

Programowanie sieciowe

email: dawid@us.edu.pl

Programowanie warstwy transportu.

- Berkeleysockets.
 - koncepcjaniazda
 - biblioteksockets (connect(),socket(),bind(),listen (),accept(),read(),write())
 - select();
- Winsock.
 - operacjeasynchroniczne
 - operacjenakładane
- NETBIOS
- JęzykC/C++.
- Język Java

Komunikacja między procesami.

- Skrzynki(mailslots)
- Łączanazwane(named pipes).

Labolatorium.

C/C++, Assembler ->(Win32VC++6.0, Linux GCC)

A.Dawid – Uniwersytet Śląski

Bibliografia

M. Gabassi,B. Dupouy „Przetwarzanie rozproszonego systemu UNIX”

W. Mielczarek „Szeregowe interfejsy cyfrowe”

Stevens,,Programowanie zastosowania sieciowych w systemie UNIX”,

WNT,Warszawa 1995

Matthew, Stones,,LINUX:Programowanie, RM”,Warszawa 1999

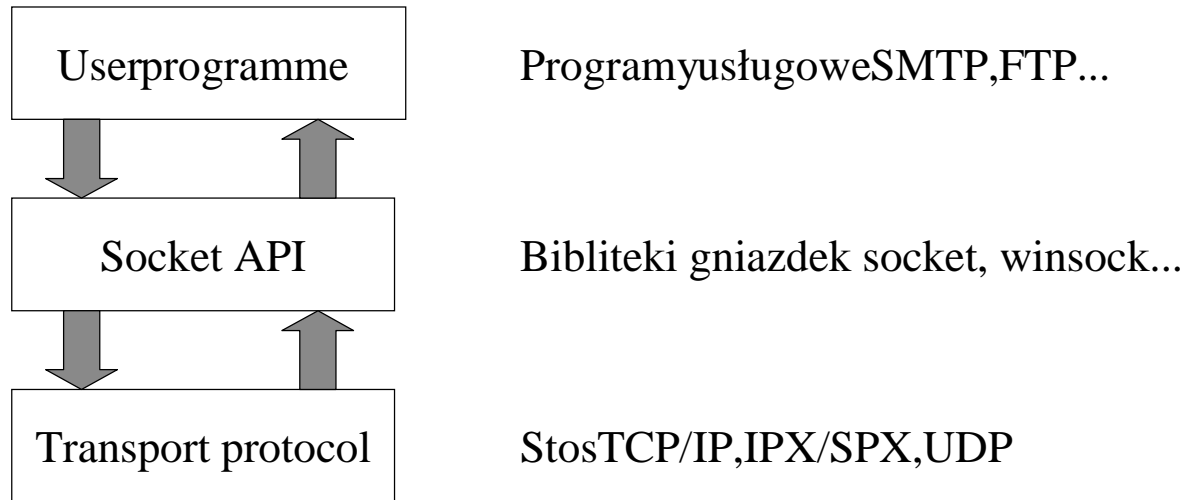
A. Jones,J. Ohlund „Programowanie Sieciowe MS Windows”

A. Sopala „Pisanie programów internetowych”

Czasopisma i Internet.

Gniazdka(sockets).

Warstwa oprogramowania znajdująca się między warstwą oprogramowania użytkownika a warstwą oprogramowania sieci.



API - Application Program Interface

Gniazdka(sockets).

Gniazdko to końcowy punkt kanału komunikacji.

Punkt ten jest identyfikowany za pomocą zmiennej całkowitej i dyskryptora pliku.

