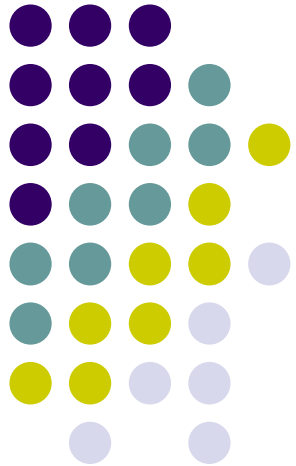
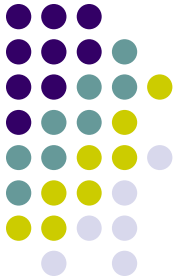
Мрежово програмиране

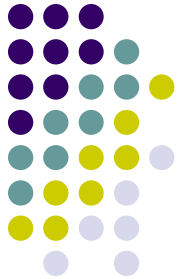
Winsock API



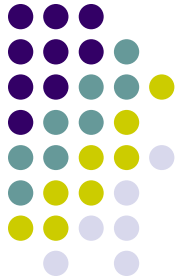


В MSWindows наборът от заглавни файлове е драстично намален. Може да се включи само един файл winsock.h или winsock2.h, в случай че ще използвате разширените възможности на Winsock 2.

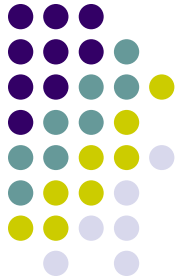
В GNULinux дескрипторите на сокетите имат тип int. В MSWindows сокетите не са файлови дескриптори, затова за тях е въведен тип SOCKET.



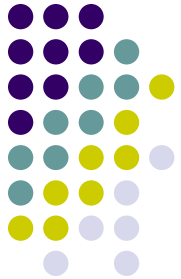
В MSWindows е необходимо библиотеката Winsock явно да се инициализира преди да се използват нейните функции.
Това се прави с функцията WSAStartup.
Преди излизането от програмата е необходимо за се извика функцията "int WSACleanup (void)" за деинициализация на библиотеката Winsock и за освобождаване на използваните от това приложение ресурси.
Приключването на процеса с функцията ExitProcess автоматично не освобождава ресурсите на сокетите.



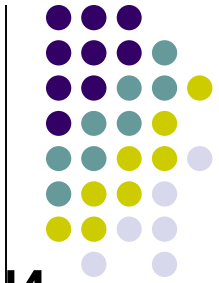
В MSWindows за работа със сокети не се използват функции на файловия вход-изход (read и write).
В MSWindows всички извиквания write и read трябва да се заменят на send и recv съответно. Вместо close се използва closesocket.



В MSWindows глобалната променлива errno не се използва. Вместо това кодът на последната грешка се запазва от системата за всеки поток отделно. За да бъде получен този код се използва функцията WSAGetLastError.



В MSWindows са въведени допълнителни константи, които следва да се използват вместо конкретни числа.
Така стойностите, връщани от функциите на Winsock, следва да се сравняват с константите INVALID_SOCKET или SOCKET_ERROR, а не с -1.



- За съжаление, разликите при socket API и Winsock не се ограничават от приведения списък.
- Често възникват проблеми, свързани с импортиране на по-сложни програми.
- Също така проблеми могат да възникнат с функции, нямащи пряко отношение към socket API.
- В MSWindows няма пряк аналог на функцията **fork**.

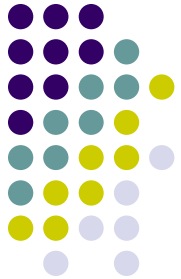
Преди да се използва кода на сокетите се налага да добавят съответните библиотеки и header-и. Кодът за инициализация и създаване на сокетите в Unix подобните системи и MSWindows изглеждат различно, понеже разработчиците на MSWindows са изнесли кода на мрежовата подсистема в отделна библиотека, затова първо програмистът трябва да направи допълнителна инициализация за да работи със сокетите.

Създаване на сокет:

```
// Listing 1 (Linux & FreeBSD)
// int socket (int domain, int type, int protocol);

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

int sd;    // нашият дескриптор
// тук няма инициализация, понеже socket()
// е системно извикване
sd = socket (PF_INET, SOCK_DGRAM, 0);
```

```
// Listing 1 (MSWindows)
// SOCKET socket (int pf, int type, int protocol);
#pragma comment (lib, "ws2_32.lib");
#include <winsock2.h // winsock2.h: typedef u_int SOCKET
WORD wVersion; // версията на winsock интерфейса
WSADATA wsaData; // записваме тук данните за сокета
wVersion = MAKEWORD (2, 0); // задаваме версията на winsock
SOCKET sd;          // нашият дескриптор на сокета
int wsaInitError = WSAStartup (wVersion, &wsaData);
                                // инициализираме winsock
if (wsaInitError != 0)     // проблеми и излизаме
   exit (1);
else    // щом инициализацията е преминала успешно,
          //тогава създаваме сокет
sd = socket (PF_INET, SOCK_DGRAM, 0);
```

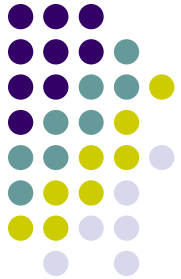
При извикване на socket() задаваме:



```
#ifdef MSWINDOWS
SOCKET sd = socket (PF_INET, SOCK_DGRAM, 0);
    //за MSWindows
#else
int sd = socket (PF_INET, SOCK_DGRAM, 0);
#endif
```

за MSWindows:

Ако функцията е приключила успешно тя връща дескриптора на сокета, ако не - INVALID_SOCKET.



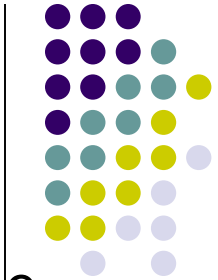
- Следва листингът да се компилира като конзолно приложение.

- Кодовете на възможните грешки и тяхното описание се проверяват или в MSDN, или в man page за конкретната функция.

