Számítógépes hálózatok

7. gyakorlat: SOCKET programozás (C)

Gyakorlatvezető: Ács Zoltán

Hatodik házi-feladat 1/2

ADAPTÍV FA

Tekintsünk `16` állomást, melyek adaptív fabejárás protokollal visznek át csomagokat. Az állomások azonosítói `0,...,15`. Szimulálja a protokoll működését, ha az állomások `1,2,3,4,6,11,12,13` egy időben akarnak csomagot átvinni! (Adja meg a verseny `slot`-okat ettől az időpillanattól addig, amíg a protokoll feloldja a `kollízió`t.)

Hatodik házi-feladat 2/2

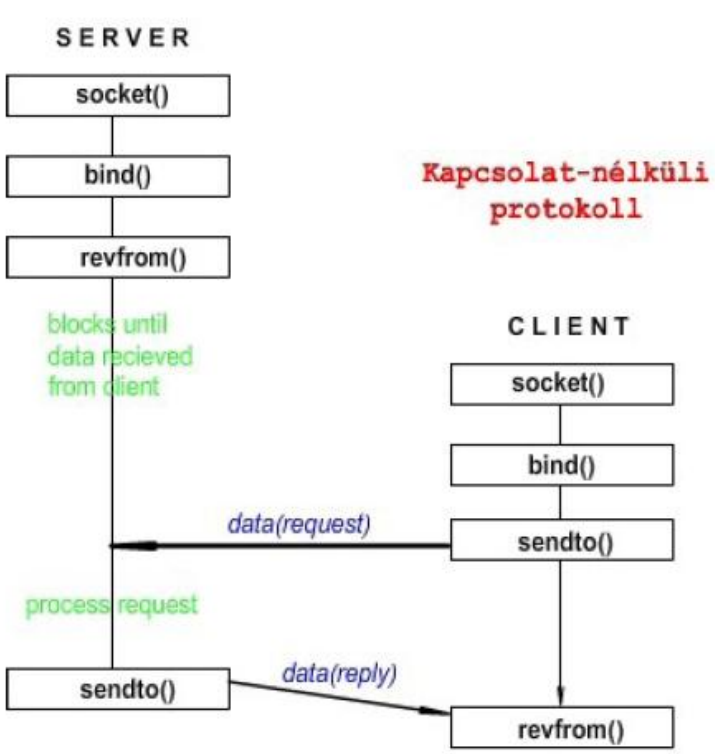
ETHERNET

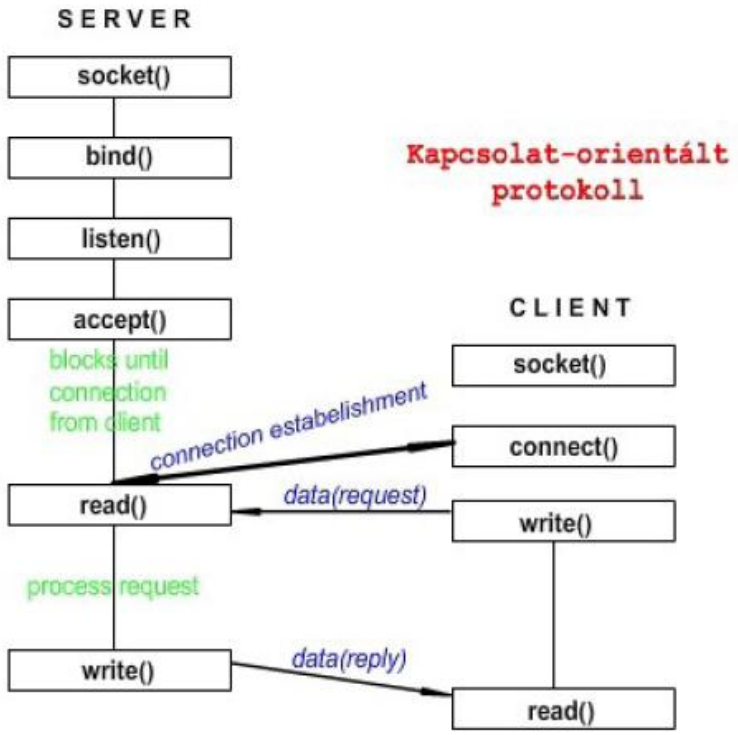
Tekintsünk egy `1 Gbps`-os `Ethernet` hálózatot, ahol a minimális keretméret `512` bájt. Mekkora lehet két eszköz között a maximális kábelhossz?

