**Wojskowa Akademia Techniczna**

**Wydział Elektroniki**

**Instytut Telekomunikacji**

**Studia II°**

**Projektowanie aplikacji sieciowych**

Materiały pomocnicze do zajęć

Cz.3.

**dr inż. Jarosław Krygier**

Warszawa 2017

Spis treści

[1 Gniazdo datagramowe – serwer UDP 3](#_Toc496605776)

[2 Gniazdo datagramowe – klient UDP 6](#_Toc496605777)

# Gniazdo datagramowe – serwer UDP

/\*

============================================================================

Name : PAS\_5\_gniazda\_UDP\_server.c

Author : J.Krygier

Version :

Copyright :

Description : Serwer UDP

============================================================================

\*/

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <netdb.h> // formaty nazw i adresow sieciowych

#include <sys/types.h> // typy zmienneych

#include <sys/socket.h> // gniazda

#include <netinet/in.h> // typy protokolow

#include <arpa/inet.h> // adresacja IPv4

#define ROZMIAR\_BUFORA 1024

// wyprowadzanie bledow

void blad (char \*msg) {

perror(msg);

exit(1);

}

int main(int argc, char \*\*argv) {

int deskryptor\_gniazda;

int nr\_portu; // nr portu na ktorym serwer nasluchuje

socklen\_t dlugosc\_adresu\_klienta; // w bajtach (unsigned int)

struct sockaddr\_in serveraddr; // adres gniazda serwera

struct sockaddr\_in clientaddr; // adres gniazda klienta

struct hostent \*hostp; // wskaznik do informacji o kliencie laczacym sie z serwerem

char buf[ROZMIAR\_BUFORA]; // bufor na wiadomosci

char \*hostaddrp; // wskaznik na tablice przetrzymujaca adres IP (d.d.d.d)

int optval; // 32b flag gniazda do ustawienia przez setsockopt

int rozmiar\_danych; // rozmiar wiadomosci [B]

// mozliwosc wprowadzenia nr portu podczas uruchamiania serwera:

#if 0

if (argc != 2) {

fprintf(stderr, "podaj nr portu: %s <port>\n", argv[0]);

exit(1);

}

nr\_portu = atoi(argv[1]);

#endif

// albo ustawmy nr portu nasluchiwania serwera na stale

nr\_portu = 8888; // uwaga: nr portu w UDP i TCP jest max 16b (65536)

/\* tworzymy gniazdo \*/

/\*

\* int socket(int domain, int type, int protocol);

\*

\* funkcja zwraca deskryptor gniazda (int)

\*

\* domain:

\* AF\_INET IPv4 Internet protocols

\* AF\_INET6 IPv6 Internet protocols

\* AF\_PACKET Low level packet interface

\*

\* type:

\* SOCK\_STREAM Provides sequenced, reliable, two-way, connection-

based byte streams. -> polaczenie TCP

\* SOCK\_DGRAM Supports datagrams -> transfer UDP

\* SOCK\_RAW Provides raw network protocol access -> gniazdo surowe -> ramki Ethenet

\*

\* protocol:

\* dla SOCK\_DGRAM:

\* IPPROTO\_UDP

\*

\* dla SOCK\_STREAM:

\* IPPROTO\_TCP

\*

\* dla SOCK\_RAW:

\* IPPROTO\_ICMP

\* PPROTO\_RAW

\*/

//deskryptor\_gniazda = socket(AF\_INET, SOCK\_DGRAM, 0);

//albo

deskryptor\_gniazda = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (deskryptor\_gniazda < 0)

blad ("BLAD podczas otwarcia gniazda");

/\* Za pomoca 'setsockopt ()' ustawimy parametr gniazda pozwalajacy na natychmiastowe

\* jego zwolnienie po wymuszonym przerwaniu programu (Ctrl+Z).

\* W przeciwnym porzypadku ponowne uruchomienie serwera mozliwe byloby po ok. 20s.

\* Unikamy chwilowego zblokowania gniazda i wypisania 'ERROR on binding: Address already in use" error'.

\*

\* int setsockopt(int sockfd, int level, int optname, const void \*optval, socklen\_t optlen);

\* level:

\* SOL\_SOCKET - opcja uzyta na poziomie gniazda (bez wzgledu na typ protokolu)

\* IPPROTO\_TCP - opcja uzyta na poziomie protokolu TCP

\* ...