Грешки при Berkeley Sockets API:

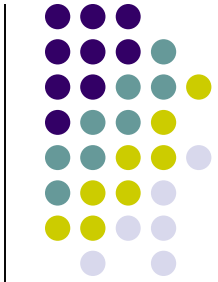
- EPROTONOSUPPORT – посоченият протокол не се поддържа
- EAFNOSUPPORT – посоченото семейство адреси не се поддържа
- EINVAL – неизвестен протокол или семейство адреси
- EACCES – забранено е създаването на сокет от зададения тип
- ENFILE, EMFILE, ENOBUFS, ENOMEM – недостиг на системни ресурси
- и др.

Грешки за системите MSWindows:



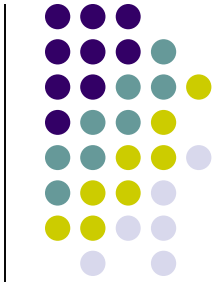
- WSANOTINITIALISED - Windows Sockets интерфейсът не е инициализиран с функцията WSAStartup
- WSAENETDOWD - срив на мрежовото програмно осигуряване
- WSAEAFNOSUPPORT – посочен е неправилен тип на адреса
- WSAEINPROGRESS – изпълнява се блокираща функция на Windows Sockets интерфейса
- WSAEMFILE - лимитът от свободни дескриптори е изчерпан
- WSAENOBUFS – няма памет за създаване на буфери
- WSAEPROTONOSUPPORT – зададен е неправилен протокол
- WSAEPROTOTYPE – посоченият протокол е несъвместим с дадения тип на сокета
- WSAESOCKNOSUPPORT – посоченият тип на сокета е несъвместим с дадения тип на адреса

Затваряне на сокет и деинициализация на библиотеките:

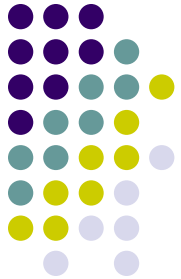


```
//Linux & FreeBSD:
  int close (int sd);

//MSWindows:
  int closesocket (SOCKET sd);
  int WSACleanup (void);
```

Как изглежда името на сокета:



```
struct sockaddr_in
{
    unsigned char  sin_len;  // само за FreeBSD
    unsigned char  sin_family;
    unsigned short   sin_port;
    struct in_addr  sin_addr;
    char  sin_zero[8];  // отсъства в Linux
};
struct in_addr
{
   unsigned long int  s_addr;  // 4 байта (int32)
 };
```



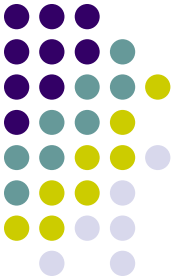
```
// Listing 2 (Linux & FreeBSD)
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 5000

struct sockaddr_in addr;
memset (&addr, 0, sizeof (struct sockaddr_in));

addr.sin_family = AF_INET;
addr.sin_port = htons (PORT);
addr.sin_addr.s_addr = inet_addr ("192.168.0.1");
```

```
// Listing 2 (Windows)
#include <string.h>
#include <winsock2.h>

#define PORT 5000

struct sockaddr_in addr;
memset (&addr, 0, sizeof (struct sockaddr_in));

addr.sin_family = AF_INET;
addr.sin_port = htons (PORT);
addr.sin_addr.s_addr = inet_addr ("192.168.0.1");
```

Настройка на сокета:

```
// Linux & FreeBSD
  int bind (int s, const struct sockaddr *addr, int addrlen);
// MSWindows
  int bind (SOCKET s, const struct sockaddr *name, int namelen);
```

Грешки при Berkeley Sockets API:



- EINVAL – сокетът вече е именуван
- EACCES – опит непривилегирован процес да получи порт с номер по-малък от 1024
- EADDRINUSE – вече има сокет със зададеното име
- EADDRNOTAVAIL – посоченият адрес не е свързан с някой от локалните мрежови интерфейси
- ENOBUFS, ENOMEM – недостиг на системни ресурси
- EBADF, ENOTSOCK – посочени са некоректни стойности за дескриптора

Грешки за системите MSWindows:

- WSANOTINITIALISED - преди използването на функцията е необходимо да се извика функцията WSAStartup
- WSAENETDOWN - срив в мрежата
- WSAEADDRINUSE - посоченият адрес вече се използва
- WSAEFAULT - стойността на параметъра namelen е по-малка от размера на структурата sockaddr
- WSAINPROGRESS - изпълнява се блокираща функция на Windows Sockets интерфейса
- WSAEAFNOSUPPORT – избраният протокол не може да работи със зададеното семейство адреси
- WSAEINVAL - сокетът вече е именуван
- WSAENOBUFS - вече са установени максимален брой съединения
- WSAENOTSOCK - посоченият в параметъра дескриптор не е дескриптор на сокет

Методи за адресните структури:

```
// Listing 3 (Windows & Linux & FreeBSD)
void _sock_addr::set_port (unsigned short port)
 {
    address->sin_port = htons (port);
 }

void _sock_addr::set_ip (const char *ip)
 {
    address->sin_addr.s_addr = inet_addr (ip);
    if (address->sin_addr.s_addr == INADDR_NONE)
     throw _sock_exception ("_sock_addr::set_ip - the
     provided IP address seems to be invalid");
  }
```

Те позволяват да се използва една и съща адресна структура за изпращане на дейтаграми към различни адреси.