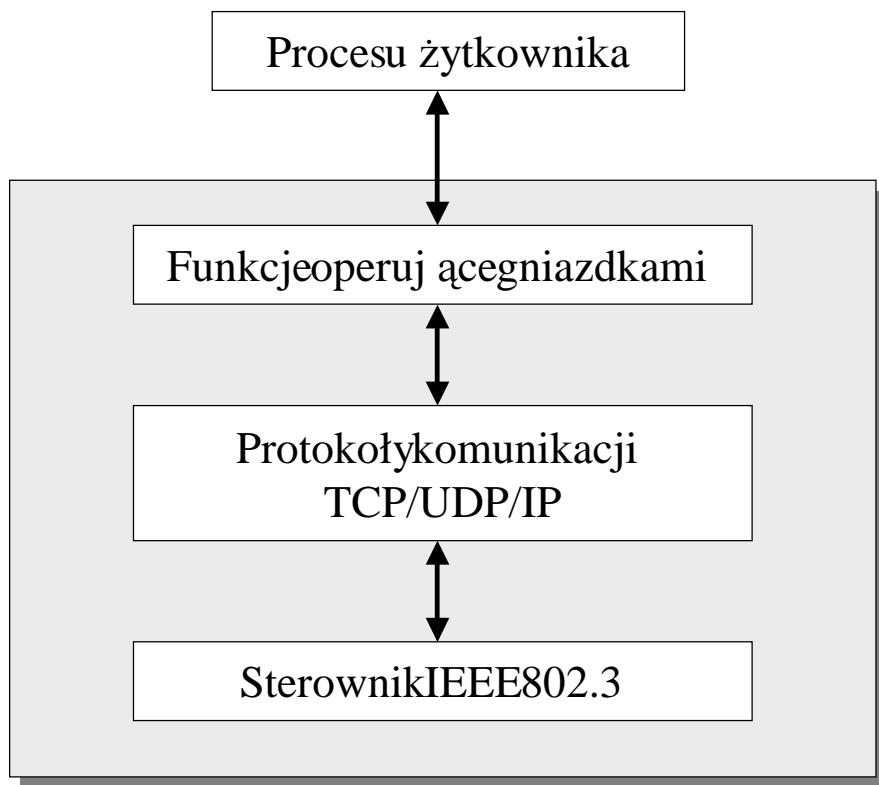
Biblioteka gniazdek maskuje warstwę transportu sieci.

Podobieństwo do dostępu do pliku.

HISTORIA

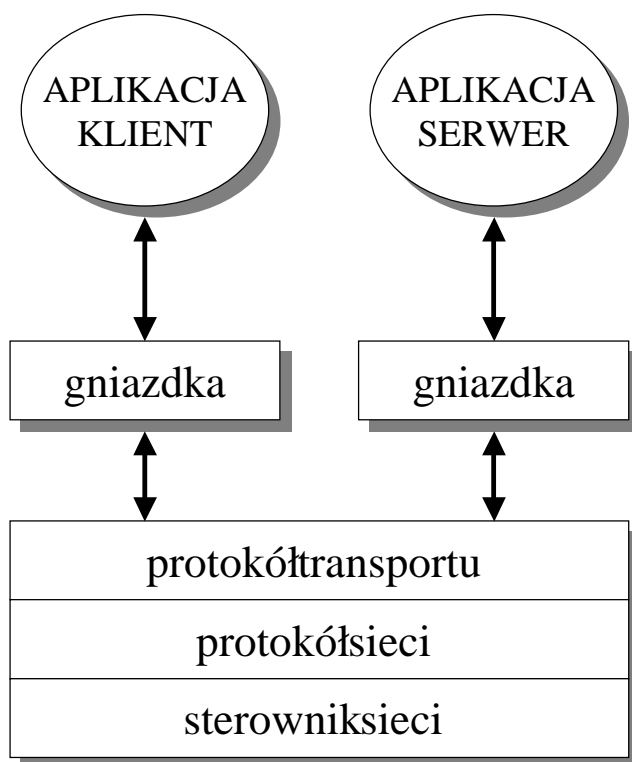
Gniazdko zostało zaimplementowane po raz pierwszy w systemie UNIX BSD 4.2 wraz z protokołem TCP/IP

Gniazdka(sockets).



