

TCP/IP SOCKET INTERFACE PROGRAMOZÁSA

Kommunikációs hálózatok I. (BMEVIHAB01)

Dr. Lencse Gábor
tudományos főmunkatárs
BME Hálózati Rendszerek és Szolgáltatások Tanszék
lencse@hit.bme.hu



- Socket: kommunikációs végpont
- Socket és fájl formális hasonlósága és különbsége
 - Kezelésük részben hasonló: megnyit, lezár, ír, olvas
 - Feladatuk más:
 - Fájl: adattárolás (helyben)
 - Socket: információcsere egyik végpontja
- Két megfelelő módon egymáshoz rendelt socket segítségével kétirányú kommunikáció valósítható meg
- TCP/IP socket interfész: a TCP/IP protocol stack programozási felülete
- Számos nyelvhez létezik: pl. C, C++, java, perl, stb.
- A C nyelvet fogjuk megismerni, elsősorban Linux alatt.

A legfontosabb lépések

- Socket létrehozása: **socket**
- Helyi IP-cím és portszám megadása: **bind**
 - kliens socketnél nem szükséges
- Kapcsolat orientált esetben
 - a kapcsolat létrehozása: **listen**, **accept**, **connect**
 - kommunikáció: **write** és **read** vagy **send** és **recv**
- Kapcsolatmentes esetben
 - kommunikáció: **sendto** és **recvfrom**
- Socket lezárása: **close**

```
int socket(int domain, int type,  
int protocol);
```

- **domain**: protokollcsalád (lásd következő fólia)
- **type**: kommunikáció jellege, például:
 - **SOCK_STREAM**: megbízható kétirányú kapcsolat, megvalósítás: TCP-vel
 - **SOCK_DGRAM**: datagram szolgálat (kapcsolatmentes, nem megbízható), megvalósítás: UDP-vel
 - **SOCK_RAW**: hozzáférés a hálózati szintű protokollhoz
- **protocol**: milyen protokoll valósítja meg (tipikusan 1 van)
- **return**: socket descriptor: socketet azonosító egész szám (mint file descriptor), hiba esetén: -1

Példák protokollcsaládra

- **AF_INET**: IPv4 protokollcsalád
- **AF_INET6**: IPv6 protokollcsalád
- **AF_UNIX**, **AF_LOCAL**: helyi kommunikáció

Nevezéktan: *Protocol Family* más néven *Address Family*

- Régebbi BSD rendszerekben **PF_** előtagok
- Modern BSD és Linux rendszerekben **AF_** előtagok
- Érdekesség: `/usr/include/bits/socket.h`:
`#define AF_INET PF_INET`

```
int bind(int sockfd,  
        struct sockaddr *my_addr,  
        socklen_t addrlen);
```

- **sockfd**: socket descriptor
- **my_addr**: mutató egy **sockaddr** típusú címstruktúrára, ami a beállítani kívánt helyi IP-címet és portszámot tartalmazza. Típusa **AF_INET** esetén **sockaddr_in**, **AF_INET6** esetén pedig **sockaddr_in6**. (Ld. köv. fóliák)
A függvény hívásakor type cast: **(struct sockaddr *)**
- **addrlen**: megadja a címstruktúra méretét bájtokban
- **return**: hiba esetén -1, egyébként 0



IPv4: struct sockaddr_in

```
struct sockaddr_in {
    sa_family_t  sin_family; /*address family: AF_INET*/
    u_int16_t    sin_port; /*port in network byte order*/
    struct in_addr sin_addr; /* IPv4 address */
    /* Pad to size of 'struct sockaddr'. */
};

/* IPv4 address: */
struct in_addr {
    u_int32_t s_addr; /*address in network byte order*/
};
```

Megjegyzés: hálózati bájtrend: MSB: Most Significant Byte first, big-endian. A legnagyobb helyi értékű bájt van elől.

De a számítógép bájtrendje eltérő lehet! Vannak konverziós függvények, lásd később.