Изпращане на съобщения с UDP протокол:

// Linux & FreeBSD

int sendto (int s, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);

// Windows

int sendto (SOCKET s, const char *buf, int len, int flags, const struct sockaddr *to, int tolen);

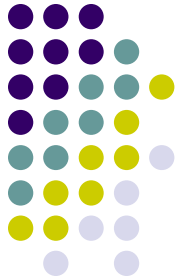
Възможни кодове на грешки за MSWindows:

- WSANOTINITIALISED, WSAENETDOWN, WSAEACCES, WSAEINVAL, WSAEINTR, WSAEINPROGRESS, WSAEFAULT, WSAENETRESET, WSAENOBUFS, WSAENOTCONN, WSAENOTSOCK, WSAEOPNOTSUPP, WSAESHUTDOWN, WSAEWOULDBLOCK, WSAEMSGSIZE, WSAEHOSTUNREACH, WSAECONNABORTED, WSAECONNRESET, WSAEADDRNOTAVAIL, WSAEAFNOSUPPORT, WSAEDESTADDRREQ, WSAENETUNREACH, WSAEHOSTUNREACH, WSAETIMEDOUT

Да се определи номера на порта и адреса, зададени на сокета, може с функцията getsockname():

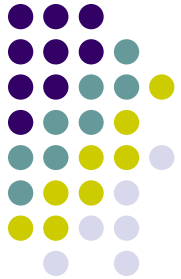
```
// Linux & FreeBSD

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

// Windows

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

Определяне адреса на сокета:



```
// Listing 4 (Windows & Linux & FreeBSD)

struct sockaddr_in *name = new struct sockaddr_in;
int namelen = sizeof (struct sockaddr_in);
int error = getsockname (sd, (struct sockaddr *) name, &namelen);
if (error == -1) //SOCKET_ERROR в WINDOWS
{
    // обработка на грешките
}
cout << " The socket IP address is: " << inet_ntoa (name->sin_addr) << endl
        << " The socket port number is: " << ntohs (name->sin_port) << endl;
```

Получаване на съобщения с UDP протокол:



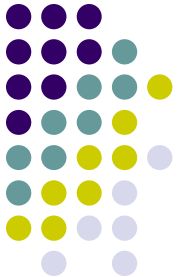
// Linux & FreeBSD

```
int recvfrom (int s, void *buf, int len, int flags, struct
    sockaddr *from, int *fromlen);
```

// MSWindows

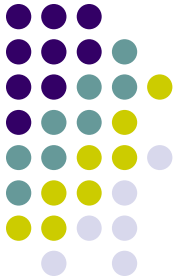
```
int recvfrom (SOCKET s, char *buf, int len, int flags,
    struct sockaddr *from, int *fromlen);
```

Възможни кодове на грешки за MSWindows:



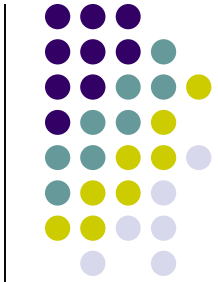
WSANOTINITIALISED
WSAENETDOWN
WSAEFAULT
WSAEINTR
WSAEINPROGRESS
WSAEINVAL
WSAEISCONN
WSAENETRESET
WSAENOTSOCK
WSAEOPNOTSUPP
WSAESHUTDOWN
WSAEWOULDBLOCK
WSAEMSGSIZE
WSAETIMEDOUT
WSAECONNRESET

Функция listen():



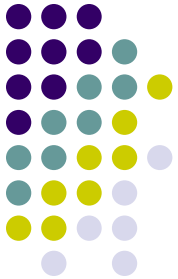
```
// Linux & FreeBSD

  int listen (int s, int backlog);

// Windows

   int listen (SOCKET s, int backlog);
```

Грешки при Berkeley Sockets API:



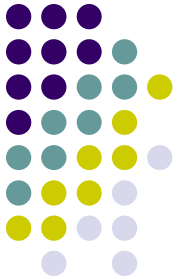
- EADDRINUSE – сокет със зададеното име вече (все още) съществува и се намира в режим на прослушване
- EBADF, ENOTSOCK – зададен е невалиден дескриптор
- EOPNOTSUPP – неправилен тип на сокета

Възможни кодове на грешки за MSWindows:



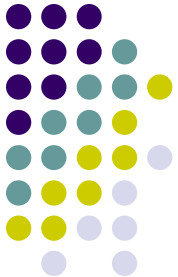
- WSANOTINITIALISED – преди използването на функцията е необходимо да се извика функцията WSAStartup
- WSAENETDOWN – срив в мрежата
- WSAEADDRINUSE – посоченият адрес вече се използва
- WSAEINPROGRESS – изпълнява се блокираща функция за интерфейса Windows Sockets
- WSAEINVAL – сокетът все още няма адрес или вече е присъединен
- WSAEISCONN - сокетът вече е присъединен
- WSAEMFILE – няма достъпни дескриптори
- WSAENOBUFS – недостиг на памет за зареждане в буфера
- WSAENOTSOCK – зададеният в параметъра дескриптор не е дескриптор на сокет
- WSAEOPNOTSUPP - функцията listen не може да работи със сокет от посочения тип

Изпращане на заявка към сървъра:



```
// Linux & FreeBSD

int connect (int s, const struct sockaddr *server_addr,
    int namelen);

// MSWindows

int connect (SOCKET s, const struct sockaddr
    *server_addr, int namelen);
```

Грешки при Berkeley Sockets API:

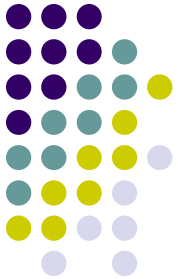


- EBADF, ENOTSOCK – невалиден дескриптор
- EISCONN – сокетът вече е съединен
- ECONNREFUSED – на отсрещната страна посоченият порт не се прослушва
- ETIMEDOUT – таймаут
- ENETUNREACH – мрежата е недостъпна
- EINPROGRESS – неблокиращ сокет, съединението все още се установява
- EALREADY – неблокиращ сокет, предишният опит все още не е приключил
- EACCES, EPERM – опит за броудкаст предаване без задаване на съответната опция на сокета

Възможни кодове на грешки за MSWindows:

WSANOTINITIALISED
WSAENETDOWN
WSAEADDRINUSE
WSAEINTR
WSAEINPROGRESS
WSAEALREADY
WSAEADDRNOTAVAIL
WSAEAFNOSUPPORT
WSAECONNREFUSED
WSAEFAULT
WSAEINVAL
WSAEISCONN
WSAENETUNREACH
WSAEHOSTUNREACH
WSAENOBUFS
WSAENOTSOCK
WSAETIMEDOUT
WSAEWOULDBLOCK
WSAEACCES

Примерен код за изпращане на заявка:



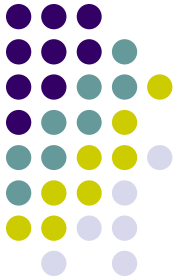
```
  #define SERVER_ADDRESS  "192.168.0.1“
  #define SERVER_PORT  10000
  struct sockaddr_in addr;
  addr.sin_family = AF_INET;
  addr.sin_port = htons (SERVER_PORT);
  addr.sin_addr.s_addr = inet_addr (SERVER_ADDRESS);
  int namelen = sizeof (struct sockaddr_in);
  int error = connect (sd, (struct sockaddr *) &addr, namelen);
if (error == -1) // SOCKET_ERROR в MSWINDOWS
    // неуспешно
else
   // приемаме-изпращаме данни
```

Функцията accept():