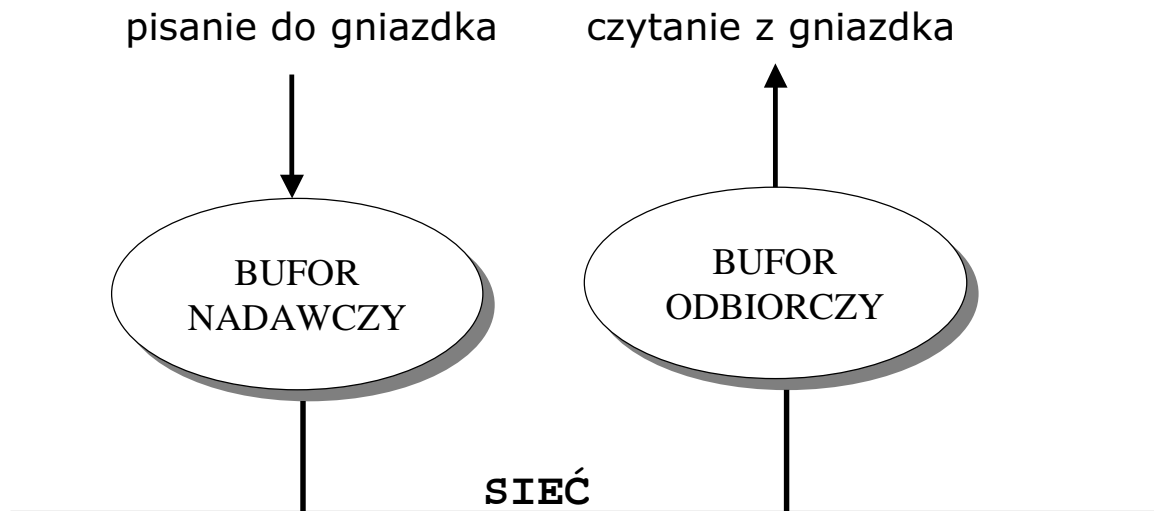
Realizacja gniazdek w BSD

Gniazdka(sockets).



Architektura klient-serwer dla gniazdek

Gniazdko(sockets).



Bufory transmisji TCP

Domyślna wartość tych buforów waha się od 4k do 16k
Windows - 8192. Maksymalna wartość to 64k 16bit

Gniazdko(sockets).

Domyślna wartość tych buforów waha się od 4k do 16k
Windows - 8192. Maksymalna wartość to 64k 16bit.
Parametr oznaczający programowo `setsockopt(...)`

Korzystanie z gniazdek

1. Dziedzina w jakiej gniazdko pracuje.

UNIX	(AF_UNIX)	Unix
TCP/IP	(AF_INET)	winsck, Unix
IPX/SPX	(AF_IPX)	winsck

1. Typ protokołu.

SOCK_DGRAM - datagramy, brak kontroli transmisji
dziedzina AF_INET, AF_IPX

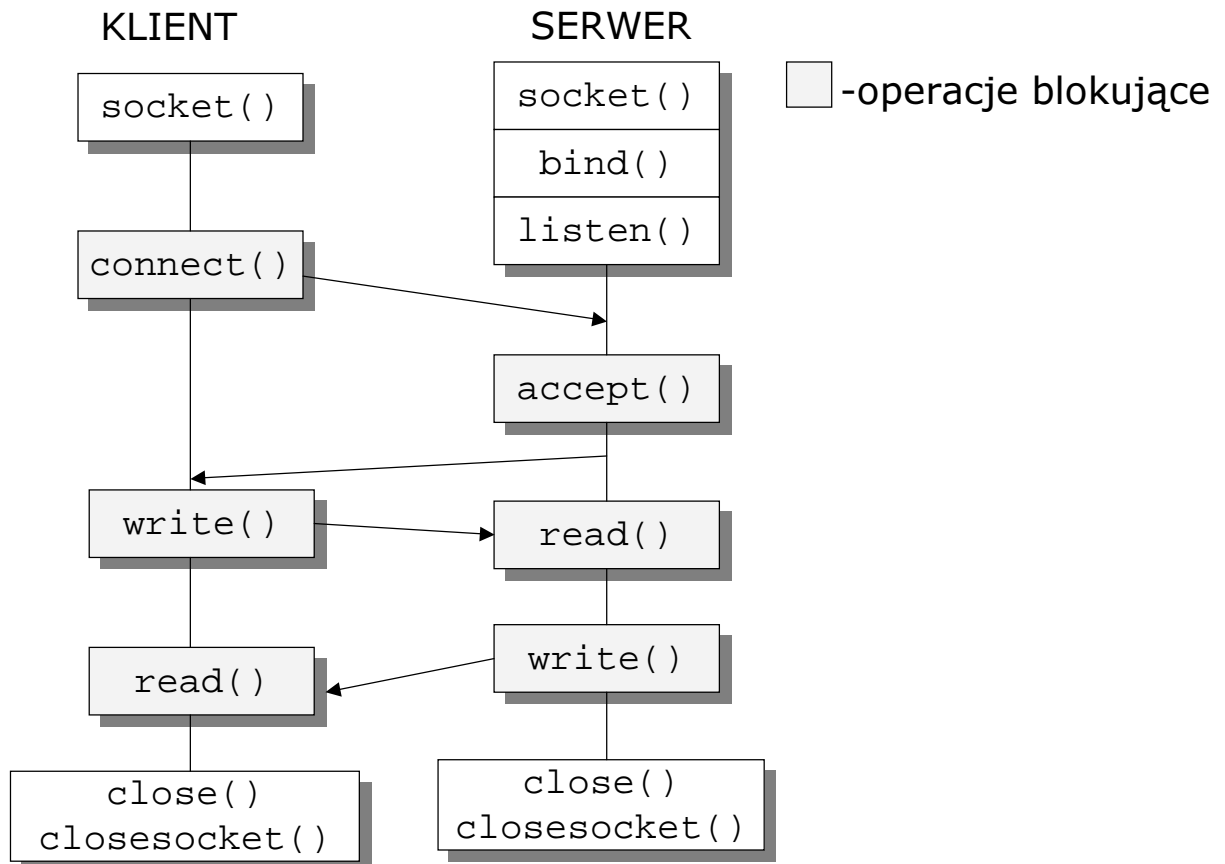
Gniazdko(sockets).

