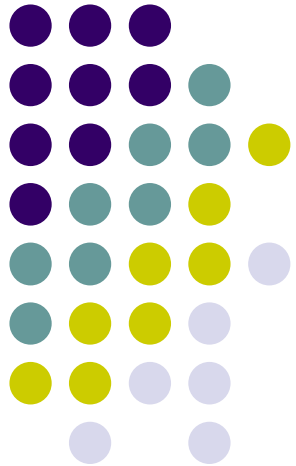
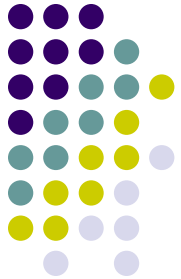
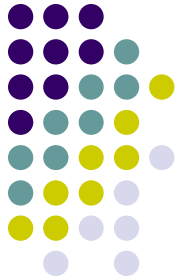
Мрежово програмиране

UDP сокети





Интерфейсът на сокетите описва набора от програмни функции или процедури, позволяващи разработването на приложения за използване в TCP/IP мрежите.

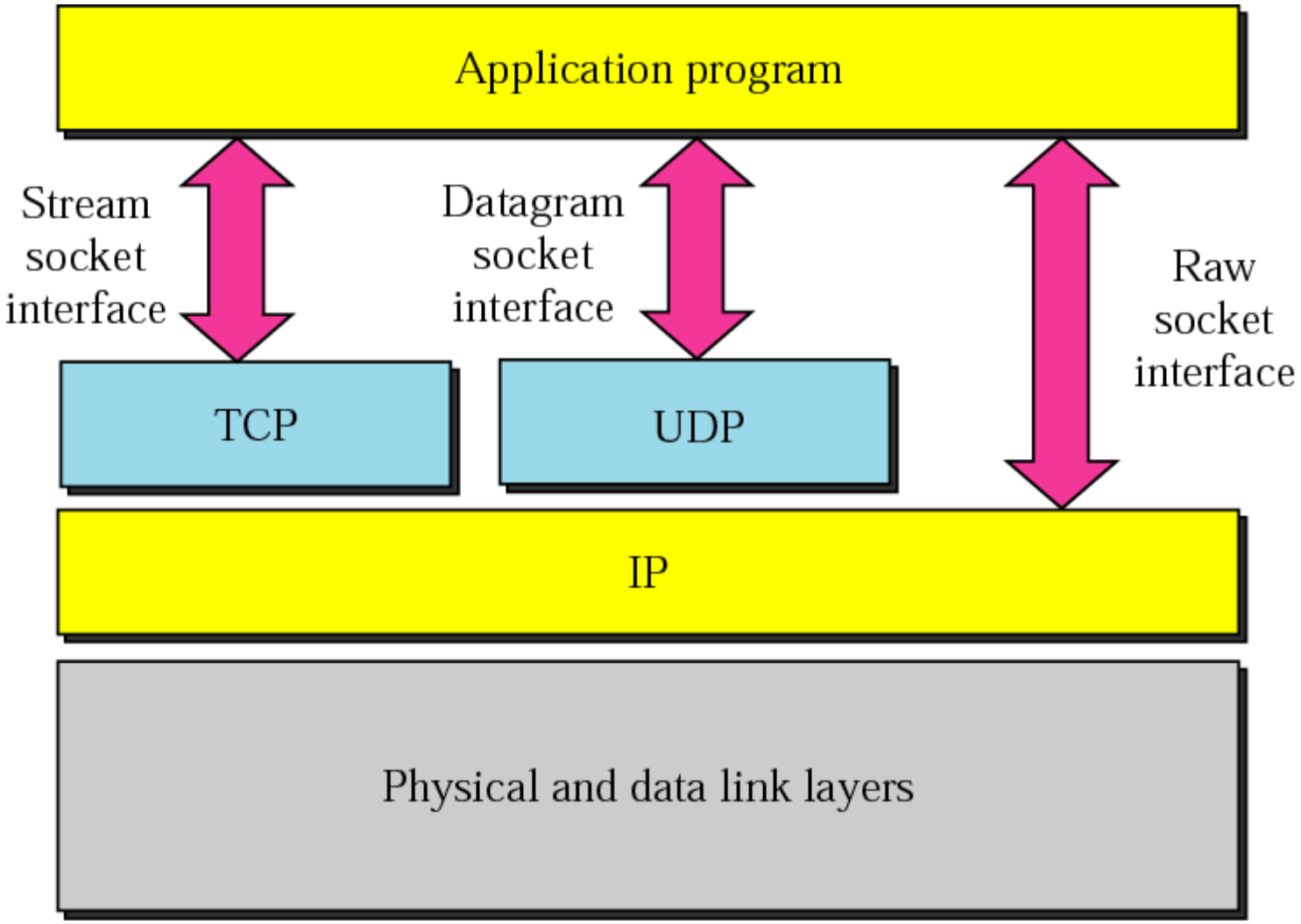


През осемдесетте години на миналия век ARPA е финансирала реализацията на TCP/IP протоколите за UNIX в Калифорнийския университет в град Бъркли.

Група от изследователи-програмисти разработва интерфейс за приложно програмиране за TCP/IP мрежови приложения (TCP/IP API).

Този интерфейс бил наречен *сокети на TCP/IP* (TCP/IP sockets).