```
// Linux & FreeBSD
  int accept (int s, struct sockaddr *addr, int
   *addrlen);
// MSWindows
   SOCKET accept (int s, struct sockaddr
   *addr, int *addrlen);
```

После решаваме дали да продължим сеанса или ще затваряме сокета.

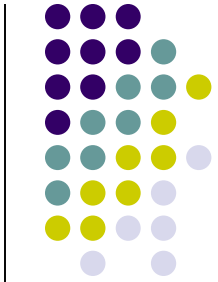
Грешки при Berkeley Sockets API:



- EBADF, ENOTSOCK – невалиден дескриптор
- EAGAIN, EWOULDBLOCK – неблокиращ сокет, няма съединение
- EOPNOTSUPP – неправилен тип на сокета
- EINTR – извикването е прекъснато от сигнал
- EINVAL – сокетът не се намира в състояние на прослушване
- ECONNABORTED – разрив на съединението
- ENFILE, EMFILE, ENOBUFS, ENOMEM – недостиг на системни ресурси

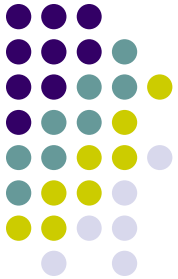
Възможни кодове на грешки за MSWindows:

- WSANOTINITIALISED - преди използването на функцията е необходимо да се извика функцията WSAStartup
- WSAENETDOWN – срив в мрежата
- WSAEFAULT – стойността на параметъра addrlen е по-малка от размера на структурата на адреса
- WSAEINTR – изпълнението на функцията е отменено от функцията WSACancelBlockingCall
- WSAEINPROGRESS – изпълнява се блокираща функция на интерфейса на Windows Sockets
- WSAEINVAL – преди извикването на функцията accept не е била извикана функцията listen
- WSAEMFILE – няма достъпни дескриптори
- WSAENOBUFS – установени са прекалено много съединения
- WSAENOTSOCK – зададеният дескриптор не е дескриптор на сокет
- WSAEOPNOTSUPP – зададеният тип на сокета не може да се използва при извикването на функция, ориентирана за работа със свързан канал
- WSAEWOULDBLOCK – сокетът е отбелязан като неблокиращ и към настоящия момент няма канали за връзка, които да е необходимо да се установяват



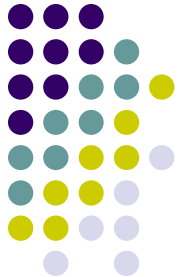
```
// Listing 5 (Windows & Linux & FreeBSD)
#ifdef _WINDOWS_
SOCKET client;
#else  int client;
#endif
struct sockaddr_in client_addr;
int client_addrlen = sizeof (struct sockaddr_in);
client = accept (sd, (struct sockaddr *) &client_addr,
&client_addrlen);
if (client_is_valid())
  transmit_data();
 else
#ifdef _WINDOWS_
 closesocket (client);
#else
  close (client);
#endif
```

Обявяването на send() изглежда по следния начин:



```
// Linux & FreeBSD

  int send (int s, const void *msg, int len, int flags);

// MSWindows

int send (SOCKET s, const char *buf, int len, int flags);
```

Berkeley Sockets API

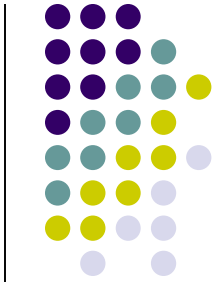


Предаване на данни в сокет

```
#include <sys/types.h>
#include <sys/socket.h>
size_t sendto(
    int sockfd,
    const void *buf,
    size_t len,
    int flags,
    const struct sockaddr *to,
    socklen_t tolen);

size_t send(
    int sockfd,
    const void *buf,
    size_t len,
    int flags);