SOCK_STREAM - strumienie, kontrola transmisji
dziedzina AF_INET, AF_IPX

SOCK_RAW - pozwala na dostęp do niższych warstw
protokołów takich jak IP w
dziedzinie AF_INET, lub na realizację
nowych protokołów.

Użycie gniazdek w trybie z kontrolą połączenia.

Gniazdko(sockets).



Użycie gniazdek w trybie z kontrolą połączenia.

Gniazdka(sockets).

KLIENT

`connect()` - blokuje wykonywanie programu dopóki serwer nie wykona `accept()`.

`write()` - dopóki bufor nadawczy jest pełny;

`read()` - dopóki co najmniej jeden bajt danych nie zostanie odebrany z serwera.

SERWER

`accept()` - blokuje wykonywanie programu dopóki klient nie wykona `connect()`.

`read()` - dopóki co najmniej jeden bajt danych nie zostanie odebrany od klienta.

`write()` - dopóki bufor nadawczy jest pełny;

Biblioteki funkcji.

Gniazdka(sockets).

WIN32

```
#include <winsock.h> Win95
```

```
#include <winsock2.h> Win98,Me,NT,2000,XP
```

UNIX (Linux)

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

```
#include <netinet/in.h>
```

```
#include <netdb.h>
```

Adresowanie.

Gniazdka(sockets).

STRUKTURA ADRESU gniazdka <sys/socket.h>

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

Adresowanie w dziedzinie AF_INET <netinet/in.h>

```
struct in_addr {
    u_long s_addr;
}

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

Adresowanie.

Gniazdka(sockets).

Adresowanie w dziedzinie AF_INET <winsock.h>

```
UNIA
struct in_addr {
    union {
        struct {
            unsigned char    s_b1,
                            s_b2,
                            s_b3,
                            s_b4;
        } s_un_b;

        struct {
            unsigned short   s_w1,
                            s_w2;
        } s_un_w;

        unsigned long       s_addr;
    } s_un;
};
```

Adresowanie.

Gniazdka(sockets).

Adresowanie w dziedzinie AF_IPX <wsipx.h>

```
struct sockaddr_ipx {
    u_short sa_family;
    u_char sa_netnum[4];
    u_char sa_nodenum[6];
    unsigned short sa_socket;
}
```

Adresowanie w dziedzinie AF_UNIX <sys/un.h>

```
struct sockaddr {
    u_short sun_family;
    char sun_path[108];
}
```

Funkcje obsługi gniazdka.

Gniazdka(sockets).

Tworzenie gniazdka

```
#define SOCKET int
```

```
SOCKET socket(
    int af,    //AF_INET, AF_UNIX, AF_IPX
    int type, //SOCK_STREAM, SOCK_DGRAM
    int protocol
);
```

Funkcja zwraca dyskryptor gniazdka wykorzystywany przez inne funkcje.

```
SOCKET s_tcp, s_spx;
s_tcp = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP);
s_spx = socket(AF_IPX, SOCK_STREAM, NSPROTO_SPX);
```

```
s_tcp=INVALID_SOCKET; //error
```

Funkcje obsługi gniazdka.

Gniazdka(sockets).

Przypisywanie adresu

```
int bind(
int sock,    //dyskryptor gniazdka
struct sockaddr *localaddr, //adres lokalny gniazdka
int addrlen  //długość adresu
);
```

Funkcja przypisuje lokalny adres do gniazdka.

