Сокеты

Наумов Д.А., доц. каф. КТ

Операционные системы и системное программное обеспечение, 2020

Содержание лекции

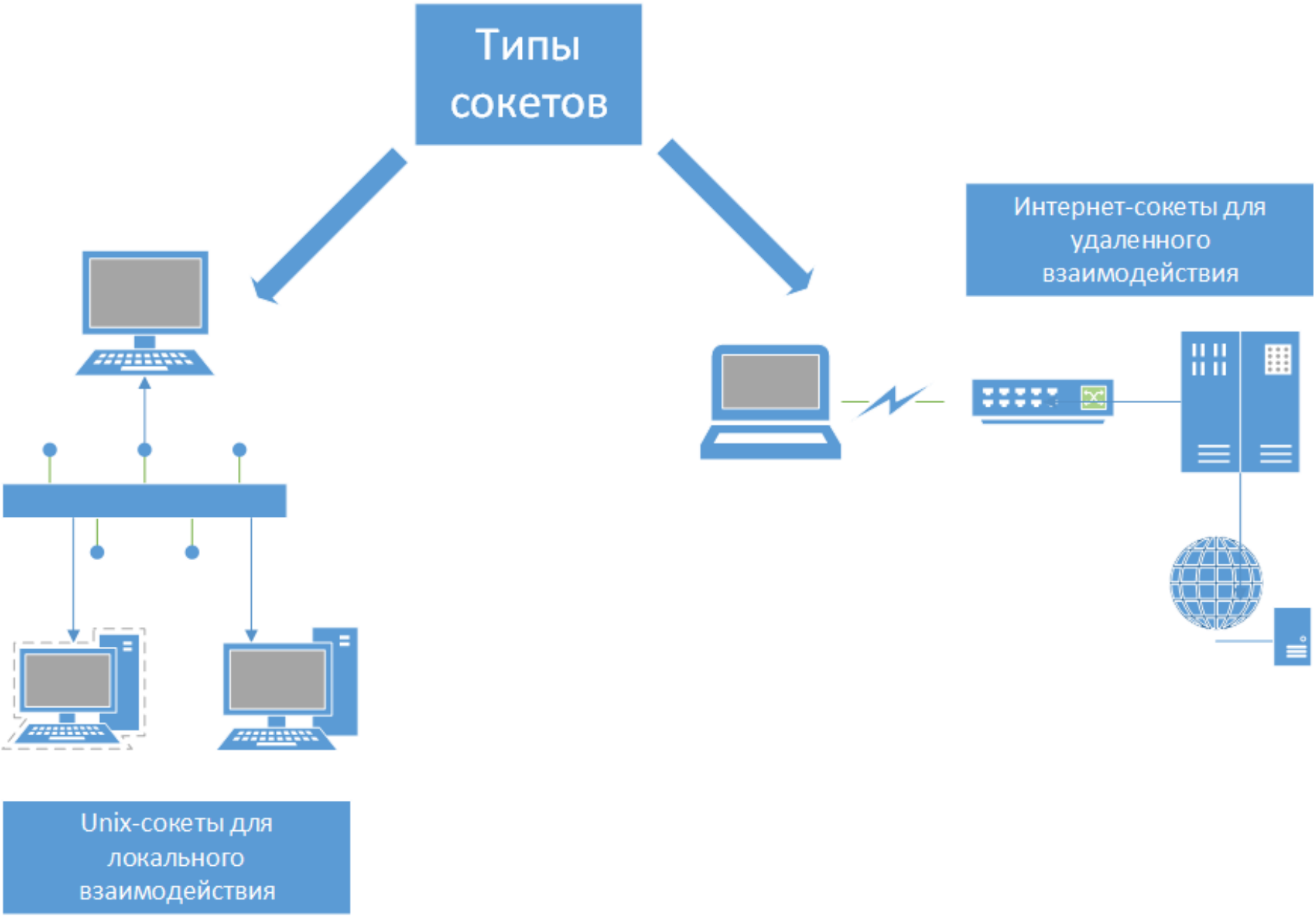
Сокеты

это механизм межпроцессного взаимодействия, который позволяет обмениваться данными между приложениями, выполняемыми как локально, так и на разных компьютерах, соединенных по сети.