\* optname:

\* SO\_REUSEADDR - pozwala na natuchmiastowe ponownie uzycie gniazda

\* SO\_BROADCAST - pozwala na wyslanie wiadomosci rogloszeniowych

\* ...

\* uzycie: https://linux.die.net/man/3/setsockopt

\*/

optval = 1;

setsockopt(deskryptor\_gniazda, SOL\_SOCKET, SO\_REUSEADDR, (const void \*)&optval , sizeof(int)); // w socket.h

// Ustawienie adresu gniazda serwera

bzero((char \*) &serveraddr, sizeof(serveraddr)); // wyzerowanie bloku pamieci opisujacego gniazdo (string.h)

serveraddr.sin\_family = AF\_INET; //ustwienie typu protokolu: IPv4

serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //kto moze sie laczyc z serwerem: INADDR\_ANY - kazdy

//lub

//serveraddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //konkretny adres klienta, //inet\_addr() - przeksztalca //'d.d.d.d' na liczbe 32b (inet.h)

serveraddr.sin\_port = htons((unsigned short)nr\_portu); //na jakim porcie nasluchuje (in.h)

//laczymy deskryptor gniazda z jego adresem

/\*

\* int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

\*

\*/

if (bind(deskryptor\_gniazda, (struct sockaddr \*) &serveraddr, sizeof(serveraddr)) < 0) //socket.h

blad ("BLAD w laczeniu (bind)");

else

printf ("SERWER UDP uruchomiony\n");

dlugosc\_adresu\_klienta = sizeof(clientaddr);

//petla do odbioru danych od klienta i wyslania echo

while (1) {

bzero(buf, ROZMIAR\_BUFORA); // wyzerowanie bufora

printf ("SERWER UDP: Oczekuje na dane...\n");

/\* pobranie wiadomosci od klienta \*/

/\* w socket.h

\*

\* recvfrom - funkcja blokujaca -> oczekiwanie do czasu nadejscia danych

\* ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);

\*/

rozmiar\_danych = recvfrom(deskryptor\_gniazda, buf, ROZMIAR\_BUFORA, 0,

(struct sockaddr \*) &clientaddr, &dlugosc\_adresu\_klienta);

if (rozmiar\_danych < 0)

blad("BLAD w recvfrom");

// identyfikacja od kogo dane odebrano

/\*

\* struct hostent \*gethostbyaddr(const void \*addr, socklen\_t len, int type);

\* addr - wskaznik na adres IPv4 - liczba 32b

\* len - dlugosc adresu - IPv4 = 32b

\* type - typ protokolu: AF\_INET = IPv4

\* struct hostent - adres struktury z informacja o hoscie

\*/

hostp = gethostbyaddr((const char \*)&clientaddr.sin\_addr.s\_addr,

sizeof(clientaddr.sin\_addr.s\_addr), AF\_INET); //netdb.h

if (hostp == NULL)

blad("BLAD pozyskania informacji o kliencie");

hostaddrp = inet\_ntoa(clientaddr.sin\_addr);

if (hostaddrp == NULL)

blad("BLAD w inet\_ntoa\n");

printf("SERWER UDP: odebrano dane od %s (%s)\n", hostp->h\_name, hostaddrp);

printf("SERWER UDP: odebrano %d/%d bajtow: %s\n", strlen(buf), rozmiar\_danych, buf);

//wyslemy echo do klienta

/\* w socket.h

\* ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

\*

\*/

rozmiar\_danych = sendto(deskryptor\_gniazda, buf, strlen(buf), 0,

(struct sockaddr \*) &clientaddr, dlugosc\_adresu\_klienta);

if (rozmiar\_danych < 0)

blad ("BLAD w sendto");

}

}

# Gniazdo datagramowe – klient UDP

/\*

============================================================================

Name : PAS\_4\_gniazda\_UDP\_klient.c

Author : J.Krygier

Version :

Copyright :

Description : Klinet UDP

============================================================================

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <arpa/inet.h> //dla inet\_addr

#define BUFSIZE 1024

// wyprowadzanie bledow

void blad(char \*msg) {

perror(msg);

exit(0);