Socket Types

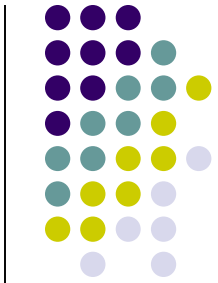


TCP протокол

- *Надежден* – ако се изгуби или повреди TCP сегмент, реализацията на TCP ще открие това и повторно ще предаде необходимия сегмент.
- *На база на логическо съединение* – преди да започне да предава данни TCP установява с отдалечената машина съединение, обменя с нея служебна информация.
- *С непрекъснат поток от данни* – TCP осигурява механизъм за предаване, позволяващ прехвърлянето на произволен брой байтове.

UDP протокол

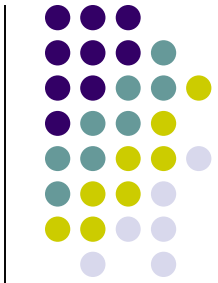
- *Ненадежден* – UDP няма вграден механизъм за откриване на грешки, нито средства за повторно предаване на повредени или изгубени данни.
- *Без установяване на логическо съединение* – преди да предава данни UDP не установява логическо съединение. Информацията се предава като се предполага, че приемащата страна я очаква.
- *Основан на дейтаграми* – UDP позволява на приложенията да прехвърлят информацията във вид на съобщения, предавани посредством дейтаграми, които се явяват единици за предаване на данни в UDP. Приложението е длъжно самостоятелно да разпределя данните в различни дейтаграми.



UDP е полезен, когато мощните механизми за осигуряване на надеждност на TCP, не са задължителни.

UDP протоколът съхранява границите на съобщенията, определяни от приложния процес.

Той никога не обединява няколко съобщения в едно цяло и не дели едно съобщение на части.

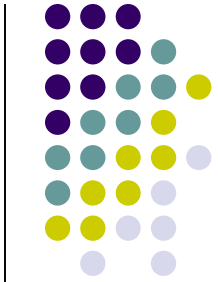


Системните примитиви за вход-изход в UNIX изглеждат като последователен цикъл, съставен от операции от типа *отваряне-четене-записване-затваряне.*

Първоначално разработващите интерфейса на сокетите изследователи са пробвали да накарат мрежовия вход-изход да функционира по същия механизъм, както всеки друг вход-изход в UNIX.

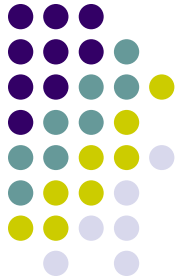
* *Everything in Unix is a file*

Проблеми



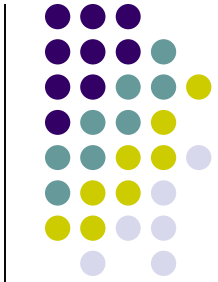
- Разработчиците на сокетите са могли лесно да реализират такъв API за входно-изходната системата на UNIX, в който програмистът да може да създаде програма-клиент, активно установяваща мрежово съединение. Но същият този API е длъжен да позволи създаването и на програми-сървъри, пасивно чакащи, докато някой не се обърне към тях. Обичайната система за вход-изход на UNIX на практика не може пасивно да въвежда и да извежда данни.
- Разработчиците са били принудени да създадат набор от нови функции за обработка на пасивни операции за въвеждане-извеждане.

- Стандартните функции за вход-изход в UNIX лошо могат да установяват съединение - те използват фиксиран адрес на файла и устройството за обръщение към него. Адресът на файла или на устройството за всеки компютър е постоянна величина. Съединението (или пътят) към файла или устройството е достъпно през целия цикъл на запис-четене, т.е. докато програмата не затвори съединението.
- Фиксираният адрес е добра идея, ако протоколът за предаване на данни е ориентиран на съединение. За неориентираните на съединение протоколи фиксираният адрес е проблемен.



В отговор на системното извикване UNIX връща така наречения *дескриптор на файла* (file descriptor); понякога го наричат указател (file handler).

