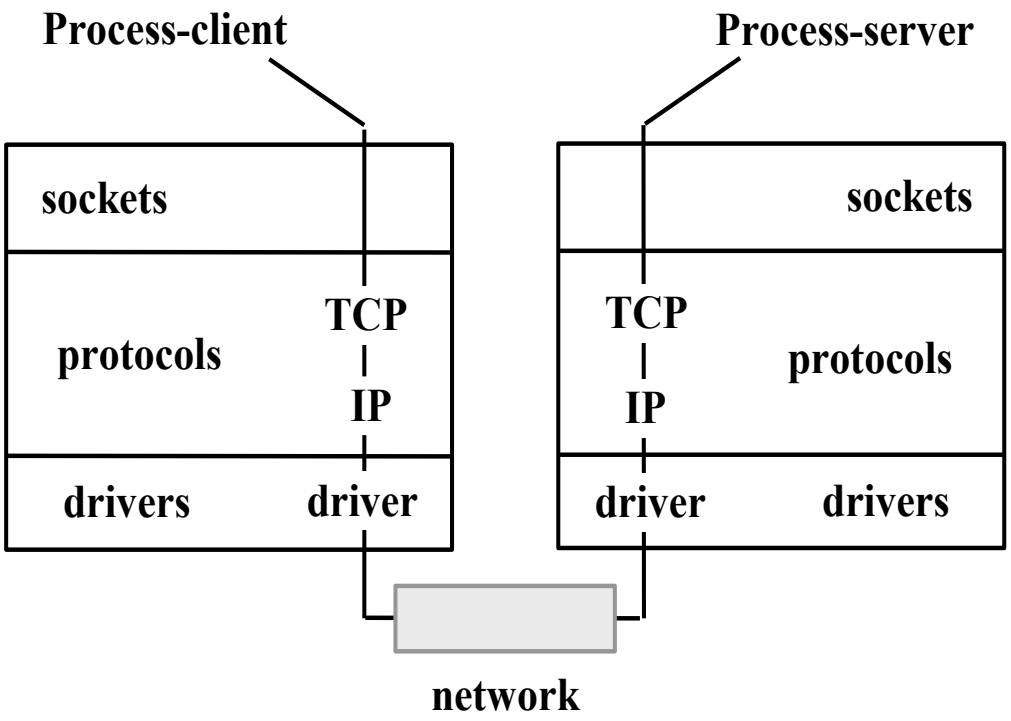
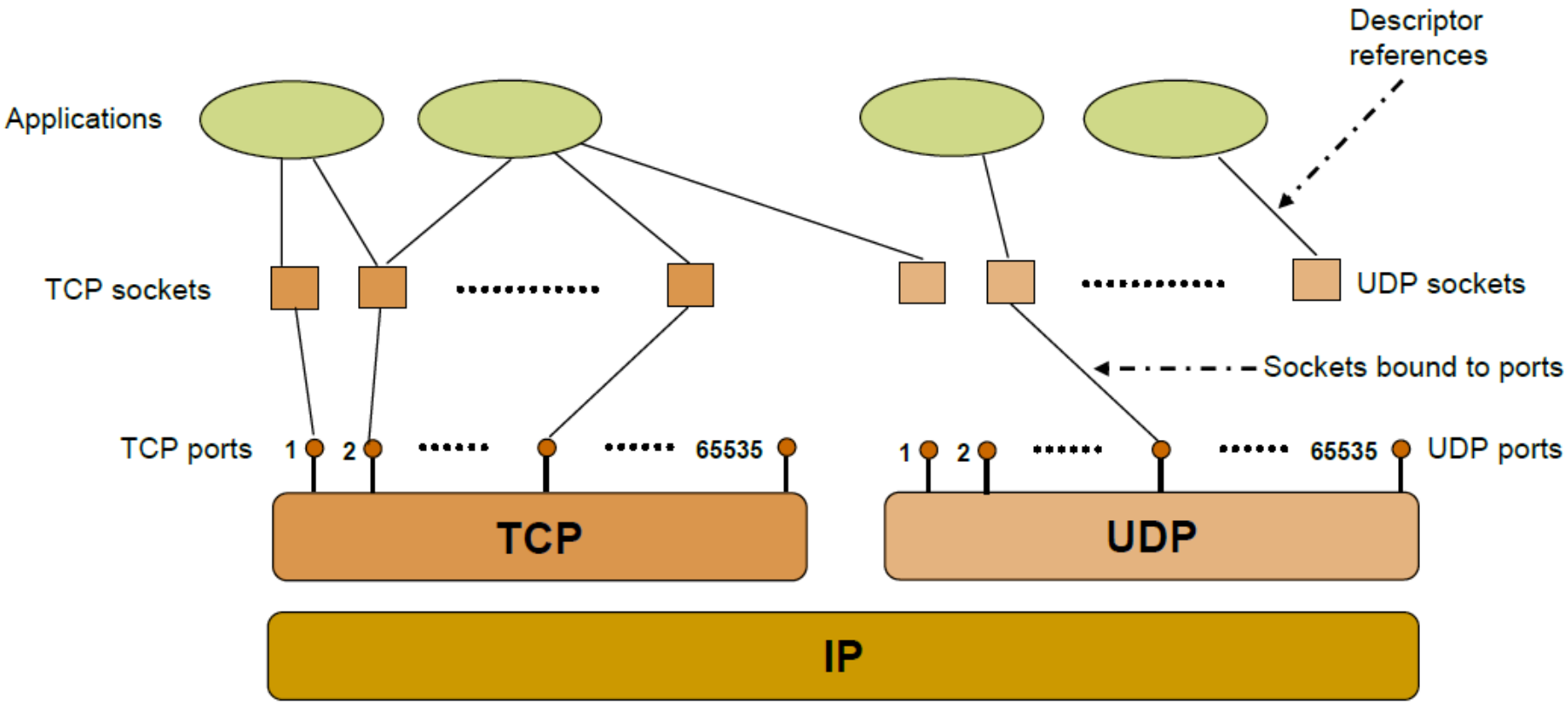
Програмиране със сокети