- сокеты фигурируют в виде *файловых дескрипторов*, над которыми можно осуществлять обычные операции чтения-записи;
- набор действий, подготавливающих файловый дескриптор к использованию, несколько сложнее, чем в случае с обычными файлами;
- при взаимодействии посредством сокетов процессы рассматриваются по схеме *«клиент — сервер»*;
- процесс-сервер устанавливает «правила общения» и предлагает всем желающим межпроцессное взаимодействие.

Сокет создается с применением системного вызова `socket()`; вся дальнейшая работа с сокетом выполняется с помощью дескриптора, возвращенного этим вызовом:

```
fd = socket(domain, type, protocol);
```


Типы сокетов
Интернет-сокеты для удаленного взаимодействия
Unix-сокеты для локального взаимодействия

Домены сокетов

Сокеты существуют внутри домена взаимодействия, определяющего:

- способ идентификации сокета (то есть формат его «адреса»);
- диапазон взаимодействия (то есть находятся ли приложения в одной системе или на разных компьютерах, соединенных по сети).

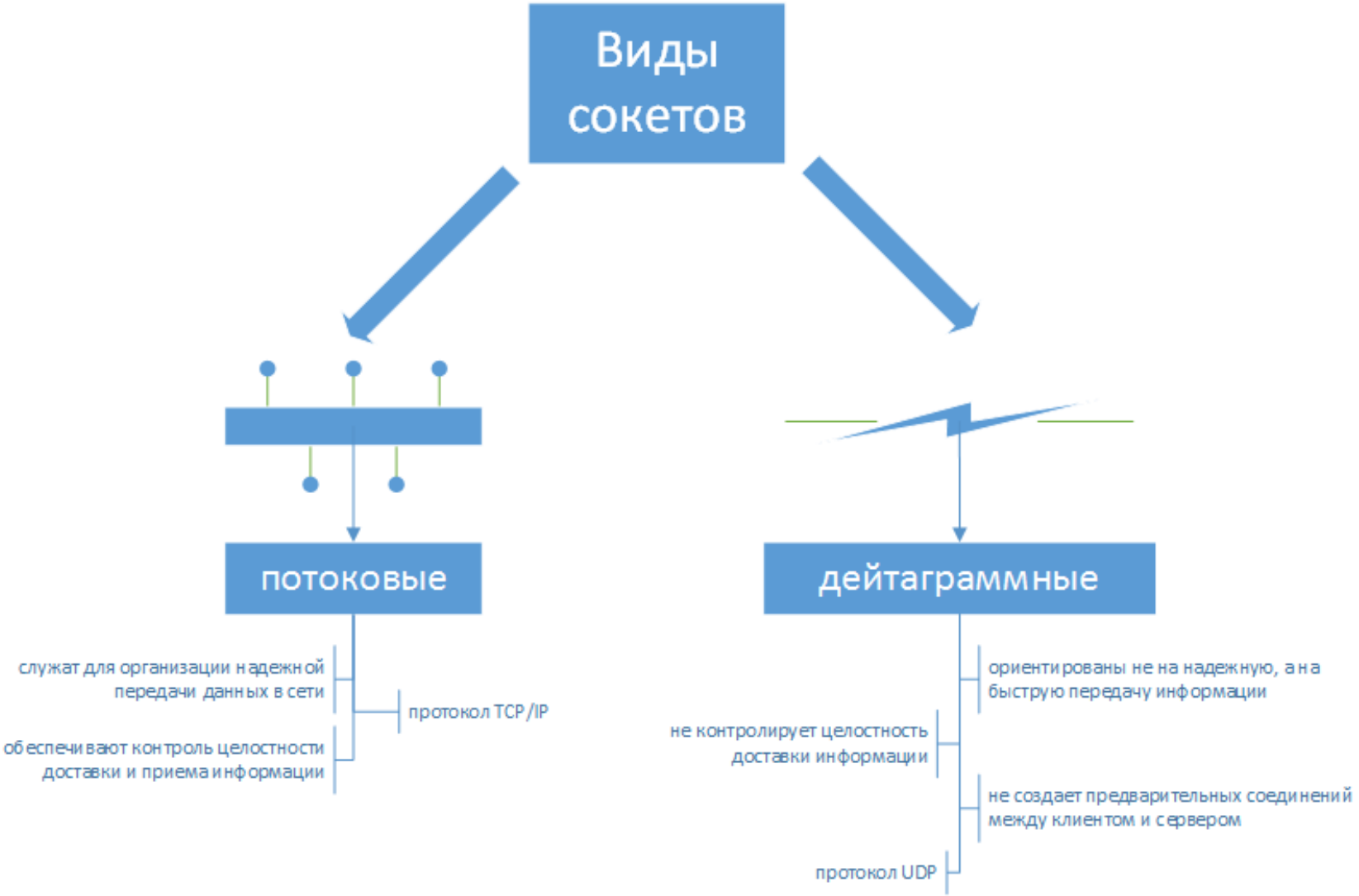
Современные операционные системы поддерживают как минимум домены следующих видов:

- UNIX-домен (AF_UNIX) позволяет взаимодействовать приложениям, находящимся на одном компьютере;
- IPv4-домен (AF_INET) позволяет взаимодействовать приложениям, которые выполняются на разных компьютерах, соединенных по сети на основе протокола IPv4;
- IPv6-домен (AF_INET6) позволяет взаимодействовать приложениям, выполняемым на разных компьютерах, соединенных по сети на основе протокола IPv6.

Домены сокетов

Домен	Взаимодействие выполняется	Взаимодействие между приложениями	Формат адреса	Структура адреса
AF_UNIX	Внутри ядра	На одном компьютере	Путь	sockaddr_un
AF_INET	Через IPv4	На компьютерах, соединенных по сети IPv4	32-разрядный адрес IPv4 + 16-разрядный номер порта	sockaddr_in
AF_INET6	Через IPv6	На компьютерах, соединенных по сети IPv6	128-разрядный адрес IPv6 + + 16-разрядный номер порта	sockaddr_in6

Любая реализация предоставляет как минимум два вида сокетов:



Эти две разновидности поддерживаются как в UNIX-, так и в интернет-доменах.

Потоковые сокеты

Потоковые сокеты (SOCK_STREAM) предоставляют надежный, двунаправленный канал взаимодействия на основе байтового потока.

- *надежный* – мы гарантируем одно из двух: либо передаваемые данные будут доставлены невредимыми приложению-адресату, либо мы получим уведомление о возможном сбое при передаче;
- *двунаправленный* -— данные могут передаваться между сокетами в любом направлении;
- *байтовый поток* — сообщения не имеют границ.

Потоковые сокеты работают парами, соединяясь друг с другом, поэтому их называют ориентированными на соединение.
В интернет-доменах потоковые сокеты задействуют протокол **TCP** (*Transmission Control Protocol* -— протокол управления передачей).

Датаграммные сокеты

Датаграммные сокеты (SOCK_DGRAM) позволяют обмениваться данными в виде сообщений, которые называются датаграммами.

- датаграммные сокеты сохраняют границы сообщений, но не обеспечивают надежную передачу данных.
- сообщения могут приходить не в том порядке, дублироваться или вовсе теряться.
- датаграммные сокеты являются общим случаем сетевого взаимодействия, не требующего соединения. В процессе работы им не нужно соединяться с другими сокетами (этим они отличаются от потоковых). Они могут быть соединены друг с другом, но по своей семантике данная процедура некоторым образом отличается от соединения потоковых сокетов.
- в интернет-доменах датаграммные сокеты задействуют протокол UDP (User Datagram Protocol — протокол пользовательских датаграмм).

