Лекция 4. Сетевое программирование.

- Сокеты
- Системные вызовы для работы с сокетами
- Настройки сокетов
- Порядок байт
- Сырые сокеты
- Работы с TCP/UDP
- Асинхронная работа

socket

```
NAME
     socket -- create an endpoint for communication

SYNOPSIS
     #include <sys/socket.h>

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

Типы доменов

```
PF_LOCAL        Host-internal protocols, formerly called PF_UNIX,
PF_UNIX         Host-internal protocols, deprecated, use PF_LOCAL,
PF_INET         Internet version 4 protocols,
PF_ROUTE        Internal Routing protocol,
PF_KEY          Internal key-management function,
PF_INET6        Internet version 6 protocols,
PF_SYSTEM       System domain,
PF_NDRV         Raw access to network device
```

Виды сокетов

SOCK_STREAM - потоки байт (TCP)
SOCK_DGRAM - датаграммы (UDP, Unix socket)
SOCK_RAW - “сырые” сокеты

bind()

```
NAME
     bind -- bind a name to a socket

SYNOPSIS
     #include <sys/socket.h>

     int
     bind(int socket, const struct sockaddr *address, socklen_t address_len);
```

Смена порядка байт

```
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

sockaddr

```
struct sockaddr {
    unsigned short    sa_family;    // address family,
AF_xxx
    char              sa_data[14];  // 14 bytes of protocol
address
};
```

sockaddr_in

```
// (IPv4 only--see struct sockaddr_in6 for IPv6)

struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

sockaddr_in6

```
struct sockaddr_in6 {
        sa_family_t     sin6_family;   /* AF_INET6 */
        in_port_t       sin6_port;     /* port number */
        uint32_t        sin6_flowinfo; /* IPv6 flow information */
        struct in6_addr sin6_addr;     /* IPv6 address */
        uint32_t        sin6_scope_id; /* Scope ID (new in 2.4) */
    };

    struct in6_addr {
        unsigned char   s6_addr[16];   /* IPv6 address */
    };
```

listen()

```
NAME
     listen -- listen for connections on a socket

SYNOPSIS
     #include <sys/socket.h>

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

accept()

```
NAME
     accept -- accept a connection on a socket

SYNOPSIS
     #include <sys/socket.h>

     int
     accept(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);
```

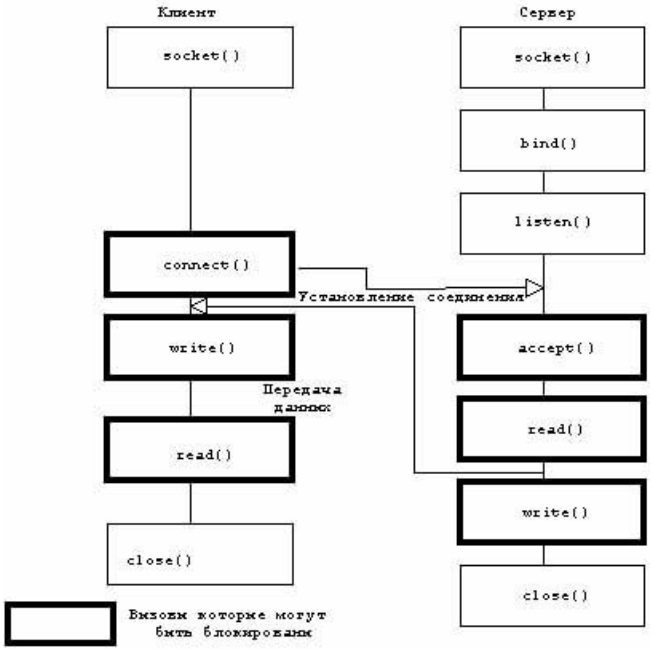
connect()

```
NAME
     connect -- initiate a connection on a socket

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

     int
     connect(int socket, const struct sockaddr *address, socklen_t address_len);
```

Полный процесс работы



getsockname()

```
NAME
     getsockname -- get socket name

SYNOPSIS
     #include <sys/socket.h>

     int
     getsockname(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);
```

Запись в сокет

```
NAME
     send, sendmsg, sendto -- send a message from a socket

SYNOPSIS
     #include <sys/socket.h>

     ssize_t
     send(int socket, const void *buffer, size_t length, int flags);

     ssize_t
     sendmsg(int socket, const struct msghdr *message, int flags);

     ssize_t
     sendto(int socket, const void *buffer, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len);
```