```
struct sockaddr_in server;
```

```
    memset(&server,0,sizeof(server)); //?
    server.sin_family=AF_INET;
    server.sin_addr.s_addr=INADDR_ANY;
    server.sin_port=htons(port);
```

INADDR_ANY - upraszcza programowanie, adres lokalny
dobierany jest automatycznie na
podstawie ustawień danego komputera

Funkcje obsługi gniazdka.

Gniazdka(sockets).

Przypisywanie adresu

```
u_short=htons(hostshort);
host to network short;
err=bind(sock,(struct sockaddr*) &server,sizeof(server));

err=0 //no error
```

adresy lokalne przypisywane są tylko w przypadku
TCP serwera i przy odbiorze danych w UDP.

W gniazdkach TCP można podać za port wartość 0
1024-5000

Połączenie klienta z serwerem

```
int connect(
int sock,    //dyskryptor gniazdka
struct sockaddr *servaddr, //adres serwera
int addrlen
);
```

Funkcje obsługi gniazdka.

Gniazdka(sockets).

Połączenie klienta z serwerem connect

```
char Mhost[]="127.0.0.1";
u_long l_adres;

l_adres=inet_addr(Mhost);
memset(&server,0,sizeof(server));
server.sin_family=AF_INET;
server.sin_port=htons(port);
memcpy ( (char *)&server.sin_addr,
         (char *)&l_adres, sizeof(u_long));
```

```
err=connect(sock,(struct sockaddr*)
&server,sizeof(server));
```

```
err=0 //no error
err=SOCKET_ERROR //winsock
```

Funkcje obsługi gniazdka.

Gniazdka(sockets).

Wprowadzenie serwera w stan nasłuchu listen

```
int listen(
    int sock, //dyskryptor gniazdka
    int cmax //maksymalna liczba oczekujących
             połączeń
);
```

```
err=listen(sock,5);
err=0 //no error
```

Funkcja powoduje, że serwer staje się gotowy na żądania klientów.

Funkcje obsługi gniazdka.

Gniazdka(sockets).

Nawiązywanie połączenia przez serwer

```
int accept(  
    int sock, //dyskryptor gniazdka  
    struct sockaddr *addrdistant, //odległy kompu.  
    int *addrlen;  
);
```

```
nsock=accept(sock,(struct sockaddr*)&addrc,&addrc_len);
```

ERROR

```
INVALID_SOCKET = -1
```

Dwa ostatnie parametry zawierają adres klienta od którego odebrano połączenie;
Funkcja tworzy nowe gniazdko do komunikacji z klientem.
Serwer współbieżny.

Standaryzacja formatu liczb całkowitych.

Gniazdka(sockets).

Numer portu i adres internetowy są liczbami całkowitymi
Wymagane jest aby zarówno klient jak i serwer stosowały
tę samą konwencję kolejności bajtów w reprezentacji
liczb. Do uniezależnienia się od formatów stosowanych
przez lokalne systemy służą funkcje do konwersji liczb
całkowitych i tzw długich całkowitych do formatu
sieciowego.

```
u_long htonl(hostlong);  
u_long hostlong;  
ushort htons(hostshort);
```

```
u_long ntohl(netlong);  
u_long netlong;
```

```
u_short ntohs(netshort);  
u_short netshort;
```

Operacje na danych.

Gniazdka(sockets).

Funkcje operujące na pamięci.

```
memset(void *dst,u_char fill, size_t size);
memcpy(void *dst, void *src, size_t size);
memcmp(const void *dst, const void *src, size_t size);
```

Konwersja formatów adresów.

```
char *address;
u_long inet_addr(address);

struct in_addr inaddress;
char *inet_ntoa(inaddress);
```

Ustalanie adresu sieciowego.

Gniazdka(sockets).

Znając nazwę komputera jego adres sieciowy możemy ustalić przez funkcję `gethostbyname()`; Ustala ona adres na podstawie wpisu w serwerze nazw.

```
char *hostname;
hostname=TEXT("uranos.cto.us.edu.pl");
Struct hostent *gethostbyname(hostname);
```

Funkcja zwraca wskaźnik do struktury `hostent`;

```
struct hostent{
    char *h_name;           //Nazwa kompytera
    char **h_aliases;      // lista pseudonimów
    int h_addrtype;        // dziedzina (AF_INET)
    int h_lenght;          // długość adresu
    char **h_addr_list;    //lista adres. Inet. komputera
};
```

Przykład.

Gniazdka(sockets).