IPv6: struct sockaddr_in6

```
struct sockaddr_in6 {
    u_int16_t sin6_family; /* address family: AF_INET6 */
    u_int16_t sin6_port; /* port in network byte order */
    u_int32_t sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr; /* IPv6 address */
    u_int32_t sin6_scope_id; /* Scope ID */
};

/* IPv6 address: */
struct in6_addr {
    unsigned char s6_addr[16]; /* addr. in netw. b. o.*/
};
```


Kapcsolat orientált eset

- A kapcsolat létrehozása során a szerver és a kliens szerepe aszimmetrikus
 - A szerver csatlakozásra vár, bárki csatlakozhat hozzá
`listen`, `accept`
 - A kliens dönti el, hogy melyik szerverhez csatlakozik
`connect`
- A létrehozott kapcsolaton keresztül a kommunikáció a „másik féllel” történik, nem kell megadni, hogy kivel
`write` , `read` , (`send` , `recv` , sőt van még `writen` , `readv` is)

```
int listen(int sockfd, int  
backlog);
```

- **sockfd**: socket descriptor
- **backlog**: hagyományosan a létrehozás alatt álló kapcsolatok (TCP 3 utas kézfogás) számának felső korlátja volt. Ha ezt kimerítették, a szerver a további kapcsolatokat visszautasította. A 2.2-es Linux kernelben ennek jelentését megváltoztatták, a már létrejött, de még **accept()**-tel el nem fogadott kapcsolatok maximális számát jelenti. (man 2 listen)
- **return**: hiba esetén -1, egyébként 0

Megjegyzés: ez egy nem blokkoló függvényhívás.

```
int accept(int sockfd,  
          struct sockaddr *addr,  
          socklen_t *addrlen);
```

- **sockfd**: socket descriptor
- **addr**: mutató egy címstruktúrára. Ide kerül be a kapcsolódó fél IP-címe és portszáma.
- **addrlen**: mutató egy kétirányú paraméterre: híváskor megadja a rendelkezésre álló hely méretét, visszatéréskor pedig a kapott cím adatok tényleges méretét bájtokban.
- **return**: siker esetén a létrejött kapcsolat eléréséhez a socket descriptor, hiba esetén -1

Megjegyzés: ez egy blokkoló függvényhívás.

```
int connect(int sockfd, const
            struct sockaddr *serv_addr,
            socklen_t addrlen);
```

- **sockfd**: socket descriptor
- **serv_addr**: mutató egy **sockaddr** típusú címstruktúrára, ami a szerver IP-címét és portszámát tartalmazza. Ez **AF_INET** esetén **sockaddr_in** típusú, **AF_INET6** esetén pedig **sockaddr_in6**, mint a **bind()**-nál is láttuk.
- **addrlen**: megadja a címstruktúra méretét bájtokban
- **return**: ha sikerült a kapcsolódás, akkor 0, hiba esetén pedig -1

Megjegyzés: ez is blokkoló függvényhívás.

```
ssize_t write(int fd,  
              const void *buf, size_t count);
```

- **fd**: socket descriptor
- **buf**: mutató arra a memóriaterületre, ahol az adatok találhatóak. Ezek, úgy ahogy vannak, átvitelre kerülnek.
- **count**: az átvinni kívánt bájtok száma
- **return**: hiba esetén -1, egyébként az elküldött bájtok száma.

Megjegyzések:

- Azonos az (alacsony szintű) fájlba írással.
- A két félnek összhangban kell használnia a **write()** és a **read()** függvényeket!

```
ssize_t read(int fd,  
             void *buf, size_t count);
```

- **fd**: socket descriptor
- **buf**: mutató arra a memóriaterületre, ahova az adatok kerülnek.
- **count**: a fogadni kívánt bájtok száma
- **return**: hiba esetén -1, egyébként a fogadott bájtok száma.

Megjegyzések:

- Szintén az (alacsony szintű) fájlból való olvasás függvénye.
- A két félnek összhangban kell használnia a **write()** és a **read()** függvényeket!