Сокети



- Предоставят двупосочна комуникация между два процеса от тип point-to-point.

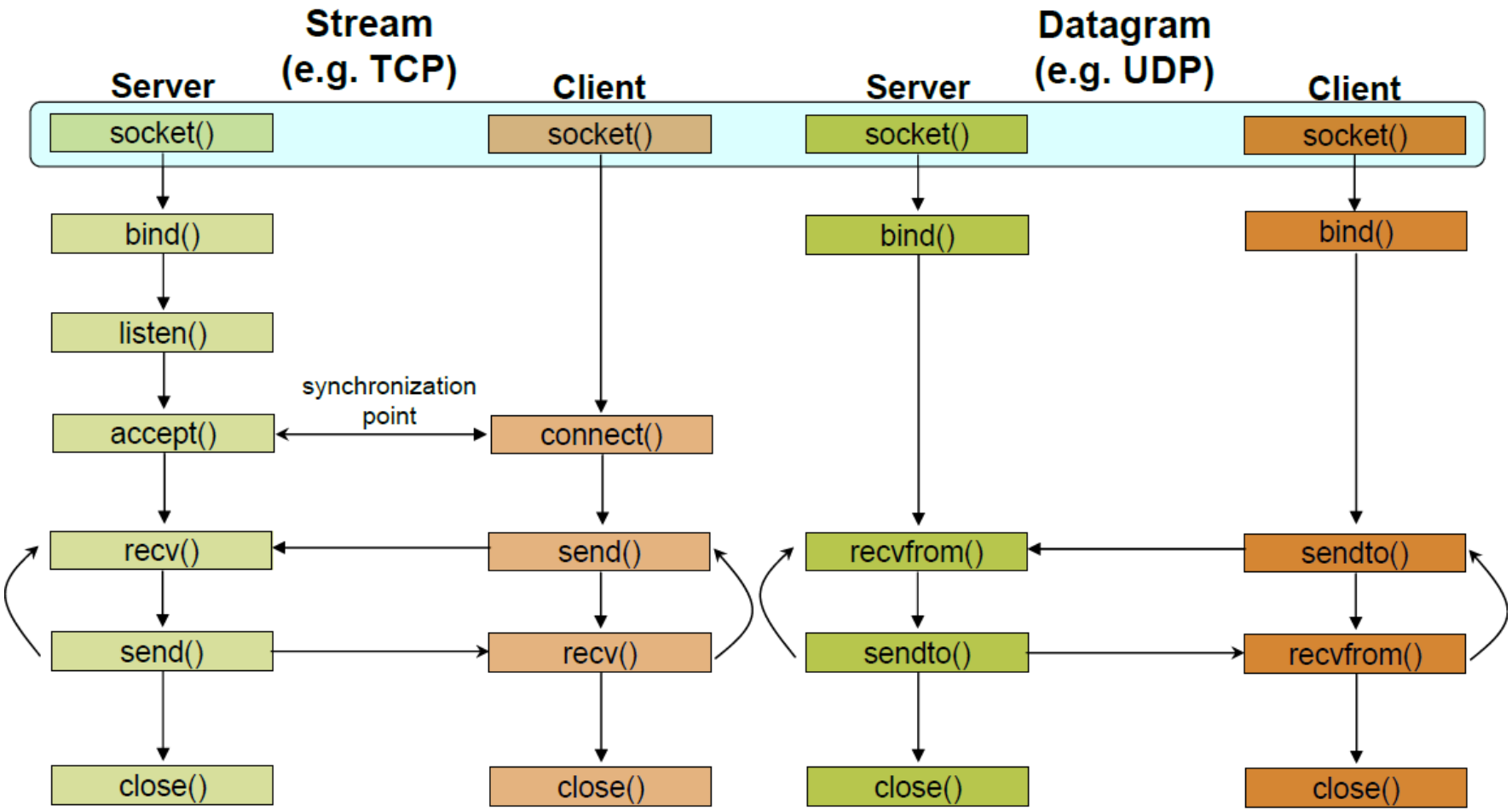
Сокети в Интернет домейна



Типове сокети

- Stream socket
- Datagram socket
- Sequential packet socket
- Raw socket

Комуникация клиент-сървър



Конструиране на съобщенията – подреждане на байтовете

- Адресите и портовете са integer
- Различните машини или ОС използват различно подреждане на байтовете:
  - Little-endian – старшите байтове напред;
  - Big-endian – младшите байтове напред.

Конструиране на съобщенията – подреждане на байтовете

192	168	0	20

Big-endian

192	A
168	A+1
0	A+2
20	A+3

192	168	0	20

Little-endian

20	A
0	A+1
168	A+2
192	A+3

Конструиране на съобщенията – решение

- **Host-byte ordering** – подреждането на байтовете в машината (Little-endian или Big-endian)
- **Network-byte ordering** – подреждането на байтовете в мрежата (винаги Big-endian)

Функции

- `u_long htonl (u_long x)`
- `u_short htons (u_short x)`
- `u_long ntohl (u_long x)`
- `u_short ntohs (u_short x)`

- При Big-endian машини не се извършват никакви действия
- При Little-endian машини се обръща реда на байтовете

Създаване на сокет

```
int socket ( int domain, int type, int protocol );
```

domain – фамилия комуникационни протоколи (AF_INET)

type – тип на сокета:

- **SOCK_STREAM – потоков сокет, базиран на TCP**
- **SOCK_DGRAM – датаграмен сокет, базиран на UDP**

