Wojskowa Akademia Techniczna
Wydział Elektroniki
Instytut Telekomunikacji

Studia II°

Języki C/C++ w zastosowaniach sieciowych

Materiały pomocnicze do zajęć
Dodatek 3

dr inż. Jarosław Krygier

Warszawa 2020

Spis treści

1 Tworzenie i zamykanie gniazd, obsługa błędów

```
            /*
============================================================================
Name        : PAS_001_gniazdko.c
Author      : Jaroslaw Krygier
Version     : v1.0
Copyright   : Jaroslaw Krygier
Description : Gniazda - utworzenie i zamkniecie, obsluga bledow
============================================================================
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>                     //close
#include <sys/socket.h>         //socket, domena, typ gniazda
#include <netinet/in.h>                 //protokoly
#include <errno.h>                      //errno
#include <unistd.h>             //getpid()
#include <sys/types.h>                  //pid_t

int main(void) {
    puts("Utworzenie, zamkniecie gniazda, bledy");
    puts("====================================");
    int socket_desc = -1;
    int out;

    //============
    socket_desc = socket(AF_INET, SOCK_STREAM, 0); // gniazdo strumieniowe (transmisja TCP)

    if (socket_desc == -1) {
        printf("(1) Nie moge utworzyc gniazda AF_INET/SOCK_STREAM\n");
    } else {
        printf("(1) Utworzono gniazdo AF_INET/SOCK_STREAM o numerze: %d \n",
                socket_desc);
    }
    //============
    socket_desc = socket(AF_INET, SOCK_DGRAM, 0); // gniazdo datagamowe (transmisja UDP)
    if (socket_desc == -1) {
        printf("(2) Nie moge utworzyc gniazda SOCK_DGRAM\n");
    } else {
        printf("(2) Utworzono gniazdo AF_INET/SOCK_DGRAM o numerze: %d \n",
                socket_desc);
    }
    //============
    socket_desc = socket(AF_INET, SOCK_RAW, IPPROTO_RAW); // gniazdo surowe (transmisja besposrednio do karty sieciowej)
    //wymaga uprawnien administratora

    if (socket_desc == -1) {
        printf("(3) Nie moge utworzyc gniazda SOCK_RAW\n");
    } else {
        printf("(4) Utworzono gniazdo AF_INET/SOCK_RAW o numerze: %d \n",
                socket_desc);
    }

    //============
    //IPv6
    //============
    socket_desc = socket(AF_INET6, SOCK_STREAM, 0); // gniazdo strumieniowe (transmisja TCP)

    if (socket_desc == -1) {
        printf("(5) Nie moge utworzyc gniazda SOCK_STREAM\n");
    } else {
        printf("(5) Utworzono gniazdo AF_INET6/SOCK_STREAM o numerze: %d \n",
                socket_desc);
    }
    //============
```

```
socket_desc = socket(AF_INET6, SOCK_DGRAM, 0); // gniazdo datagamowe (transmisja UDP)
if (socket_desc == -1) {
    printf("(6) Nie moge utworzyc gniazda SOCK_DGRAM\n");
} else {
    printf("(6) Utworzono gniazdo AF_INET6/SOCK_DGRAM o numerze: %d \n",
            socket_desc);
}
//============
socket_desc = socket(AF_INET6, SOCK_RAW, IPPROTO_RAW); // gniazdo surowe (transmisja besposrednio do karty sieciowej)
//wymaga uprawnien administratora

if (socket_desc == -1) {
    printf("(7) Nie moge utworzyc gniazda SOCK_RAW\n");
} else {
    printf("(7) Utworzono gniazdo AF_INET6/SOCK_RAW o numerze: %d \n",
            socket_desc);
}
// gniaza powyzsze sie zamkna po zakonczeniu programu

//============
//IPv4 - a teraz beda sie zamykac od razu
//============
socket_desc = socket(AF_INET, SOCK_STREAM, 0); // gniazdo strumieniowe (transmisja TCP)

if (socket_desc == -1) {
    printf("(8) Nie moge utworzyc gniazda AF_INET/SOCK_STREAM\n");
} else {
    printf("(8) Utworzono gniazdo AF_INET/SOCK_STREAM o numerze: %d \n",
            socket_desc);
}
out = shutdown(socket_desc, SHUT_RDWR);
//Tutaj nie bede mogl zamknac gniazda poniewaz shutdown zamyka gniazdo gdy jest aktywne polaczenie TCP
if (out == -1) {
    printf("(8.1) (shutdown) Nie moge zamknac gniazda AF_INET/SOCK_STREAM\n");
} else {
    printf("(8.1) (shutdown) Zamknieto gniazdo AF_INET/SOCK_STREAM o numerze: %d \n",
            socket_desc);
}
//============
socket_desc = socket(AF_INET, SOCK_STREAM, 0); // gniazdo strumieniowe (transmisja TCP)

if (socket_desc == -1) {
    printf("(9) Nie moge utworzyc gniazda AF_INET/SOCK_STREAM\n");
} else {
    printf("(9) Utworzono gniazdo AF_INET/SOCK_STREAM o numerze: %d \n",
            socket_desc);
}
out = close(socket_desc);
if (out == -1) {
    printf("(10) Nie moge zamknac gniazda AF_INET/SOCK_STREAM\n");
} else {
    printf("(10) Zamknieto gniazdo AF_INET/SOCK_STREAM o numerze: %d \n", socket_desc);
}
//============
socket_desc = socket(AF_INET, SOCK_DGRAM, 0); // gniazdo datagamowe (transmisja UDP)
if (socket_desc == -1) {
    printf("(11) Nie moge utworzyc gniazda SOCK_DGRAM\n");
} else {
    printf("(11) Utworzono gniazdo AF_INET/SOCK_DGRAM o numerze: %d \n",
            socket_desc);
}
out = close(socket_desc);
if (out == -1) {
    printf("(12) Nie moge zamknac gniazda AF_INET/SOCK_DGRAM \n");
} else {
    printf("(13) Zamknieto gniazdo AF_INET/SOCK_DGRAM  o numerze: %d \n", socket_desc);
}
//============
socket_desc = socket(AF_INET, SOCK_RAW, IPPROTO_RAW); // gniazdo surowe (transmisja besposrednio do karty sieciowej)
```

```
    //wymaga uprawnien administratora

    if (socket_desc == -1) {
        printf("(14) Nie moge utworzyc gniazda SOCK_RAW\n");
    } else {
        printf("(14) Utworzono gniazdo AF_INET/SOCK_RAW o numerze: %d \n",
                socket_desc);
    }
    out = close(socket_desc);
    if (out == -1) {
        printf("(15) Nie moge zamknac gniazda INET/SOCK_RAW \n");
    } else {
        printf("(15) Zamknieto gniazdo AF_INET/SOCK_RAW  o numerze: %d \n",
                socket_desc);
    }
    //===============
    //Bledy - musi byc dolaczony plik naglowokowy <errno.h>

    socket_desc = socket(AF_INET, 500, 0); // niepoprawny argument
    if (socket_desc == -1) {
        printf("(16) Nie moge utworzyc gniazda \n");
        printf("(16) Numer bledu %d\n ", errno);
    } else {
        printf("(16) Utworzono gniazdo o numerze: %d \n", socket_desc);
    }

    socket_desc = socket(AF_INET, SOCK_STREAM, IPPROTO_UDP); // niepoprawny protokol dla gniazda strumieniowego
    if (socket_desc == -1) {
        printf("(17) Nie moge utworzyc gniazda \n");
        printf("(17) Numer bledu %d\n ", errno);
    } else {
        printf("(17) Utworzono gniazdo o numerze: %d \n", socket_desc);
    }

    socket_desc = socket(AF_AX25, SOCK_STREAM, IPPROTO_UDP); // niepoprawna domena adresowa (X25)
    if (socket_desc == -1) {
        printf("(18) Nie moge utworzyc gniazda \n");
        printf("(18) Numer bledu %d\n ", errno);
    } else {
        printf("(18) Utworzono gniazdo o numerze: %d \n", socket_desc);
    }

    out = close(1000); // nie ma takiego gniazda otwartego
    if (out == -1) {
        printf("(19) Nie moge zamknac gniazda o numerze deskryptora: %d \n",
                1000);
        printf("(19) Numer bledu %d\n ", errno);
    } else {
        printf("(19) Zamknieto gniazdo o numerze: %d \n", 1000);
    }

    //===============
    //wypiszmy nr otwartych gniazd za pomoca komendy systemowej ls -l /proc/[PID]/fd
    pid_t pid = getpid();
    char command[20];
    printf("Identyfikagtor procesu PID: %d \n", pid);
    sprintf(command, "ls -l /proc/%d/fd", pid);
    printf("Aktywne gniazda: \n");
    system(command);

    //sleep(100);

    return 0;
}
```

