Казахстанский филиал МГУ им. М.В. Ломоносова

Операционные системы
лекции 22, 23

Сетевое взаимодействие в UNIX

11.11.2010

Проблемы взаимодействия в рамках сети

1. У каждой машины своя ОС, своя нумерация процессов Взаимодействие с помощью сигналов - не подходит.

2. Средства IPC — не подходят (уникальность именования обеспечивается только для конкретной машины, реализация не позволяет их использовать для сетевого взаимодействия)

НУЖНО:

1. Общий подход к организации взаимодействия.
2. Его поддержка для некоторого языка программирования.

Сокеты

Сокеты - универсальный механизм межпроцессного взаимодействия

- **Используется для локального взаимодействия (аналогично именованным каналам)**

- **Используется для сетевого взаимодействия**

Общая схема работы с сокетами любого типа:

- **каждый из взаимодействующих процессов на своей стороне создает и настраивает сокет,**
- **затем процессы выполняют соединение, используя пару настроенных сокетов,**
- **по окончании взаимодействия сокеты уничтожаются.**

Типы сокетов: потоковые и дейтаграммные

Потоковые сокеты

Соединение с использованием виртуального канала
передается последовательный поток байтов.
– гарантируется надежная доставка
– сохраняется порядок следования сообщений

Данные начинают передаваться только после того, как виртуальный канал установлен, и канал не разрывается, пока все данные не будут переданы.

Примеры: механизм каналов в UNIX,
телефонный разговор.

Границы сообщений при таком виде соединений не сохраняются (принимающее приложение само определяет границы сообщения).

Дейтаграммные сокеты

Дейтаграммное соединение используется для передачи отдельных пакетов, содержащих порции данных – дейтаграмм.
– Не гарантируется доставка в том же порядке, в каком дейтаграммы были отправлены.
– Не гарантируется доставка дейтаграмм.

Надежность соединения ниже, чем при установлении виртуального канала, но дейтаграммные соединения, более быстрые.

Пример дейтаграммного соединения: почта (письма и посылки могут приходить адресату не в порядке их отправки, а некоторые из них могут и пропадать).

Создание сокета

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

- Возвращается файловый дескриптор сокета

- В зависимости от типа создаваемого сокета может потребоваться дополнительная настройка параметров

- Файловый дескриптор может копироваться с помощью dup, наследоваться через fork/exec

- В конце использования файловый дескриптор должен быть закрыт с помощью close

Параметры создания сокета

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

- domain — домен сокета
  - PF_UNIX, PF_LOCAL – локальный сокет
  - PF_INET – IPv4
  - PF_INET6 – IPv6
- type — тип соединения
  - SOCK_STREAM – потоковый сокет
  - SOCK_DGRAM – дейтаграммный сокет
- Параметр protocol уточняет используемый протокол для пары (domain,type).
  - 0 — выбрать протокол по умолчанию

Примеры создания сокетов

```
fd = socket(PF_LOCAL, SOCK_STREAM, 0);
```
Локальный потоковый сокет (аналог именованных каналов)

```
fd = socket(PF_INET, SOCK_STREAM, 0);
```
Сокет для работы по протоколу TCP

```
fd = socket(PF_INET, SOCK_DGRAM, 0);
```
Сокет для работы по протоколу UDP

Адрес соединения

- Используется структура адреса - своя для конкр. домена

```
#include <netinet/in.h>
struct sockaddr_in {
  sa_family_t sin_family; /* AF_INET */
  uint16_t sin_port;
  struct in_addr sin_addr; /* internet address */
};
struct in_addr {
  uint32_t s_addr;
};

#include <sys/un.h>
struct sockaddr_un {
   short sun_family; /* == AF_UNIX */
   char sun_PATH[108];
};
```

Адрес соединения

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(const char *cp,struct in_addr *inp);