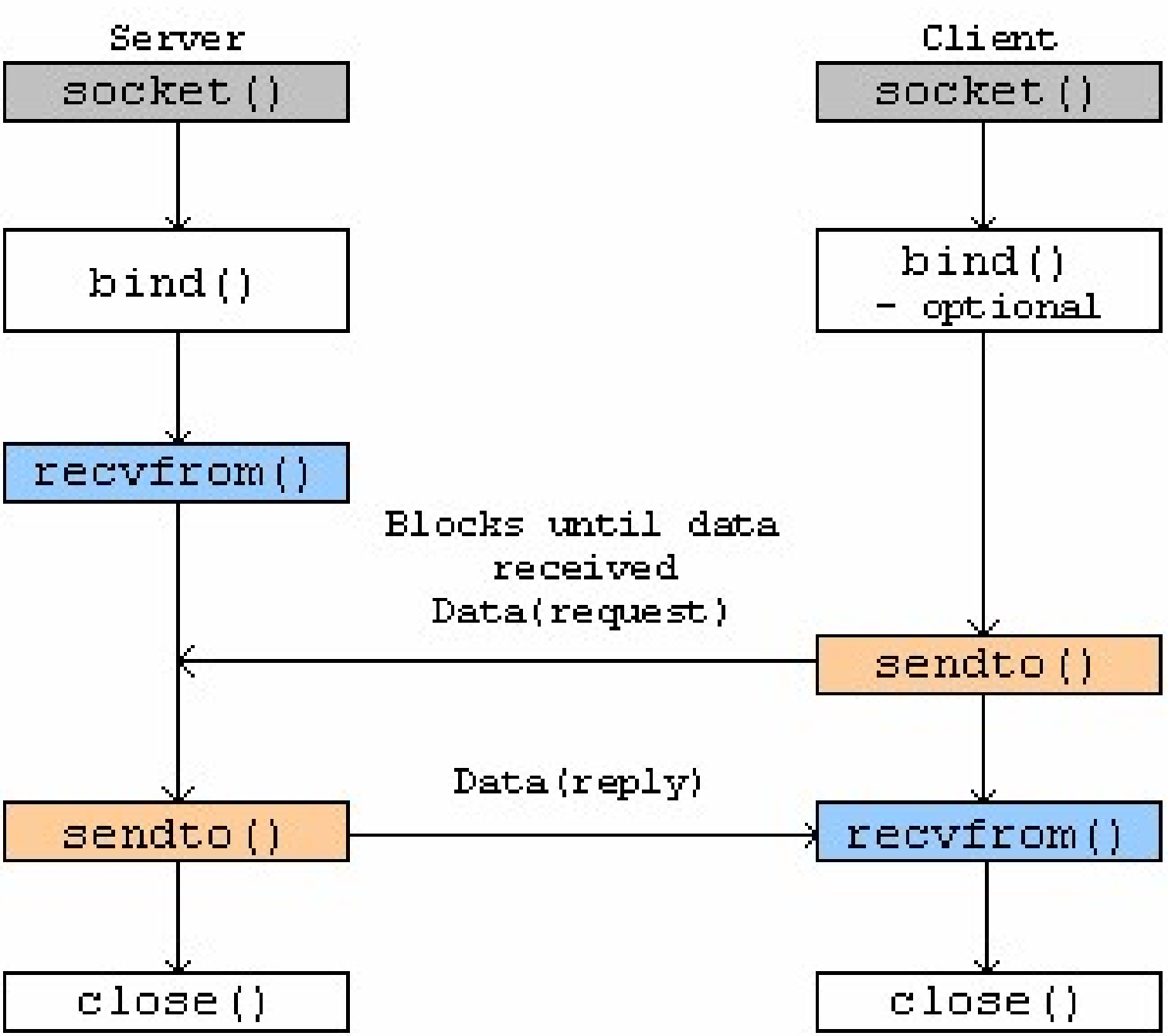
Дескрипторът на сокетите в интерфейса както по-рано се е наричал дескриптор на файла и информацията за сокета се съхранява в същата таблица, в която и дескрипторите на файловете.



Разликите между дескриптора на сокета и дескриптора на файла

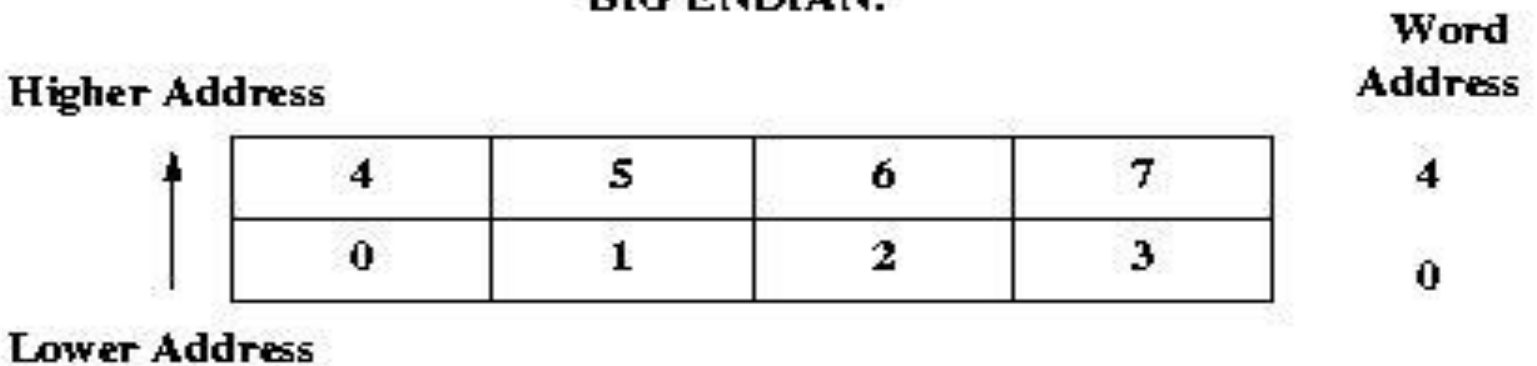
- дескрипторът на сокета не съдържа някакви определени адреси или целеви пунктове на мрежовото съединение- дескрипторът на сокета не представлява определена мрежова точка (endpoint)

Опростено представяне на структурата от данни на сокета


Server
socket()
bind()
recvfrom()
Blocks until data
received
Data(request)
sendto()
Data(reply)
sendto()
close()
Client
socket()
bind()
- optional
sendto()
recvfrom()
close()