protocol – мрежов протокол (0 – по подразбиране за фамилията)

Създава се сокет. Като резултат се връща дескрипторът на сокет или -1 при грешка.

Създаване на сокет

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

int main(int argc , char *argv[]) {
   int s;

   ...

   if ( socket(AF_INET,SOCK_STREAM,0)==-1 ){
        perror("Socket:");
        exit(1);
   }

   printf("Socket created.\n");

   return 0;
}
```

Структура за свързване през сокет

```
struct sockaddr {
    u_short    sa_family;       // communication domain
    char       sa_data[14];     // protocol-specific address
};
```

Структура за свързване през сокет

```
struct sockaddr_in {
   short       sin_family;     // Комуникационна област (AF_INET)
   u_short     sin_port;       // Номер на порт
   struct in_addr sin_addr;    // IP адрес
   char sin_zero[8];           // Не се използва
};

struct in_addr {
   u_long  s_addr;             // 32 битов IP адрес
};
```

Инициализиране на данните (при сървър)

```
#define  MYPORT   2345          // порт по който ще чака заявки

.  .  .

struct   sockaddr_in  myaddress;

myaddress.sin_family = AF_INET;       // Internet фамилия
myaddress.sin_port = htons(MYPORT);   // network byte order

myaddress.sin_addr.s_addr = INADDR_ANY; // ще
                                 // използва локалния си IP адрес

 .  .  .
```

Инициализиране на данните (при клиент)

```
. . .

