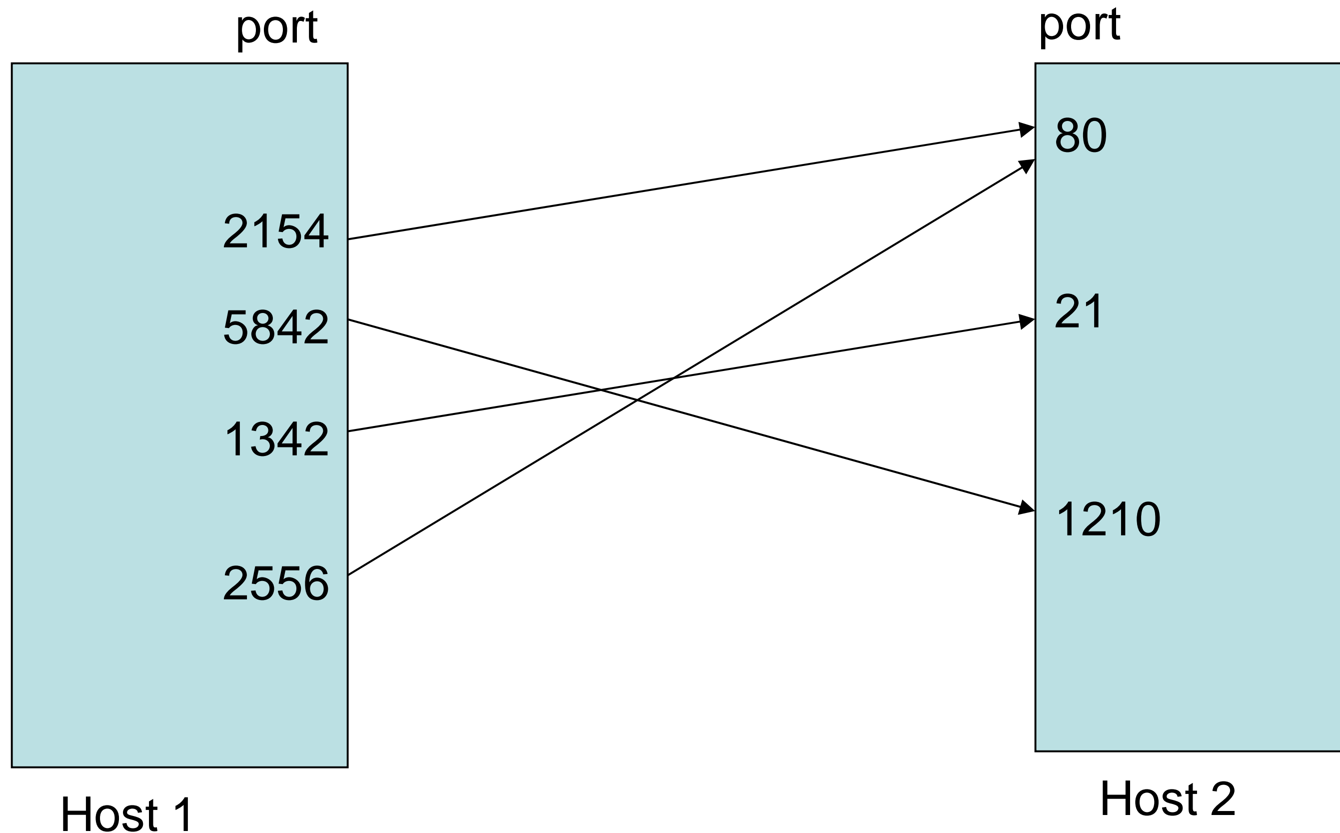
INET socket

- Lehetővé teszi különböző platformokon futó programok kommunikációját
- Többféle INET socket létezik:
  - STREAM (TCP)
  - DGRAM (UDP)
  - RAW (IP)
- Kis eltérésekkel ugyanúgy működik Linuxon mint Windows alatt

Kliens-Szerver modell

- Szerver:
  - Várakozik egy előre ismert porton
  - Nagyszámú egyidejű kapcsolatot kezel
  - Tipikusan beérkező kérésekre válaszol
- Kliens:
  - Csatlakozik a szerver ismert portjára
  - Kéréseket küld és választ vár
- Egy alkalmazás mindkettő lehet egyszerre

Virtuális kapcsolatok



1024-nél kisebb portok – “well known”, ismert portok

Szolgáltatások (services)

- A szolgáltatásokat név szerint ismerjük
  - Pl. ftp, telnet
- A portokat szám szerint tartjuk nyilván
  - Az ftp szolgáltatás pl. a 21-es portot használja
- Az összerendelés egy táblázat alapján
  - /etc/services
- „Well-known portok” – 1024-ig (RFC)
- Az összerendelés lekérdezhető
  - getservbyname(const char *name, const char *proto)