2 Wiązanie deskryptora gniazda z adresem gniazda, opcje gniazda

```
/*
 ============================================================================
 Name        : PAS_002_adres_gniazda_bind.c
 Author      : Jarosaw Krygier
 Version     : 1.0
 Copyright   : Jarosaw Krygier
 Description : Adres gniazda, powiazanie z deskryptorem, opcje, obsluga bledow
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>    //close()
#include <sys/socket.h>
#include <arpa/inet.h>  //struct sockaddr, inet_addr()
#include <sys/types.h>
#include <errno.h>     // errno
#include <netinet/ip.h> //ToS...
#include <sys/types.h> //pid_t

int main(void) {
    puts("Adres gniazda, powiazanie z deskryptorem, opcje");
    puts("========================================");

    int socket_desc = -1;
    int socket_desc1 = -1;
    int socket_desc2 = -1;
    struct sockaddr_in serwer, klient; //struct sockaddr_in - struktura gniazda internetowego IPv4
    int opt_val, ret;
    int opt_len;
    int sock_addr_len;
    struct sockaddr_in pobrana_nazwa_gn;

    //=========================================
    //Jestem klientem TCP - przygotuj addres gniazda
    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc == -1) {
        printf("Nie moge utworzyc gniazda (1) \n");
    } else {
        printf("Utworzylem gniazdo SOCK_STREAM (1):%d \n", socket_desc);
    }

    //Adres gniazda (nazwa gniazda)
    klient.sin_addr.s_addr = inet_addr("127.0.0.1"); //moj adres
    klient.sin_family = AF_INET;
    klient.sin_port = htons(8765);

    //Powiaz adres gniazda z deskryptorem
    if (bind(socket_desc, (struct sockaddr *) &klient, sizeof(klient)) < 0) {
        puts("Problem dolaczenia gniazda");
    }

    //Odczytanie adresu (nazwy) gniazda powiazdanego z deskryptorem
    //rozmiar struktury adresu gniazda
    sock_addr_len = sizeof(struct sockaddr);
    getsockname(socket_desc, (struct sockaddr *) &pobrana_nazwa_gn,
            (socklen_t *) &sock_addr_len);

    printf("Adres IP w dowiazanym gniezdzie: %s\n",
            inet_ntoa(pobrana_nazwa_gn.sin_addr));
    printf("Port w dowiazanym gniezdzie: %d\n",
            (int) ntohs(pobrana_nazwa_gn.sin_port));

    //=========================================
```

```
//nowe gniazdo
socket_desc1 = socket(AF_INET, SOCK_STREAM, 0);
if (socket_desc1 == -1) {
    printf("Nie moge utworzyc gniazda (2) \n");
} else {
    printf("Utworzylem gniazdo SOCK_STREAM (2):%d \n", socket_desc1);
}

//Jestem serwerem TCP - Przygotuj addres gniazda
serwer.sin_family = AF_INET;
serwer.sin_addr.s_addr = INADDR_ANY;
serwer.sin_port = htons(8888);

//Powiaz adres gniazda z deskryptorem
if (bind(socket_desc1, (struct sockaddr *) &serwer, sizeof(serwer)) < 0) {
    puts("Problem dowiazaniem gniazda");
}

//Odczytamy zwiazany adres (nazwe) gniazda
sock_addr_len = sizeof(struct sockaddr);
getsockname(socket_desc1, (struct sockaddr *) &pobrana_nazwa_gn, (socklen_t *) &sock_addr_len);

printf("Adres IP w dowiazanym gniezdzie: %s\n",
        inet_ntoa(pobrana_nazwa_gn.sin_addr));
printf("Port w dowiazanym gniezdzie: %d\n",
        (int) ntohs(pobrana_nazwa_gn.sin_port));

//=====
//nowe gniazdo
socket_desc2 = socket(AF_INET, SOCK_STREAM, 0);
if (socket_desc2 == -1) {
    printf("Nie moge utworzyc gniazda (3)\n");
} else {
    printf("Utworzylem gniazdo SOCK_STREAM (3):%d \n", socket_desc2);
}

//Powiazemy gnizado z nazwa
klient.sin_addr.s_addr = inet_addr("127.0.0.1"); //moj adres
//klient.sin_addr.s_addr = inet_addr("10.1.1.1"); //adres ktorego nie mam na interfejsie - bedzie blad
klient.sin_family = AF_INET;
klient.sin_port = htons(9999);
//klient.sin_port = htons(1); //bedzie blad bindowania - bez uprawnien admina takiego portu nie ustawimy
//klient.sin_port = htons(8765); //bedzie blad - powtorzony adres gniazda IP/port

//Powiaz adres gniazda z deskryptorem
if (bind(socket_desc2, (struct sockaddr *) &klient, sizeof(klient)) < 0) {
    puts("Problem dolaczenia gniazda");
    printf("Numer bledu %d\n ", errno); //jesli nr bledu 99, to EADDRNOTAVAIL - Cannot assign requested address
}

//Odczytamy zwiazany adres (nazwe) gniazda
sock_addr_len = sizeof(struct sockaddr);
getsockname(socket_desc2, (struct sockaddr *) &pobrana_nazwa_gn,
        (socklen_t *) &sock_addr_len);

printf("Adres IP w dowiazanym gniezdzie: %s\n",
        inet_ntoa(pobrana_nazwa_gn.sin_addr));
printf("Port w dowiazanym gniezdzie: %d\n",
        (int) ntohs(pobrana_nazwa_gn.sin_port));


//======================
//Obsluga opcji
//======================
opt_len = sizeof(opt_val);
ret = getsockopt(socket_desc2, SOL_SOCKET, SO_RCVBUF, &opt_val,
        (socklen_t*) &opt_len);

if (ret == -1) {
```

```
    printf("Nie moge odczytac opcji \n");
} else {
    printf("GET: Dugosc bufora odbiorczego=%d \n", opt_val);
}
//--------
ret = getsockopt(socket_desc2, SOL_SOCKET, SO_SNDBUF, &opt_val,
        (socklen_t*) &opt_len);

if (ret == -1) {
    printf("Nie moge odczytac opcji \n");
} else {
    printf("GET: Dugosc bufora nadawczego=%d\n", opt_val);
}

//--------
opt_val = 30000;
setsockopt(socket_desc2, SOL_SOCKET, SO_RCVBUF, &opt_val,
        (socklen_t) sizeof(int));
printf("SET: Dugosc bufora odbiorczego=%d \n", opt_val);

//---
opt_len = sizeof(opt_val);
ret = getsockopt(socket_desc2, SOL_SOCKET, SO_RCVBUF, &opt_val,
        (socklen_t*) &opt_len);
if (ret == -1) {
    printf("Nie moge odczytac opcji \n");
} else {
    printf("GET: Dugosc bufora odbiorczego=%d \n", opt_val);
}

//--------
opt_len = sizeof(opt_val);
ret = getsockopt(socket_desc2, SOL_SOCKET, SO_TYPE, &opt_val,
        (socklen_t*) &opt_len);
if (ret == -1) {
    printf("Nie moge odczytac opcji \n");
} else {
    printf("GET: Typ gniazda=%d \n", opt_val); //1 SOCK_STREAM, 2 SOCK_DGRAM
}

//--------
opt_len = sizeof(opt_val);
ret = getsockopt(socket_desc2, SOL_SOCKET, SO_ERROR, &opt_val,
        (socklen_t*) &opt_len);
if (ret == -1) {
    printf("Nie moge odczytac opcji \n");
} else {
    printf("GET: Socket error=%d \n", opt_val);
}

//--------
opt_len = sizeof(opt_val);
ret = getsockopt(socket_desc2, SOL_SOCKET, SO_TYPE, &opt_val,
        (socklen_t*) &opt_len);
if (ret == -1) {
    printf("Nie moge odczytac opcji \n");
} else {
    printf("GET: Typ gniazda=%d \n", opt_val); //1 SOCK_STREAM, 2 SOCK_DGRAM
}

//--------
opt_val = 100;
setsockopt(socket_desc2, IPPROTO_IP, IP_TTL, &opt_val,
        (socklen_t) sizeof(int));
printf("SET: TTL=%d \n", opt_val);

//--------
opt_val = IPTOS_DSCP_EF; //wartosci zdefiniowane w ip.h
setsockopt(socket_desc2, IPPROTO_IP, IP_TOS, &opt_val,
```

```
        (socklen_t) sizeof(int));
    printf("SET: TOS=%d \n", opt_val);

    //--------
    opt_len = sizeof(opt_val);
    ret = getsockopt(socket_desc2, IPPROTO_IP, IP_TTL, &opt_val,
        (socklen_t*) &opt_len);
    if (ret == -1) {
        printf("Nie moge odczytac opcji X\n");
    } else {
        printf("GET: TTL=%d \n", opt_val);
    }

    //--------
    ret = getsockopt(socket_desc2, IPPROTO_IP, IP_TOS, &opt_val,
        (socklen_t*) &opt_len);
    if (ret == -1) {
        printf("Nie moge odczytac opcji TOS\n");
    } else {
        printf("GET: TOS=%d \n", opt_val);
    }

    //--------
    //odblokowanie gniazda po czasie timeout (funkcje np. recvfrom nie beda czekac)
    struct timeval timeout;
    timeout.tv_sec = 2; //po 2 sek
    timeout.tv_usec = 0;
    ret = setsockopt(socket_desc2, SOL_SOCKET, SO_RCVTIMEO, &timeout,
        sizeof(struct timeval));
    if (ret == -1) {
        printf("Nie moge odblokowac gniazda\n");
    } else {
        printf("SET: Gniazdo zostanie odlokowane po czasie=%d.%d sec\n",
            (int) timeout.tv_sec, (int) timeout.tv_usec);
    }
    //Sprawdzmy czy opcja zostala wprowadzona
    struct timeval opt_val_1;
    int opt_len_1 = sizeof(struct timeval);
    ret = getsockopt(socket_desc2, SOL_SOCKET, SO_RCVTIMEO,
        (struct timeval*) &opt_val_1, (socklen_t*) &opt_len_1);
    if (ret == -1) {
        printf("Nie moge odczytac opcji TOS\n");
    } else {
        printf("GET: Gniazdo zostanie odlokowane po czasie=%d.%d sec\n",
            (int) opt_val_1.tv_sec, (int) opt_val_1.tv_usec);
    }

    //Wypiszmy nr otwartych gniazd za pomoca komendy systemowej ls -l /proc/[PID]/fd
    pid_t pid = getpid();
    char command[20];
    printf("Identyfikagtor procesu PID: %d \n", pid);
    sprintf(command, "ls -l /proc/%d/fd", pid);
    printf("Aktywne gniazda: \n");
    system(command);

    //zamkniecie gniazd
    close(socket_desc);
    printf("Zamknalem gniazdo: %d\n", socket_desc);
    close(socket_desc1);
    printf("Zamknalem gniazdo: %d\n", socket_desc1);
    close(socket_desc2);
    printf("Zamknalem gniazdo: %d\n", socket_desc2);

    return EXIT_SUCCESS;
}
```

3 Gniazdo datagramowe - serwer

```
/*
 ============================================================================
 Name        : PAS_5_gniazda_UDP_server.c
 Author      : J.Krygier
 Version     : 1.0
 Copyright   : J.Krygier
 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]

  //netstat -vaun: mozna sprawdzic aktywne polaczenia UDP

  // 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 rozgloszeniowych
   * ...
   * 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 domeny: IPv4
  serveraddr.sin_addr.s_addr = htonl(INADDR_ANY); //na jaki adres beda przychodzily dane? INADDR_ANY - obojetnie
  //lub
  //serveraddr.sin_addr.s_addr = inet_addr("127.0.0.1");  //lub na jakis adres mojego interfejsu; 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 (wiazemy) deskryptor gniazda z jego adresem (nadajemy nazwe)
  /*
   * 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);
   * funkcja dosteje adres IP, zwraca nazwe hosta (system moze skorzystac z zapytan do DNS)
   * addr - wskaznik na adres IP
   * len - dlugosc adresu IP
   * 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");
  else {
   printf ("Wyslano: %d bajtow \n", rozmiar_danych);
   printf ("====================\n");
  }
 }
 return 0;
}
```