Запись в сокет: флаги

```
#define MSG_OOB         0x1  /* process out-of-band
data */
    #define MSG_DONTROUTE  0x4  /* bypass
routing, use direct interface */
```

sendmsg пример

```
struct iovec iov[1];
iov[0].iov_base=content;
iov[0].iov_len=sizeof(content);

struct msghdr message;
message.msg_name=res->ai_addr;
message.msg_namelen=res->ai_addrlen;
message.msg_iov=iov;
message.msg_iovlen=1;
message.msg_control=0;
message.msg_controllen=0;

if (sendmsg(fd,&message,0)==-1) {
    die("%s",strerror(errno));
}
```

Вычитывание из сокета

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

     ssize_t
     recv(int socket, void *buffer, size_t length, int flags);

     ssize_t
     recvfrom(int socket, void *restrict buffer, size_t length, int flags, struct sockaddr *restrict address, socklen_t *restrict address_len);

     ssize_t
     recvmsg(int socket, struct msghdr *message, int flags);
```

Флаги вычитывания

MSG_OOB	Обработка внеочередных данных
MSG_PEEK	Не вычитывать из сокета
MSG_WAITALL	Вычитать все данные или ошибка

msghdr

```
struct msghdr {
        void        *msg_name;      /* optional address */
        socklen_t       msg_namelen;    /* size of address */
        struct       iovec *msg_iov; /* scatter/gather array */
        int          msg_iovlen;     /* # elements in msg_iov */
        void         *msg_control;   /* ancillary data, see below */
        socklen_t       msg_controllen; /* ancillary data buffer len */
        int          msg_flags;      /* flags on received message */
};
```

recvmsg: пример

```
char buffer[548];
struct sockaddr_storage src_addr;

struct iovec iov[1];
iov[0].iov_base=buffer;
iov[0].iov_len=sizeof(buffer);

struct msghdr message;
message.msg_name=&src_addr;
message.msg_namelen=sizeof(src_addr);
message.msg_iov=iov;
message.msg_iovlen=1;
message.msg_control=0;
message.msg_controllen=0;

ssize_t count=recvmsg(fd,&message,0);
if (count==-1) {
    die("%s",strerror(errno));
} else if (message.msg_flags&MSG_TRUNC) {
    warn("datagram too large for buffer: truncated");
} else {
    handle_datagram(buffer,count);
}
```

Внеполосные данные

- В poll через флаг POLLPRI
- В epoll через флаг EPOLLPRI
- Через сигнал SIGURG, если установлен флаг F_SETOWN через fcntl

Сырые сокеты

- Поддерживает те же опции, что и любые датаграммные сокеты.
- Доступно только для программ, запущенных от рута или с флагом CAP_NET_RAW

Интерфейс к DNS

```
SYNOPSIS
     #include <netdb.h>

     int h_errno;

     struct hostent *
     gethostbyname(const char *name);

     struct hostent *
     gethostbyname2(const char *name, int af);

     struct hostent *
     gethostbyaddr(const void *addr, socklen_t len, int type);

     struct hostent *
     gethostent(void);

     void
     sethostent(int stayopen);

     void
     endhostent(void);

     void
     herror(const char *string);

     const char *
     hstrerror(int err);
```

Интерфейс к DNS

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* host address type */
        int     h_length;       /* length of address */
        char    **h_addr_list;  /* list of addresses from name server */
};
#define h_addr  h_addr_list[0]  /* address, for backward compatibility */
```

Пример

```
const char *ipstr = "127.0.0.1";
struct in_addr ip;
struct hostent *hp;

if (!inet_aton(ipstr, &ip))
        errx(1, "can't parse IP address %s", ipstr);

if ((hp = gethostbyaddr((const void *)&ip,
    sizeof ip, AF_INET)) == NULL)
        errx(1, "no name associated with %s", ipstr);

printf("name associated with %s is %s\n", ipstr, hp->h_name);
```

getaddrinfo

```
NAME
     getaddrinfo, freeaddrinfo -- socket address structure to host and service name

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

     int
     getaddrinfo(const char *hostname, const char *servname, const struct addrinfo *hints, struct addrinfo **res);

     void
     freeaddrinfo(struct addrinfo *ai);
```

getaddrinfo

```
x =  getaddrinfo(hostname, port, 0, &addr);
fd = socket(addr->ai_family, SOCK_STREAM, 0);
x = connect(fd, addr->ai_addr,
(int)addr->ai_addrlen);
```

shutdown

```
NAME
     shutdown -- shut down part of a full-duplex connection

SYNOPSIS
     #include <sys/socket.h>

     int
     shutdown(int socket, int how);
```