Port menedzsment

- A portokat egy kapcsolat számára megfelelően kell megválasztani
  - Szerver esetében a fogadó port általában adott
  - Kliens számára a lokális port adott (0)
- Néha célszerű a lokális port kiválasztása
  - Pl. ha ugyanazt a portot szeretnénk küldésre fogadásra használni: SO_REUSEADDR
    - UDP esetében különösen hasznos lehet
- Egy TCP port csak akkor használható újra ha a kapcsolat CLOSED állapotba kerül (vagy SO_REUSEADDR)

Big- és Little Endian

- Különböző rendszerek – különböző számábrázolás (**Host Byte Order**)
  - Little Endian – Intel (alacsony helyiérték előbb)
  - Big Endian – Sun Microsystems stb. (magas helyiérték előbb)
- Hálózati szabvány: (**Network Byte Order**)
  - Big Endian
- Konverziós függvények:
  - network-to-host (beérkező csomagokra)
  - host-to-network (kiküldött csomagokra)
  - ntohs(), ntohl()
  - htons(), htonl()

Áttekintés

- Socketek általánosan
- Fontosabb rendszerhívások és struktúrák
- Socket programozási technikák
- Példák

Socket

- UNIX – a socket egy “file descriptor”, FD
  - Egy egész számként ábrázolva
  - UNIX: „minden egy fájl” filozófia
- Windows – külön típus – SOCKET
  - Tulajdonképpen itt is integer
- Típus:
  - AF_ -address family
  - PF_ -protocol family
  - Jelenleg ugyanaz a két kategória
  - (eredeti gondolat: egy AF akár különböző protokollokat is használhat... nem jött be...)

Fontosabb struktúrák

- **struct sockaddr** – Internet címek kezelésére szolgál
  - Protokoll- és socket típustól független

```
struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol address
};
```

- **struct sockaddr_in** – INET socket
  - sockaddr struktúrára kasztolható (kell is)

```
struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

Fontosabb struktúrák

- **struct addrinfo** – újabb típus cím információk tárolására