4 Gniazdo datagramowe - klient

```
/*
 ============================================================================
 Name        : PAS_4_gniazda_UDP_klient.c
 Author      : J.Krygier
 Version     : 1.0
 Copyright   : J.Krygier
 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> //protocols
#include <netdb.h> //gethostbyname()
#include <arpa/inet.h> //inet_addr()
#define BUFSIZE 1024
#define UZYWAJ_DNS0

// 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;
    char buf[BUFSIZE];
    char *ip_addr;
    unsigned short int port;
    //jesli nie bedziemy uzywac DNSu to te zmienne nie sa potrzebne
#if UZYWAJ_DNS
    struct hostent *server;
    char *nazwa_hosta;
#endif

    //Timeout do odblokowania recvfrom
    struct timeval timeout;

#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"; //do uzycia w fukcji gethostbyname()
    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 (0: UDP)
    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*/
    /*wylaczyc jesli adres IP serwera podany bedzie lokalnie*/
#if UZYWAJ_DNS
    server = gethostbyname(nazwa_hosta);
    if (server == NULL) {
        fprintf(stderr,"BLAD, nie ma takiego hosta %s\n", nazwa_hosta);
        exit(0);
    }
#endif

    /* Konfiguracja adresu gniazda docelowego (serwera UDP) */
    // Ropoczynamy od wyczyszczenia gniazda serwera
    bzero((char *) &serveraddr, sizeof(serveraddr)); //to samo co 'memset () do wyzerowania bloku pamieci

    //USTAWIENIE domeny: IPv4
    serveraddr.sin_family = AF_INET;

    //USTAWIENIE adresu IP serwera pobranego z DNS za pomoca gethostbyname()
#if UZYWAJ_DNS
    bcopy((char *)server->h_addr, (char *)&serveraddr.sin_addr.s_addr, server->h_length); //prawe to samo co 'memcpy'
#endif
    //albo bezposrednio podanie adresu IP serwera tutaj:
    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);

    //Odblokowanie gniazda po czasie timeout,
    //zeby recvfrom() nie czekalo za dlugo, gdy nie ma odpowiedzi z serwera
    timeout.tv_sec = 1; //sekundy
    timeout.tv_usec = 0; //mikrosekundy
    setsockopt(deskryptor_gniazda, SOL_SOCKET, SO_RCVTIMEO, &timeout,
            sizeof(struct timeval));

    while (1) {
        bzero(buf, BUFSIZE); //wyzeruj bufor znakowy
        printf("Wprowadz wiadmomosc: ");
        fgets(buf, BUFSIZE, stdin); // bobierz tekst z klawiatury i umiesc w buforze

        /* 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 - gdy gniazdo blokujace -> oczekiwanie do czasu nadejscia danych
         * ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
         struct sockaddr *src_addr, socklen_t *addrlen);
         * My odblokowalismy je po jakims czasie za pomoca setsockopt()
         */

        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("nie odebralem danych: blad lub timeout \n");
        } else {
            //recvfrom podaje od kogo otrzymuje dane: gniazdo zrodlowe
            // zobaczmy czy na pewno z adresu gniazda serwera
            ip_addr = inet_ntoa(serveraddr.sin_addr);
            port = ntohs(serveraddr.sin_port);
            printf("Odpowiedz z serwera UDP (IP:port): %s:%d \n", ip_addr, port);
            printf("ECHO z serwera UDP (%d bajtow): %s", rozmiar_danych, buf);
            printf("=====================\n");
        }
    }
    close(deskryptor_gniazda);
    return 0;
}
```

5 Gniazdo strumieniowe - serwer

```
/*
 ============================================================================
 Name        : PAS_5_gniazda_serwer.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Serwer TCP:
 ============================================================================
 */
#include <stdio.h>
#include <string.h>    //strlen
#include <sys/socket.h>
#include <arpa/inet.h> //inet_addr
#include <unistd.h>    //write
#include <errno.h>     // errno
#include <signal.h> //do obsugi sygnalu przerwania
#define      SERWER_SIE_ZGLASZA 1
#define      PRACA_W_PETLI 1

int end = 0; //flaga ustawiana po odebraniu sygnalu SIGINT (CtrC)

void obsluga_sygnalu_ctrC(int signal) {
    printf("Odebralem sygnal SIGINT \n");
    end = 1;
}
void obsluga_sygnalu_term(int signal) {
    printf("Odebralem sygnal SIGTERM \n");
    end = 1;
}
void obsluga_sygnalu_kill(int signal) {
    printf("Odebralem sygnal SIGKILL \n");
    end = 1;
}

int main(int argc, char *argv[]) {
    int socket_desc, new_socket, c;
    struct sockaddr_in server, client;
    int read_size;
    char *message, client_message[3000];
    char *ip_addr;
    unsigned short int port;
    int out;

    //netstat -vatn: mozna sprawdzic aktywne polaczenia              TCP

    //obsluga sygnalow wymuszonego zamkniecia programu
    signal(SIGINT, obsluga_sygnalu_ctrC); // obsuga sygnalu SIGINT
    signal(SIGKILL, obsluga_sygnalu_kill);
    signal(SIGTERM, obsluga_sygnalu_term);

    //Utworz gniazdo struminiowe
    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_desc == -1) {
        printf("Nie moge utworzyc gniazda");
    }

    //Przygotuj addres gniazda
    server.sin_family = AF_INET;
    //server.sin_addr.s_addr = INADDR_ANY; // Obojetnie na jaki adres przyjda dane (moich interfejsow) INADDR_ANY = 0.0.0.0
    //server.sin_addr.s_addr = inet_addr("192.168.110.130"); // Moj adres interfejsu
    server.sin_addr.s_addr = inet_addr("127.0.0.1"); // Moj adres interfejsu
    server.sin_port = htons(8888); //Moj port


    //Polacz  deskrypor gniazda (gniazdo) z przypisanym adresem gniazda (wiazanie)
    out = bind(socket_desc, (struct sockaddr *) &server, sizeof(server));
```

```
    if (out < 0) {
        puts("Problem dolaczenia gniazda");
        printf("Numer bledu %d\n ", errno);
        return 0; //wyjdziemy z programu
    } else {
        puts("Dolaczono gniazdo strumieniowe");
    }

    //Przygotowanie gniazda do nasluchiwania (serwer)
    listen(socket_desc, 3);

    //Akceptuj polaczenie przychodzace
    puts("SERWER TCP: Czekam na polaczenie...");
    c = sizeof(struct sockaddr_in);
    new_socket = accept(socket_desc, (struct sockaddr *) &client,
            (socklen_t*) &c);
    if (new_socket < 0) {
        perror("Problem z akceptacja poloczenia");
    }

    ip_addr = inet_ntoa(client.sin_addr);
    port = ntohs(client.sin_port);
    printf("Polaczenie zaakceptowane od klienta o adresie IP: %s i porcie zrodlowym %d\n", ip_addr, port);
    printf("Moj adres gniazda (IP: %s, port %d)  \n", inet_ntoa(server.sin_addr), ntohs(server.sin_port));

#if SERWER_SIE_ZGLASZA
    //Odpowiedz do klienta
    message = "Witam, poloczeyles sie z serwerem TCP. Czekam na dane...\n";
    write(new_socket, message, strlen(message));
#endif

#if PRACA_W_PETLI
    while (1) {
        while ((read_size
                = recv(new_socket, client_message, 3000, MSG_DONTWAIT)) > 0) {
            puts("SERWER: Dane odebrano");
            puts(client_message);
            message = "Info od serwera: odebralem dane";
            //Wyslij echo
            write(new_socket, message, strlen(message));
            //albo send()
            //send(new_socket, message, strlen(message), 0);
        }

        if (end == 1) {
            puts("Koniec programu");
            break;
        }
    }
#endif

    puts("Zamykam gniazda");
    close(new_socket);
    close(socket_desc);

    return 0;
}
```

6 Gniazdo strumieniowe - klinet

```
/*
 ============================================================================
 Name        : PAS_4_gniazda.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Gniazda: kilent - gniazdo strumieniowe (TCP)
 ============================================================================
 */

#include <stdio.h>
#include <sys/socket.h>  //socket
#include <arpa/inet.h> //inet_addr
#include <string.h>    //strlen
#include <unistd.h>  //close
#define LOCAL_BIND  0

int main(int argc, char *argv[]) {
    int socket_desc = -1;
    struct sockaddr_in server, moj_addr_gniazda;
    char *message, server_reply[2000];

    socket_desc = socket(AF_INET, SOCK_STREAM, 0); // gniazdo strumieniowe (transmisja TCP)

    if (socket_desc == -1) {
        printf("Nie moge utworzyc gniazda \n");
    } else {
        printf("Utworzono gniazdo o numerze: %d \n", socket_desc);
    }

#if LOCAL_BIND
    //Przygotowuje adres gniazda lokalnego - czynnosc opcjonalna,jak bede chcial wymusic konkretny port zrodlowy, to musze wykonac wiazanie
    moj_addr_gniazda.sin_family = AF_INET;
    moj_addr_gniazda.sin_addr.s_addr = INADDR_ANY; // na jakikolwiek moj adres przyjda pakiety (moich interfejsow)
    moj_addr_gniazda.sin_port = htons(8000); //wymuszam zastosoweanie portu zrodlowego 8000, bede odbieral na porcie 8000

    //wymuszam wiazanie adresu gniazda lokalnego z deskryptorem do komunikacji z serwerem - bede uzywal konkretnego portu zrodlowego
    //do mojej aplikacji (poprzez deskryptor) beda kierwane dane przeslane na jakikolwiek adres IP i wymuszony na port
    if (bind(socket_desc, (struct sockaddr *) &moj_addr_gniazda,
            sizeof(moj_addr_gniazda)) < 0) {
        puts("Problem dolaczenia gniazda");
    }
#endif

    //Przygotowuje adres gniazda serwera zdalnego
    //server.sin_addr.s_addr = inet_addr("172.217.20.206");
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    //server.sin_addr.s_addr = inet_addr("192.168.110.130");
    server.sin_family = AF_INET;
    //server.sin_port = htons( 80 );
    server.sin_port = htons(8888);

    //Polacz z serwerem zdalnym (TCP)
    if (connect(socket_desc, (struct sockaddr *) &server, sizeof(server)) < 0) {
        puts("Blad polaczenia");
        return 1;
    }

    puts("Polaczono");

    sleep(5);

#if 1
    //Wyslij jakies dane
    message = "GET / HTTP/1.1\r\n\r\n";
    // funkcja do wysyania danych (TCP)
```

```
    if (send(socket_desc, message, strlen(message), 0) < 0) {
        puts("Blad wyslania danych");
        return 1;
    }
    puts("Dane wyslano\n");
#endif

    while (1) {
        //Nasluchiwanie i odbior danych z serwera (TCP)
        if (recv(socket_desc, server_reply, 2000, 0) < 0) {
            puts("Blad odbioru danych");
        }
        puts("Dane odebrano\n");
        puts(server_reply);
        memset(server_reply, 0, sizeof(server_reply));
        sleep(1);
        //wysylanie
        if (send(socket_desc, message, strlen(message), 0) < 0) {
            puts("Blad wyslania danych");
            return 1;
        }
        puts("Dane wyslano\n");

    }
    close(socket_desc);
    return 0;
}
```