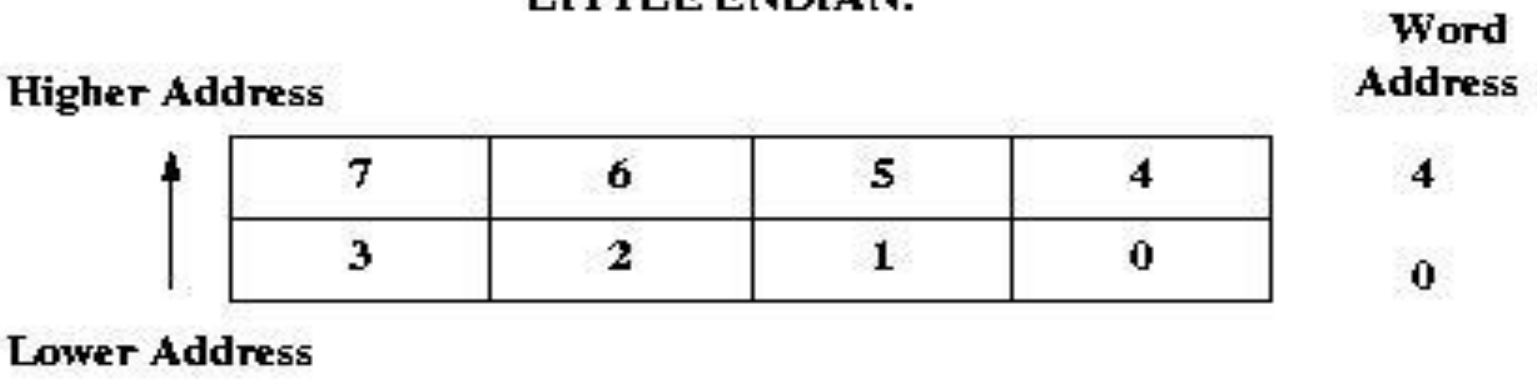
Проблеми на представянето

BIG ENDIAN:

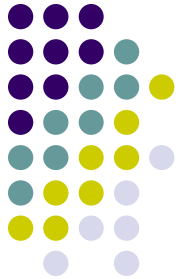


* Most-significant byte at lowest address.
* Word is addressed by most-significant byte

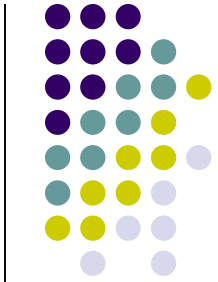
LITTLE ENDIAN:



* Least-significant byte at lowest address.
* Word is addressed by least-significant byte



- Предаването от един изчислителен комплекс към друг на символна информация, като правило (когато един символ заема един байт), не поражда проблеми.
- Обаче за числова информация ситуацията се усложнява.
- Целите числови данни от представянето, прието на компютъра-изпращач, се преобразуват от потребителския процес в мрежова подредба на байтовете, в такъв вид пътешестват по мрежата и се превеждат в необходимата подредба на байтовете на машината-получател от процеса, за който те са предназначени.



В i80x86 е приета подредба на байтовете, при която младшите байтове на цяло число имат младши адреси. При мрежовата подредба на байтовете, приета в Интернет, младши адреси имат старшите байтове на числото.

Нека имаме две цели 32-битови числа:

0XFFC3B2A7 и 0X21F2CE07,

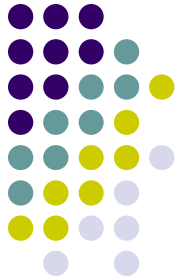
които са записани в паметта последователно.

little-endian = host order A7 B2 C3 FF 07 CE F2 21

big-endian=network order FF C3 B2 A7 21 F2 CE 07

Функции за преобразуването на реда на байтовете

```
#include <netinet/in.h>
unsigned long int htonl(
    unsigned long int hostlong);
unsigned short int htons(
    unsigned short int hostshort);
unsigned long int ntohl(
    unsigned long int netlong);
unsigned short int ntohs(
    unsigned short int netshort);
```



- За числата с плаваща точка всичко е в пъти по-сложно. На различните машини могат да се различава не само реда на байтовете, но и формата на представяне на това число. Прости функции за тяхното коригиране не съществуват.
- Ако се налага да се обменят реални данни, тогава или това следва да се прави в хомогенна мрежа, съставена от еднакви компютри, или да се използват символни и цели данни за предаването на реални стойности.

Функции за преобразуване на IP адреси

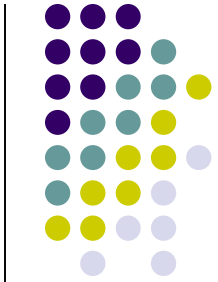
```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
int inet_aton(const char *strptr,
    struct in_addr *addrptr);
char *inet_ntoa(struct in_addr *addrptr);
```

Функцията inet_aton преобразува символен IP адрес, разположен според указателя strptr, в числово представяне в мрежова подредба на байтовете и го записва в структурата, разположена на адреса addrptr. Функцията връща стойност 1, ако в стринга е записан правилен IP адрес и стойност 0 в противен случай.

Структурата от типа struct in_addr се използва за съхранение на IP адресите в мрежова подредба на байтовете:

```
struct in_addr {
    in_addr_t s_addr;
};
```

Това, че се използва структура, състояща от една променлива, а не самата 32-битова променлива, се е получило исторически.



Функцията inet_addr често пъти се използва в програмите. Тя получава стринг и връща адрес (вече в мрежова последователност на байтовете). Проблемът при тази функция е в това, че стойността -1, връщана при грешка, представлява коректен адрес 255.255.255.255

За обратното преобразуване се прилага функцията inet_ntoa(). Числовото представяне на адреса в мрежова подредба на байтовете е длъжно да бъде записано в структурата от типа struct in_addr, адресът на която addrptr предава на функцията като аргумент. Функцията връща указател към стринг, съдържащ символно представяне на адреса. Този стринг се поставя в статичен буфер, като при следващи извиквания новото съдържимо заменя старото.