```
#include "header.h"

void main(int argc, char *argv[])
{
    struct hostent *hostp;

    if((hostp=gethostbyname(argv[1])!=NULL){
        printf("\nNie znaleziono komputera: %s",argv[1]);
    }else{
        printf("\nKomputer: %s (IP: %s)",hostp->h_name,
            inet_ntoa(*hostp->h_addr_list[0]));
    }
}
```

Ustalanie adresu sieciowego.

Gniazdka(sockets).

Znając adres komputera jego nazwę sieciową możemy ustalić przez funkcję `gethostbyaddr()`;

```
struct HOSTENT FAR * gethostbyaddr ( const char FAR *  
addr, int len, int type );
```

`addr` - adres w notacji sieciowej.

`Len` - rozmiar adresu

`type` - dziedzina

Numer portu danej usługi.

```
struct servent FAR * getservbyname (  
    const char FAR * name, const char FAR * proto  
);
```

Struktura.

Gniazdka(sockets).

```
struct servent {
    char FAR *      s_name;      //nazwa usługi
    char FAR * FAR * s_aliases; //lista pseudonimów
    short          s_port;      //numer portu
    char FAR *      s_proto;    //używany protokół
};
```

Przykład nazwa serwisu.

Gniazdka(sockets).

```
#include "header.h"

void main(int argc, char *argv[])
{
    struct servent *servp;

    if((servp=getservbyname(argv[1],"tcp"))==NULL
        && (servp=getservbyname(argv[1],"udp"))==NULL){
        printf("\nUsługi %s nie znaleziono",argv[1]);
    }else{
        printf("\nUsługa: %s port: %d proto: %s",
            servp->s_name,ntohs(servp->s_port), servp->s_proto);
    }
}
```

Adres przypisany gniazdku.

Gniazdka(sockets).

```
int getsockname ( SOCKET s, struct sockaddr FAR* name,  
int FAR* namelen );
```

Gniazdko, wskaźnik do adresu, długość adresu.
Na podstawie gniazdka funkcja potrafi określić
przypisany adres.

Funkcja działa jedynie na gniazdka gdzie występuje
bind() lub connect().

```
int getpeername ( SOCKET s, struct sockaddr FAR* name,  
int FAR* namelen );
```

Funkcja zwraca nazwę klienta podłączonego do gniazdka s.

Zamykanie połączenia.

Gniazdka(sockets).

```
closesocket(SOCKET s),close(int sock)
```

Brutalne zakończenie.

```
int shutdown(int sock, int control)
```

Zamknięcie kontrolowane połączenia.

0-uniemożliwienie odbierania danych przez gniazdko

1-uniemożliwienie wysyłania danych przez gniazdko

2-uniemożliwienie odbierania i wysyłania danych przez
gniazdko

Nadawanie numerów portom.

Gniazdka(sockets).

Porty zarezerwowane przez system 1-1023

Konfiguracja gniazdek.

```
setsockopt(  
int sock;      //deskryptor gniazdka  
int level;     //SOL_SOCKET  
int option;    //parametr gniazdka  
char *p_arg;   //wskaźnik do argumentu  
int lg;       //długość argumentu  
)  
  
getsockopt(  
int sock;      //deskryptor gniazdka  
int level;     //SOL_SOCKET  
int option;    //parametr gniazdka  
char *p_arg;   //wskaźnik do zwracanych danych  
int *lg;      //długość zwracanych danych  
)
```

Parametry konfiguracyjne gniazdek.

Gniazdka(sockets).

TCP_NODELAY - wstrzymanie porządkowania danych w buforach TCP (*level* = IPPROTO_TCP)
level = SOL_SOCKET

SO_RECVBUF - określenie wielkości bufora odbiorczego

SO_SNDBUF - określenie wielkości bufora nadawczego

SO_KEEPALIVE - regularna transmisja komunikatów kontrolnych z kontrolą połączenia. Jeśli adresat nie odpowiada to połączenie jest uważane za przerwane.

SO_RECVTIMEO - ograniczony czas na odbiór.

SO_SNDTIMEO - ograniczony czas na wysyłanie.

Parametry konfiguracyjne gniazdek.

Gniazdka(sockets).

Ex. Ograniczenie czasowe na odbiór danych .