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

Примеры заполнения структуры с адресом

```
ifconfig   (UNIX)
ipconfig   (Windows)

"127.0.0.1"
short port = 3000;
struct sockaddr_in S;

S.sin_family = AF_INET;
S.sin_port = htons(port);
if (! inet_aton("127.0.0.1", &S.sin_addr))/*error*/;
-----------------------------------------
S.sin_family = AF_INET;
S.sin_port = htons(port);
S.sin_addr = INADDR_ANY;
```

Связывание сокета с адресом

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
                     socklen_t addrlen);
```

- addr – структура, задающая параметры привязки
- addrlen – размер структуры адреса

Пример:

```
short port = 3000;
struct sockaddr_in S;
/* .... fd = socket ....*/
S.sin_family = AF_INET;
S.sin_port = htons(port);
if (! inet_aton("127.0.0.1",&S.sin_addr))/*error*/;
bind(fd,(struct sockaddr*) &s, sizeof(s));
```

Идентификация соединения для приема / передачи сообщений

- Для полной идентификации дейтаграммы или соединения необходим номер исходящего порта

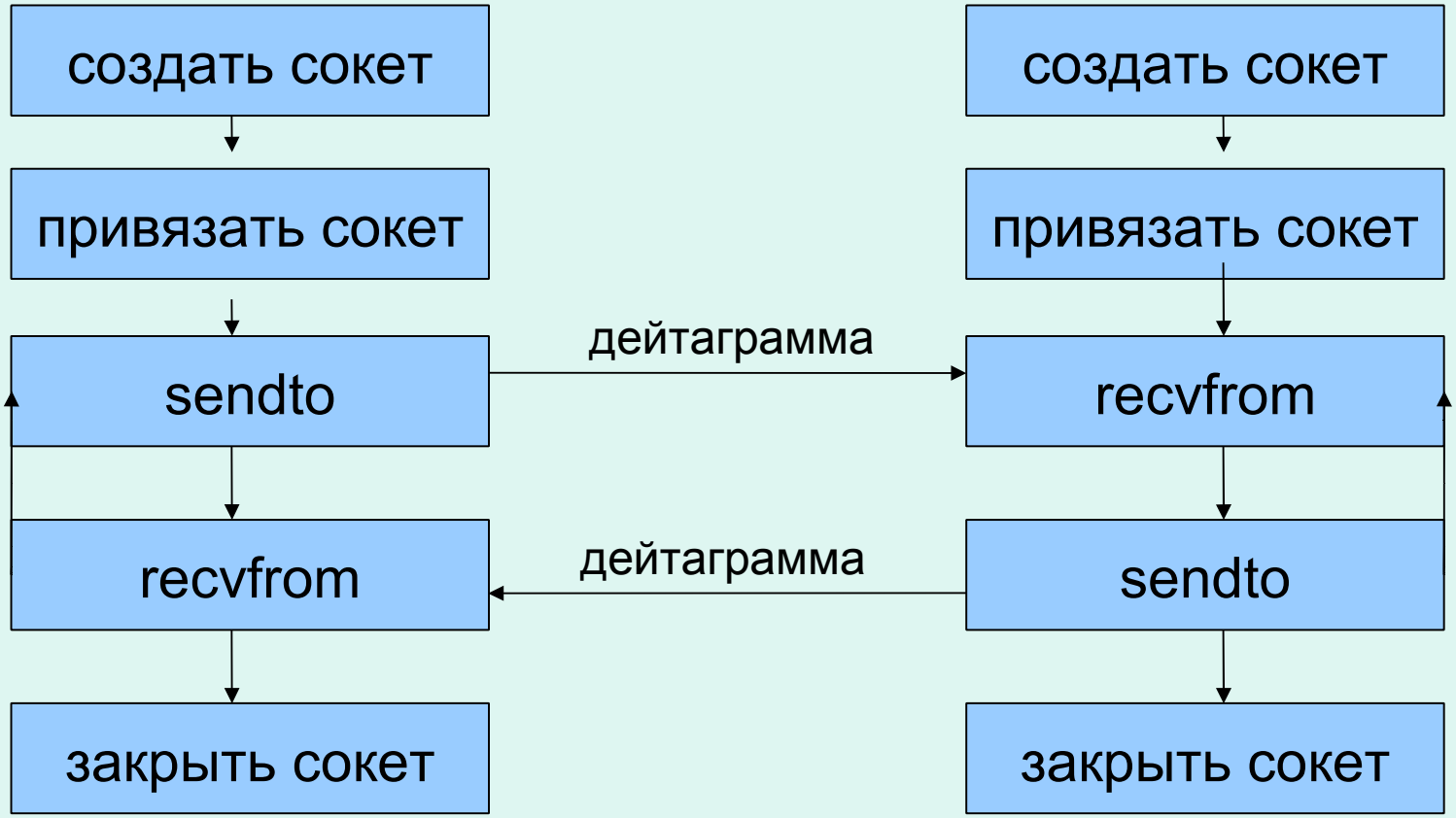
- Если номер исходящего порта не задан, он назначается автоматически

- Для взаимодействия по протоколу TCP исходящий порт, как правило, не требуется

- Для взаимодействия по протоколу UDP может потребоваться назначить исходящий порт

- Привязку необходимо выполнять, если планируется принимать сообщения

Схема взаимодействия без установления соединения



Прием и передача данных

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

ssize_t sendto(int s, const void *buf, size_t len,
    int flags, const struct sockaddr *to,
    socklen_t tolen);

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

flags позволяет задавать флаги посылки/приема:
MSG_DONTWAIT — неблокирующий прием
MSG_PEEK — не удалять данные из буфера приема
MSG_TRUNC — обрезать входной пакет

Прием и передача данных. Пример.