TCP vs UDP

TCP

UDP

Содержание лекции

Адреса сокетов

Адрес сокета в домене UNIX представляет собой путь к файлу, а структура, предназначенная для его хранения, имеет следующий вид:

```
struct sockaddr_un {
    sa_family_t sun_family;  /* Всегда равно AF_UNIX */
    char sun_path[108];  /* Путь к сокету */
};
```

- Для привязки сокета UNIX-домена к адресу нужно инициализировать структуру `sockaddr_un`, передать указатель на нее аргументу `addr` вызова `bind()` и указать `addrlen` в качестве размера этой структуры.
- Привязывая сокет домена UNIX, вызов `bind()` создает запись в файловой системе. Сам файл помечается как сокет.

Привязка сокета домена UNIX

Необходимо отметить несколько моментов, касающихся привязки сокетов UNIX-домена.

- Сокет нельзя привязать к существующему пути.
- Обычно сокет привязывают к полному пути для фиксации его местоположения в файловой системе.
- Сокет можно привязать только к одному пути; и наоборот — путь может быть привязан только к одному сокету.
- Сокет нельзя открыть с помощью вызова `open()`.
- Когда сокет больше не нужен, его запись в файловой системе можно удалить, используя такие вызовы, как `unlink()` или `remove()`.

Привязка сокета домена UNIX

```
const char *SOCKNAME = "/tmp/mysock";
struct sockaddr_un addr;
int sfd = socket(AF_UNIX, SOCK_STREAM, 0); /* Создаем сокет */
if (sfd == -1)
    return -1; /* Ошибка создания сокета */

/* Очищаем структуру */
memset(&addr, 0, sizeof(struct sockaddr_un));
addr.sun_family = AF_UNIX; /* Адрес домена UNIX */
strncpy(addr.sun_path, SOCKNAME, sizeof(addr.sun_path) - 1);
int s = bind(sfd, (struct sockaddr *) &addr,
             sizeof(struct sockaddr_un));
if (s == -1)
    return -1; /* Ошибка привязки сокета */
```

Привязывание адреса: локальное взаимодействие

При локальном взаимодействии на основе Unix-сокетов в качестве второго аргумента bind() используется указатель на структуру sockaddr_un с полями:

- sun_family - семейство адресов (константа AF_LOCAL или AF_UNIX);
- sun_path - обычная строка, содержащая путь к файлу сокета.

Пример: socket1.c

Проверка:

```
$ gcc -o socket1 socket1.c
$ ./socket1 mysocket
Press <Enter> to continue...
```

Теперь откроем другое терминальное окно и посмотрим на наш сокет:

```
$ ls -l mysocket
srwxr-xr-x 1 nn nn 0 2019-12-11 10:18 mysocket
```

Назначение адресов: интернет-сокеты

При локальном взаимодействии на основе Unix-сокетов в качестве второго аргумента bind() используется указатель на структуру sockaddr_in с полями:

- sun_family - семейство адресов (константа AF_INET);
- sin_addr - адрес сокета;
- sin_port - порт сокета.

Поле sin_addr - это приведенный к целому числу IP-адрес серверного узла. Пример:

- IP-адрес: 192.168.0.1
- 192 = b11000000
- 168 = b10101000
- 0 = b00000000
- 1 = b00000001
- IP-адрес: b11000000101010000000000000000001 = 0xC0A80001

Преобразование имени в адрес

Система доменных имен (DNS, Domain Name System) сопоставляет IP-адресам удобочитаемые имена (домены). Для преобразования IP-адреса или доменного имени в числовой адрес используется функция gethostbyname():

```
struct hostent * gethostbyname (const char * NAME)
```

- адрес узла обычно находится в элементе host->h_addr_list[0] структуры hostent;
- аргумент NAME - это доменное имя или IP-адрес.

```
host = gethostbyname (argv[1]);
if (host == NULL) {
    fprintf (stderr, "Unknown server");
    return 1;
}
```

Назначение адресов: порты

Порт

число, идентифицирующее сеанс межпроцессного взаимодействия.

- в поле sin_port структуры sockaddr_in содержится 16-разрядный номер порта.
- порядок записи байтов и битов на конкретном компьютере может отличаться от той, что принята в сети общего пользования.
- для преобразования локального числа в 16-разрядный «сетевой» формат необходима функция htons().