$$\frac{512 * 8}{2 * 10^9} = d_{MAX\_PROP}$$

$$l_{MAX} = 1{,}8 * 10^8 * \frac{512 * 8}{2 * 10^9} = 368{,}64 \text{ m}$$

Alapok – egyszerű modell





Alapok – szükséges adatok

	protocol	local_addr, local_process	foreign_addr, foreign_process
Kapcsolat-orientált szerver	socket()	bind()	accept()
Kapcsolat-orientált kliens	socket()	connect()	
Kapcsolat-nélküli szerver	socket()	bind()	recvfrom()
Kapcsolat-nélküli kliens	socket()	bind()	sendto()

Alapok - struktúra

```
struct sockaddr {
    unsigned short sa_family;
    char           sa_data[14];
};

struct sockaddr_in {
    short          sin_family;
    u_short        sin_port;
    struct in_addr sin_addr ;
    char           sin_zero[8];
}
```

Alapok – fontosabb hívások 1/6

SOCKET létrehozása (`sys/socket.h`)

SZINTAXIS → **`int socket(int domain, int type, int protocol)`**

A hívás hatására kapunk egy fájl leírót a socket-hez. Ha -1-es értéket kapunk vissza, akkor hiba történt a létrehozás során.

SOCKET neve

SZINTAXIS → **`int getsockname(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);`**

A socket nevét adja meg, amelyet az *address struktúrában tárol le. A cím hosszát* az address_len argumentumban rögzíti. Alapvetően 0-ával kell visszatérjen, hiba esetén -1 a visszatérési értéke.

SOCKET opciók

SZINTAXIS → **`int setsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);`**

SOCKET tulajdonság beállítása. SOCKET level: `SOL_SOCKET, IPPROTO_TCP, …` (`netinet/in.h`). Option_name: `SO_REUSEADDR, SO_REUSEADDR, SO_BROADCAST,…`

Alapok – fontosabb hívások 2/6

`Host` nevének meghatározása

SZINTAXIS → **`int gethostname(char *name, size_t namelen)`**

Az aktuális `host` nevét adja meg. Egy `host` neve maximum `255` karakter hosszú lehet. Hiba esetén `-1`-et ad vissza egyébként `0`-át.

`Host` adatainak meghatározása

SZINTAXIS → **`struct hostent *gethostbyname(const char *name)`**

Információkat ad meg az adott `host`ról. (`#include <netdb.h>`) A `hostent` struktúra:

- **`char *h_name`** – *név*
- **`char **h_aliases`** – *alternatív nevek*
- **`int h_addrtype`** – *a cím típusa*
- **`int h_length`** – *a cím bájt mérete*
- **`char **h_addr_list`** – *a címek vektora*
- **`char *h_addr`** – *a `h_addr_list[0]` szinonimája*

Példa - gethostbyname

```
struct hostent *hp =
   gethostbyname(”pandora.inf.elte.hu”);

memcpy( (void *) &pandora.sin_addr,
   (void *) hp->h_addr, hp->h_length );
```

Alapok - fontosabb hívások 3/6

SOCKET címzés

SZINTAXIS → `int bind(int socket, const struct sockaddr *address, socklen_t address_len);`

A lokális cím még nem elegendő a `socket` használatához, ezért a használat előtt `bind` rendszerhívással nevesíthetjük a `socket`-et. Hiba esetén `-1`-gyel tér vissza.

Adatok fogadása (UDP)

SZINTAXIS → `ssize_t recvfrom(int socket, void *buffer, size_t length, int flags,struct sockaddr *address, socklen_t *address_len);`

Adatok fogadása. `UDP` kapcsolatok esetén szokták használni. A fogadott bájtok számával tér vissza, ha hiba volt, akkor pedig `-1`-gyel. Az `address` *értéket is kitölti* adat fogadása után.