```
struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname

    struct addrinfo *ai_next;      // linked list, next node
};
```

Névfeloldás

- **gethostbyname**
  - Név feloldása, IP cím stringre nem működik
- **gethostbyaddr**
  - Csak IP cím string feloldására
- HOSTENT struktúrát adnak vissza
  - Tartalma:
    - Hivatalos domain nevek (FQDN) listája
    - IP címek listája
- majd a szükséges struktúrák feltöltése kézzel
- újabb lehetőség: **getaddrinfo**

Címkezelési segédfüggvények

- inet_addr()
  - Egy pontokkal elválasztott IP címet alakít intre
- inet_aton()
  - "ascii to network", hasonló az előzőhöz, jobb hibakezeléssel
- inet_ntoa()
  - "network to ascii", előállítja az IP cím stringet

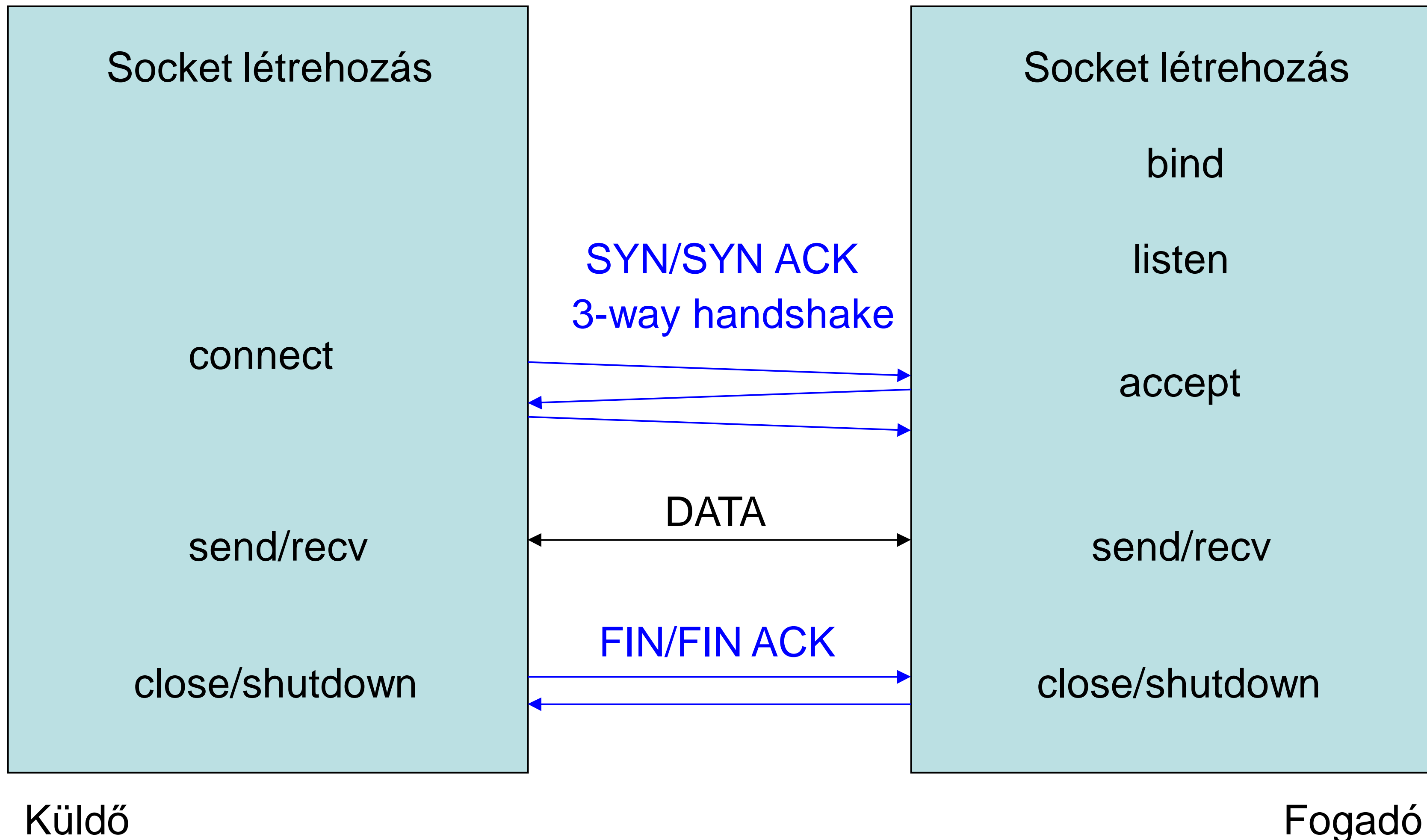
Socket létrehozása

- **SOCKET socket (int** *af*, **int** *type*, **int** *protocol* **)**
- Af:
  - **AF_INET**, AF_IPX, AF_ATM, AF_IRDA
- Típus:
  - SOCK_STREAM,SOCK_DGRAM,SOCK_RAW
- Protokoll:
  - IPPROTO_IP, IPPROTO_RAW
  - IPPROTO_ICMP,IPPROTO_UDP,IPPROTO_TCP
  - Általában nem specifikáljuk (0), az OS kitalálja
  - (adott PF-en belül egy típushoz általában egy protokoll)

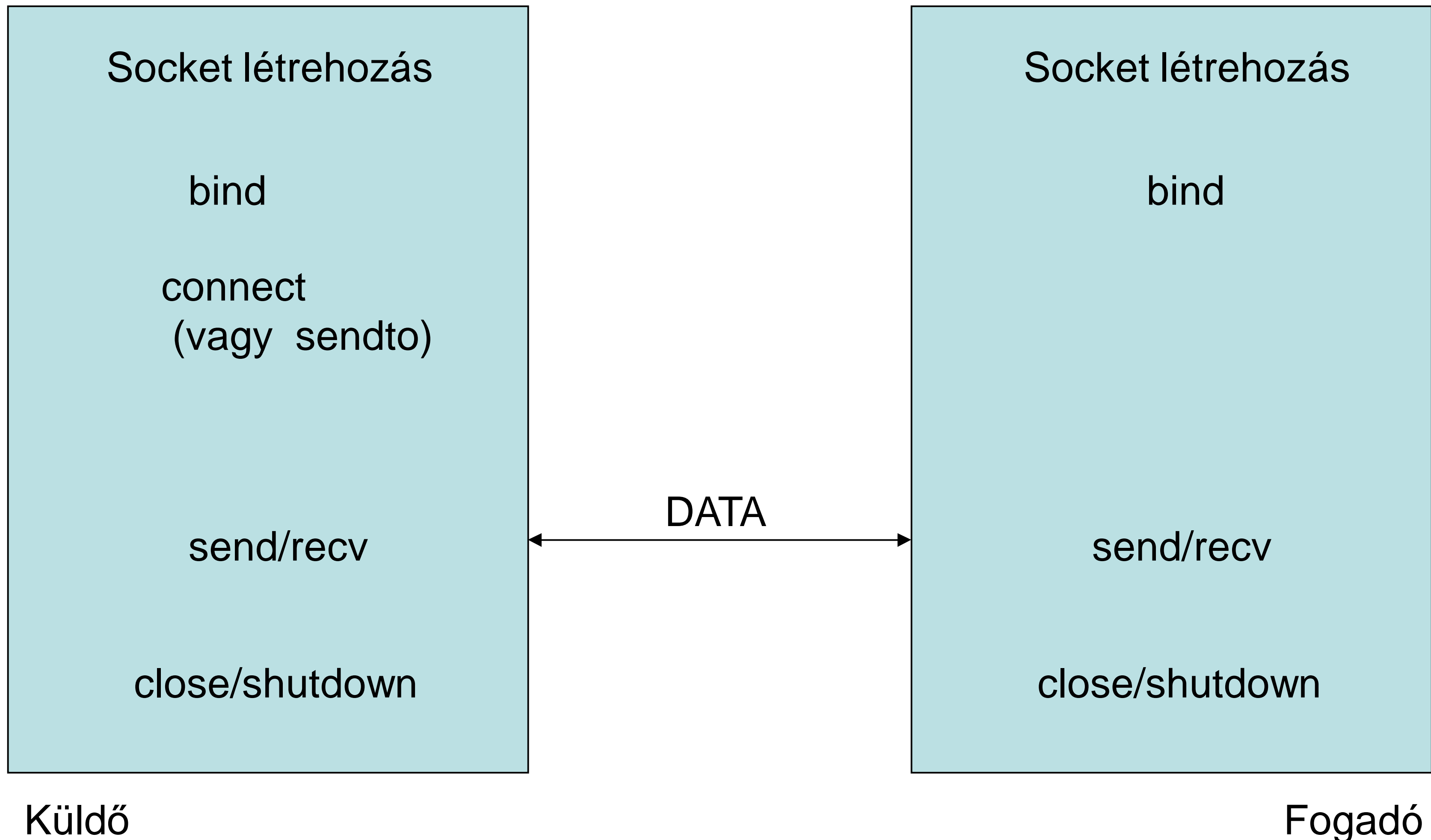
Socket létrehozása