Содержание лекции

Системные вызовы

Ключевые системные вызовы для работы с сокетами:

- Системный вызов `socket()` создает новый сокет.
- Системный вызов `bind()` привязывает сокет к адресу. Обычно он используется сервером для привязки сокета к общеизвестному адресу, чтобы клиенты могли его найти.
- Системный вызов `listen()` позволяет потоковому сокету принимать входящие соединения от других сокетов.
- Системный вызов `accept()` принимает соединение от удаленного приложения в «слушающий» потоковый сокет и опционально возвращает адрес удаленного сокета.
- Системный вызов `connect()` устанавливает соединение с другим сокетом.

Ввод/вывод через сокеты может быть выполнен с помощью традиционных операций `read()` и `write()` или же специальных системных вызовов таких как `recv()`, `recvfrom()`, `send()`, `sendto()`.

Создание сокета

Системный вызов `socket()` создает новый сокет.

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

int socket(int domain, int type int protocol);
// Возвращает файловый дескриптор или -1, если произошла ошибка
```

- Аргументы `domain` и `type` обозначают соответственно домен соединения сокета и его тип.
- Параметр `protocol` задает конкретный протокол, который работает с сокетом.
- Обычно существует только один протокол, задающий конкретный тип сокета в определенном семействе протоколов, в этом случае `protocol` может быть определено, как 0.

Привязывание к адресу

Системный вызов bind() привязывает сокет к заданному адресу.

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

int bind(int sockfd, const struct sockaddr *addr,
         socklen_t addrlen);
// Возвращает 0 при успешном завершении или -1 при ошибке
```

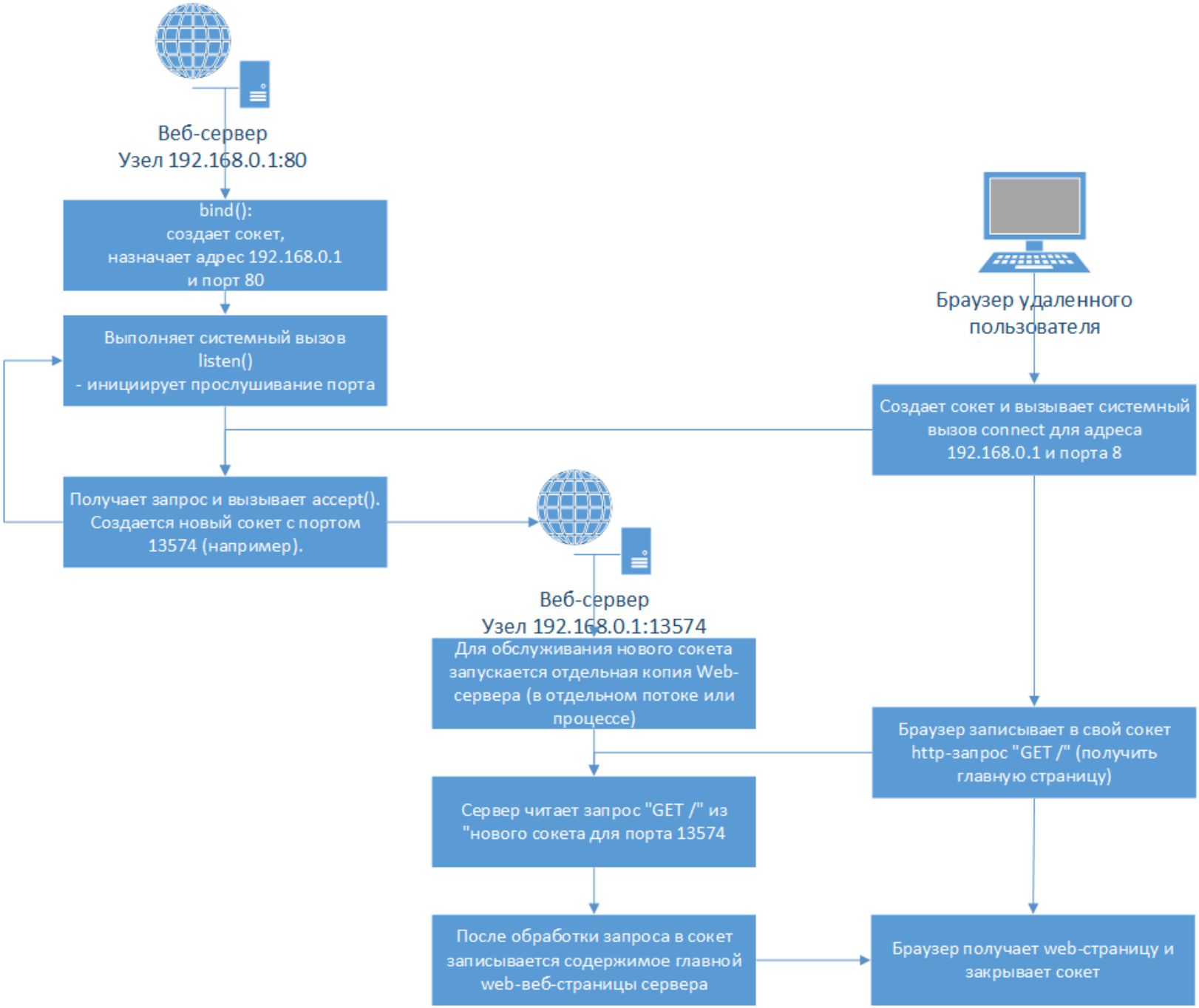
- Аргумент sockfd представляет собой файловый дескриптор, полученный из предыдущего вызова socket().
- Аргумент addr является указателем на структуру, описывающую адрес привязки сокета.
- Тип структуры, передаваемой в этом аргументе, зависит от домена сокета.
- Аргумент addrlen обозначает размер структуры с адресом; он имеет тип socklen_t, должен быть целым числом.