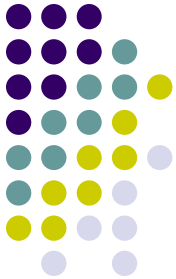
Функция bzero()

```
#include <string.h>
void bzero(void *addr, int n);
```

Функция bzero запълва първите n байта, започвайки от адреса addr с нулеви стойности. Функцията нищо не връща.

bzero (), bcopy () и bcmp () позволяват да се инициализира символен низ от нули, да се копира един низ в друг и да се сравнят два стринга, съответно. В дадения случай, става дума за примитивите в BSD. Подобни примитиви има и в системите на базата на System V: memset (), memcpy () и memcmp ().

Създаване на сокет
Системен примитив socket()



```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type,
           int protocol);
```

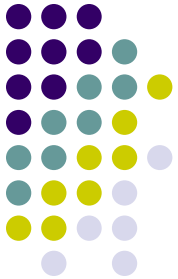
- С всеки сокет се свързват три атрибута: домейн, тип и протокол.
- Тези атрибути се задават при създаването на сокета и остават непроменени през цялото време на съществуването на сокета.

Параметърът domain определя семейството протоколи

- PF_INET- за TCP/IP семейството протоколи;
- PF_UNIX- за семейството вътрешни протоколи на UNIX, още наричано UNIX domain.

Параметърът type определя семантиката на обмена на информацията

- SOCK_STREAM- за свързване чрез установено виртуално съединение;
- SOCK_DGRAM- за обмен на информация чрез съобщения.



SOCK_STREAM

Data streams

Потоково предаване

- *Мрежовата подсистема на операционната система създава два потока, по които данните се предават в двете посоки (скоростта варира)*.
- В TCP/IP протоколния стек съществува само един протокол за потокови сокети – TCP (затова третият параметър за транспортните протоколи на TCP/IP се игнорира).

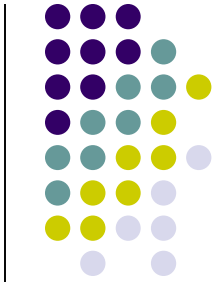
SOCK_DGRAM

Data packets

Пакетно предаване

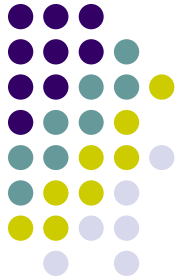
- *Данните се предават на порции*.
- В TCP/IP протоколния стек съществува само един протокол за дейтаграмни сокети – UDP (затова третият параметър за транспортните протоколи на TCP/IP се игнорира).

Параметърът protocol



- този параметър ще го задаваме равен на 0.

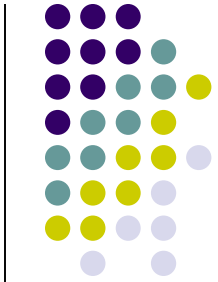
Иначе използваните протоколи са няколко и трябва или много да се задълбочим в реализацията на сокетите или да създаваме свой собствен протокол.



- В случай на успешно приключване системният примитив връща файлов дескриптор (стойност по-голяма или равна на 0), който ще се използва като *указател към създадения комуникационен възел* при всички следващи мрежови извиквания. При възникване на някаква грешка връща -1.

- Указателят към структурата от данни за създадения сокет се разполага в таблицата на отворените файлове на процеса, подобно на това, както работи за програмните канали (pip’ове, FIFO).
- Системният примитив връща на потребителя файлов дескриптор, съответстващ на попълнения елемент на таблицата, който наричаме дескриптор на сокета.
- Такъв начин за съхраняване на информация за сокета позволява първо: процесите-деца да я наследяват от процесите-родители и второ: да се използват за сокети част от системните примитиви, вече изучавани при работа с програмните канали (pip’ове и FIFO): close(), read(), write().

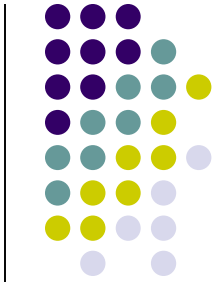
Настройка на адреса на сокета. Системен примитив bind()



```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockd,
        struct sockaddr *my_addr,
        int addrlen);
```

Първият параметър на примитива sockd е длъжен да съдържа дескриптора на сокета, за който се прави тази настройка на адреса.

Параметърът my_addr представлява адрес на структурата struct sockaddr, съдържаща локалната част на пълния адрес на сокета

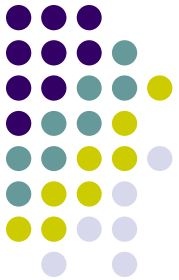


Структурата-шаблон struct sockaddr е описана във файла <sys/socket.h> по начина:

```
struct sockaddr {
    short sa_family; // Семейство адреси, AF_xxx
    char sa_data[14]; // 14 байта за разполагане на адреса
};
```

Тя трябва да е конкретизирана (в зависимост от използваното семейство протоколи) и запълнена преди извикването.

Параметърът addrlen трябва да съдържа фактическата дължина на структурата, адреса на която се предава като втори параметър.



- Тази дължина при различните семейства протоколи и даже в рамките на едно семейство протоколи може да бъде различна (например за UNIX Domain).
- Размерът на структурата, съдържаща адрес на сокета, за TCP/IP семейството протоколи може да бъде определен като sizeof(struct sockaddr_in).

Системният примитив връща стойност 0 при нормално приключване и -1 при регистрирана грешка.