```
struct addrinfo hints, *res;
int sockfd;

// first, load up address structs with getaddrinfo():

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;     // AF_INET, AF_INET6, or AF_UNSPEC
hints.ai_socktype = SOCK_STREAM; // SOCK_STREAM or SOCK_DGRAM

getaddrinfo("www.example.com", "3490", &hints, &res);

// make a socket using the information gleaned from getaddrinfo():
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
```

A kapcsolat - TCP



A kapcsolat - UDP



Aktív illetve passzív kezdés

- Passzív Open
  - Kiválasztja a lokális cím/port párt
  - Hozzárendeli a TCP stack-ben és várja az érkező kapcsolatokat
  - Nincs még hálózati forgalom
- Aktív Open
  - Kapcsolatot kezdeményez egy távoli gép IP/port párosa felé
  - Hálózati forgalommal jár

Bind

- Lokálisan hozzárendeli a cím/port párost a socket struktúrához
  - általában fogadó oldalon szükséges
  - Küldőnél automatikusan érdemes, vagy ha több IP címmel rendelkezik
  - Default értékek:
    - INADDR_ANY – bármely cím (0.0.0.0)
    - 0 port – automatikus portválasztás
- **int bind (SOCKET** *s*, **const struct sockaddr FAR** * *name*, **int** *namelen***);**

Connect

- Létrehoz egy kapcsolatot a távoli hoszt felé
  - UDP: Hozzárendeli a távoli címet
  - TCP: létrehozza a kapcsolatot
    - A túloldalnak fogadni kell
    - Időbe telik - blokkolhat
- **int connect (SOCKET** *s*, **const struct sockaddr FAR*** *name*, **int** *namelen* **);**

Listen

- Az adott socketet várakozó állapotba helyezi
  - csak TCP-vel használatos
  - A socketet először bind utasítással egy lokális porthoz kötjük
- **int listen (SOCKET *s*, int *backlog*);**
  - A backlog a várakozó kapcsolatok számát adja meg (egy kapcsolat addig várakozik míg accept-tel nem fogadjuk)

Accept

- Fogadja a várakozó TCP kapcsolatot
  - Létrehoz egy sockaddr struktúrát
  - Hozzárendeli a TCP kapcsolatot
  - child socket
  - Az eredeti “listening” socket megmarad
  - Blokkoló függvény
- **SOCKET accept (SOCKET** ***s*****, struct sockaddr FAR*** ***addr*****, int FAR*** ***addrlen*** **);**

Adat küldése

- **int send (SOCKET** *s*, **const char FAR *** *buf*, **int** *len*, **int** *flags* **);**
  - Az s socketen a buf tartalmából próbál elküldeni len bájtot
  - Nem feltétlenül sikerül egyből elküldeni, ilyenkor a visszaadott érték < len
  - Flag: MSG_OOB – TCP esetén sürgős adat küldése (urgent bit)
- DGRAM soket esetén a max küldhető adat korlátozott

Adat fogadása

- **int recv (SOCKET** *s*, **char FAR*** *buf*, **int** *len*, **int** *flags* **);**
  - Az s socketről olvas max. len bájtnyi adatot a buf bufferbe
  - Ha nincs adat – blokkol
  - Flag: MSG_PEEK, MSG_OOB
- Visszaadott érték: a beolvasott bájtok száma, vagy 0 – a kapcsolat befejezésekor

SendTo/RecvFrom

- Csak összekapcsolatlan socketek esetén
  - SendTo: adott cím/portra küld adatot
    - STREAM esetén a to mező figyelmen kívül marad
  - RecvFrom: fogadja az adatot, és visszaadja a küldő címét is
- A hívás után a socket kapcsolódik
- Broadcast üzenet küldésére is alkalmas

Socket bezárása

- Close() vagy closesocket () [Windows]
  - Stream socket esetében blokkoló
- A másik oldalon stream socket esetében 0 adat olvasásával jelentkezik
- DGRAM socket esetén a másik oldal számára nincs jelzés

Blokkoló/nem blokkoló mód

- Küldés/fogadás esetén sok esetben nem megengedett a befejezés megvárása
  - Az alapértelmezett a blokkoló
  - a nem-blokkoló üzemmód kiválasztható
  - A hívás után EWOULDBLOCK hibával tér vissza
- Nem-blokkoló módban az állapot select() függvénnyel kérhető le

Blokkolás engedélyezés/tiltás

- **int ioctl (SOCKET** *s*, **long** *cmd*, **u_long FAR*** *argp* **);**
  - Cmd:
    - FIONBIO
    - FIONREAD
  - Argp:
    - Pointer az paraméterre
- Példa blocking I/O beállításra:

```
unsigned long n=1;
ioctl(socket, FIONBIO, &n);
```

Socket opciók

- Protokoll és socket specifikus értékek beállítására/lekérdezésére szolgál
- Fontosabb opciók:
  - SO_BROADCAST
  - SO_RCVBUF
  - SO_SNDBUF
  - SO_REUSEADDR
  - TCP_NODELAY

setsockopt()

- **int setsockopt (SOCKET** *s*, **int** *level*, **int** *optname*, **const char FAR *** *optval*, **int** *optlen* **);**
  - Level: SOL_SOCKET vagy IPPROTO_TCP
  - Optname: opció
  - Az értékek az opciótól függőek
- Általában a kapcsolódás előtt kell meghívni

getsockopt()

- Socket beállítások lekérdezése
- Használata hasonló
- Fontosabb lekérdezhető paraméterek
  (a beállíthatókon kívül)
  - SO_ERROR
  - SO_MAX_MSG_SIZE
  - SO_TYPE
  - SO_PROTOCOL_INFO vagy TCP_MAXSEG

Socket opciók 1

- SO_RCVBUF és SO_SNDBUF
  - A küldő és fogadó buffer méretét állítja
  - (semmi köze a TCP ablakokhoz!!!)
- SO_REUSEADDR – socket cím újrahasznosítás
  - Pl.ha ugyanazon a porton küldünk/fogadunk
  - Gyors egymásután nyitunk/zárunk egy kapcsolatot

Socket opciók 2

- SO_BROADCAST
  - Engedélyezi a broadcast üzenetek küldését
- SO_LINGER kapcsolat zárásra vonatkozik
  - Beállítható hogy mennyi ideig várjon adatot zárás után
  - close előtt állítjuk
- SO_SNDTIMEO és SO_RCVTIMEO
  - **send**, **sendto**, **recv**, és **recvfrom** esetén meghatároz egy timeoutot
  - A select()-nél optimálisabb bizonyos esetekben

IP opciók

- IP_TTL
  - A TTL mező értékének állítása
- IP_TOS
  - A TOS mező értékének állítása
- IP_OPTIONS
  - IP opciók küldése. A támogatott opciókat ellenőrizni kell.

TCP opciók

- Maximális sávszélesség eléréséhez
  - Bandwidth – delay product:

    ADV.Window => bandwidth * delay
  - A SO_RCVBUF / SO_SNDBUF használata nélkül nem optimális
- TCP Nagle algoritmus tiltása
  - TCP_NODELAY – interaktív kapcsolatoknál

UDP opciók

- SO_MAX_MSG_SIZE
  - A DGRAM socketek nem küldhetnek akármekkora adattömböket
  - A maximális csomagméret a hálózati rétegtől függ
  - A maximális csomagméret meghatározása fontos
- UDP_NOCHECKSUM
  - Ne számoljon UDP checksumot
  - UDP checksum = 0

Hibakezelés

- A függvények általában -1 értéket (SOCKET_ERROR) adnak vissza hiba esetén
- A tényleges hiba lekérdezhető az *errno* megvizsgálásával
  - Windows: WSAGetLastError
- Az EWOULDBLOCK nem hiba - lekezelendő

Eltérések Unix/Windows között

- Az alapvető függvények gyakorlatilag egyeznek – a Windows socket implementáció BSD alapú
- A Windows a blokkoló/nem blokkoló módokon kívül lehetőséget ad az ún. Overlapped módra
  - Speciális függvényhívások
  - Jelzések – Üzenetek, PL: ON_READ

Eltérések a függvénynevekben

- int - SOCKET
- errno – WSAGetLastError()
- ioctl() – ioctlsocket()
- close() – closesocket()

- A legtöbb eltérés néhány #define direktívával megoldható

Debug lehetőségek

- TCPDump vagy Wireshark
  - A teljes kommunikáció vizsgálata
- Netstat parancs
  - Az adott gépen megmutatja az összes kapcsolatot
- TCP socketek – telnet!
  - Kapcsolat létrehozás tesztelés
  - telnet hoszt port

Áttekintés

- Socketek általánosan
- Fontosabb rendszerhívások és struktúrák
- Socket programozási technikák
- Példák

Programozási Technikák

- Blokkoló
  - Egyszerű, nem javasolt
  - multithreaded
    - Minden kapcsolat egy szál
- Nem blokkoló
  - Polling alapú
  - Jelzés alapú (notification)
  - Aszinkron –AIO interfészt igényel
  - Call-back alapú
    - Csak néhány oprendszer támogatja
    - Windows, X-Motif

Blokkoló modell

- A szerver figyel egy adott porton
  - Új kapcsolat esetén fork()-ol
  - Az új szál accept()-tel fogadja a kapcsolatot
- Egyszerűen implementálható
- Nehéz megoldani az időzítést
  - (timeout vagy hiba esetén blokkol)

Polling modell

- Nem blokkoló módban használjuk
- Periodikusan ellenőrizzük hogy befejeződött-e a művelet
- CPU pazarló lehet

- Előny:
  - Kihasználhatjuk az időt másra
- Hátrány
  - Blokkolhatja a főprogramot

- Megjegyzés:
  - Pollozhatunk select()-tel is ha 0 a várakozási idő

Polling modell

- A hálózat kezelés egy (végtelen) ciklus
  - A polling függvényt hívjuk az állapotok lekérdezésére
    - Egyidejűleg több socket ellenőrzése
  - Átadja a vezérlést az OS számára ameddig
    - Hálózati esemény - vagy -
    - timeout - vagy -
    - Megszakítás érkezik
- A függvény kiválasztja azon leírókat, melyek:
  - küldhetnek, adat érkezett vagy jelzés érkezett (hiba vagy OOB)

Polling módszerek

- select()
  - Minden implementációban elérhető
  - limitálva FD_SETSIZE handle-re (beépítve)
- poll()
  - Minden Unix implementációban elérhető
  - nincs limit, de nagyon lelassul nagyszámú FD kezelésével
    - A keresés sok FD között időt igényel
    - Különböző technikákkal gyorsítható
- /dev/poll – Solaris, epoll() – Linux
- kqueue() – BSD

A select lekérdezés

- A select() függvény egyszerre több socket figyelését végezheti (igazából általános FD)
- Működése:
  - Paraméterei
    - a figyelendő socketek listája (file descriptorok)
    - Várakozási idő
  - A select() a megadott időre átadja a vezérlést az op. rendszernek (sleep), vagy valamely esemény hatására azonnal visszatér
- Visszaadja azon socketek listáját ahol valami esemény történt

A select() módszere

- Egy szálon fut, végtelen ciklusban
  - FD_SET halmazok létrehozása
  - select() hívása (nincs aktív loop), visszatér, ha:
    - megszakították a várakozást (INTERRUPT)
    - adat érkezett
    - letelt az idő (timeout)
  - A visszaadott FD_SET-ek ellenőrzése
    - Ha volt adatforgalom akkor a kezelő függvények hívása
    - Ha bezárultak kapcsolatok az FD_SET módosítása
  - A bemenő FD_SET visszaállítása
- Egyetlen loop kezel kiépítés-bontást, küldés-fogadást

Előnyök:

- Nem blokkol a főprogram
- Jól skálázható
- vezérelhető a timeout

Hátrányok:

- Komplexebb megoldás
- Egyenként ellenőrizendő az összes kapcsolat

FD_ makrók

- A socket leírók listájának kezelését könnyítik meg
- fd_set változó – egy socket listát határoz meg
- Makrók:
  - FD_CLR(s, *set)
  - FD_SET(s, *set)
  - FD_ZERO(*set)
  - FD_ISSET(s, *set)

select()

- **int select (int** *nfds*, **fd_set FAR** * *readfds*, **fd_set FAR** * *writefds*, **fd_set FAR** * *exceptfds*, **const struct timeval FAR** * *timeout* **);**
  - nfds: a legnagyobb socket fd +1
  - read, write, except fd_set-ek ki-bemenő paraméterek
  - Timeout – a várakozási idő.
- Ha a select() rendben visszatér, FD_ISSET makróval ellenőrizhetjük hogy mely socketen történt változás

epoll()

- A Linux 2.6 verziótól elérhető
- 3 új rendszerhívást valósít meg:
  - int *epoll_create*(int maxfds);
    - Létrehozza az epoll set-et és hozzárendeli egy FD-hez
  - int *epoll_ctl*(int epfd, int op, int fd, unsigned int events);
    - FD-k regisztrációja az FD set-be
  - int *epoll_wait*(int epfd, struct pollfd *events, int maxevents, int timeout);
    - Lekérdezés indítása: hasonló a select-hez

Áttekintés

- Socketek általánosan
- Fontosabb rendszerhívások és struktúrák
- Socket programozási technikák
- Példák

Példa: szerver