7 Gniazdo surowe (RAW) – odbieranie danych

```
/*
 * Oprogramownie na podstawie:
 *
 *  Packet Sniffer Code in C using sockets | Linux
 *  Author: By Silver Moon
 *  Apr 26, 2009
 *  http://www.binarytides.com/packet-sniffer-code-c-linux/
 */

#include <stdio.h> //For standard things
#include <stdlib.h>    //malloc
#include <string.h>    //memset
#include <netinet/ip_icmp.h>   //Provides declarations for icmp header
#include <netinet/udp.h>   //Provides declarations for udp header
#include <netinet/tcp.h>   //Provides declarations for tcp header
#include <netinet/ip.h>    //Provides declarations for ip header
#include <sys/socket.h>
#include <arpa/inet.h>
#include <net/if.h>
#include <unistd.h>

void ProcessPacket(unsigned char* , int);
void print_ip_header(unsigned char* , int);
void print_tcp_packet(unsigned char* , int);
void print_tcp_packet_to_terminal(unsigned char* , int);
void print_udp_packet(unsigned char * , int);
void print_icmp_packet(unsigned char* , int);
void PrintData (unsigned char* , int);

int sock_raw;
FILE *logfile;
int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;
struct sockaddr_in source,dest;

int main()
{
        socklen_t saddr_size;
        int data_size;
   struct sockaddr saddr;
   //struct in_addr in;

   unsigned char *buffer = (unsigned char *)malloc(65536); //Its Big!

   logfile=fopen("log.txt","w");
   if(logfile==NULL) printf("Unable to create file.");
   printf("Starting...\n");
   //Create a raw socket that shall sniff
   sock_raw = socket(AF_INET , SOCK_RAW , IPPROTO_TCP);

   struct ifreq ifr;
   memset(&ifr, 0, sizeof(ifr));
   snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "lo");
   setsockopt(sock_raw, SOL_SOCKET, SO_BINDTODEVICE, (void *)&ifr, sizeof(ifr));

   if(sock_raw < 0)
   {
      printf("Socket Error\n");
      return 1;
   }
   while(1)
   {
      saddr_size = sizeof saddr;
      //Receive a packet
      data_size = recvfrom(sock_raw , buffer , 65536 , 0 , &saddr , &saddr_size);
      if(data_size <0 )
      {
```

```
            printf("Recvfrom error , failed to get packets\n");
            return 1;
        }
        //Now process the packet
        ProcessPacket(buffer , data_size);
    }
    close(sock_raw);
    printf("Finished");
    return 0;
}

void ProcessPacket(unsigned char* buffer, int size)
{
    //Get the IP Header part of this packet
    struct iphdr *iph = (struct iphdr*)buffer;
    ++total;
    switch (iph->protocol) //Check the Protocol and do accordingly...
    {
        case 1:  //ICMP Protocol
            ++icmp;
            //PrintIcmpPacket(Buffer,Size);
            break;

        case 2:  //IGMP Protocol
            ++igmp;
            break;

        case 6:  //TCP Protocol
            ++tcp;
            //print_tcp_packet(buffer , size);
            print_tcp_packet_to_terminal(buffer , size);
            break;

        case 17: //UDP Protocol
            ++udp;
            print_udp_packet(buffer , size);
            break;

        default: //Some Other Protocol like ARP etc.
            ++others;
            break;
    }
    printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d   Total : %d\r",tcp,udp,icmp,igmp,others,total);
}

void print_ip_header(unsigned char* Buffer, int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)Buffer;
    iphdrlen = iph->ihl*4;

    memset(&source, 0, sizeof(source));
    source.sin_addr.s_addr = iph->saddr;

    memset(&dest, 0, sizeof(dest));
    dest.sin_addr.s_addr = iph->daddr;

    fprintf(logfile,"\n");
    fprintf(logfile,"IP Header\n");
    fprintf(logfile,"   |-IP Version        : %d\n",(unsigned int)iph->version);
    fprintf(logfile,"   |-IP Header Length  : %d DWORDS or %d Bytes\n",(unsigned int)iph->ihl,((unsigned int)(iph->ihl))*4);
    fprintf(logfile,"   |-Type Of Service   : %d\n",(unsigned int)iph->tos);
    fprintf(logfile,"   |-IP Total Length   : %d  Bytes(Size of Packet)\n",ntohs(iph->tot_len));
    fprintf(logfile,"   |-Identification    : %d\n",ntohs(iph->id));
    //fprintf(logfile,"   |-Reserved ZERO Field   : %d\n",(unsigned int)iphdr->ip_reserved_zero);
    //fprintf(logfile,"   |-Dont Fragment Field   : %d\n",(unsigned int)iphdr->ip_dont_fragment);
    //fprintf(logfile,"   |-More Fragment Field   : %d\n",(unsigned int)iphdr->ip_more_fragment);
    fprintf(logfile,"   |-TTL      : %d\n",(unsigned int)iph->ttl);
```

```
    fprintf(logfile,"   |-Protocol : %d\n",(unsigned int)iph->protocol);
    fprintf(logfile,"   |-Checksum : %d\n",ntohs(iph->check));
    fprintf(logfile,"   |-Source IP        : %s\n",inet_ntoa(source.sin_addr));
    fprintf(logfile,"   |-Destination IP   : %s\n",inet_ntoa(dest.sin_addr));
}

void print_tcp_packet(unsigned char* Buffer, int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)Buffer;
    iphdrlen = iph->ihl*4;

    struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen);

    fprintf(logfile,"\n\n**********************TCP Packet************************\n");

    print_ip_header(Buffer,Size);

    fprintf(logfile,"\n");
    fprintf(logfile,"TCP Header\n");
    fprintf(logfile,"   |-Source Port      : %u\n",ntohs(tcph->source));
    fprintf(logfile,"   |-Destination Port : %u\n",ntohs(tcph->dest));
    fprintf(logfile,"   |-Sequence Number    : %u\n",ntohl(tcph->seq));
    fprintf(logfile,"   |-Acknowledge Number : %u\n",ntohl(tcph->ack_seq));
    fprintf(logfile,"   |-Header Length      : %d DWORDS or %d BYTES\n" ,(unsigned int)tcph->doff,(unsigned int)tcph->doff*4);
    //fprintf(logfile,"   |-CWR Flag : %d\n",(unsigned int)tcph->cwr);
    //fprintf(logfile,"   |-ECN Flag : %d\n",(unsigned int)tcph->ece);
    fprintf(logfile,"   |-Urgent Flag          : %d\n",(unsigned int)tcph->urg);
    fprintf(logfile,"   |-Acknowledgement Flag : %d\n",(unsigned int)tcph->ack);
    fprintf(logfile,"   |-Push Flag            : %d\n",(unsigned int)tcph->psh);
    fprintf(logfile,"   |-Reset Flag           : %d\n",(unsigned int)tcph->rst);
    fprintf(logfile,"   |-Synchronise Flag     : %d\n",(unsigned int)tcph->syn);
    fprintf(logfile,"   |-Finish Flag          : %d\n",(unsigned int)tcph->fin);
    fprintf(logfile,"   |-Window         : %d\n",ntohs(tcph->window));
    fprintf(logfile,"   |-Checksum       : %d\n",ntohs(tcph->check));
    fprintf(logfile,"   |-Urgent Pointer : %d\n",tcph->urg_ptr);
    fprintf(logfile,"\n");
    fprintf(logfile,"                    DATA Dump                     ");
    fprintf(logfile,"\n");

    fprintf(logfile,"IP Header\n");
    PrintData(Buffer,iphdrlen);

    fprintf(logfile,"TCP Header\n");
    PrintData(Buffer+iphdrlen,tcph->doff*4);

    fprintf(logfile,"Data Payload\n");
    PrintData(Buffer + iphdrlen + tcph->doff*4 , (Size - tcph->doff*4-iph->ihl*4) );

    fprintf(logfile,"\n###########################################################\n");
}

void print_tcp_packet_to_terminal(unsigned char* Buffer, int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)Buffer;
    iphdrlen = iph->ihl*4;

    struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen);

    printf("\n\n**********************TCP Packet************************\n");

    //print_ip_header(Buffer,Size);

    printf("\n");
    printf("TCP Header\n");
    printf("   |-Source Port      : %u\n",ntohs(tcph->source));
```

```
    printf("   |-Destination Port : %u\n",ntohs(tcph->dest));
    printf("   |-Sequence Number    : %u\n",ntohl(tcph->seq));
    printf("   |-Acknowledge Number : %u\n",ntohl(tcph->ack_seq));
    printf("   |-Header Length      : %d DWORDS or %d BYTES\n" ,(unsigned int)tcph->doff,(unsigned int)tcph->doff*4);
    //fprintf(logfile,"   |-CWR Flag : %d\n",(unsigned int)tcph->cwr);
    //fprintf(logfile,"   |-ECN Flag : %d\n",(unsigned int)tcph->ece);
    printf("   |-Urgent Flag          : %d\n",(unsigned int)tcph->urg);
    printf("   |-Acknowledgement Flag : %d\n",(unsigned int)tcph->ack);
    printf("   |-Push Flag            : %d\n",(unsigned int)tcph->psh);
    printf("   |-Reset Flag           : %d\n",(unsigned int)tcph->rst);
    printf("   |-Synchronise Flag     : %d\n",(unsigned int)tcph->syn);
    printf("   |-Finish Flag          : %d\n",(unsigned int)tcph->fin);
    printf("   |-Window         : %d\n",ntohs(tcph->window));
    printf("   |-Checksum       : %d\n",ntohs(tcph->check));
    printf("   |-Urgent Pointer : %d\n",tcph->urg_ptr);
    printf("\n");
    //printf("                    DATA Dump                     ");
    //printf("\n");

    //fprintf(logfile,"IP Header\n");
    //PrintData(Buffer,iphdrlen);

    //fprintf(logfile,"TCP Header\n");
    //PrintData(Buffer+iphdrlen,tcph->doff*4);

    //fprintf(logfile,"Data Payload\n");
    //PrintData(Buffer + iphdrlen + tcph->doff*4 , (Size - tcph->doff*4-iph->ihl*4) );

    printf("\n###########################################################\n");
}

void print_udp_packet(unsigned char *Buffer , int Size)
{

    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)Buffer;
    iphdrlen = iph->ihl*4;

    struct udphdr *udph = (struct udphdr*)(Buffer + iphdrlen);

    fprintf(logfile,"\n\n***********************UDP Packet*************************\n");

    print_ip_header(Buffer,Size);

    fprintf(logfile,"\nUDP Header\n");
    fprintf(logfile,"   |-Source Port      : %d\n" , ntohs(udph->source));
    fprintf(logfile,"   |-Destination Port : %d\n" , ntohs(udph->dest));
    fprintf(logfile,"   |-UDP Length       : %d\n" , ntohs(udph->len));
    fprintf(logfile,"   |-UDP Checksum     : %d\n" , ntohs(udph->check));

    fprintf(logfile,"\n");
    fprintf(logfile,"IP Header\n");
    PrintData(Buffer , iphdrlen);

    fprintf(logfile,"UDP Header\n");
    PrintData(Buffer+iphdrlen , sizeof udph);

    fprintf(logfile,"Data Payload\n");
    PrintData(Buffer + iphdrlen + sizeof udph ,( Size - sizeof udph - iph->ihl * 4 ));

    fprintf(logfile,"\n###########################################################");
}

void print_icmp_packet(unsigned char* Buffer , int Size)
{
    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)Buffer;
```

```
    iphdrlen = iph->ihl*4;

    struct icmphdr *icmph = (struct icmphdr *)(Buffer + iphdrlen);

    fprintf(logfile,"\n\n**********************ICMP Packet*************************\n");

    print_ip_header(Buffer , Size);

    fprintf(logfile,"\n");

    fprintf(logfile,"ICMP Header\n");
    fprintf(logfile,"   |-Type : %d",(unsigned int)(icmph->type));

    if((unsigned int)(icmph->type) == 11)
       fprintf(logfile,"  (TTL Expired)\n");
    else if((unsigned int)(icmph->type) == ICMP_ECHOREPLY)
       fprintf(logfile,"  (ICMP Echo Reply)\n");
    fprintf(logfile,"   |-Code : %d\n",(unsigned int)(icmph->code));
    fprintf(logfile,"   |-Checksum : %d\n",ntohs(icmph->checksum));
    //fprintf(logfile,"   |-ID      : %d\n",ntohs(icmph->id));
    //fprintf(logfile,"   |-Sequence : %d\n",ntohs(icmph->sequence));
    fprintf(logfile,"\n");

    fprintf(logfile,"IP Header\n");
    PrintData(Buffer,iphdrlen);

    fprintf(logfile,"UDP Header\n");
    PrintData(Buffer + iphdrlen , sizeof icmph);

    fprintf(logfile,"Data Payload\n");
    PrintData(Buffer + iphdrlen + sizeof icmph , (Size - sizeof icmph - iph->ihl * 4));

    fprintf(logfile,"\n###########################################################");
}

void PrintData (unsigned char* data , int Size)
{
    for(i=0 ; i < Size ; i++)
    {
       if( i!=0 && i%16==0)   //if one line of hex printing is complete...
       {
          fprintf(logfile,"         ");
          for(j=i-16 ; j<i ; j++)
          {
             if(data[j]>=32 && data[j]<=128)
                fprintf(logfile,"%c",(unsigned char)data[j]); //if its a number or alphabet

             else fprintf(logfile,"."); //otherwise print a dot
          }
          fprintf(logfile,"\n");
       }
       if(i%16==0) fprintf(logfile,"   ");
          fprintf(logfile," %02X",(unsigned int)data[i]);

       if( i==Size-1)  //print the last spaces
       {
          for(j=0;j<15-i%16;j++) fprintf(logfile,"   "); //extra spaces

          fprintf(logfile,"         ");

          for(j=i-i%16 ; j<=i ; j++)
          {
             if(data[j]>=32 && data[j]<=128) fprintf(logfile,"%c",(unsigned char)data[j]);
             else fprintf(logfile,".");
          }
          fprintf(logfile,"\n");
       }
    }
}
```