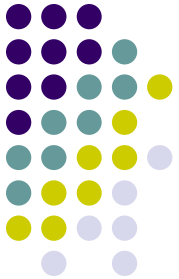
Ще използваме адрес на сокета от следния вид, описан във файла <netinet/in.h>:

```
struct sockaddr_in{
    short sin_family;
    /* Избраното семейство протоколи за TCP/IP - PF_INET */
    unsigned short sin_port;
    /* 16 битов № на порт в мрежова подредба на байтове */
    struct in_addr sin_addr;
    /* Адрес на мрежовия интерфейс */
    char sin_zero[8];
    /* Това поле е задължително да бъде запълнено с нули */
};
```

Номер на порт

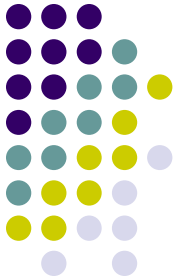
- 0 - избира се от ОС;
- от 1 до 1023 – забранено;
- от 1024 до 49151 – нежелателно;
- 49152 до 65535 – разрешено за потребителски процеси.

Съществуват два варианта за задаване на номер на порт: фиксиран порт по желание на потребителя и порт, който произволно е зададен от операционната система (0).

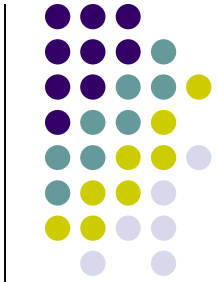


IP адресът при настройката може да бъде на конкретен мрежов интерфейс, а може да бъде зададен за цялата изчислителна система. В първия случай в качеството на стойност на полето на структурата sin_addr.s_addr се използва числова стойност на IP адреса на конкретния мрежов интерфейс в мрежова подредба на байтовете. Във втория случай тази стойност трябва да е равна на стойността на константата INADDR_ANY, приведена към мрежова подредба на байтовете.

Системни примитиви sendto() и recvfrom()

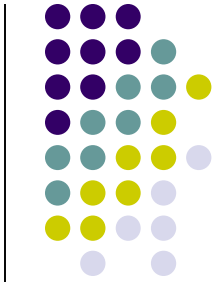


```
#include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockd, char *buff,
    int nbytes, int flags,
    struct sockaddr *to, int addrlen);
int recvfrom(int sockd, char *buff,
    int nbytes, int flags,
    struct sockaddr *from, int addrlen);
```

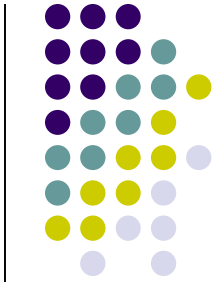


За изпращане на дейтаграми се прилага системния примитив sendto(). В него участват следните параметри:

- дескриптор на сокета, чрез който се изпраща дейтаграмата;
- адрес на област от паметта, където се намират данните, които са длъжни да съставляват съдържателната част на дейтаграмата и тяхната дължина;
- флагове, определящи поведението на системния примитив (в нашия случай те винаги ще са 0);
- указател към структурата, съдържаща адреса на сокета на получателя и нейната фактическа дължина.

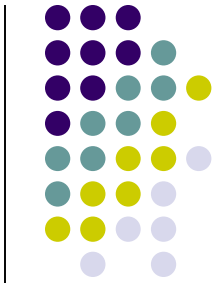


- Системният примитив връща -1 при възникване на грешка и броя на реално изпратените байтове при нормална работа. Нормалното приключване на системния примитив не означава, че дейтаграмата вече е напуснала вашия компютър. Дейтаграмата първо се помества в системния мрежов буфер, а нейното реално изпращане може да стане след възвратa (return) от системния примитив. Примитивът sendto() може да се блокира, ако в мрежовия буфер не достига място за дейтаграмата.

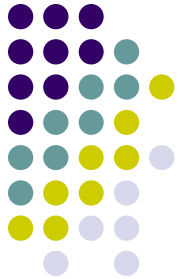


За четене на приетите дейтаграми и определене на адреса на получателя (при необходимост) служи системния примитив recvfrom(). Параметри на този примитив:

- Дескриптор на сокета, чрез който се приема дейтаграмата.
- Адрес на областта от паметта, където следва да се заредят данните, съставляващи съдържателната част на дейтаграмата.
- Максимална дъжина, допустима за дейтаграмата. В случай, че броят на данните в дейтаграмата превишава зададената максимална дължина, тогава примитивът по премълчаване разглежда това като грешна ситуация.
- Флагове, определящи поведението на системния примитив (в нашия случай те ще се задават равни на 0).

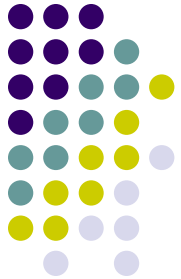


- Указател към структурата, в която при необходимост може да бъде записан адреса на сокета на изпращача. Ако този адрес не е нужен, тогава може да се посочи стойността NULL.
- Указател към променлива, съдържаща максимално възможната дължина на адреса на изпращача. След връщането от системния примитив в нея ще бъде записана фактическата дължина на структурата, съдържаща адреса на изпращача. Ако предишният параметър има стойност NULL, тогава и този параметър може да има стойност NULL.



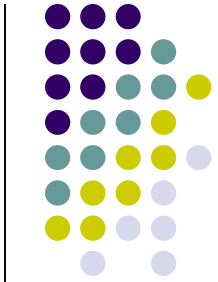
Системният примитив recvfrom() по премълчаване се блокира, щом отсъстват приети дейтаграми, докато не се появи дейтаграма.
При възникване на грешка той връща -1, при нормална работа- дължината на приетата дейтаграма.