struct   sockaddr_in  myaddress;

myaddress.sin_family = AF_INET;  // Internet фамилия
myaddress.sin_port = htons(0);   // ще получи автоматично от ОС
                                 // първия свободен порт

myaddress.sin_addr.s_addr = INADDR_ANY; // ще
                              // използва локалния си IP адрес

 . . .
```

Свързване към сокет (binding)

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

int bind ( int sockfd, struct sockaddr *my_addr,
                                int addrlen );
```

sockfd – дескриптор на създаден сокет
my_addr – променлива, съдържаща данни за сокета
addrlen – дължината на тези данни

Свързване към сокет (binding)

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


struct sockaddr_in  myaddress;

sockfd = socket(AF_INET, SOCK_STREAM, 0);

myaddress.sin_family = AF_INET;
myaddress.sin_port = htons(0);        // за клиент
myaddress.sin_addr.s_addr = INADDR_ANY;

if ( bind( sockfd, (struct sockaddr *)&myaddress,
                                sizeof(myaddress) ) == -1 ) {
   perror("Bind error:");
   exit (1)
}
```

Свързване към сървър

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

int connect ( int sockfd, struct sockaddr *srv_addr,
                                int srv_addrlen );
```

sockfd – дескриптор на създаден сокет

srv_addr – променлива, съдържаща данни за връзка със сървър

srv_addrlen – дължината на тези данни

Клиентът се блокира до създаване на конекция със сървъра. След създаването на конекцията, функцията завършва и клиентът продължава изпълнението си.

Свързване към сървър

```
#define  SRVPORT   2345        // порт на сървъра
#define  SRVIP "200.100.50.1" // IP адрес на сървъра

.  .  .

struct   sockaddr_in  srvaddress;

srvaddress.sin_family = AF_INET;      // Internet фамилия
srvaddress.sin_port = htons(SRVPORT);
srvaddress.sin_addr.s_addr = inet_addr(SRVIP);

if ( connect( sockfd, (struct sockaddr *)&srvaddress,
                      sizeof(srvaddress) ) == -1 ) {
       perror("Connect:");
       exit(1);
}

.  .  .
```

Изпращане и получаване на данни

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

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

sd – дескриптор на създадения сокет

buf – адрес в паметта, където се записват/четат данните

len – дължината на данните

flags – флагове

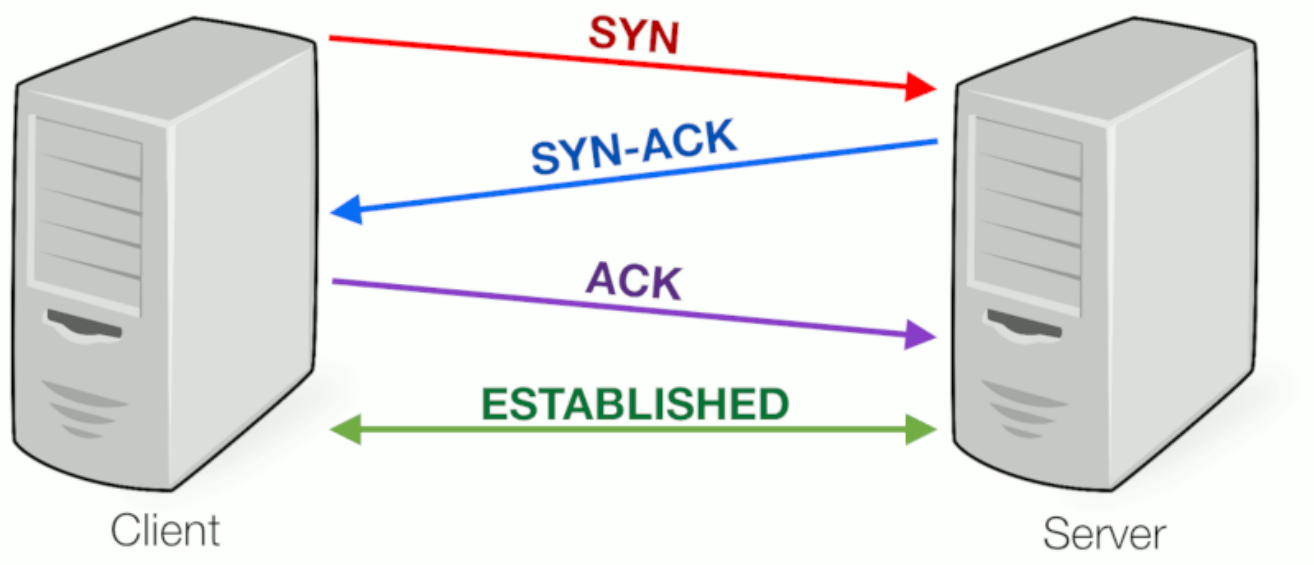
Изпращат се по сокета **len** байта от паметта, сочена от **buf**.