#include <unistd.h>
size_t write(int fd, const void *buf, size_t len);
```

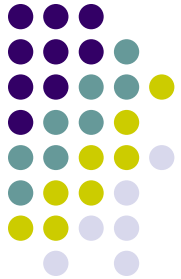
Грешки при предаване на данни в сокет при Berkeley Sockets API:
write(sock,buf,len) <=> send(sock,buf,len,0) <=> sendto(sock,buf,len,0,NULL,0);

- EBADF, ENOTSOCK – посоченият дескриптор не е коректен
- EAGAIN, EWOULDBLOCK – неблокиращ се сокет, предаването на данни би го блокирало
- EISCONN – сокетът вече е бил съединен, а в sendto() е посочен получател
- ECONNRESET – отсрещната страна е изпратила флаг RST
- ENOTCONN, EDESTADDRREQ – сокетът не е съединен, адресът на получателя в sendto() не е показан
- EINTR – извикването е прекъснато от сигнал
- ENOBUFS, ENOMEM – недостатиг на системни ресурси
- EFAULT – невалиден адрес на буфера
- EOPNOTSUPP – невалидно съчетание на флаговете за дадения сокет
- EPIPE – от локалната страна сокетът е бил затворен за предаване на пакети



Възможни кодове на грешки за *send* (в системите MSWindows може да бъде получен с помощта на функцията WSAGetLastError):

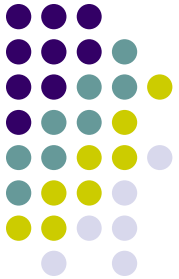
WSANOTINITIALISED, WSAENETDOWN, WSAEACCES, WSAEINTR, WSAEINPROGRESS, WSAEFAULT, WSAENETRESET, WSAENOBUFS, WSAENOTCONN, WSAENOTSOCK, WSAEOPNOTSUPP, WSAESHUTDOWN, WSAEWOULDBLOCK, WSAEMSGSIZE, WSAEHOSTUNREACH, WSAEINVAL, WSAECONNABORTED, WSAECONNRESET, WSAETIMEDOUT.



За sendto е възможен следния код на грешката:

WSANOTINITIALISED
WSAENETDOWN
WSAEACCES
WSAEINVAL
WSAEINTR
WSAEINPROGRESS
WSAEFAULT
WSAENETRESET
WSAENOBUFS
WSAENOTCONN
WSAENOTSOCK
WSAEOPNOTSUPP
WSAESHUTDOWN
WSAEWOULDBLOCK
WSAEMSGSIZE
WSAEHOSTUNREACH
WSAECONNABORTED
WSAECONNRESET
WSAEADDRNOTAVAIL
WSAEAFNOSUPPORT
WSAEDESTADDRREQ
WSAENETUNREACH
WSAEHOSTUNREACH
WSAETIMEDOUT

интерфейс на сокетите, Функция	Описание
send	Предава данните на сокет с установено логическо съединение.
write	Предава данните на сокет с установено логическо съединение. За предаването се използва буфер с данни.
writev	Предава данни на сокет с установено логическо съединение. В качеството на буфер се използат разделно разположени блокове от памет.
sendto	Предава данни на UDP сокет. Използва буфер за данни.
sendmsg	Предава данни на UDP сокет. В качеството на буфер се използва гъвкава структура на съобщението.



writev(socket_handle, io_vector, vector_length) ;

- Също както в случая с write, функцията writev изисква първият параметър да сочи дескриптора на сокета.
- Вторият параметър, векторът за вход-изход, сочи към масив от указатели. Нека да предположим, че данните за предаването са разположени в различни области на паметта. В този случай всеки член на масива с указатели представлява указател към област от паметта, съдържаща данните за предаването. Когато функцията writev предава данните, тя ги открива според зададените в приложната програма в масива с указатели на адресите. Данните се изпращат в същия ред, в който техните адреси са посочени в масива с указателите.
- Третият параметър на функцията writev определя броя на указателите в масива с указатели, зададен от вектора за вход-изход.

sendmsg(socket_handle, message_structure, special_flags);

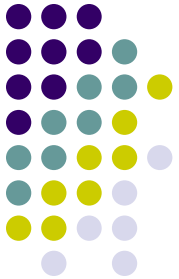
- Структурата на съобщението позволява на програмата гъвкаво да размества дълги списъци с параметри на съобщенията в единна структура от данни.
- Функцията sendmsg прилича на writev по това, че приложната програма може да разположи своите данни в няколко разделно разположени блокове в паметта. Или както и при функцията writev, структурата на съобщението съдържа указател към масив с адреси в паметта.

Формат на структурата на съобщението, използван от функцията sendmsg

---32 бита---

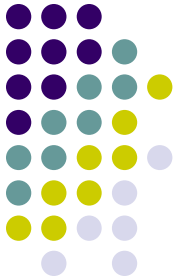
- Указател към структурата от данни на сокета
- Дължина на структурата от данни на сокета
- Указател към списък от вектори за вход/изход
- Дължина на списъка от вектори за вход/изход
- Указател към списък с права за достъп
- Дължина на списъка с правата за достъп

Съответстващи функции за предаване и приемане на данни в интерфейса на сокетите



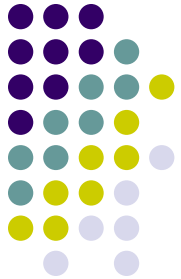
Функция за предаване на данни	Съответстваща функция за приемане на данни
send	recv
write	read
writev	readv
sendto	recvfrom
sendmsg	recvmsg

Обявяването на recv() изглежда по начина:



```
// Linux & FreeBSD

 int recv (int s, void *buf, int len, int flags);

// MSWindows

 int recv (SOCKET s, char *buf, int len, int flags);
```

Berkeley Sockets API



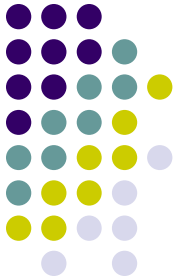
Приемане на данни от сокет

```
#include <sys/types.h>
#include <sys/socket.h>
size_t recvfrom(
    int sockfd,
    const void *buf,
    size_t len,
    int flags,
    const struct sockaddr *from,
    socklen_t fromlen);

size_t recv(
    int sockfd,
    const void *buf,
    size_t len,
    int flags);