Kapcsolatmentes eset

- Minden egyes küldésnél megmondjuk, hogy kinek szeretnék datagramot küldeni: `sendto()`
- Minden egyes vételnél megtudjuk, hogy kitől kaptuk a datagramot: `recvfrom()`

```
ssize_t sendto(int sockfd, const
void *buf, size_t len, int flags,
const struct sockaddr *dest_addr,
socklen_t addrlen);
```

- **sockfd**: socket descriptor
- **buf**: mutató arra a memóriaterületre, ahol az adatok találhatóak
- **len**: az átvinni kívánt bájtok száma
- **flags**: megadható számos opció (lásd: man 2 sendto), ha nem kívánunk élvitellel, írhatunk egyszerűen 0-t.
- **dest_addr**: a korábban megismert struktúra a cél megadására
- **addrlen**: a fenti struktúra mérete
- **return**: hiba esetén -1, egyébként az elküldött bájtok száma.


```
ssize_t recvfrom(int sockfd,  
void *buf, size_t len, int flags,  
struct sockaddr *src_addr,  
socklen_t *addrlen);
```

- **sockfd**: socket descriptor
- **buf**: mutató arra a memóriaterületre, ahova az adatok kerülnek
- **len**: a fogadni kívánt bájtok száma
- **flags**: megadható számos opció, hasonlóan, mint **sendto()**
- **src_addr**: a korábban megismert struktúra a forrás címadatainak fogadására (NULL: nem kérjük az információt)
- **addrlen**: a fenti struktúra mérete – mindkét irányban!
- **return**: hiba esetén -1, egyébként a fogadott bájtok száma.

```
ssize_t recvfrom( ... );
```

- Alap esetben blokkoló függvényhívás.
- Nem blokkoló mód engedélyezése **MSG_DONTWAIT** flag használatával. Kezelés: `man 2 recvfrom`
- És lehetőség van arra is, hogy adott idő letelte után tovább ne várjon. Ehhez előre be kell állítani a timeout értékét az `int setsockopt(...)` függvény használatával.

```
int setsockopt(int sockfd,  
int level, int optname,  
const void *optval,  
socklen_t optlen);
```

- `sockfd`: socket descriptor
- `level`: az opció beállítás szintje. Mi most socket API szintjén szeretnénk, ezért: `SOL_SOCKET`.
- `optname`: az opció neve, most: `SO_RCVTIMEO`.
- `optval`: az opció értéke. Ehhez `struct timeval` kell.
- `optlen`: az opció értékét megadó struktúra mérete, itt: `sizeof(struct timeval)`
- `return`: hiba esetén -1, különben 0.

```
int close(int fd) ;
```

- **sockfd**: socket descriptor
- **return**: hiba esetén -1, egyébként 0.

- Mind a **socket ()** függvényhívással megnyitott, mind az **accept ()** -tel „kapott” socketeket ezzel a függvénnyel kell lezárni.

- FONTOS, hogy a függvények visszatérési értékét mindig ellenőrizzük!
- Hibakezelésre használhatjuk a jól ismert **errno** változót, amit a rendszerhívások és bizonyos könyvtári függvények állítanak.
 - Ennek értékét összehasonlíthatjuk a jól ismert POSIX szimbolikus nevekkel, pl. **EAGAIN** vagy **EWOULDBLOCK**.
 - Hibaüzenet kiírására használhatjuk az alábbi függvényt:
void perror(const char *s) ;
 - Ez függvény is az **errno** változót használja
 - Argumentumként annak a függvénynek a nevét szokták neki megadni, ami a hibát okozta.

- Alapvető 16 és 32 bites bájt sorrend konverziós függvények.