Четат се от сокета **len** байта и се записват в паметта, сочена от **buf**.

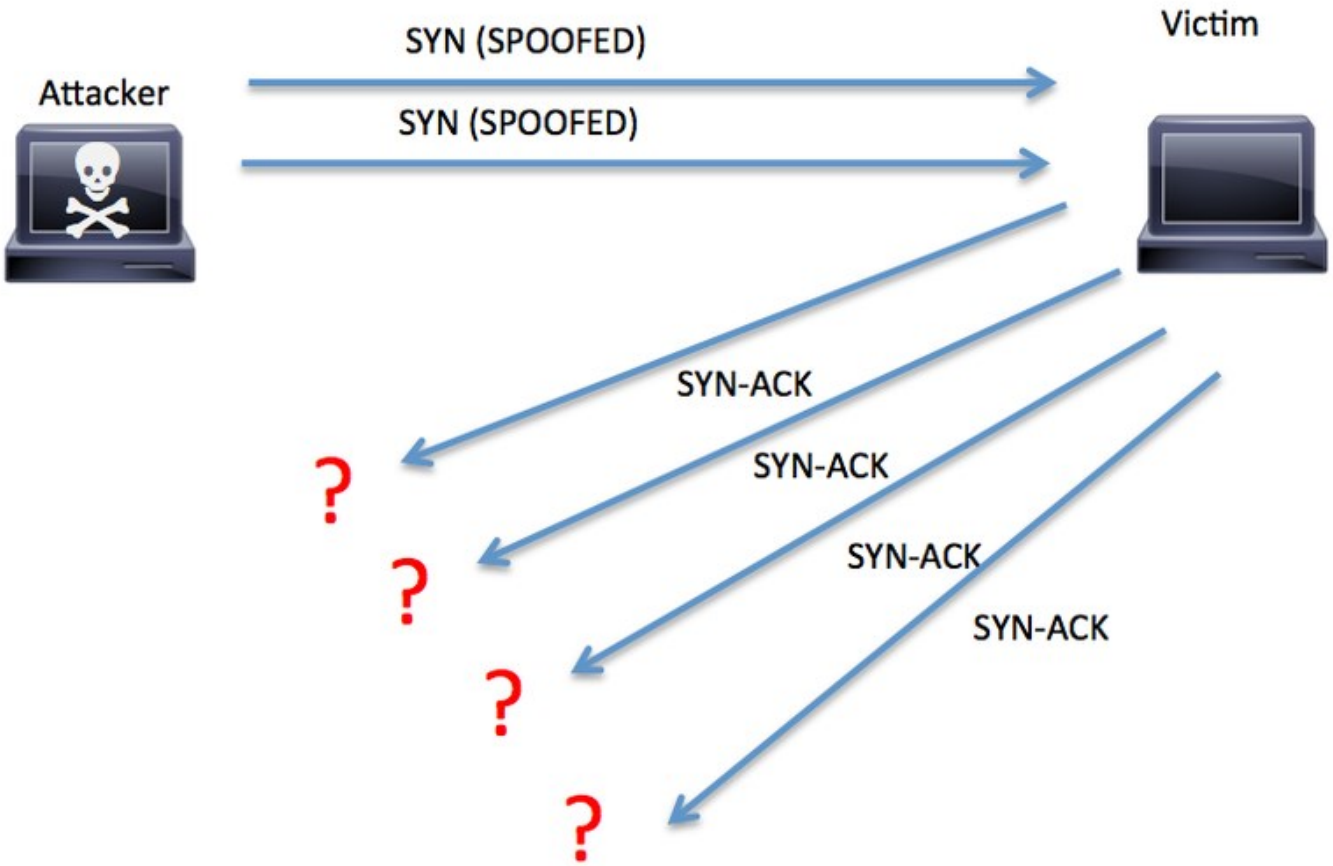
Изпращане на данни (клиент)

```
#define  SRVPORT   2345        // порт на сървъра
#define  SRVIP "200.100.50.1" // IP адрес на сървъра

struct   sockaddr_in  srvaddress;
char *msg="Hello server";
int len;

srvaddress.sin_family = AF_INET;      // Internet фамилия
srvaddress.sin_port = htons(SRVPORT);
srvaddress.sin_addr.s_addr = inet_addr(SRVIP);

connect(sockfd, (struct sockaddr *)&srvaddress,sizeof(srvaddress));

len = strlen(msg);

send(sockfd, msg, len, 0);

...
```