#include <unistd.h>
size_t read(int fd, const void *buf, size_t len);
```

Грешки при получаване (приемане) на данни от сокет при Berkeley Sockets API: read(sock,buf,len) <=> recv(sock,buf,len,0) <=> recvfrom(sock,buf,len,0,NULL,0);

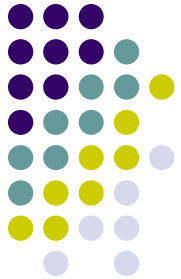
- EBADF, ENOTSOCK – посочен е некоректен дескриптор
- EAGAIN, EWOULDBLOCK – неблокиращ сокет, като получаването на данните ще го блокира
- ENOTCONN – сокетът не е съединен
- EINTR – извикването е прекъснато от сигнал
- ENOBUFS, ENOMEM – недостиг на системни ресурси
- EFAULT – невалиден адрес на буфера

За recv кодът на грешката може да бъде получен с помощта на функцията WSAGetLastError:



WSANOTINITIALISED, WSAENETDOWN, WSAEFAULT, WSAENOTCONN, WSAEINTR, WSAEINPROGRESS, WSAENETRESET, WSAENOTSOCK, WSAEOPNOTSUPP, WSAESHUTDOWN, WSAEWOULDBLOCK, WSAEMSGSIZE, WSAEINVAL, WSAECONNABORTED, WSAETIMEDOUT, WSAECONNRESET.

За recvfrom възможни кодове на грешка:



WSANOTINITIALISED, WSAENETDOWN, WSAEFAULT, WSAEINTR, WSAEINPROGRESS, WSAEINVAL, WSAEISCONN, WSAENETRESET, WSAENOTSOCK, WSAEOPNOTSUPP, WSAESHUTDOWN, SAEWOULDBLOCK, WSAEMSGSIZE, WSAETIMEDOUT, WSAECONNRESET

Функции за четене и запис в потоков сокет:
int readn(SOCKET fd, char *bp, int len);
int writen(SOCKET fd, char *bp, int len);

- функциите връщат броя на прочетените

или записани байтове, разположени в буфера bp.

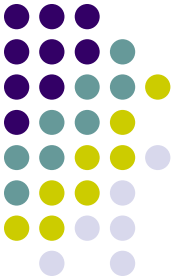
int readLine(SOCKET fd, char *buf);

- Функцията readLine чете данните побайтово, за да не пропусне признака за край на записа – нулев байт;
- Функцията връща броя на приетите байтове, включително и нулевия байт.

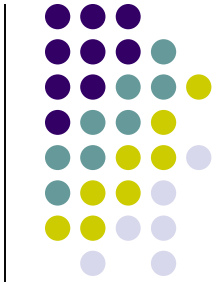
```
int sendall(int s, char *buf, int len, int flags)
{
int total = 0;
int n;
while(total < len)
{
n = send(s, buf+total, len-total, flags);
if(n == -1) { break; }
total += n;
}
return (n==-1 ? -1 : total);
}
```

sendall осигурява изпращането на всички данни от буфера

В UNIX всички операции за вход/изход са синхронни



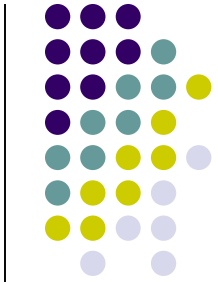
- Синхронните функции използват блокиращи сокети.
  - Те работят по-добре в command-line многозадачни системи (като UNIX).
- Режимът без блокиране е по-сложен за използване.
  - Той осигурява същите възможности, както и режимът с блокиране плюс още някои предимства.



Синхронните Window Sockets API са моделирани на базата на стандартните Berkeley Sockets API.