```
int inline SetSockTimeOut(SOCKET sock,DWORD time)
{
int err;
char test[256];
DWORD optval;

optval=time*1000;
err = setsockopt(sock,SOL_SOCKET,SO_RCVTIMEO,(const char
*)&optval,sizeof(optval));

if(err==SOCKET_ERROR){
    return -1;
}
return 0;
}
```

Parametry konfiguracyjne gniazdek.

Gniazdka(sockets).

Ex Konstrukcja.

```
SetSockTimeOut(sock,180);

nread=recv(sock,buf,nbytes,NULL);

if(nread==SOCKET_ERROR)
{
    err=WSAGetLastError();

    if(err==WSAETIMEDOUT)
    {
        //Send timeout
        return -2;
    }
    return -1;
}
```

SELECT().

Gniazdka(sockets).

```
int select (  
    int nfds,  
    fd_set FAR * readfds,  
    fd_set FAR * writefds,  
    fd_set FAR * exceptfds,  
    const struct timeval FAR * timeout  
);
```

nfds

[in] Parametr ignorowany dodany ze względu na zgodność z Berkeley sockets.

readfds

[in/out] wskaźnik do zbioru gniazd sprawdzanych pod kątem możliwości odczytu.

writefds

[in/out] wskaźnik do zbioru gniazd sprawdzanych pod kątem możliwości zapisu.

exceptfds

[in/out] wskaźnik do zbioru gniazd sprawdzanych pod kątem błędów.

timeout

[in] Maksymalny czas oczekiwania select, NULL operacja blokująca.

SELECT().

Gniazdka(sockets).

Ex Kontrola czy w gniazdku TCP znajdują się dane gotowe do czytania.

```
main(){  
    int sock;  
    int nsock;  
    int rtnr;  
    struct sockaddr_in server;  
    fd_set readf;  
    fd_set writef;  
    struct timeval to;  
    while(1){  
        FD_ZERO(&readf);  
        FD_ZERO(&writef);  
        FD_SET(sock,&readf);  
        FD_SET(sock,&writef);  
        to.tv_sec=10;  
        rtnr=select(sock,&readf,&writef,0,&to);  
    }  
}
```

SELECT().

Gniazdka(sockets).

```
if(rtrn==0){
    printf("TIMEOUT");
    continue;
}

if( FD_ISSET(sock,&readf) || FD_ISSET(sock,&writef))
{
    nsock=accept(sock,(struct sockaddr *) 0, (int *)0);
    server(nsock);
    closesocket(nsock);
}else{
    printf("To nie dyskryptor gniazdka");
}
} //while(1)
} //main
```

Operacje nie blokujące.

Gniazdka(sockets).

```
u_long non_block=1;
int h_error;
err = ioctlsocket(sockData, FIONBIO, &non_block);

do{

    nsockData=accept(sockData,(struct sockaddr*) &addrc,
&addrc_len);
    //Tutaj serwer w chwilach wolnych może przetwarzać
    h_error=WSAGetLastError();

}while(h_error!=WSAEWOULDBLOCK);
```

Gniazdka w języku Java.

Klasa Socket.

KONSTRUKTORY

```
Socket()  
Socket(SocketImpl impl)  
-----Gniazda klienckie-----  
Socket(InetAddress address, int port)  
Socket(InetAddress address, int port, InetAddress  
localAddr, int localPort)  
Socket(String host, int port);  
Socket(String host, int port, InetAddress localAddr, int  
localPort)  
  
-----Deprecated-----  
Socket(InetAddress host, int port, boolean stream)  
Socket(String host, int port, boolean stream)
```

Klasa Socket.

Java Gniazdka(sockets).

METODY

```
void close();  
InetAddress getAddress()  
InetAddress getLocalAddress()  
int getPort()  
int getLocalPort();  
  
InputStream getInputStream()  
OutputStream getOutputStream()  
  
int getReceiveBufferSize()  
int getSendBufferSize()  
int getSoLinger()  
int getSoTimeout()  
boolean getTcpNoDelay()
```

Klasa Socket.

Java Gniazdka(sockets).

METODY

```
void setReceiveBufferSize(int size)
void setSendBufferSize(int size)
void setSoLinger(boolean on, int linger)
void setSoTimeout(int timeout)
void setTcpNoDelay(boolean on)

static void setSocketImplFactory(SocketImplFactory fac)
```

Klasa ServerSocket.

Java Gniazdka(sockets).

KONSTRUKTORY