Сървър – поддръжка на конекциите



Сървър – поддръжка на конекциите



Сървър – поддръжка на конекциите

```
int listen(int sockfd, int backlog);
```

sockfd – дескриптор на създадения сокет

backlog – брой на едновременно поддържани конекции

Указва максималният брой конекции, които могат да се обслужват едновременно. Всяка нова конекция след този брой ще бъде отхвърлена.

Сървър – поддръжка на конекциите

```
#include <sys/types.h>
#include <sys/socket.h>
#define  MYPORT   2345

struct sockaddr_in  myaddress;

sockfd = socket(AF_INET, SOCK_STREAM, 0);

myaddress.sin_family = AF_INET;
myaddress.sin_port = htons(MYPORT);
myaddress.sin_addr.s_addr = INADDR_ANY;

bind( sockfd, (struct sockaddr *)&myaddress,
                                    sizeof(myaddress) );

if ( listen(sockfd, 5) == -1 ) {
   perror("Listen:");
   exit(1);
}
```

Сървър – очакване на заявки

```
int accept(int sockfd, struct sockaddr *cl_addr,
                    int *cl_addrlen );
```

sockfd – дескриптор на създадения сокет

cl_addr – променлива, в която се получават данни за клиента

cl_addrlen – размер на получените данни

Блокира се сървърът до получаване на конекция от клиента. След осъществяване на връзка, функцията завършва и връща копие на сокета, **с което трябва да се работи нататък**. В **cl_addr** се връща информация за клиента, а в **cl_addrlen** се връща размера на тази информация.

Сървър – очакване на заявки

```
#include <sys/types.h>
#include <sys/socket.h>
#define  MYPORT   2345

struct sockaddr_in  myaddress, claddress;
int sock_copy, cl_addrlen;;

sockfd = socket(AF_INET, SOCK_STREAM, 0);

myaddress.sin_family = AF_INET;
myaddress.sin_port = htons(MYPORT);
myaddress.sin_addr.s_addr = INADDR_ANY;

bind( sockfd, (struct sockaddr *)&myaddress, sizeof(myaddress) );

listen(sockfd, 5);

if ( sock_copy=accept(sockfd, (struct sockaddr *)&claddress,
                                  &cl_addrlen) == -1 ) {
   perror("Accept:");
   exit(1);
}
```

Обмен при datagram сокети

```
int sendto ( int sockdesc, const void *msg, int msglength,
        int flags, const struct sockaddr *dest_address, int len );

int recvfrom ( int sockdesc, void *buf, int buflen, int flags,
       struct sockaddr *dest_address, int len );
```

Формиране на съобщенията – кодиране на данните

Клиент трябва да изпрати две цели числа. Как да се укажат в съобщението?

Решение 1: Кодиране на символите.

- Използване на ASCII кодове
- Същото представяне се използва и за извеждане на екрана
- Позволява предаване на големи числа

Формиране на съобщенията – кодиране на данните

x = 17 998 720, y = 47 034 615

49	55	57	57	56	55	50	48	32	52	55	48	51	52	54	49	53	32
1	7	9	9	8	7	2	0	_	4	7	0	3	4	6	1	5	_

```
sprintf(msgBuffer, "%d %d ", x, y);
send(clientSocket, strlen(msgBuffer), 0);
```

Формиране на съобщенията – кодиране на данните

Проблеми:

- Трябва допълнителен разделител
- *MsgBuffer* трябва да е достатъчно голям
- Неефикасно – за всяка цифра са достатъчни 4 бита, а се използва 1 байт
- Невъзможност за манипулиране с числата

Формиране на съобщенията – кодиране на данните

<u>Решение 2:</u> Изпращане на стойностите на **x** и **y** .