Използването на **асинхронното разширение на Window Socket API** е най-доброто решение за програмиране на сокети в MSWindows. То позволява изпълнението едновременно на много мрежови задачи без голямо натоварване на процесора. Използвайки асинхронното разширение потребителят може да изпраща писмо и да проверява за наличие на нови писма на отдалечен сървър едновременно с една и съща програма.

Асинхронни се наричат, понеже изпълнението им е свързано с определен диалог, като нито началото, нито приключването не е ограничено от някакви времеви рамки.



- Синхронните функции на Window Sockets API представляват клон на стандартните функции на Berkeley Sockets API. Синхронните функции на Window Sockets API са ориентирани към процедурно програмиране, като се използват блокиращи сокети.
- **Асинхронният модел** за вход-изход е един от главните модели за изпълнение на операциите за вход-изход в MSWindows. Затова е разработено асинхронното разширение на Window Socket API.
- Winsock предоставя няколко модела за вход–изход, които подпомагат приложенията при управлението на входно-изходните операции за няколко сокета едновременно чрез асинхронен режим: блокиране, select, WSAAsyncSelect, WSAEventSelect, overlapped I/O и completion port.

Функционален аналог – асинхронен еквивалент

WSAAsyncGetHostByAddr - gethostbyaddr

WSAAsyncGetHostByName - gethostbyname

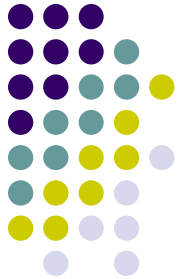
WSAAsyncGetProtoByName - getprotobyname

WSAAsyncGetProtoByNumber - getprotobynumber

WSAAsyncGetServByName - getservbyname

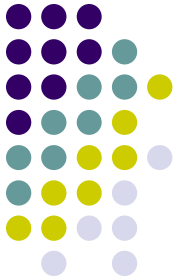
WSAAsyncGetServByPort - getservbyport

WSAAsyncSelect - select



- WSAAsyncSelect е аналог на select, но има съществени разлики
- WSAAsyncSelect привежда сокета в неблокиращо състояние
- Ако select контролира състоянието на няколко сокета, за да бъде достигнат същия резултат с wsaasyncselect трябва да се реализират няколко извиквания - толкова, колкото следва да бъдат анализирани

Форматът на WSAAsyncSelect има следния вид:

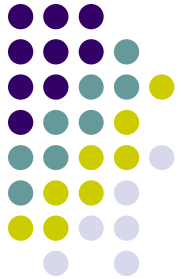


WSAAsyncSelect(SOCKETs, HWND *hWnd*, unsigned int *wMsg*, long *lEvent*),

- *s* - дескриптор на съединения сокет, състоянието на който искаме да контролираме;
- *hWnd* - дескриптор на прозореца на получателя на съобщението;
- *wMsg* - определя типа на изпращаното съобщение;
- *lEvent* - битова маска, определяща типа на събитията, които ни интересуват.

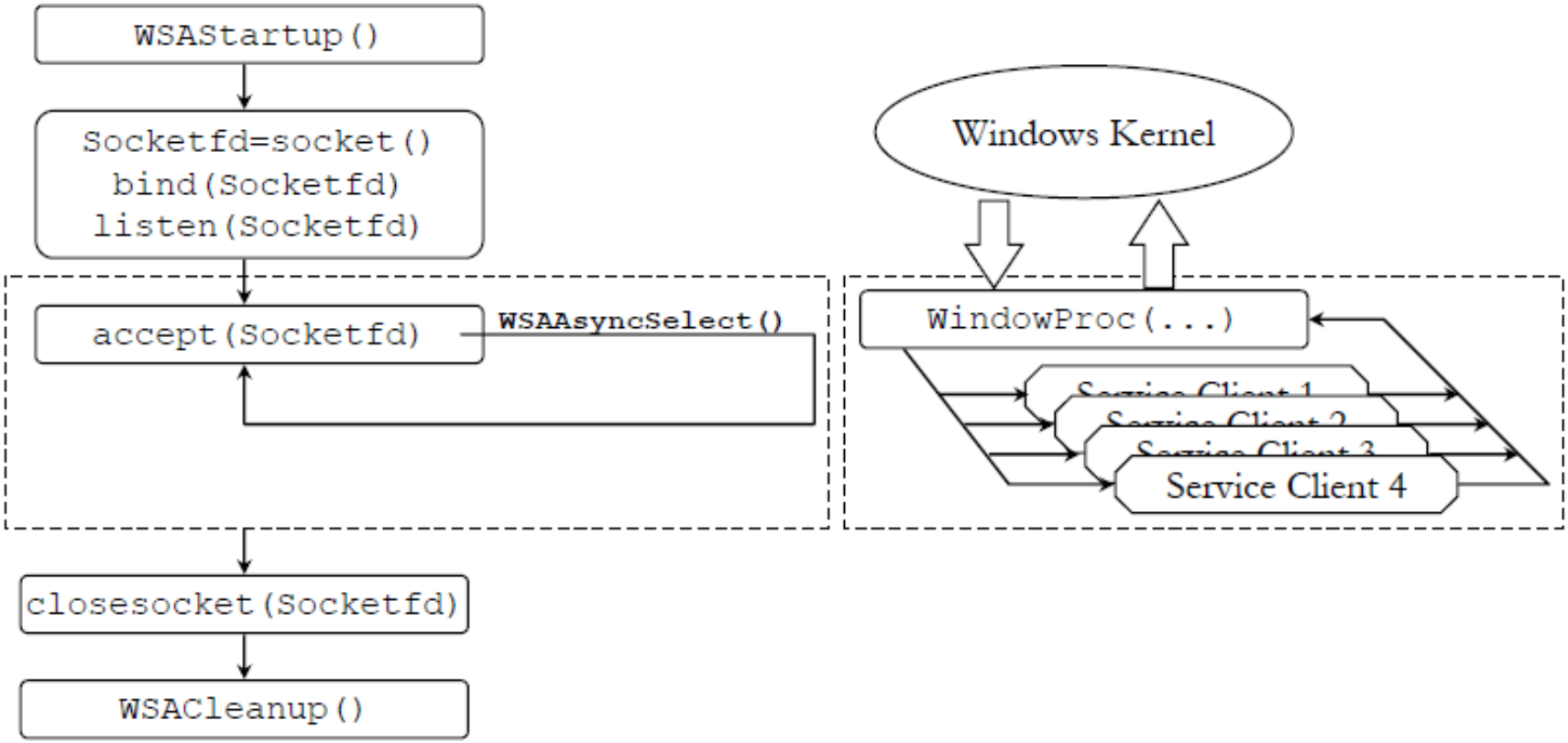
Възможни стойности на параметъра lEvent за WSAAsyncSelect