```
ServerSocket(int port);
ServerSocket(int port,int backlog);
ServerSocket(int port,int backlog,InetAddress bindAddr);
```

KONSTRUKTORY

```
Socket accept()
void close()
InetAddress getAddress()
int getLocalPort()

int getSoTimeout()
void setSoTimeout(int timeout)
protected void implAccept(Socket s)
static void setSocketFactory(SocketImplFactory fac)
```

NetBIOS (*Network Basic Input/Output System*)

IBM 1983 (Sytek Corporation)

1985 - interfejs NetBEUI (NetBIOS Extended User Interface)

NUMERY LANA (LAN Adapter numbers)

Numer LANA to niepowtarzalne połączenie numeru karty sieciowej oraz protokołu transportowego.

LANA

0. TCP/IP - karta sieciowa 1
1. NetBEUI - karta sieciowa 1
2. TCP/IP - karta sieciowa 2
3. NetBEUI - karta sieciowa 2

Przedział 0-9

NetBIOS (*Network Basic Input/Output System*)

KOMUNIKACJA

Wystarczy jeden wspólny protokół aby nastąpiła komunikacja między stacjami.

NAZWY NETBIOSOWE

Proces rejestruje nazwę dla każdego numeru LANA z którym chce się komunikować.

Nazwa ta ma długość maksymalnie 16 znaków plus 16 znaków do celów specjalnych.

Maksymalna liczba nazw jakie można dodać do każdego numeru lana wynosi 254 (0 i 255 są zarezerwowane dla systemu).

Komputery w sieci zgłaszają swoje nazwy jeżeli nie ma zastrzeżeń to nazwa jest akceptowana.

Serwer WINS

NetBIOS (*Network Basic Input/Output System*)

SZESNASTY ZNAK

- 00 - Netbiosowa nazwa komputera.
- 03 - nazwa usługi komunikacyjnej.
- 1B - nazwa głównej przeglądarki domeny
- 06 - usługa serwera RAS
- 1F - usługa NetDDE(Network Dynamic Data Exchange)
- 20 - usługa umożliwiająca współużytkowanie plików.
- 21 - klient RAS
- BE - Network Monitor Agent
- BF - Narzędzie Network Monitor

NetBIOS (*Network Basic Input/Output System*)

Właściwości NetBIOS-u

1. Usługi połączeniowe
2. Usługi bezpołączeniowe (datagramy).

NetBIOS (Network Basic Input/Output System)

Podstawy programowania NetBIOS-u

Zestaw funkcji API

```
UCHAR Netbios(PNCB pNCB);
```

```
typedef struct _NCB{
  UCHAR ncb_command; //Polecenie NETBIOS z ^ ASYNCH (0x80) - asynchro
  UCHAR ncb_retcode; //zwracany kod przez operację
  UCHAR ncb_lsn; //identyfikacja numeru sesji lokalnej
  UCHAR ncb_num; //określenie numeru nazwy sieci lokalnej
  PCHAR ncb_buffer; //wskazanie bufora danych
  WORD ncb_length; //długość bufora w bajtach
  UCHAR ncb_callname[NCBNAMSZ]; //określenie nazwy odległej aplikacji
  UCHAR ncb_name[NCBNAMSZ]; //określenie nazwy pod którą wyst. Aplikac.
  UCHAR ncb_rto; //limit oczekiwania na odbiór. [500 ms]
  UCHAR ncb_sto; //limit oczekiwania na nadawanie [500 ms]
  void (*ncb_post) (struct _NCB *); adres procedury nadawczej.
  UCHAR ncb_lana_num; //określenie numeru LANA
  UCHAR ncb_cmd_cplt; //określenie kodu zwracanego dla operacji
  UCHAR ncb_reserve[10]; //zarezerwowane
  HANDLE ncb_event; //uchwyt obiektu zdarzenia Windows.
}*PNCB, NCB;
```

NetBIOS (Network Basic Input/Output System)

Podstawy programowania NetBIOS-u

Funkcja zwrotna

```
VOID CALLBACK PostRoutine(PNCB pncb);
```

Transmisja synchroniczna a transmisja asynchroniczna

Polecenie (OR) ASYNCH

Wartość zwracana NRC_GOODRET (0x00);

ale ncb_cm_cplt ma wartość NRC_PENDING.