Névkonvenció: <honnan>to<hova><hossz>: {h|n}to{n|h}{s|l}
h=host, n=network; s=short, l=long (az adott helyen értve)

```
uint32_t htonl (uint32_t hostlong) ;  
uint16_t htons (uint16_t hostshort) ;  
uint32_t ntohl (uint32_t netlong) ;  
uint16_t ntohs (uint16_t netshort) ;
```

Bővebben: man 3 byteorder

IP-cím konverziók (csak IPv4)

```
in_addr_t inet_addr(const char *cp);
```

A `cp` stringben megadott IPv4 címet adja vissza binárisan, hálózati bájtrendben. Problémát okozhat, hogy a -1 értéket használja hiba jelzésére, de ez megegyezik az érvényes 255.255.255.225 érték ábrázolásával. Helyette:

```
int inet_aton(const char *cp,  
             struct in_addr *inp);
```

Az eredménynek külön van helye (`inp`), és az érvénytelen inputot a 0 visszatérési értékkel jelzi. A másik irány:

```
char *inet_ntoa(struct in_addr in);
```

Bináris, hálózati bájtrendben megadott IPv4 címet szövegesen ad vissza. FIGYELEM: statikus puffert használ, tehát ki kell másolni, mert legközelebb felülírja!

IP-cím konverziók (IPv4 és IPv6)

```
int inet_pton(int af,  
             const char *src, void *dst);
```

- **af**: címcsalád: **AF_INET**: IPv4, **AF_INET6**: IPv6.
- **src**: szabványos formában megadott IPv4 vagy IPv6-cím.
- **dst**: eredmény: az IP-cím binárisan, hálózati bájtsorrendben.
- **return**: FIGYELEM, kissé szokatlan!
 - Ha a konverzió sikeres, akkor: 1.
 - Ha nem talált a megadott címcsaládnak megfelelő érvényes IP-címet, akkor: 0.
 - Ha **af** nem felel meg egy érvényes címcsaládnak sem, akkor: -1, és az **errno** értéke: **EAFNOSUPPORT**.

IP-cím konverziók (IPv4 és IPv6)

```
const char *inet_ntop(int af,  
    const void *src, char *dst,  
    socklen_t size);
```

- **af**: címcsalád: **AF_INET**: IPv4, **AF_INET6**: IPv6.
- **src**: az **af** címcsaládbeli IP-cím binárisan
- **dst**: puffer az eredmény számára
- **size**: a hívó fél adja meg benne a puffer méretét
- **return**: FIGYELEM, kissé szokatlan!
 - Ha nincs hiba: nem NULL pointer **dst**-re mutat
 - Hiba esetén NULL pointer és az **errno** értéke adja meg a hibát **EAFNOSUPPORT** (érvénytelen címcsalád) vagy **ENOSPC** (nem fér el a cím a pufferben).

Ha valaki programozni szeretne...

```
int gethostname(char *name, size_t len);
```

A saját gépünk nevének lekérdezése a `name` stringbe, ami `len` hosszú.

Példaprogramokban előfordulnak:

```
gethostbyname(); gethostbyaddr();
```

De ne használjuk, mert elavultak (obsolete)!

Helyette:

```
getaddrinfo(); getnameinfo();
```

Kicsit bonyolultak, de ott a man! 😊

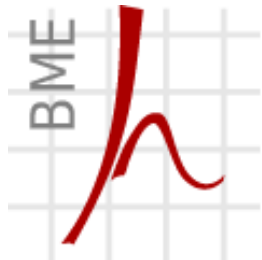
Cserében reentránsak, IPv4-hez és IPv6-hoz is használhatók.

Példaprogramok: <http://whale.hit.bme.hu/socket>

- Megismertük:
 - A socket interface programozás lépéseit
 - kapcsolatorientált esetben
 - kapcsolatmentes esetben
 - Az alapvető socket kezelő függvényeket
 - A konverziós függvényeket
 - Több bájtos adatok helyi és hálózati bájtsorrendje
 - IP-címek (ASCII és bináris formátum között)
 - A további szükséges függvényeket

Kérdések?

KÖSZÖNÖM A FIGYELMET!



Hálózati Rendszerek és
Szolgáltatások Tanszék

Dr. Lencse Gábor
tudományos főmunkatárs
BME Hálózati Rendszerek és Szolgáltatások Tanszék
lencse@hit.bme.hu