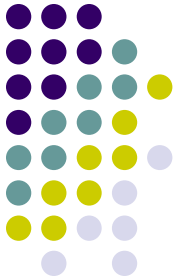
- FD_READ - Готовност за четене
- FD_WRITE - Готовност за запис
- FD_OOB – Постъпване на Out_Of_Band данни
- FD_ACCEPT – Контрол за установяване на входящо съединение
- FD_CONNECT – Контрол за установени съединения
- FD_CLOSE – Контрол за затваряне на съединения



- При необходимост от контрол на комбинация от тези състояния маските могат да се обединяват с ИЛИ
- Когато състоянието на сокета съответства на избраната маска, тогава MSWindows ще изпрати на приложната програма съответното съобщение
- Това съобщение съдържа информация за това, откъде е реализирано извикването на WSAAsyncSelect, идентификатора на съобщението и 16-битови и 32-битови параметри на това съобщение
- Първият от тях е дескриптора на сокета, където е регистрирано събитието. Младшите 16 бита на втория параметър са код на събитието, а старшите са предназначени за записване на кода на грешките, ако има такива

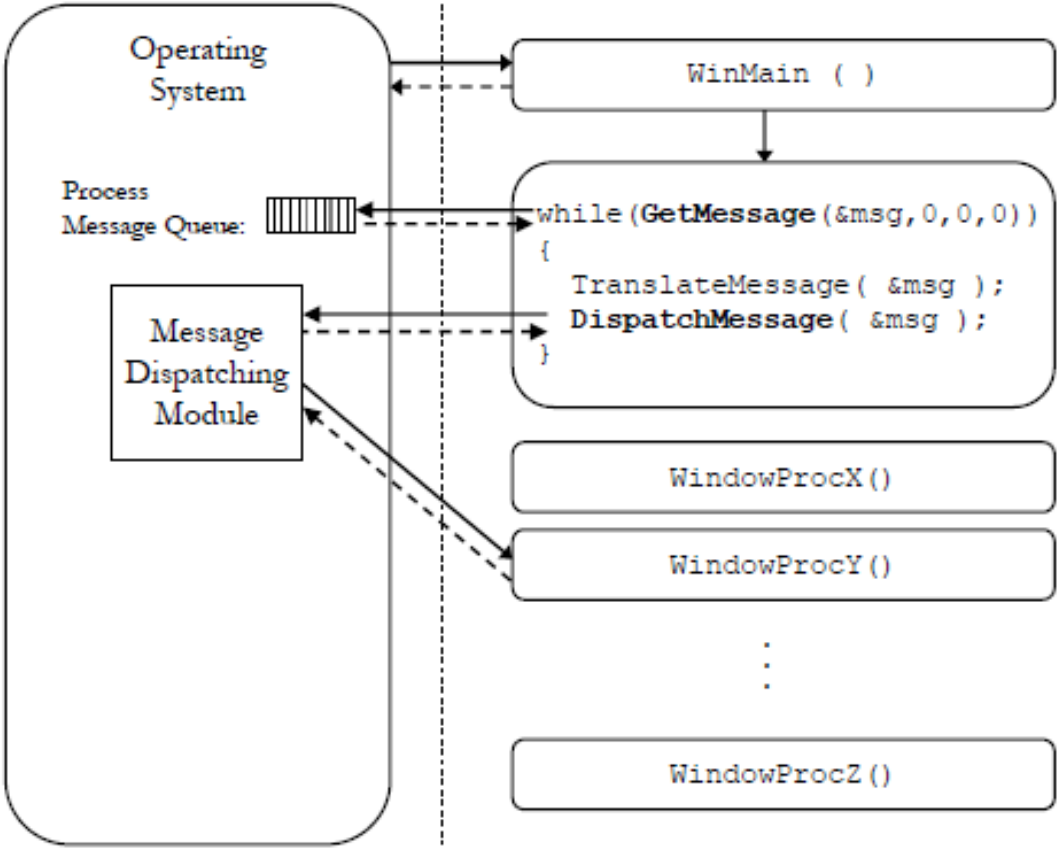
Управляван от съобщения паралелен сървър



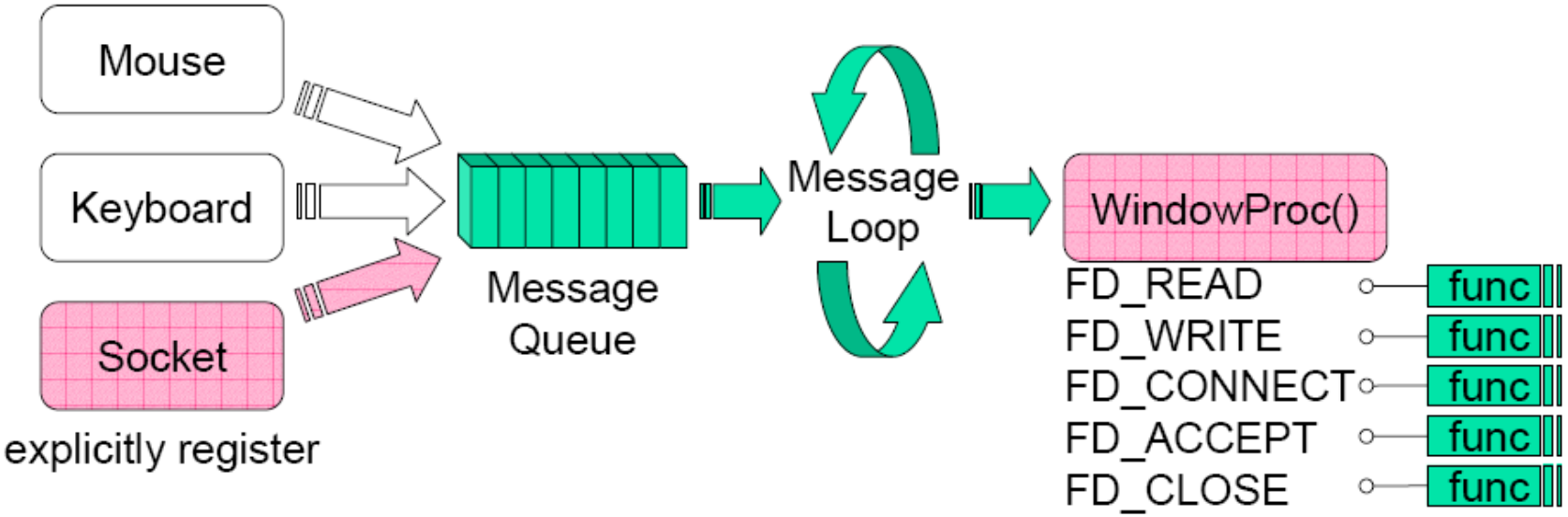


- Синхронните съобщения са такива съобщения, които MSWindows разполага в опашка на съобщенията на приложението. Тези съобщения се извличат и се диспетчеризират в цикъл за обработване на съобщенията.
- Асинхронните съобщения се предават непосредствено на прозореца, когато MSWindows извиква прозоречна процедура.
  - Съобщенията не приличат на апаратните прекъсвания. Докато прозоречната процедура обработва съобщението, програмата не може да бъде прекъсната от друго съобщение. Само в случай, че функцията, която се изпълнява в тялото на прозоречната процедура, генерира ново асинхронно съобщение, то извиква повторно прозоречна процедура, и когато приключи неговата обработка, тогава изпълнението на прекъсната функция ще продължи.

Управлявано от съобщения асинхронно изпълнение



Поток от съобщения



Управляван от съобщения вход-изход

- Функциите за вход-изход продължават да работят във фонов режим и програмата е уведомена за:
  - Приключила I/O операция
  - I/O операцията може да се изпълни
  - Сработила е грешка на I/O
- Уведомяването се реализира чрез прозоречни съобщения. Всяко Windows приложение (има GUI) използва цикъл за обработване на съобщенията
  - Когато се използва управляем от съобщения вход-изход, ние определяме потребителското съобщение, което стои в цикъл от съобщения, които трябва да уведомят за състоянието на операция за IO.

Регистрация на съобщенията при Winsock