#include <sys/socket.h>

PF_... → protocol family
AF_... → address format

SOCK_STREAM
Działa poprzez połączenie ze zdalnym urządzeniem i niezawodnie przesyła dane w postaci bajtów

SOCK_DGRAM
Dane mogą nie dotrzeć do odbiorcy. Pakiety wysyłane są indywidualnie

SOCK_RAW
Zapewnia dostęp do niskopoziomowych protokołów i interfejsów

Struct sockaddr → typ danych, który przedstawia adres typu socket

Pola:
short int sa_family → kod formatu adresu
char sa_data[14] → „prawdziwe" dane adresu

Internet namespace - Address Format
AF_LOCAL → wyznacza format adresu dla lokalnej przestrzeni nazw
AF_UNIX → to co wyżej, tylko zwiększona kompatybilność
AF_FILE → kolejny synonim kompatybilności
AF_INET → wyznacza format adresu, który pochodzi z internetowej przestrzeni nazw
AF_INET6 → to co wyżej, tyle że dla IPv6
AF_UNSPEC → w sumie nie wyznacza formatu adresu

int bind(int socket, struct sockaddr* addr, socklen_t n)

przypisuje adres obiektowi typu socket. Gdy udało zwraca 0, w przeciwnym wypadku -1

---

* Pomijam lokalne przestrzenie nazw

Internet namespace -> protocols

int PF_INET -> wyznacza IPv4 internet namespace i jest kojarzony z rodziną protokołów

int PF_INET6 -> -11- IPv6 -11-

#include <netinet/in.h>

struct sockaddr_in -> reprezentuje adresy socket w internetowej przestrzeni nazw

Pola:
- sa_family_t sin_family -> tu trzymamy AF_INET
- struct in_addr sin_addr -> adres IP hosta
- unsigned short int sin_port -> numer portu

Gdy wywołujemy bind lub getsockname powinniśmy użyć sizeof(struct sockaddr_in)

Struct in_addr -> używane do przetrzymywania adresu hosta

Pola:
- uint32_t s_addr

include <arpa/inet.h>

int inet_aton(const char* name, struct in_addr* addr)

konwertuje IPv4 address name z standardowego "192.168.1.1" do binarnego rozwiązania i trzyma struct in_addr

uint32_t inet_addr(const char* name)

to co wyżej bez "trzymania"

uint32_t inet_network(const char* name)

rozpakowuje numer sieci z nazwy adresu

char* inet_ntoa(struct in_addr addr)

konwertuje adres hosta do "stringa"

---

include <netdb.h>

struct hostent -> typ danych używany do reprezentowania wejście w bazie danych hosta

Pola:

char* h_name -> nazwa hosta

int h_addrtype -> typ adresu hosta

int h_length -> długość adresu

char** h_addr_list -> vector adresów danych

struct hostent* gethostbyname(const char* name)

zwraca informacje o nazwie hosta

struct hostent* gethostbyaddr(const char* addr, size_t length, int format)

zwraca informacje o hoście z jego adresem

void sethostent(int stayopen)

otwiera baze danych hosts i rozpoczyna skanowanie

void endhostent(void)

zamyka bazę danych hosts

Port Numbers

int IP_PORT_RESERVED

numery mniejsze od tego są zarezerwowane dla roota.

Gdy tworzymy połączenie poprzez socket, dane w sin_port oraz sin_addr muszą być reprezentowane w bajtach sieciowych.

Gdy używamy gethostbyname lub inet_addr, możemy od razu dane skopiować do struktury sock_addr_in, w innym przypadku korzystamy z htons lub htohs.

uint16_t htons(uint16_t hostshort)
uint16_t ntons(uint16_t netshort)
uint32_t htonl(uint32_t hostlong)
uint32_t ntohl(uint32_t netlong)

- Tworzenie socketa

  int socket(int namespace, int style, int protocol)

- Zamykanie

  int shutdown(int socket, int how)

- Parowanie

  int socketpair(int namespace, int style, int protocol, int filedes[2])

- Połączenie

  int connect(int socket, struct sockaddr* addr, socklen_t length)

- Nasłuchiwanie

  int listen(int socket, unsigned int n)

- Akceptowanie

  int accept(int socket, struct sockaddr* addr, socklen_t* length)

- Kto się połączył?

  int getpeername(int socket, struct addr* addr, socklen_t* length)

- Wysyłanie danych

  int send(int socket, void* buffer, size_t size, int flags)

- Odbieranie

  int recv(int socket, void* buffer, size_t size, int flags)

SOCKET DATA OPTIONS

Argumenty "flags"

int MSG_OBB
int MSG_PEEK
int MSG_DONTROUTE

- Wysyłanie datagramów

int sendto (int socket, void* buffer, size_t size, int flags,
struct sockaddr* addr, socklen_t length)

- odbieranie datagramów

int recvfrom (/* tak samo jak wyżej */)