```
.....
char buf[16]="test";
int buf_len=strlen(buf);
n = sendto(fd, buf, buf_len, 0,
            (struct sockaddr *) S,sizeof(S));
if (n!=buf_len) /*ошибка errno==EMSGSIZE */
.....
---------------------------------
.....
struct sockaddr_in inaddr;
socklen_t inlen;
char buf[LEN];
int n;
n = recvfrom(fd, buf, LEN, MSG_TRUNC,
              (struct sockaddr*) &inaddr,&inlen);
printf("Message from %s, port %d\n",
inet_ntoa(inaddr.sin_addr),ntohs(inaddr.sin_port));
```

Архитектура клиент - сервер.

- **Техническое понимание:**

Клиент — сторона, которая инициирует соединение

Сервер — сторона, которая ожидает подключения

- **Логическое понимание**

Клиент — сторона, запрашивающая выполнение некоторого сервиса

Сервер — сторона, предоставляющая сервис

Схема взаимодействия с установлением соединения

Завершение соединения

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

int shutdown (int sockfd, int mode);
```

fd - дескриптор сокета

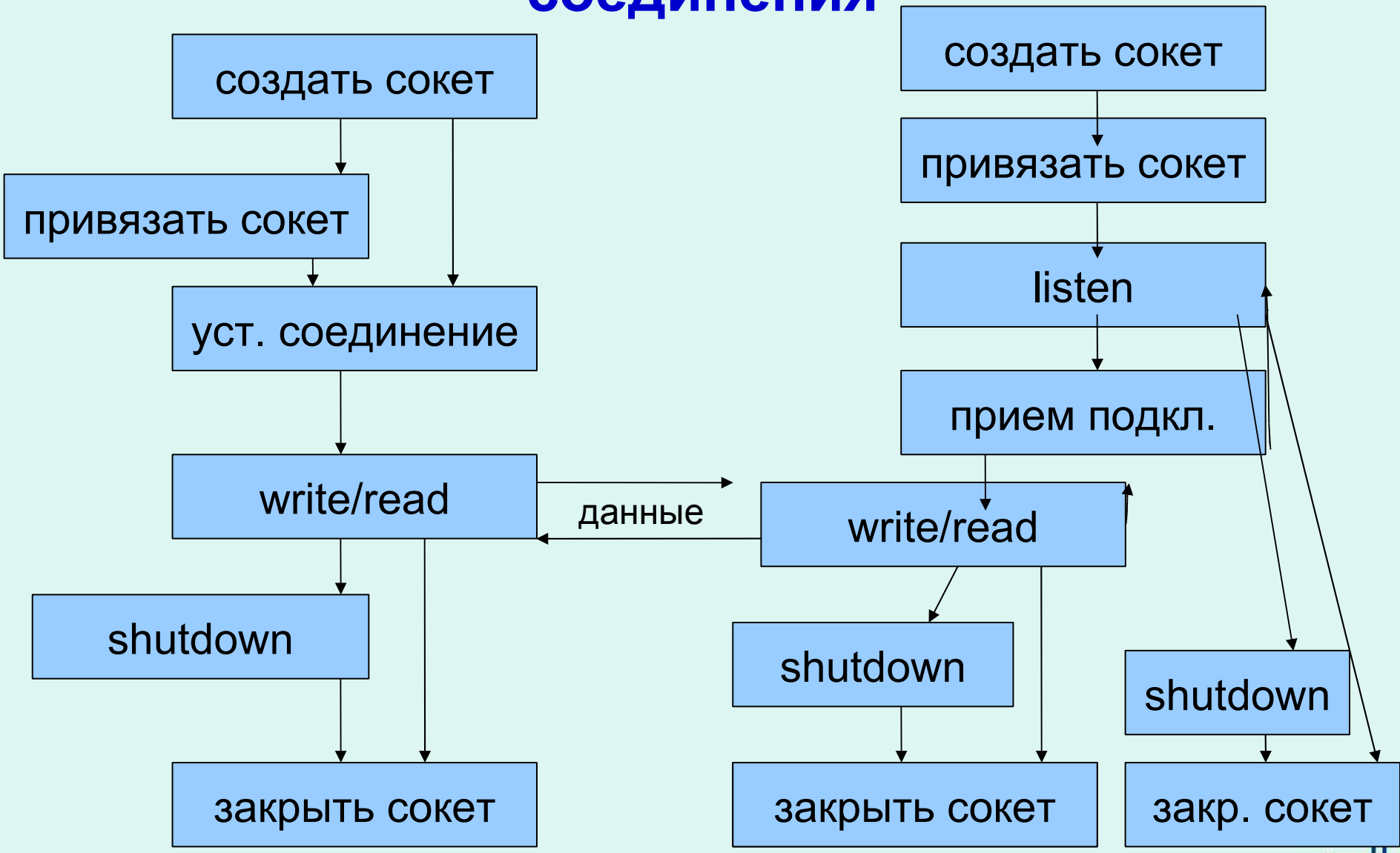
mode

Если mode=0 - сокет закрывается для чтения, (все дальнейшие попытки чтения вернут EOF)

Если mode=1, то сокет закрывается для записи (все дальнейшие попытки передать данные будут завершены неудачно)

Если mode=2, то сокет закрывается и для чтения, и для записи.

Схема взаимодействия с установлением соединения



Подключение к серверу

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

int connect(int sockfd,struct sockaddr * serv_addr,
                                int addrlen);
```

Ожидание и прием подключений на сервере

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

int listen (int sockfd, int backlog);

ECONNREFUSED
ETIMEDOUT
```

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

int accept (int sockfd, struct sockaddr *addr,
                                    int *addrlen);
```

Возвр. значение – новый дескриптор.

Обмен данными

read/write

sendto/recvfrom

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

int  send(int  sockfd,  const  void *msg, int len,
                                unsigned int flags);
int recv(int sockfd, void *buf, int len,
                                unsigned int flags);
```

Пример программы - клиента.

В командной строке получен IP-адрес сервера и порт. Нужно выполнить подключение, передать серверу целое число и получить в ответ тоже целое число (int).

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <time.h>
```

Пример программы - клиента.