8 Gniazdo surowe (RAW) – transmisja danych

```
// PAS_4_gniazda_RAW_TCP.c
// UWAGA! gniazdo surowe wymaga uprawnien administratora
// Na podstawie:
// Silver Moon (m00n.silv3r@gmail.com)
// http://www.binarytides.com/raw-sockets-c-code-linux/
/*
 Naglowek IPv4:
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |Version|  IHL  |Type of Service|          Total Length         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |         Identification        |Flags|      Fragment Offset    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Time to Live |    Protocol   |         Header Checksum       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Source Address                          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    Destination Address                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    Options                    |    Padding    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 Naglowek TCP:
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          Source Port          |       Destination Port        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        Sequence Number                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    Acknowledgment Number                      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Data |           |U|A|P|R|S|F|                               |
 | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
 |       |           |G|K|H|T|N|N|                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |           Checksum            |         Urgent Pointer        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    Options                    |    Padding    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                             data                              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*/

#include <stdio.h>              //for printf
#include <string.h>             //memset
#include <sys/socket.h>         //for socket ofcourse
#include <stdlib.h>             //for exit(0);
#include <errno.h>              //For errno - the error number
#include <netinet/tcp.h>        //Provides declarations for tcp header
#include <netinet/ip.h>         //Provides declarations for ip header
#include <arpa/inet.h>          // inet_addr
#include <unistd.h>             // sleap()
/*
 96 bit (12 bytes) pseudo header needed for tcp header checksum calculation
 */
struct pseudo_header {
    u_int32_t source_address;
    u_int32_t dest_address;
    u_int8_t placeholder;
    u_int8_t protocol;
    u_int16_t tcp_length;
};

/*
 Generic checksum calculation function
 */
unsigned short csum(unsigned short *ptr, int nbytes) {
```

```
    register long sum;
    unsigned short oddbyte;
    register short answer;

    sum = 0;
    while (nbytes > 1) {
        sum += *ptr++;
        nbytes -= 2;
    }
    if (nbytes == 1) {
        oddbyte = 0;
        *((u_char*) &oddbyte) = *(u_char*) ptr;
        sum += oddbyte;
    }

    sum = (sum >> 16) + (sum & 0xffff);
    sum = sum + (sum >> 16);
    answer = (short) ~sum;

    return (answer);
}

int main(void) {
    //Create a raw socket
        int s = socket(PF_INET, SOCK_RAW, IPPROTO_TCP);
    //int s = socket(PF_INET, SOCK_RAW, IPPROTO_RAW);

    if (s == -1) {
        //socket creation failed, may be because of non-root privileges
        perror("Failed to create socket");
        exit(1);
    }

    //Datagram to represent the packet
    char datagram[4096], source_ip[32], *data, *pseudogram;

    //zero out the packet buffer
    memset(datagram, 0, 4096);

    //IP header
    struct iphdr *iph = (struct iphdr *) datagram;

    //TCP header
    struct tcphdr *tcph = (struct tcphdr *) (datagram + sizeof(struct ip));
    struct sockaddr_in sin;
    struct pseudo_header psh;

    //Data part
    data = datagram + sizeof(struct iphdr) + sizeof(struct tcphdr);
    strcpy(data, "ABCDEFGHIJKLMNOPQRSTUVWXYZ");

    //some address resolution
    strcpy(source_ip, "127.0.0.1"); //adres zrodlowy
    sin.sin_family = AF_INET;
    sin.sin_port = htons(80);
    sin.sin_addr.s_addr = inet_addr("127.0.0.1"); //adres docelowy
    strcpy(source_ip, "127.1.1.1"); //adres zrodlowy

    //Fill in the IP Header
    iph->ihl = 5;
    iph->version = 4;
    iph->tos = 0;
    iph->tot_len = sizeof(struct iphdr) + sizeof(struct tcphdr) + strlen(data);
    iph->id = htonl(54321); //Id of this packet
    iph->frag_off = 0;
    iph->ttl = 255;
    iph->protocol = IPPROTO_TCP;
    iph->check = 0; //Set to 0 before calculating checksum
    iph->saddr = inet_addr(source_ip); //Spoof the source ip address
```

```
    iph->daddr = sin.sin_addr.s_addr;

    //Ip checksum
    iph->check = csum((unsigned short *) datagram, iph->tot_len);

    //TCP Header
    tcph->source = htons(1234);
    tcph->dest = htons(80);
    tcph->seq = 0;
    tcph->ack_seq = 0;
    tcph->doff = 5; //tcp header size
    tcph->fin = 0;
    tcph->syn = 1;
    tcph->rst = 0;
    tcph->psh = 0;
    tcph->ack = 0;
    tcph->urg = 0;
    tcph->window = htons(5840); /* maximum allowed window size */
    tcph->check = 0; //leave checksum 0 now, filled later by pseudo header
    tcph->urg_ptr = 0;

    //Now the TCP checksum
    psh.source_address = inet_addr(source_ip);
    psh.dest_address = sin.sin_addr.s_addr;
    psh.placeholder = 0;
    psh.protocol = IPPROTO_TCP;
    psh.tcp_length = htons(sizeof(struct tcphdr) + strlen(data));

    int psize = sizeof(struct pseudo_header) + sizeof(struct tcphdr) + strlen(
            data);
    pseudogram = malloc(psize);

    memcpy(pseudogram, (char*) &psh, sizeof(struct pseudo_header));
    memcpy(pseudogram + sizeof(struct pseudo_header), tcph,
            sizeof(struct tcphdr) + strlen(data));

    tcph->check = csum((unsigned short*) pseudogram, psize);

    //IP_HDRINCL to tell the kernel that headers are included in the packet
    // Zamiast tego mozna uzyc gniazda:
    // int s = socket (AF_INET, SOCK_RAW, IPPROTO_RAW);
    // IPPROTO_RAW - oznacza ze zarowno IP jak i naglowki warstw wyzszych trzeba zbudowac

    int one = 1;
    const int *val = &one;

    //Jesli wylaczymy ponizsze naglowek ip i tcp beda budowane przez jadro systemu
#if 0
    if (setsockopt(s, IPPROTO_IP, IP_HDRINCL, val, sizeof(one)) < 0) {
        perror("Error setting IP_HDRINCL");
        exit(0);
    }
#endif
    //wyzylanie w petli
    while (1) {
        //Send the packet
        if (sendto(s, datagram, iph->tot_len, 0, (struct sockaddr *) &sin,
                sizeof(sin)) < 0) {
            perror("sendto failed");
        }
        //Data send successfully
        else {
            printf("Packet Send. Length : %d \n", iph->tot_len);
        }
        sleep(1);
    }

    return 0;
}
```