Проблем: Използва се естествен целочислен формат на данните

- Подреждане на байтовете

Формиране на съобщенията – кодиране на данните

Реализация 1:

```
typedef struct {
  int x;
  int y;
} msgstruct;

msgstruct.x = X;
msgstruct.y = Y;

send( sockfd, &msgstruct, sizeof(msgstruct), 0 );
```

Формиране на съобщенията – кодиране на данните

Реализация 2:

```
send( sockfd, &X, sizeof(X), 0 );
send( sockfd, &Y, sizeof(Y), 0 );
```

Опции на сокетите

```
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

sockfd – дескриптор на създадения сокет

level – нивото на протоколния стек

optname – конкретната опция

optval – връща се стойността на опцията

optlen – размер на стойността

Установява опцията указана от **optname** за протоколното ниво **level** на стойността, указана от **optval** за зададения сокет **sockfd**.

Повторно използване на сокета

Ако сървърът се стартира повторно след като е изпълнил *bind()*, и не е изтекъл тайм-аута на сокета, ще настъпи грешка по време на изпълнение.

За избягване на това, сокетът се конфигурира с опция да може да се използва повторно. Конфигурирането трябва да се извърши преди *bind().*

```
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
                    &tmp, sizeof(int)) == -1 ) {
      perror(„Setsckopt");
      exit(1);
}
```

Пример – Echo сървър

Сървър, получаващ символ от клиент. Връща обратно символа.

```
// Echo server

#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define PORT     2346

int main (void){
  int sockfd, client, claddrlen, len, tmp;
  struct sockaddr addr;
  char ch;
  struct sockaddr_in   srvaddr;    // server address

          // create a socket
          sockfd=socket(AF_INET,SOCK_STREAM,0);
          if (sockfd == -1) {
            perror ("socket");
            exit (1);
          }
```

Пример – Echo сървър

```
/* set socket to be with REUSABLE port */
if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &tmp,
                          sizeof(int)) == -1 ) {
  perror("setsckopt");
  exit(1);
}

/* filling the server address */
srvaddr.sin_family = AF_INET;     // Internet domain
srvaddr.sin_port = htons (PORT);  // in network byte order
srvaddr.sin_addr.s_addr = INADDR_ANY; //auto-fill with local IP

/* attach address to the socket */
if (bind(sockfd, (struct sockaddr*)&srvaddr, sizeof(srvaddr)) == -1) {
  perror ("bind");
  exit(1);
}

/* create a pending queue */
if (listen(sockfd, 5) == -1) {
  perror ("listen");
  exit(1);
}
```

Пример – Echo сървър

```
/* main body of the server */

   claddrlen = sizeof(addr);
   client = accept(sockfd, &addr, &claddrlen); // wait

   recv (client, &ch, sizeof(char), 0);  // get a char

   send (client, &ch, sizeof(char), 0);  // return the answer

   close(client);

   return (0);

}
```

Пример – Echo клиент

Клиент, изпращащ символ.

```
/* Echo client */

#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>

#define       SVPORT 2346          // server port
#define       SVIP   "127.0.0.1"   // server IP address

   int main (void) {

     int sockfd, len;
     char ch;                       // input buffer
     struct sockaddr_in srvaddr;    // server address
     struct sockaddr_in claddr;     // client address

     sockfd=socket(AF_INET,SOCK_STREAM,0); // create a socket
     if (sockfd == -1) {
       perror ("socket");
       exit (1);
     }
```

Пример – Echo клиент

```
 /* filling the client address */
 claddr.sin_family = AF_INET;  // Internet domain
 claddr.sin_port = htons (0);  // port number in network byte order
 claddr.sin_addr.s_addr = INADDR_ANY;    // auto-fill with local IP

/* attach address to the socket */
if (bind(sockfd, (struct sockaddr*)&claddr,sizeof(claddr)) == -1) {
  perror ("bind");
  exit(1);
}

/* filling the server address */
srvaddr.sin_family = AF_INET;              // Internet domain
srvaddr.sin_port = htons (SVPORT);         // server port number
srvaddr.sin_addr.s_addr = inet_addr(SVIP); // server IP address

/* connect to the server */
if (connect(sockfd,(struct sockaddr*)&srvaddr, sizeof(srvaddr)) == -1) {
  printf("ERROR connection\n");
  exit(1);
}
```

Пример – Echo клиент

```
/* main body of the client */

    ch = getchar();
    send(sockfd, &ch, sizeof(char), 0);

    recv(sockfd, &ch, sizeof(char), 0);
    printf("%c\n", ch);

   return 0;
 }
```

Въпроси?