Заглавни файлове:



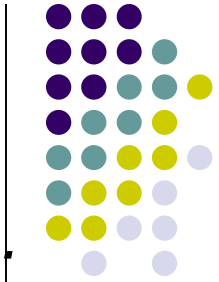
- **<sys/socket.h>** - основни функции за работа със сокети и базови структури от данни
- **<netinet/in.h>** - структури от данни, специфични за мрежовите сокети
- **<sys/un.h>** - структури от данни, специфични за локалните сокети
- **<arpa/inet.h>** - функции за манипулиране с IP адреси
- **<netdb.h>** - функции за преобразуване имената на протоколите и хостовете в числови стойности (в това число и DNS заявки)

Алтернативата TCP – UDP позволява на програмиста гъвкаво и рационално да използва предоставените ресурси, отчитайки конкретните възможности и потребности



- Сокетите позволяват да се представи мрежовия интерфейс като просто устройство за вход–изход и да се работи с него по начин, подобен на работа с обикновен файл.
- Понятието сокет не се ограничава само в рамките на TCP/IP.
- Сокетите са станали стандарт де-факто.
- Във версията BSD 4.3. на системата UNIX сокетите са реализирани в ядрото. Във версията System V Release 4, сокетите са реализирани като библиотека-надстройка над TLI (Transport Level Interface).

New address family: AF_INET6



New address data type:

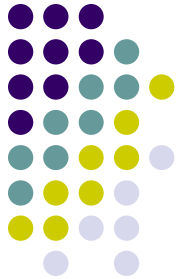
in6_addr

```
struct   in6_addr {
    uint8_t   s6_addr[16];
};
```

New address structure:

```
struct sockaddr_in6 {
uint8_t          sin6_len;
sa_family_t      sin6_family;
in_port_t        sin6_port;
uint32_t         sin6_flowinfo;
struct in6_addr  sin6_addr;
};
```

Функции за преобразуване на IP адреси

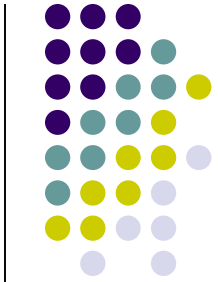


Стандартът POSIX 1003.1-2001 препоръчва вместо функцията inet_aton() да се използва функцията inet_pton, понеже тя поддържа както IPv4, така и IPv6:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
int inet_pton(int af, const char *src, void *dst);
```

af: AF_INET (dst: struct in_addr*), AF_INET6 (dst: struct in6_addr*)

Функцията inet_pton() преобразува IP адрес, задаван с аргумента src, от стрингов формат в мрежова последователност на байтовете в съответствие със зададеното семейство адреси af.



Стандартът POSIX 1003.1-2001 препоръчва вместо функцията inet_ntoa() да се използва функцията inet_ntop, понеже тя поддържа както IPv4, така и IPv6:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
const char *inet_ntop(
    int af, const void *src, char *dst, size_t cnt);
```

af: AF_INET (src: struct in_addr*), AF_INET6 (src: struct in6_addr*)

Функцията inet_ntop() преобразува IP адрес, задаван с аргумента src, от мрежова подредба на байтовете в стрингов формат. Полученият ASCIIZ стринг се записва в зададения от потребителя буфер dst.

Функции за работа с БД на възлите в мрежата (DNS)

```
#include <netdb.h>
#include <sys/socket.h>
struct hostent *gethostbyname(const char *cp);
struct hostent *gethostbyaddr(
    const char *addr, int len, int type);
```

За преобразуване на доменното име в IP адрес се използва функцията gethostbyname.

Функцията търси в БД на възлите (DNS) информация за посочения хост. За gethostbyname() хостът се задава с име на домейн или с IP адрес в стрингов формат.

Тази функция получава името на хоста и връща указател към структура с неговото описание hostent.

За gethostbyaddr() хостът се задава с IP адрес в бинарен вид (addr), вторият аргумент определя дължината на адреса, а третият е AF_INET или AF_INET6.

```
struct hostent {
    char *h_name;        /*официално име на хоста*/
    char **h_aliases;    /*масив с псевдоними*/
    int h_addrtype;      /*тип на адреса - AF_INET*/
    int h_length;        /*дължина на адреса в байтове*/
    char **h_addr_list;  /*масив с адреси*/
};
```

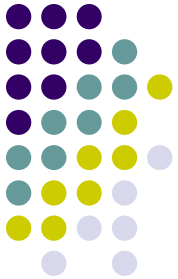
Масивите h_aliases и h_addr_list са ограничени от елемент със стойност 0. Адресите се съхраняват в мрежова подредба на байтовете. Функциите gethostbyname() и gethostbyaddr() връщат указател към структура, поместване в областта с данни на ядрото на ОС. Ако за gethostbyname() в качеството на параметър се зададе IP адресът на хоста, тогава той ще се копира в полето h_name (няма да се изпълни DNS заявка). Ако се налага да се получи името на домейна от IP адреса, използвайте gethostbyaddr().

Функции за работа с БД за информация за мрежовите услуги

```
#include <netdb.h>
struct servent *getservbyname(
    const char *name, const char *proto);
struct servent *getservbyport(
    int port, const char *proto);
```

Функцията getservbyname() търси в БД (/etc/services) информация (номер на порта) за зададената мрежова услуга. Функцията getservbyport() връща информация за мрежовата услуга според номера на нейния порт (port - в мрежова подредба на байтовете).