```
int sockfd, new_fd, sin_size;
struct sockaddr_in my_addr, their_addr;

sockfd = socket(AF_INET, SOCK_STREAM, 0));

memset(&my_addr, 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(8888);
my_addr.sin_addr.s_addr = INADDR_ANY;

bind(sockfd,
     (struct sockaddr *)&my_addr,
     sizeof(struct sockaddr));

listen(sockfd, 5);

while(true){
  sin_size = sizeof(struct sockaddr_in);
  new_fd = accept(sockfd,
                  (struct sockaddr *) &their_addr,
                  &sin_size);

  send(new_fd, "Hello, world!\n", 14, 0);

  close(new_fd);
}

close(sockfd);
```

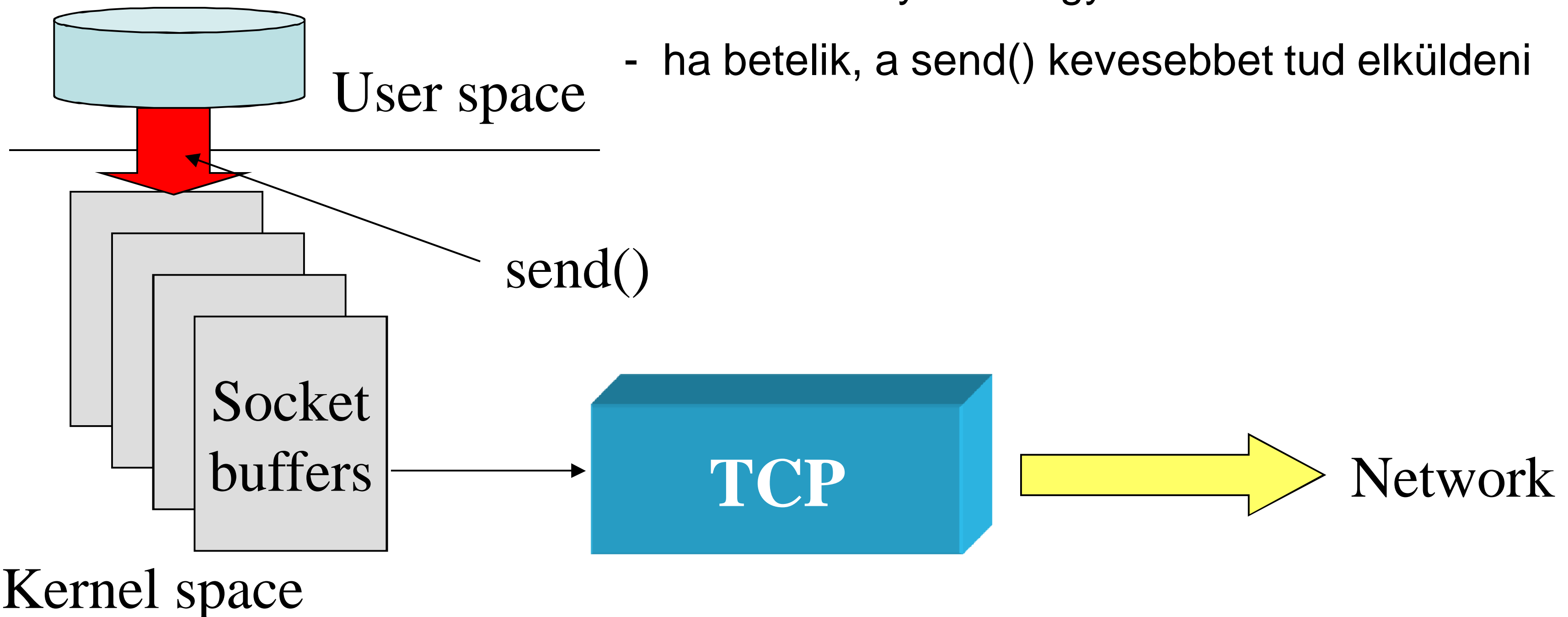
Példa: kliens

```
int sockfd,numbytes;
struct sockaddr_in my_addr, their_addr;

sockfd = socket(AF_INET, SOCK_STREAM, 0));

memset(&their_addr, 0, sizeof(their_addr));
their_addr.sin_family = AF_INET;
their_addr.sin_port = htons(8888);
their_addr.sin_addr.s_addr = inet_addr("192.168.1.2");

connect(sockfd,
        (struct sockaddr *) &their_addr,
        sizeof(struct sockaddr));
// now the socket is connected!
char buf[255];

numbytes=recv(sockfd, buf, 254, 0));