```
int main(int argc, char const *argv[]){
int port, n, sfd; struct sockaddr_in sin;
if (argc != 3) {/*error*/}
if (! inet_aton (argv[1],&sin.sin_addr)){/*error*/}
if (sscanf(argv[2],"%d%n",&port,&n)!=1 || argv[2][n] ||
port <= 0 || port > 65535)  {/*error*/}
sin.sin_family = AF_INET;
sin.sin_port = htons(port);

if ((sfd = socket(PF_INET,SOCK_STREAM,0))<0){/*error*/}
if (connect(sfd, (struct sockaddr*) &sin,sizeof(sin)) < 0)
{/*error*/}
   {int t = 1;
    write(sfd, &t, sizeof(t));
    if (read(sfd, &t, sizeof(t)) != sizeof(t)) {/*error*/}
     else printf("Server number: %d", t);
    }
close(sfd);
return 0;
}
```

Еще один пример программы — клиента.

В командной строке получен IP-адрес сервера и порт.
Нужно выполнить подключение и
всю информацию от сервера выдавать в стандартный вывод,
а всю информацию со стандартного ввода отправлять на сервер.

Использование select

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int n, fd_set *rfds, fd_set *wfds,
           fd_set *efds, struct timeval *timeout);


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

```
{ fd_set fds;
/*sfd – дескриптор сокета  */
while(1){
 FD_ZERO(&fds); FD_SET(sfd, &fds); FD_SET(0, &fds);
 t=select(FD_SETSIZE, &fds, NULL,NULL,NULL);
 if (t<=0){/* ошибка */}

 if (FD_ISSET(sfd, &fds)) {
    sz = read(sfd, buf, sizeof(buf));
    if (!sz) break;
    write(1, buf, sz);
 }
 if (FD_ISSET(0, &fds)) {
   sz = read(0, buf, sizeof(buf));
   if (!sz) break;
   write(sfd, buf, sz);
  }
 }
}
```

Организация работы сервера.

- При каждом успешном выполнении accept создается новый файловый дескриптор для обмена данными с клиентами

- Сервер должен выполнять операции ввода/вывода с несколькими файловыми дескрипторами и обрабатывать новые подключения

- Каждая операция может заблокировать процесс на неопределенное время

Архитектура сервера

- Формировать по одному процессу для взаимодействия с каждым подключившимся клиентом.

- Решение на основе select

Схема реализации сервера

```
int main(int argc, char  *argv[]) {
 int lfd, n, port, alen;
 struct sockaddr_in b, a;
 if (argc != 2) {/* error */ }
 if (sscanf(argv[1],"%d%n",&port, &n)!=1 || argv[1][n] ||
        port <= 0 || port > 65535) {/*error*/}
 if ((lfd = socket(PF_INET, SOCK_STREAM, 0))<0){/*error*/}
  b.sin_family = AF_INET;
  b.sin_port = htons(port);
  b.sin_addr.s_addr = INADDR_ANY;
 if (bind(lfd,(struct sockaddr*)&b,sizeof(b))<0){/*err*/}
 if (listen(lfd, 5) < 0) {/*error*/}
 while (1) {
   int res; fd_set r;
   FD_ZERO(&r);FD_SET(lfd,&r);
/*добавить в r все дескр., работающие с подкл. клиентами*/
   res = select(FD_SETSIZE,&r,NULL,NULL,NULL);
   if (FD_ISSET(lfd,&r)){
     afd = accept(lfd, (struct sockaddr*)&a, &alen))
 .........
```

Установка режимов работы сокета

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

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

fd – дескриптор сокета
level - режимы какого уровня нужно изменить
  изменение работы сокета SOL_SOCKET
  изменение работы протокола IPPROTO_TCP
optname - имя изменяемого параметра

Устранение «залипания» порта после завершения сервера
(выполняется после создания сокета, до bind)

```
int opt = 1;
setsockopt(sfd,SOL_SOCKET, SO_REUSEADDR,&opt,sizeof(opt));
```

Событийно-ориентированное программирование

Событийно-ориентированные программы построены по принципу автоматов:

- **Выделяются состояния, в которых может находится автомат**
- **Выделяются все типы событий**
- **Описываются переходы между состояниями по приходу всех типов событий**
- **Требования: действия во время выполнения переходов между состояниями не должны блокировать процесс**