Sockets API: header datoteke

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
```

Sockets API: tipovi podataka

```
struct in_addr {
    unsigned long s_addr;
};
```
. . . struktura za IP-adresu zapisanu u binarnom obliku

```
struct hostent {
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_length;
    char **h_addr_list;
};
```
. . . struktura za konverziju host-name-a u binarni IP i obratno

- `char *h_name;` — službeni host-name računala (string)
- `char **h_aliases;` — polje alternativnih host-nameova (polje stringova, zadnji je `NULL`)
- `int h_addrtype;` — tip adrese, kod nas je uvijek `AF_INET`
- `int h_length;` — duljina adrese u byteovima, kod nas uvijek 4
- `char **h_addr_list;` — polje binarnih zapisa IP-adresa računala (zadnja je `NULL`; svaki element treba *cast* u `struct in_addr *`)

```
#define h_addr h_addr_list[0]
```
prvi binarni zapis u polju `h_addr_list`

Sockets API: funkcije za konverziju adresa

```
int inet_aton( const char *dekadskiIP, struct in_addr *binarniIP );
```
. . . prima dekadskiIP, konvertira ga u binarni tako da napuni binarniIP. Ako ne uspije, vrati 0; inače vrati ne-nulu.

```
char *inet_ntoa( struct in_addr binarniIP );
```
. . . prima binarnu IP-adresu, vraća string koji sadrži dekadsku (ne alocira memoriju već koristi statičko spremište unutar funkcije za dekadsku adresu!). Vraća `NULL` ako je došlo do greške.

```
struct hostent *gethostbyname( const char *hostName );
```
. . . prima string u kojem je host-name, vraća popunjenu `hostent` strukturu (ne alocira memoriju već koristi statičko spremište unutar funkcije za `hostent`!). Vraća `NULL` ako je došlo do greške.

```
struct hostent *gethostbyaddr( const char *binarniIP, int duljina, int tipAdrese );
```
. . . prima binarnu IP adresu tipa `struct in_addr` (treba cast u `const char *`), duljina je uvijek `sizeof( binarniIP )`, tipAdrese je uvijek `AF_INET`. Vraća popunjenu `hostent` strukturu (ne alocira memoriju već koristi statičko spremište unutar funkcije za `hostent`!). Vraća `NULL` ako je došlo do greške.

Sockets API: strukture i funkcije za klijent/server

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

. . . struktura za definiranje porta i IP-adrese servera
postavljamo na `AF_INET`
port na serveru na koji se spajamo
binarna IP-adresa servera (server kod `bind` može ovo postaviti na `INADDR_ANY`)
polje od 8 znakova koje sve treba postaviti na `'\0'`

```
int socket( int domena, int tip, int protokol );
```

. . . stvara novu utičnicu. domena je `PF_INET`, tip je `SOCK_STREAM` za TCP-protokol, a `SOCK_DGRAM` za UDP-protokol, protokol postavljamo na 0. Vraća -1 ako nije uspjelo, inače novu utičnicu.

```
int bind( int sock, struct sockaddr *servAddr, socklen_t addrlen );
```

. . . specificira na kojem će portu i kojoj IP-adresi komunicirati utičnica sock. sock je utičnica napravljena sa socket. servAddr je napunjena `struct sockaddr_in` struktura (treba *cast* na `struct sockaddr`). addrlen postavljamo na `sizeof( servAddr )`. Vraća -1 ako nije uspjelo, 0 inače.

```
int connect( int sock, struct sockaddr *servAddr, int lenAddr );
```

. . . ostvaruje konekciju između klijenta i udaljenog računala. sock je utičnica napravljena sa socket, servAddr popunjena `struct sockaddr_in` struktura (treba *cast* u `struct sockaddr *`), lenAddr postavljamo na `sizeof( servAddr )`. Vraća -1 ako nije uspjelo, 0 inače.

```
int listen( int sock, int maxKonekcija );
```

. . . rezervira utičnicu koja će serveru služiti za osluškivanje nadolazećeg kontakta na danom portu. sock je utičnica napravljena sa socket i povezana sa portom i IP-adresom sa bind. maxKonekcija je maksimalan broj klijenata koji može odjednom čekati na uslugu servera. Vraća -1 ako nije uspjelo, 0 inače.

```
int accept( int listenerSock, struct sockaddr *klijentAddr, unsigned int *lenAddr );
```

. . . prihvaća prijedlog konekcije od strane klijenta. Daje klijentovu IP-adresu i novu utičnicu za komunikaciju. listenerSock je *listener* utičnica na kojoj su pozvani socket, bind i listen. Funkcija popunjava klijentAddr podacima o klijentu. lenAddr trebamo postaviti na adresu `unsigned int`-a u kojem piše `sizeof( klijentAddr )`. Vraća -1 ako nije uspjelo, inače novu utičnicu preko koje možemo komunicirati sa klijentom.

```
ssize_t recv( int sock, void *buffer, size_t duljinaBuffera, int opcije );
```

. . . prima podatke od udaljenog računala. sock je utičnica napravljena sa socket i povezana sa connect ili dobivena od accept. buffer je mjesto u memoriji na koje se spremaju dobiveni podaci, duljinaBuffera veličina spremišta u byte-ovima. opcije postavljamo na 0. Vraća broj primljenih byte-ova, 0 ako je druga strana prekinula komunikaciju, -1 ako je došlo do greške.

```
ssize_t send( int sock, void *buffer, size_t duljinaBuffera, int opcije );
```

. . . šalje podatke sa lokalnog računala na udaljeno. sock je utičnica napravljena sa socket i povezana sa connect ili dobivena od accept. buffer su podaci koji se šalju, duljinaBuffera veličina u byte-ovima podataka. opcije postavljamo na 0. Vraća broj poslanih byte-ova, 0 ako je druga strana prekinula komunikaciju, -1 ako je došlo do greške.

```
int close( int sock );
```

. . . zatvara utičnicu sock tako da više nije moguća komunikacija preko nje. Vraća -1 ako nije uspjelo, 0 inače.

Višedretveni programi: biblioteka `pthreads`

```
#include <pthread.h>
```

```
int pthread_create( pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg );
```
. . . stvara novu dretvu čiji ID sprema u `thread`, `attr` uvijek postavljamo na `NULL`, pokreće u novoj dretvi funkciju `start_routine` sa parametrom `arg`. Vraća 0 ako je uspjelo.

```
int pthread_join( pthread_t thread, void **value_ptr );
```
. . . čeka da sa izvođenjem završi dretva čiji ID je jednak `thread`; povratna vrijednost iz dretve sprema se u `value_ptr`. Vraća 0 ako je uspjelo.

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```
. . . deklaracija i inicijalizacija mutexa `mutex`

```
int pthread_mutex_lock( pthread_mutex_t *mutex );
```
. . . "zaključava" dio koda tako da mu pristup ima samo jedna dretva; "lokot" pripada mutexu `mutex`

```
int pthread_mutex_unlock( pthread_mutex_t *mutex );
```
. . . "otključava" dio koda koji je ranije bio "zaključan" pomoću mutexa `mutex`