buf[numbytes] = '\0';
printf("Received: %s",buf);
close(sockfd);
```

Parciális send()

- A send() függvény STREAM socket esetén nem feltétlen küldi el az összes adatot
  - Általában kisebb mint a TCP ablak
  - fordított irányban is így van
  - ha betelik, a send() kevesebbet tud elküldeni

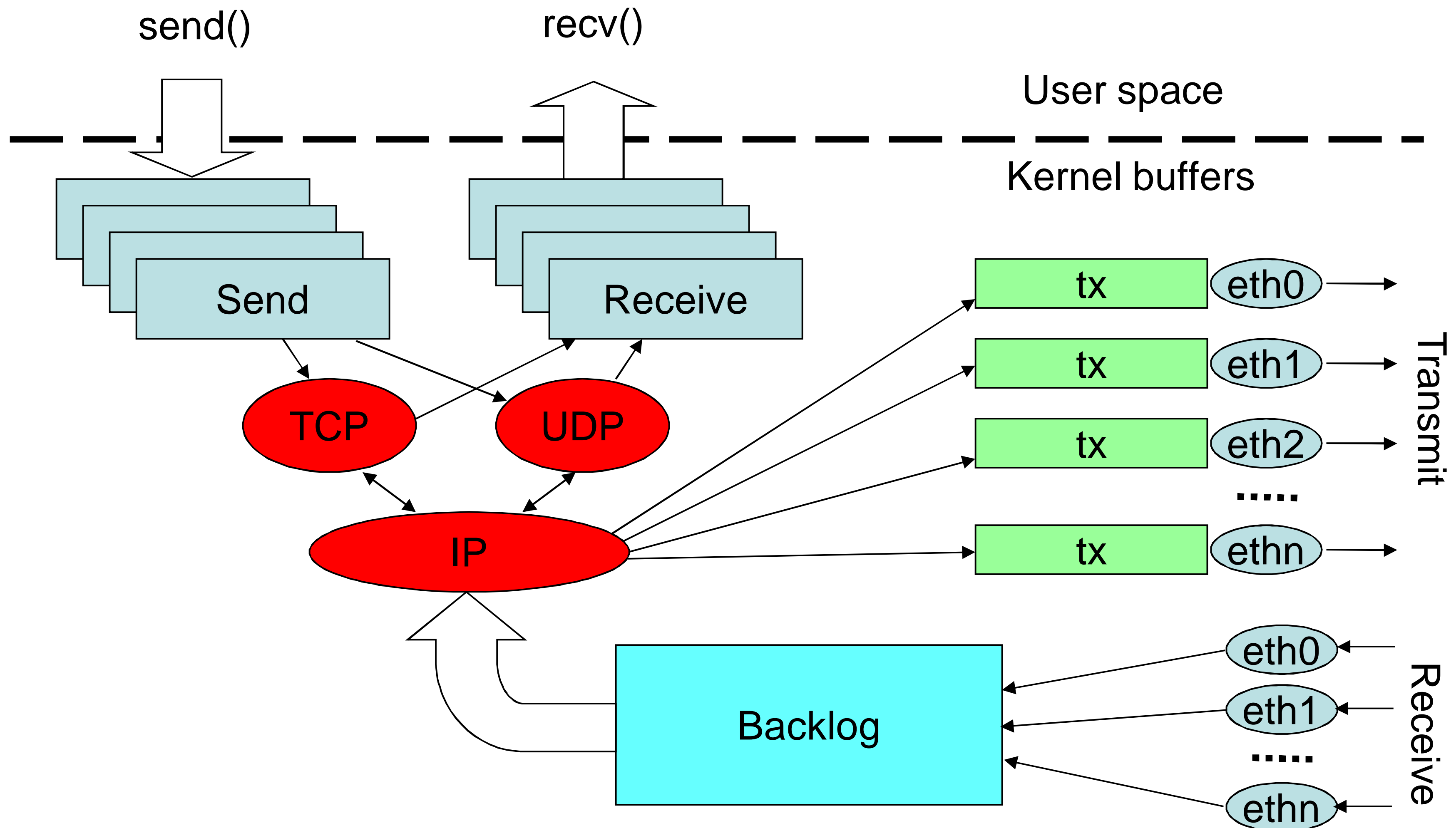
Application buffer



Parciális recv()

- A recv() függvény szintén adhat vissza kevesebbet mint amennyi adat van
  - UDP-re kifejezetten érvényes, pl. 2 datagram csak két readdel olvasható ki
- A gyors adatátvitel érdekében a fogadó buffert üríteni kell

Linux Bufferek



Adatátvitel - példa

- Minden adat elküldése egy pufferből:

```
int total = 0; // hány bájtot küldtünk el
int bytesleft = *len; // mennyi maradt még
int n;

while(total < *len) {
  n = send(s,
           buf+total,
           bytesleft,
           0);

  if (n == -1)
    break;

  total += n;
  bytesleft -= n;
}
```

Olvasnivaló

- http://beej.us/guide/bgnet/