Опции сокетов

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

     int
     getsockopt(int socket, int level, int option_name, void *restrict option_value, socklen_t *restrict option_len);

     int
     setsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);
```

Базовые опции уровня SOL_SOCKET

SO_ERROR
SO_DONTROUTE
SO_KEEPALIVE
SO_REUSEADDR
SO_REUSEPORT
SO_BROADCAST
SO_OOBINLINE
SO_SNDTIMEO
SO_RCVTIMEO
SO_LINGER

Опции TCP

TCP_CORK
TCP_NODELAY
TCP_QUICKACK
TCP_SYNCNT
TCP_MAXSEG

Отключение алгоритма Нейгла

```
flag = 1;
ret = setsockopt( sock, IPPROTO_TCP, TCP_NODELAY, (char
*)&flag, sizeof(flag) );
```

sendfile

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

ssize_t sendfile(int out_fd, int in_fd, off_t *offset, size_t count);
```

Правильные таймауты

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

int timerfd_create(int clockid, int flags);

int timerfd_settime(int fd, int flags,
                    const struct itimerspec *new_value,
                    struct itimerspec *old_value);

int timerfd_gettime(int fd, struct itimerspec *curr_value);
```

select

```
SYNOPSIS
       /* According to POSIX.1-2001 */
       #include <sys/select.h>

       /* According to earlier standards */
       #include <sys/time.h>
       #include <sys/types.h>
       #include <unistd.h>

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

       void FD_CLR(int fd, fd_set *set);
       int  FD_ISSET(int fd, fd_set *set);
       void FD_SET(int fd, fd_set *set);
       void FD_ZERO(fd_set *set);
```

select: пример

DEMO: select.c

select: факты

Работает с неблокирующими сокетами
При fd > 1024 бьет стек.

poll

```
NAME
       poll, ppoll - wait for some event on a file descriptor

SYNOPSIS
       #include <poll.h>

       int poll(struct pollfd *fds, nfds_t nfds, int timeout);

       #define _GNU_SOURCE         /* See feature_test_macros(7) */
       #include <poll.h>

       int ppoll(struct pollfd *fds, nfds_t nfds,
               const struct timespec *timeout_ts, const sigset_t *sigmask);

DESCRIPTION
       poll() performs a similar task to select(2): it waits for one of a set of file descriptors to become ready to perform I/O.

       The set of file descriptors to be monitored is specified in the fds argument, which is an array of structures of the following form:

           struct pollfd {
               int   fd;         /* file descriptor */
               short events;     /* requested events */
               short revents;    /* returned events */
           };
```

poll: события

POLLIN - есть данные на чтение/входящее соединение
POLLPRI - есть OOB данные
POLLOUT - запись не приведет к блокировке
POLLRDHUP - был вызван shutdown
POLLERR* - ошибка на сокете
POLLHUP* - разрыв связи
POLLNVAL* - сокет не открыт

* - output only

epoll

```
#include <sys/epoll.h>
int epoll_create(int size);
int epoll_ctl(int epfd, int op, int fd, struct epoll_event
*event);
int epoll_wait(int epfd, struct epoll_event *events,
                    int maxevents, int timeout);
```

epoll

```
typedef union epoll_data {
        void        *ptr;
        int          fd;
        uint32_t     u32;
        uint64_t     u64;
    } epoll_data_t;

    struct epoll_event {
        uint32_t     events;      /* Epoll events */
        epoll_data_t data;        /* User data variable */
    };
```

epoll_ctl: команды

EPOLL_CTL_ADD
EPOLL_CTL_DEL
EPOLL_CTL_MOD

epoll: события

EPOLLIN - готовы к read/accept
EPOLLOUT - готовы к write
EPOLLRDHUP - другая сторона закрыла соединение
EPOLLPRI - OOB данные
EPOLLERR* - ошибка на сокете
EPOLLHUP* - разрыв соединения
EPOLLET - включить Edge-triggered
EPOLLONESHOT - удалять сокет из наблюдаемых после события

Задания

1) Сниффером проверить работу TCP_NODELAY, TCP_CORK и обычный режим.
2) Сниффером проверить работу TCP_KEEPALIVE
3) Реализовать работу с OOB с использованием poll