9 Procesy potomne

9.1 Procesy potomne – tworzenie cz. 1

```
/*
 ============================================================================
 Name        : PAS_1.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Procesy potomne
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //fork(), getpid()
#include <sys/types.h> //pid_t


int main(void) {

    //deklaracja zmiennych
    pid_t pid; // zmianna na identyfikator procesu (tak naprawde to int)
    int a = 1; //zmianna manipulacyjna

    //pobierz identyfikator procesu glownego (macierzystego)
    printf ("Identyfikator procesu glownego (macierzystego): %d \n", getpid());

    sleep (5);

    printf ("Utworz proces potomny (fork()) \n \n");
    //Uwaga pamiec procesu macierzystego zostala skopiowana do potomnego
    pid = fork ();

    if (pid == -1) {
        //jesteśmy w rodzicu, ale blad w utworzeniu nowego procesu
        printf ("Blad w utworzeniu procesu \n");
        exit (0); // przerywam program
    } else if (pid == 0){
        ////////////////////PROCES POTOMNY//////////////////////////
        printf ("--------------------\n");
        printf ("Identyfikator procesu potomnego: %d \n", getpid());
        //Jestem w procesie potomnym
        //Jaka jest wartosc zmiennej a?
        printf ("Proces potomny -> a: %d \n", a);
        //Zmieniam wartosc a w procesie macierzystym
        a = 2;
        printf ("Proces potomny (zmienilem a) -> a: %d \n", a);
        int b = 200;
        printf ("Dostep do zmiennej zainicjowanej rowniez w procesie mac. -> b: %d \n", b);
        ////////////////////PROCES POTOMNY//////////////////////////

    } else {
        ////////////////////PROCES MACIERZYSTY//////////////////////////
        printf ("--------------------\n");
        printf ("Identyfikator procesu macierzystego: %d \n", getpid());
        //Jestem w procesie macierzystym
        //Jaka jest wartosc zmiennej a?
        printf ("Proces macierzysty -> a: %d \n", a);
        //Zmieniam wartosc a w procesie potomnym
        a = 3;
        printf ("Proces macierzysty (zmienilem a) -> a: %d \n", a);

        int b = 100; //druga deklaracja
        printf ("Dostep do zmiennej zainicjowanej rowniez w procesie pot. -> b: %d \n", b);
        ////////////////////PROCES MACIERZYSTY//////////////////////////
    }
```

```
    sleep (5);
    printf ("==================\n");
    //Teraz wykonuje sie to samo w pbu procesach
    printf ("Odczyt w obu procesach (pid: %d) ->  a: %d \n",getpid (), a);
    //Zmieniam wartosc a w pamieci obu procesow:
    a = 10;
    printf ("Zmiana w obu proceach (pid: %d) teraz -> a: %d \n",getpid (), a);

    while (1) {
        sleep (10); // petla nieskonczona zeby zobaczyc pid w termianlu
    }

    return EXIT_SUCCESS;
}
```

9.2 Procesy potomne – tworzenie cz. 2

```
/*
 ============================================================================
 Name        : PAS_1_A.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Procesy potomne _ case zamiast if
 ============================================================================
 */
// INNY SPOSOB NA STRUKTUTE PROGRAMU Z FORK()
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //fork(), getpid()
#include <sys/types.h> //pid_t


int main(void) {

    //deklaracja zmiennych
    //pid_t pid; // zmianna na identyfikator procesu (tak naprawde to int)
    int a = 1; //zmianna manipulacyjna

    //pobierz identyfikator procesu glownego (macierzystego)
    printf ("Identyfikator procesu glownego (macierzystego): %d \n", getpid());

    sleep (5);

    printf ("Utworz proces potomny (fork()) \n \n");
    //Uwaga pamiec procesu macierzystego zostala skopiowana do potomnego

    switch(fork()) {
    case -1:
        //jesteśmy w rodzicu, ale blad w utworzeniu nowego procesu
        printf ("Blad w utworzeniu procesu \n");
        exit (0); // przerywam program (zamiast brake;)
    case 0:
        //jesteśmy w procesie potomnym
        ////////////////////PROCES POTOMNY//////////////////////////////
        printf ("--------------------\n");
        printf ("Identyfikator procesu potomnego: %d \n", getpid());
        //Jestem w procesie potomnym
        //Jaka jest wartosc zmiennej a?
        printf ("Proces potomny -> a: %d \n", a);
        //Zmieniam wartosc a w procesie macierzystym
        a = 2;
        printf ("Proces potomny (zmienilem a) -> a: %d \n", a);

        break;
        ////////////////////PROCES POTOMNY//////////////////////////////

    default:
        //jesteśmy w procesie macierzystym
        ////////////////////PROCES MACIERZYSTY//////////////////////////
        printf ("--------------------\n");
        printf ("Identyfikator procesu macierzystego: %d \n", getpid());
        //Jestem w procesie macierzystym
        //Jaka jest wartosc zmiennej a?
        printf ("Proces macierzysty -> a: %d \n", a);
        //Zmieniam wartosc a w procesie potomnym
        a = 3;
        printf ("Proces macierzysty (zmienilem a) -> a: %d \n", a);

        break;

        ////////////////////PROCES MACIERZYSTY//////////////////////////
        }
```

```
    sleep (5);
    printf ("==================\n");
    //Teraz wykonuje sie to samo w pbu procesach
    printf ("Odczyt w obu procesach (pid: %d) ->  a: %d \n",getpid (), a);
    //Zmieniam wartosc a w pamieci obu procesow:
    a = 10;
    printf ("Zmiana w obu proceach (pid: %d) teraz -> a: %d \n",getpid (), a);

    while (1) {
        sleep (10); // petla nieskonczona zeby zobaczyc pid w termianlu
    }

    return EXIT_SUCCESS;
}
```

9.3 Wielokrotne tworzenie procesów potomnych

```
/*
 ============================================================================
 Name        : PAS_2.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : Do wykorzystania w ramach zajec
 Description : Wielokrotne tworzenie procesow potomnych - dziecko ma dziecko
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //fork(), getpid()
#include <sys/types.h> //pid_t


int main(void) {

    //deklaracja zmiennych
    pid_t pid; // zmianna na identyfikator procesu (tak naprawde to int)

    sleep (1);

    printf ("Utworz proces potomny (fork()) \n \n");
    pid = fork ();

    if (pid == 0){
        ////////////////////PROCES POTOMNY//////////////////////////////
        printf ("-------------------\n");
        printf ("Identyfikator procesu potomnego: %d \n", getpid());
        //Jestem w procesie potomnym 1
        //.....
        ////////////////////PROCES POTOMNY//////////////////////////////

    } else {
        ////////////////////PROCES MACIERZYSTY////////////////////////
        //|
        printf ("==============\n");
        printf ("Identyfikator procesu macierzystego: %d \n", getpid());
        //Jestem w procesie macierzystym
        //|.....
        ////////////////////PROCES MACIERZYSTY////////////////////////
        //utworze nowy proces potomy
        if (fork()==0) {
            //+++++++NOWY PROCES POTOMNY++++++++++++
            printf ("++++++++++++++\n");
            printf ("Identyfikator nowego procesu potomnego: %d \n", getpid());
            //Jestem w procesie potomnym 2
            //.....
            //+++++++NOWY PROCES POTOMNY++++++++++++
        } else {
            ////////////////////PROCES MACIERZYSTY////////////////////////
            printf ("==============\n");
            printf ("Identyfikator procesu macierzystego: %d \n", getpid());
            //....
            ////////////////////PROCES MACIERZYSTY////////////////////////
        }
    }


    while (1);

    return EXIT_SUCCESS;
}
```

9.4 Kończenie procesów potomnych

```
/*
 ============================================================================
 Name        : PAS_2.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Konczenie procesow potomnych
 ============================================================================
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //fork(), getpid()
#include <sys/types.h> //pid_t
#include <sys/wait.h>  //wait()

int main(void) {

    // ZAMYKANIE (KONCZENIE) PROCESOW  - synchronizacja zakonczenia
    // !!!! W pierwszej kolejnosci koncz procesy potomne a potem macierzyste

    //deklaracja zmiennych
    pid_t pid, id; // zmianna na identyfikator procesu (tak naprawde to int)

    sleep (1);
    int i; // zmienna pomocnicza
    int status; //zmienna statusu zakoczenia procesu

    printf ("Utworz proces potomny (fork()) \n \n");
    pid = fork ();



    if (pid == 0){
        ////////////////////PROCES POTOMNY//////////////////////////////
        printf ("-------------------\n");
        printf ("Identyfikator procesu potomnego: %d \n", getpid());
        //Jestem w procesie potomnym 1

        printf ("pid%d: Proces potomny wykonuje sie powoli.\n", getpid());

        //wywolanie instruckji  systemowej: wyswietli procesy zaczynajace sie od "PAS"
        system ("ps -A | grep PAS");

        for (i=0; i<10; i++) {
            printf (".\n");
            sleep (2);
        }
        printf ("pid%d: Koncze proces potomny z kodem wyjscia %d\n", getpid(), 0xaa);
        exit (0xaa);  //kod powrotu 0xaa = 170
        ////////////////////PROCES POTOMNY//////////////////////////////

    } else {
        ////////////////////PROCES MACIERZYSTY//////////////////////////
        //|
        printf ("==============\n");
        printf ("Identyfikator procesu macierzystego: %d \n", getpid());
        //Jestem w procesie macierzystym
        //|.....


        // gdyby proces potomny skonczyl sie zanim proces macierzysty dotrze do 'wait()',
        // proces potomny staje sie tzw. procesem zombi (nieczynnym), ktory powinien sie zakonczyc
        // w momencie wywolania fukcji 'wait()'
#if 0
        sleep (20); // mozna to sprawdzic dodajac opoznienie proceu macierzystego
#endif
```

```
        printf ("pid%d: Wstrzymuje proces macierzysty do zakonczenia procesu potomnego\n", getpid());
        id = wait (&status);

        //wywolanie instruckji  systemowej: wyswietli procesy zaczynajace sie od "PAS"
        system ("ps -A | grep PAS");

        // pod zmienna ststus ukryty jest kod powrotu exit() z proc. potomnego w przedostatnim bajcie ('0xaa')
        // oststni bajt dodaje wystem operacyjny: przyczyna zakonczenia procesu potomnego
        printf ("pid%d: Koncze proc. macierzysty po zakonczeniu procesu potomnego, ktorego pid byl %d (status:0x%x)\n", getpid(),id,status);

        // istnieja makra (w stdlib.h) do sprawdzenia przyczyny zakonczenia procesu potomnego
        if (WIFEXITED(status)) {
            printf("Proces potomny zakonczony exitem, status=%d\n", WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf("Proces potomny zabity sygnalem %d\n", WTERMSIG(status));
        } else if (WIFSTOPPED(status)) {
            printf("Proces potomny zakonczony sygnalem %d\n", WSTOPSIG(status));
        } else if (WIFCONTINUED(status)) {
            printf("Proces potomny dziala dalej\n");
        }


        exit (0);
        ////////////////////PROCES MACIERZYSTY//////////////////////
    }

    printf ("to juz sie nie wyswietli\n");

    return EXIT_SUCCESS; // to tez nie bedzie wykorzystane
}
```