}

int main(int argc, char \*\*argv) {

int deskryptor\_gniazda, nr\_portu, rozmiar\_danych;

socklen\_t dlugosc\_adresu\_serwera;

struct sockaddr\_in serveraddr;

struct hostent \*server;

char \*nazwa\_hosta;

char buf[BUFSIZE];

#if 0

/\* sparwdz czy podano adres serwera i port\*/

if (argc != 3) {

fprintf(stderr,"podaj: %s <nazwe serwera> <port>\n", argv[0]);

exit(0);

}

nazwa\_hosta = argv[1];

nr\_portu = atoi(argv[2]);

#endif

// na sztywno podajemy adres serwera i port

nazwa\_hosta = "localhost";

nr\_portu = 8888;

/\* tworzymy gniazdo \*/

/\*

\* int socket(int domain, int type, int protocol);

\*

\* funkcja zwraca deskryptor gniazda (int)

\*

\* domain:

\* AF\_INET IPv4 Internet protocols

\* AF\_INET6 IPv6 Internet protocols

\* AF\_PACKET Low level packet interface

\*

\* type:

\* SOCK\_STREAM Provides sequenced, reliable, two-way, connection-

based byte streams. -> polaczenie TCP

\* SOCK\_DGRAM Supports datagrams -> transfer UDP

\* SOCK\_RAW Provides raw network protocol access -> gniazdo surowe -> ramki Ethenet

\*

\* protocol:

\* dla SOCK\_DGRAM:

\* IPPROTO\_UDP

\*

\* dla SOCK\_STREAM:

\* IPPROTO\_TCP

\*

\* dla SOCK\_RAW:

\* IPPROTO\_ICMP

\* PPROTO\_RAW

\*/

//sockfd = socket(AF\_INET, SOCK\_DGRAM, 0); // mozna tez z domyslnym protokolem

deskryptor\_gniazda = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

if (deskryptor\_gniazda < 0)

blad("BLAD w otwarciu gniazda");

/\* gethostbyname(): zapytanie do serwera DNS o adres serwera\*/

server = gethostbyname(nazwa\_hosta);

if (server == NULL) {

fprintf(stderr,"BLAD, nie ma takiego hosta %s\n", nazwa\_hosta);

exit(0);

}

/\* Konfiguracja adresu gniazda serwera UDP \*/

// ropoczynamy od wyczyszczenia gniazda serwera

bzero((char \*) &serveraddr, sizeof(serveraddr)); //to samo co 'memset () do wyzerowania bloku pamieci

//USTAWIENIE typu protokolu: IPv4

serveraddr.sin\_family = AF\_INET;

//USTAWIENIE adresu IP serwera z pobranego z DNS

//bcopy((char \*)server->h\_addr, (char \*)&serveraddr.sin\_addr.s\_addr, server->h\_length); //prawe to samo co 'memcpy'

//albo bezposrenido podanie adresu IP serwera:

serveraddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); //inet\_addr przeksztalca string do liczby 32b

//USTAWIENIE nr portu, na ktorym nasluchuje serwer

serveraddr.sin\_port = htons(nr\_portu);

while (1) {

/\* get a message from the user \*/

bzero(buf, BUFSIZE);

printf("Wprowadz wiadmomosc: ");

fgets(buf, BUFSIZE, stdin);

/\* Wyslanie wiadomosci do serwera UDP \*/

/\*

\* ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

\*

\*/

dlugosc\_adresu\_serwera = sizeof(serveraddr);

rozmiar\_danych = sendto(deskryptor\_gniazda , buf, strlen(buf), 0,

(struct sockaddr \*) &serveraddr, dlugosc\_adresu\_serwera);

if (rozmiar\_danych < 0)

blad("BLAD wysylania danych do serwera");

/\* pobranie wiadomosci z serwera \*/

/\*

\* recvfrom - funkcja blokujaca -> oczekiwanie do czasu nadejscia danych

\* ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);

\*/

rozmiar\_danych = recvfrom(deskryptor\_gniazda , buf, strlen(buf), 0,

(struct sockaddr \*) &serveraddr, &dlugosc\_adresu\_serwera);

if (rozmiar\_danych < 0)

blad("BLAD odbioru z gniazda");

printf("ECHO z serwera UDP: %s", buf);

}

return 0;

}