Adatok küldése (UDP)

SZINTAXIS → `ssize_t sendto(int socket, void *buffer, size_t length, int flags,struct sockaddr *dest_addr, socklen_t dest_len);`

Adatok küldése. `UDP` kapcsolatok esetén szokták használni. Az elküldött bájtok számával tér vissza, ha hiba volt, akkor pedig `-1`-gyel.

Alapok - fontosabb hívások 4/6

Bejövő kapcsolatok figyelése

SZINTAXIS → `int listen( int socket, int qlength)`

A `listen` rendszer hívás a kapcsolat-orientált szerverek esetén azt fejezi ki, hogy a kiszolgáló hajlandó a kapcsolódási kérések fogadására. Ez a SOCKET-et passzív állapotba helyezi. Lehet többszörös egyidejű kérés. (`OS`).

Kapcsolatok fogadása

SZINTAXIS → `newsockfd = accept(sockfd,peer,addrlen)`

Az `accept` alkalmas a beérkező kérések fogadására. A kapcsolódási kérésig ez blokkolt marad. Az `addrlen` egy egész értékre mutató pointer. A rendszer létrehoz egy új SOCKET-et és visszatér ennek a leírójával. (mind az öt komponens kitöltve, három örökölt)

Kapcsolatok kezdeményezése

SZINTAXIS → `int connect(int sockfd, struct sockaddr_in *servaddr, int addrlen);`

A kliens folyamat egy SOCKET leírón át kapcsolódik a szerverhez történő kapcsolódási hívás hatására (`connect`). Egy kapcsolat-orientált kliens esetén a -szerver oldalról is – elfogadott kapcsolódás mind a négy cím és folyamat komponenst kitölti.

Alapok – fontosabb hívások 5/6

Adatfogadása (TCP)

SZINTAXIS → `int recv(int s, char *buf, int len, int flags);`

Üzenet küldésére alkalmas hívás. Sikertelen fogadás esetén -1-et ad vissza. A fogadott bájtok számát adja vissza.

Adatok küldése (TCP)

SZINTAXIS → `int send(int s, const char *msg, int len, int flags);`

Nem garantált a megérkezés, illetve a -1 visszatérési érték csak a lokális hibákat jelzi. Az elküldött bájtok számát adja vissza.

SOCKET lezárása

SZINTAXIS → `void close(int socket);`

A SOCKET lezárása. Ha végeztünk akkor le kell zárni a fájlleírót.

Alapok – fontosabb hívások 6/6

Egyéb hívások

SZINTAXIS → `inet_ntoa(struct in_addr *in*), *inet_addr(const char *cp* ) (#include <arpa/inet.h>);`

SZINTAXIS → `atoi(const char *str ); (#include <stdlib.h>)`

SZINTAXIS → `htons(uint16_t *hostshort*),*ntohl(uint32_t hostlong*), ntohs(uint16_t *hostshort*), *htonl(uint32_t netlong* )`

`(#include <arpa/inet.h>)`

`SELECT` rendszerhívás

SZINTAXIS → `int select(int *nfds*, fd_set *readfds*, fd_set *writefds*, fd_set *errorfds*, struct timeval *timeout*);`

`void FD_CLR(int *fd*, fd_set *fdset*), int FD_ISSET(int *fd*, fd_set *fdset*), void FD_SET(int *fd*, fd_set *fdset*), void FD_ZERO(fd_set *fdset*)`

Feladatok

UDP HELLO WORLD

Készítsünk el egy egyszerű kliens illetve szerver alkalmazást. A kliens küldjön egy „`Hello server`” üzenetet a szervernek majd várjon a szerver válaszára végül lépjen ki. A kliens a szerver adatait parancssori argumentumként kérje be. A szerver fogadjon üzeneteket, és reagáljon is rájuk egy „`Hello <ip>:<port>`” válasz üzenettel.

TCP HELLO WORLD

Az előző feladatot írjuk meg `TCP` alapú kommunikációra is.

Hasznos linkek

- Fordítás `gcc`-vel
- http://www.cprogramming.com/function.html
- http://people.inf.elte.hu/acszolta/halozatok/ske/udp.c
- http://people.inf.elte.hu/acszolta/halozatok/ske/tcp.c

7. Gyakorlat vége

Köszönöm a figyelmet.