10 Wątki

```
/*
 ============================================================================
 Name        : PAS_6_watki.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Watki
 ============================================================================
 */
//
//Funkcje
/*
pthread_create                  Tworzenie watku
pthread_exit                    Zakonczenie wątku biezącego
pthread_join                    Czekanie na zakończenie watku
pthread_self                    Pobranie identyfikatora biezacego watku
pthread_cancel                  Kasowanie innego watku
.....................................................
pthread_attr_init       Inicjacja atrybutów wątku
*/

//UWAGA: dodac biblioteke do linkera: -l pthread (wlasciwosci-settings-linker libraries) lub w czasie kompilacji
//cc  -o"PAS_6_watki"  ./src/PAS_6_watki.o   -l pthread
// wyswietlenie w konsoli watkow: ps -eL | grep PAS

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>  //fukcje watkow
#include <unistd.h>   //sleep

// deklaracja funkcji obslugi watku
// przygotuj je w pierwszej kolejnosci
void* fukcja_obslugi_watku1 ();
void* fukcja_obslugi_watku2 ();

int main(void) {
    puts("Watki: threads");

    int i;
    int kod_powrotu;   //zmienna do przechowywania wartosci fukcji pthread_create ()
    pthread_t thread1, thread2;    // id watkow: typ zmiennej zdefiniowany w  <pthread.h>
    int argument_przekazany_do_watku1 = 10;
    int argument_przekazany_do_watku2 = 20;
    int wynik; // wynik zwrocony przez watek

    // utworz watek
    /* int pthread_create(pthread_t *id, const pthread_attr_t *attr, void* (*fun)(void*), void* arg)
    * id - identyfikator wątku;
    * attr - wskaźnik na atrybuty wątku, można podać NULL - zostaną użyte domyślne wartości;
    * fun - funkcja wykonywana w wątku; przyjmuje argument typu void* i zwraca wartość tego samego typu;
    * arg - przekazywany do funkcji.
    */

    printf ("proces: %d, ID watku w0: %lu\n", getpid (), pthread_self());

#if 0
    pthread_attr_t attr;  // atrybuty obslugi watku
    int    detachstate;  // do sprawdzenia statusu zwolnienia pamiaci dla watka
    //inicjacja atrybutow watku - gdy ich nie bedzie mozna przyjac standardowe (przekazac NULL do 'pthread_create')
    pthread_attr_init(&attr);
    pthread_attr_getdetachstate(&attr, &detachstate); // sprawdzenia standardowego statusu zwolnienia pamiaci dla watka
    if (detachstate == PTHREAD_CREATE_JOINABLE)
        printf ("Standardowy argument: PTHREAD_CREATE_JOINABLE\n");  // pamiec watkow sie zwolni po osiagnieciu funkcji 'pthread_join'
    else if (detachstate == PTHREAD_CREATE_JOINABLE)
        printf ("Standardowy argument: PTHREAD_CREATE_DETACHED\n");  // pamiec watkow sie zwolni zaraz po zakonczeniu watka
'pthread_exit'
```

```
    else
        printf ("w zasadzie innego satusu odlaczenia nie ma i to nie zostanie wypisane\n");
    //moge zmianic status zwalniania pamieci poprzez zmiane atrybutu

    pthread_attr_setdetachstate (&attr, PTHREAD_CREATE_DETACHED);
    // i przekazanie attr do 'pthread_create'
    // kod_powrotu = pthread_create( &thread1, &attr,  &fukcja_obslugi_watku1, &argument_przekazany_do_watku1);
    // po czym zwolnienie pamieci atrybutu:
    pthread_attr_destroy(&attr);
#endif


    kod_powrotu = pthread_create( &thread1, NULL,  &fukcja_obslugi_watku1, &argument_przekazany_do_watku1);
    if(kod_powrotu){
        printf("Error - pthread_create() zwraca kod: %d\n",kod_powrotu);
        exit(EXIT_FAILURE);
    } else {
         printf("watek1: %lu pthread_create() zwraca: %d\n", (unsigned long int) thread1, kod_powrotu);
    }
    kod_powrotu = pthread_create( &thread2, NULL, &fukcja_obslugi_watku2, &argument_przekazany_do_watku2);
        if(kod_powrotu){
            printf("Error - pthread_create() zwraca kod: %d\n",kod_powrotu); // kod powrotu inny niz 0 jesli blad
            exit(EXIT_FAILURE);
        } else {
            printf("watek2: %lu pthread_create() zwraca: %d\n", (unsigned long int) thread2, kod_powrotu);  // kod powrotu 0 jesli sukces
        }

#if 0
        //skasowanie innego watku
        //wykonac na koniec
        //unikac kasowania innych watkow przed ich zakonczeniem
        pthread_cancel(thread1);
        pthread_cancel(thread2);
#endif


    for (i=0; i< 30; i++) {
        printf ("w0\n");
        sleep (1);
    }

    //poczekamy na zakonczenie watkow w1 i w2 przed zakonczeniem funkcji main (watku 0)
    pthread_join( thread1, NULL);
    // watek w1 nie zwrocil nic
    pthread_join( thread2, (void*)&wynik);
    // watek w2 zwrocil wynik
    printf ("Po skonczeniu watek 2 zwrocil: %d \n", wynik);

    for (i=0; i< 5; i++) {
        printf ("w0\n");
        sleep (1);
    }

    printf ("Koncze program glowny (watek 0) \n");
    return EXIT_SUCCESS;
}


void* fukcja_obslugi_watku1 (void *arg) {

    int *ptr;
    ptr =  (int *) arg;
    int i;
    int liczba_kropek = 0;
    pthread_t id_watku;

    liczba_kropek = (*ptr);

    printf ("Wykonuje watek 1 (liczba przekazana:%d)\n", liczba_kropek);
```

```
    // pobierz id tego watku
    id_watku = pthread_self();
    printf ("proces: %d, ID watku w1: %lu\n", getpid (), id_watku);  // id_watku jest (unsigned long int), dlatego 'lu'

    sleep (1);

    for (i=0; i< liczba_kropek; i++) {
        printf ("w1\n");
        sleep (1);
    }

    printf ("Koncze watek 1\n");
    return NULL;
}

void* fukcja_obslugi_watku2 (void *arg) {

    int *ptr;
    ptr =  (int *) arg;
    int i;
    int liczba_kropek = 0;
    int wynik;
    pthread_t id_watku;

    liczba_kropek = (*ptr);

    printf ("Wykonuje watek 2 (liczba przekazana:%d)\n", liczba_kropek);

    // pobierz id tego watku
    id_watku = pthread_self();
    printf ("proces: %d, ID watku w2: %lu \n", getpid (), id_watku);

    sleep (1);

#if 0
    //wczesniejsze zakoczenie watku
    printf ("Przerwalem watek 2 'pthread_exit()' \n");
    pthread_exit(NULL);
#endif
    //lub
#if 0
    //wczesniejsze zakoczenie watku
    printf ("Przerwalem watek 2 (RETURN) \n");
    return NULL;
#endif

    for (i=0; i< liczba_kropek; i++) {
        printf ("w2\n");
        sleep (1);
    }

    wynik = 100;
    printf ("Koncze watek 1\n");
#if 0
    //zwrocenie wyniki przez watek
    return ((void*) wynik);
#endif
    // lub
#if 1
    //zwrocenie wyniki przez watek
    pthread_exit((void*) wynik);
#endif
}
```

11 Zabezpieczenie przed zakleszczeniem w wątkach (MUTEX)

```
/*============================================================================
 Name        : PAS_6A_mutex.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : J. Krygier
 Description : Watki Mutexy
 ============================================================================
 */

#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

#define WLACZ_MUTEX 1

int zmiennaglobalna=0;

#if WLACZ_MUTEX
pthread_mutex_t mojmuteks=PTHREAD_MUTEX_INITIALIZER;
#endif

void *Watek1(void *arg) {
    int i;
    for ( i=0; i<20; i++ ) {

#if WLACZ_MUTEX
        pthread_mutex_lock(&mojmuteks);
#endif
        printf("1(%d)\n", zmiennaglobalna);
        fflush(stdout);

        zmiennaglobalna=zmiennaglobalna+1;

        sleep(1);

#if WLACZ_MUTEX
        pthread_mutex_unlock(&mojmuteks);
#endif
    }
    return NULL;
}

int main(void) {
    pthread_t w1;
    int i;
    if ( pthread_create( &w1, NULL, Watek1, NULL) ) {
        printf("Blad przy tworzeniu watku.");
        abort();
    }

    for ( i=0; i<20; i++) {
#if WLACZ_MUTEX
        pthread_mutex_lock(&mojmuteks);
#endif
        printf("0(%d)\n", zmiennaglobalna);
        fflush(stdout);

        zmiennaglobalna=zmiennaglobalna-1;
#if WLACZ_MUTEX
        pthread_mutex_unlock(&mojmuteks);
#endif

        sleep(1);
    }
    if ( pthread_join (w1, NULL ) ) {
```

```
        printf("Blad przy konczeniu watku w1.");
        abort();
    }
    printf("\nMoja zmienna globalna wynosi %d\n",zmiennaglobalna);
    exit(0);
}
```