Функции за работа с БД за информация за протоколите



```
#include <netdb.h>
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent {
    char *p_name;       /*официално име*/
    char **p_aliases;   /*псевдоними*/
    int p_proto;        /*номер на протокола*/
};
```

Структурата servent представя запис за дадена услуга:

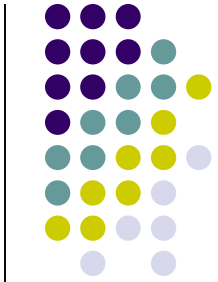
```
struct servent {
char *s_name;     // официалното име на услугата
char **s_aliases; // масив от стрингове, съдържащи псевдоними
                  // (алтернативни имена на услугата), завършващи с 0
int s_port;       // номер на порт в network byte order
char *s_proto;    // име на протокол
}
struct servent *getservent(void);
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
```

Функцията getservent прочита следващия ред от файла /etc/services и връща структура servent, описваща тази услуга. Ако се налага, файлът /etc/services първо се отваря за четене.

Четири типа данни, отнасящи се до мрежите

Тип на данните	Файл	Структура	Функции за търсене по ключ
възли	/etc/hosts	hostent	gethostbyaddr, gethostbyname
мрежи	/etc/networks	netent	getnetbyaddr, getnetbyname
протоколи	/etc/protocols	protoent	getprotobyname, getprotobynumber
услуги	/etc/services	servent	getservbyname, getservbyport

За всеки от четирите типа данни съществува собствена структура (изисква се включване на <netdb.h>). За всеки от четирите типа са определени три функции: getXXXent (чете следващия запис във файла и при необходимост затваря файла), setXXXent (отваря файла, ако все още не е отворен и преминава в началото на файла) и endXXXent (затваря файла). За всеки от четирите типа данни има фукции за търсене по ключ (keyed lookup). Те последователно преминават през файла като извикват getXXXent и търсят елемент, съвпадащ с аргумента. Търсенето по ключ има вида: getXXXbyYYY (като gethostbyname и gethostbyaddr).



#include <unistd.h>

int **gethostname**(char *hostname, size_t size);

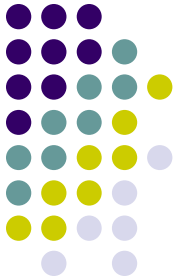
Функцията *gethostname* се използва за получаване името на локалния хост. После то може да се преобразува в адрес с помощта на gethostbyname. Така можем да получим адреса на машината, на която се изпълнява нашата програма.

#include <sys/socket.h>

int **getpeername**(int sockfd, struct sockaddr *addr, int *addrlen);

Функцията *getpeername* позволява да се научи адреса на сокета на отдалечената страна. Тя получава дескриптора на сокета, съединен с отдалечения хост и записва адреса на този хост в структура, към която сочи addr. Фактическият брой записани байтове се разполага според адреса addrlen (като там следва да стои размера на структурата addr преди извикването на *getpeername*). Полученият адрес при необходимост може да се преобразува в стринг, използвайки inet_ntoa или gethostbyaddr.

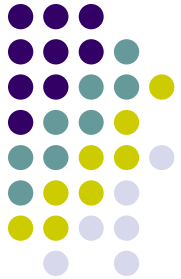
Функцията **getsockname** по предназначение е обратна на *getpeername* и позволява да се определи адреса на сокета на локалната страна.



struct netent ***getnetbyname** (const char *name)
функцията *getnetbyname* връща информация за мрежа, именувана с name.
struct netent ***getnetbyaddr** (long net, int type)
Функцията *getnetbyaddr* връща информация за мрежа от зададен тип с номер net.

може да се преглежда база от данни за мрежите, използвайки setnetent, getnetent, и endnetent.
Тези функции не са предназначени за повторно използване.
void setnetent (int stayopen)
функцията отваря базата от данни за мрежите.
Ако аргументът stayopen е различен от нула,
тя установява флага така, че последващите обръщения към *getnetbyname* или *getnetbyaddr* да не затварят базата данни.
Така ако се извикват тези функции многократно се избягва повторно отваряне на базата данни за всяко обръщение.
struct netent *getnetent (void)
функцията връща следващия вход в базата данни за мрежите.
void endnetent (void)
функцията затваря базата данни за мрежите.

Четене или установяване на параметри, свързани със сокета
Функции getsockopt и setsockopt



#include <sys/socket.h>
int **getsockopt**(int *sockfd*, int *level*, int *optname*, void **optval*, socklen_t **optlen*);
int **setsockopt**(int *sockfd*, int *level*, int *optname*, const void **optval*, socklen_t *optlen*);

И двете функции връщат 0 при успех и -1 при грешка.

optname и всички посочени параметри без промяна се предават за интерпретация на съответстващия модул на протоколите.
sockfd - задава дескриптора на сокета;
level - определя кода, с който да се интерпретира параметъра;
optval - указател към променлива, от която се извлича новата стойност на параметъра с помощта на функцията *setsockopt* или в която се съхранява текущата стойност на параметъра с помощта на функцията *getsockopt*; Размерът на тази променлива се задава от последния аргумент.

Някои параметри на сокетите за функциите getsockopt и setsockopt

level	optname	get	set	описание
SOL_SOCKET	SO_RCVBUF	*	*	размер на вх.буфер
	SO_SNDBUF	*	*	размер на изх.буфер
	SO_RCVLOWAT	*	*	min p-p на вх.буфер
	SO_SNDLOWAT	*	*	max p-p на изх.буфер
	SO_BROADCAST	*	*	разрешава broadcast
	SO_DEBUG	*	*	разрешава debug
IPPROTO_IP	IP_TTL	*	*	време на живот
	IP_TOS	*	*	TOS