12 Serwer wielowątkowy

```
/*
 ============================================================================
 Name        : PAS_7_serwer_wielowatkowy.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : na podstawie: http://www.binarytides.com/socket-programming-c-linux-tutorial/
 Description : Serwer wielowatkowy
 ============================================================================
 */

#include <stdio.h>
#include <string.h>    //strlen
#include <stdlib.h>    //strlen
#include <sys/socket.h>            //gniazda
#include <arpa/inet.h>             //inet_addr
#include <unistd.h>    //write
#include <pthread.h> //watki, pamietac o lpthread

//pamietac o dodaniu pthread jako opcji linkera: gcc program.c -l pthread
// wyswietlenie w konsoli watkow: ps -eL | grep PAS\

// wiele informacji mozna znalezc tutaj: http://www.binarytides.com/socket-programming-c-linux-tutorial/

// deklaracja fukcji
void *watek_obslugi_polaczenia(void *);  // funckajs - watek obslugi polaczenia

int main(void) {

   int     deskryptor_gniazda , nowe_gniazdo , c;
   int     *nowe_gniazdo_ptr;
   struct     sockaddr_in strukt_gniazdo_serwera; // struktura adresacji gniazda serwera
   struct     sockaddr_in strukt_gniazdo_klienta; // struktura adresacji gniazda klienta
   char   *wiadomosc;
   int     optval;

   //Create socket
   deskryptor_gniazda = socket(AF_INET , SOCK_STREAM , 0);
   if (deskryptor_gniazda == -1) {
      printf("SERWER: blad funkcji soclet() \n");
   }

   //zapewnienie nieblokowalnosci gniazda po zwolnieniu
   optval = 1;
   setsockopt(deskryptor_gniazda, SOL_SOCKET, SO_REUSEADDR, (const void *)&optval , sizeof(int));  // w socket.h


   //Przygotowanie adresacji gniazda
   strukt_gniazdo_serwera.sin_family = AF_INET;                              //IPv4
   strukt_gniazdo_serwera.sin_addr.s_addr = INADDR_ANY;          //dowolny klinet moze sie polaczyc
   strukt_gniazdo_serwera.sin_port = htons( 8888 );                  //na jakim porcie nasluchuje serwer



   //Powiaz deskryptor gniazda z adresem gniazda
   if( bind(deskryptor_gniazda,(struct sockaddr *)&strukt_gniazdo_serwera ,
         sizeof(strukt_gniazdo_serwera)) < 0) {
    printf ("SERWER: blad funkcji bind() \n");
      return 1;
   } else {
    printf ("SERWER: Polaczono sie z gniazdem. \n");
   }

   //Nasluchiwanie na porcie
   listen(deskryptor_gniazda , 3);

   //Akceptuj połaczenia
```

```
    printf ("SERWER: Czekam na klianta ... \n");

    c = sizeof(struct sockaddr_in); //rozmiar struktury dla adresacji gniazda klienta

    while( (nowe_gniazdo = accept(deskryptor_gniazda, (struct sockaddr *) &strukt_gniazdo_klienta,
        (socklen_t*) &c)) ) {

     printf ("SERWER: Polaczenie zaakceptowano. \n");

       //Odpowiedz do klienta
       wiadomosc = "Polaczyles sie z serwerem TCP. Przydzielono watek obslugi polaczenia ...\n";

       // wyslij do klienta wiadomosc
       write(nowe_gniazdo , wiadomosc , strlen(wiadomosc));

       pthread_t id_watku; // deklaracja id_watku

       // jako argument do fukncji watku przekazemy wskaznik na strukture_adresu_gniazda
       // do komunikacji z klientem utworzonego przez 'accept()'

       nowe_gniazdo_ptr = malloc(sizeof(int));
       *nowe_gniazdo_ptr = nowe_gniazdo;
#if 1
       if (pthread_create( &id_watku , NULL ,  watek_obslugi_polaczenia ,
                    (void*) nowe_gniazdo_ptr) < 0) {
         printf ("SERWER: Nie moge utworzyc watku. \n");
           return 1;
       }
#endif
    }

    //Czekamy na zakoczenie watku
    //pthread_join( id_watku, NULL);

    if (nowe_gniazdo < 0)
    {
     printf ("SERWER: Nie zaakceptowano polaczenia. \n");
       return 1;
    }

    return 0;
}

/*
 * Obsluga polaczenia przez watek
 * */
void *watek_obslugi_polaczenia(void *gniazdo_do_komunikacji_z_klientem_ptr)
{
    //Get the socket descriptor
    int gniazdo;
    int i, rozmiar_wiad;
    char *wiadomosc;
    char wiadomosc1 [100];

    gniazdo = *(int*)gniazdo_do_komunikacji_z_klientem_ptr;

    bzero(wiadomosc1, 100);
    //Wyslemy cos do klienta
    sprintf (wiadomosc1, "SERWER: Obsluguje cie watek z id gniazda: %d \n", gniazdo);
    //wiadomosc = "SERWER: Obsluguje cie watek\n";
    write(gniazdo , wiadomosc1 , 100);
    //write(gniazdo , wiadomosc , strlen(wiadomosc));

    for (i=0;i<5; i++) {
     wiadomosc = ".";
     write(gniazdo , wiadomosc , strlen(wiadomosc));
     sleep(1);
     wiadomosc = "_";
     write(gniazdo , wiadomosc , strlen(wiadomosc));
```

```
    sleep(1);
   }

#if 0
   //jak chcesz dluzej pogadac z klientem to tutaj
   //pobierz wiadomosc od klienta
   bzero(wiadomosc1, 100);
   while( 1 ) {
    wiadomosc = "\nMozesz cos napisac...\n";
    write(gniazdo , wiadomosc , strlen(wiadomosc));
    rozmiar_wiad = recv(gniazdo , wiadomosc1 , 2000 , 0);
      //wyslij echo
    wiadomosc = "\nOdebralem: ";
    write(gniazdo , wiadomosc , strlen(wiadomosc));
      write(gniazdo , wiadomosc1 , strlen(wiadomosc1));
      bzero(wiadomosc1, 100);
   }
#endif

   //rozlacz
   close (gniazdo);
   //zwolnij wskaznik
   free(gniazdo_do_komunikacji_z_klientem_ptr);
   printf ("SERWER: Zakonczono polaczenie. \n");
   return 0;
}
```

13 Serwer – wykorzystanie funkcji select()

```
/*
 ============================================================================
 Name        : PAS_8_select.c
 Author      : J. Krygier
 Version     : 1.0
 Copyright   : Na podstawie: http://www.binarytides.com/multiple-socket-connections-fdset-select-linux/
 Description : Wykorzystanie select do polaczen z gniazdami. Serwer TCP
 ============================================================================
*/

// UWAGA: W celu spradzenia dzialania, lacz sie z serwerem za pomoca Telnet

#include <stdio.h>
#include <string.h>                 //strlen
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>                 //close
#include <arpa/inet.h>              //inet
#include <sys/types.h>                      //typy zmiennych
#include <sys/socket.h>             //gniazda
#include <netinet/in.h>                     //adresy gniazd
#include <sys/time.h>                       //Makra FD_SET, FD_ISSET, FD_ZERO

#define TRUE   1
#define FALSE  0
#define PORT 8888

int main(void) {

   int opt = TRUE;
   int master_socket , new_socket;   //deskryptor gniazda
   int addrlen;
   int client_socket[30];                                   //tablica na gniazda klientow
   int max_clients = 30 , activity, i , valread , sd;
   int max_sd;
   struct sockaddr_in address;     //struktura na adres gniazda

   char buffer[1025];  //bufor

   fd_set readfds; //ustawienia deskryptora pliku dla fukcji select (select.h)

   //wiadomosc
   char *message = "SERWER TCP: Serwer odpowiada echem. Napisz cos.\r\n";

   //inicjalizacja: w client_socket[], same 0
   for (i = 0; i < max_clients; i++) {
      client_socket[i] = 0;
   }

   //utworzenie gniazda strumieniowego (TCP) - do nasluchu
   if( (master_socket = socket(AF_INET , SOCK_STREAM , 0)) == 0) {
      perror("blad w socket ()");
      exit(EXIT_FAILURE);
   } else {
    printf ("SERWER: Gniazdo nasluchujace: %d \n", master_socket);
   }

   // odblokujemy gnizdo zaraz po zamknieciu do ponownego uzycia
   //zezwolenie na wiele polaczen z gniazdem
   if( setsockopt(master_socket, SOL_SOCKET, SO_REUSEADDR, (char *)&opt, sizeof(opt)) < 0 ) {
      perror("blad w setsockopt()");
      exit(EXIT_FAILURE);
   }

   //Ustawienie adresu gniazda
   address.sin_family = AF_INET;
```

```
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

//skojarzenie utwozronego gniazda z adresem gniazda
if (bind(master_socket, (struct sockaddr *)&address, sizeof(address))<0) {
   perror("blad bind ()");
   exit(EXIT_FAILURE);
}
printf("SERWER: Nasluchuje na porcie %d \n", PORT);

// Maksimum 3 aktywne polaczenia na tym gniezdzie
if (listen(master_socket, 3) < 0) {
   perror("blad listen()");
   exit(EXIT_FAILURE);
}

// Akceptacja przychodzacego polaczenia
addrlen = sizeof(address);
puts("SERWER: Czekam na polaczenie ...");

while(TRUE) {
   // wyczyszczenie ustawien deskryptora pliku
   FD_ZERO(&readfds);   // FD_ZERO - makro w select.h

   // dodanie gniazda nasuchujacego w celu wysterowania jego ustawien przez f. select
   FD_SET(master_socket, &readfds);
   max_sd = master_socket;

   // Dodanie gniazd klientow (jak sie pojawia to beda zapisane w client_socket[])
   // w celu wysterowania ich ustawien przez f. select
   for ( i = 0 ; i < max_clients ; i++) {
      //odczytanie deskrypora gniaza z client_socket[i]
      sd = client_socket[i];

      // Jesli kolejny i-ty klient sie polaczy to w client_socket[i] deskryptor gniazda bedzie > 0
      // i nastapi dodanie deskrytora do ustawien select
      if(sd > 0)
         FD_SET( sd , &readfds);

      //znalezenie najwiekszego przydzielonego dla klienta deskryptora gniaza
      if(sd > max_sd)
         max_sd = sd;
   }

   // czakaj/sprawdzaj aktwynosc na gniazdach klientow
   // timeout is NULL , so wait indefinitely
   activity = select( max_sd + 1 , &readfds , NULL , NULL , NULL);

   if ((activity < 0) && (errno!=EINTR)) {
      printf("blad select ()");
   }

   //Jesli cos wykryto w gniezdzie nasuczujacym, oznacza to nowe polaczenie
   if (FD_ISSET(master_socket, &readfds)) {
      if ((new_socket = accept(master_socket, (struct sockaddr *)&address, (socklen_t*)&addrlen))<0) {
         perror("blad accept ()");
         exit(EXIT_FAILURE);
      }

      //Wypiszemy info o kliencie
      printf("SERWER: Nowe polaczenie. \n");
      printf("SERWER: Deskryptor gniazda do komunikacji z klientem:  %d , Adres IP klienta: %s , Nr portu klienta: %d \n" ,
            new_socket , inet_ntoa(address.sin_addr) , ntohs(address.sin_port));

      //wysli powitanie do klinta
      if( send(new_socket, message, strlen(message), 0) != strlen(message) ) {
         perror("blad w send ()");
      }
```

```
            puts("SERWER: Wyslano powitanie do klienta");

            //Dodamy teraz gnizado do komunikacji z klientem do tablicy gniazd: client_socket[]
            for (i = 0; i < max_clients; i++) {
               //na wolnej pozycji
               if( client_socket[i] == 0 ) {
                  client_socket[i] = new_socket;
                  printf("SERWER: Dodano gniazdo do komunikacji z klinetem na pozycje %d\n" , i);

                  break;
               }
            }
         }

         // sparwdz czy cos sie zmienilo na innych aktywnych gniazdach
         for (i = 0; i < max_clients; i++) {
            sd = client_socket[i];

            if (FD_ISSET( sd , &readfds)) {
               //Jesli czekaja tam dane odczytaj je, ajsli klient zamknal polaczenie zareaguj
               if ((valread = read( sd , buffer, 1024)) == 0) {
                  //read zwraca 0 - to znaczy polczenie zostalo zamkniete przez klienta
                  // odcztaj dane klineta: getpeername ()
                  getpeername(sd , (struct sockaddr*)&address , (socklen_t*)&addrlen);
                  printf("SERWER: Klient o adresie %s (port: %d) sie rozlaczyl \n" , inet_ntoa(address.sin_addr) , ntohs(address.sin_port));

                  //zamknij gniazdo do komunikacji z klientem i wyzeruj wpis w client_socket[]
                  close( sd );
                  client_socket[i] = 0;
               } else {
                  // Wyslij echo
                  // na koncu dodaj '/n'
                  buffer[valread] = '\0';
                  send(sd , buffer , strlen(buffer) , 0 );
               }
            }
         }
      }

      return 0;
   }
```