Содержание лекции

Принцип работы

Принцип работы потоковых сокетов:

1. Системным вызовом `socket()` создается сокет. Чтобы приложения могли взаимодействовать друг с другом, каждое из них должно создать свой сокет.
2. Прежде чем начать общение, приложения должны соединить свои сокеты. Это делается следующим образом.
   - Одно приложение делает вызов `bind()`, чтобы привязать свой сокет к общеизвестному адресу, и затем вызывает `listen()` для уведомления ядра о своей готовности принимать входящие соединения.
   - Другое приложение устанавливает соединение с помощью вызова `connect()`, указывая адрес сокета, к которому оно хочет подключиться.
   - Затем приложение, вызвавшее `listen()`, принимает соединение, используя вызов `accept()`. Вызов `accept()` блокируется, если сделать его до того, как другое приложение выполнит `connect()`.
3. Подключившись, можно передавать данные в обоих направлениях, пока одно из приложений не закроет соединение с помощью вызова `close()`.


Веб-сервер
Узел 192.168.0.1:80
bind():
создает сокет,
назначает адрес 192.168.0.1
и порт 80
Выполняет системный вызов
listen()
- инициирует прослушивание порта
Получает запрос и вызывает accept().
Создается новый сокет с портом
13574 (например).
Браузер удаленного
пользователя
Создает сокет и вызывает системный
вызов connect для адреса
192.168.0.1 и порта 8
Веб-сервер
Узел 192.168.0.1:13574
Для обслуживания нового сокета
запускается отдельная копия Web-
сервера (в отдельном потоке или
процессе)
Браузер записывает в свой сокет
http-запрос "GET /" (получить
главную страницу)
Сервер читает запрос "GET /" из
"нового сокета для порта 13574
После обработки запроса в сокет
записывается содержимое главной
web-веб-страницы сервера
Браузер получает web-страницу и
закрывает сокет

Соединение сокетов

При использовании потоковых сокетов между взаимодействующими процессами должно сначала установиться соединение. Для этого клиент вызывает системный вызов connect():

```
int connect (int sockfd, const struct sockaddr * addr,
             socklen_t addrlen);
```

- sockfd - это дескриптор сокета;
- addr - указатель на структуру, содержащую сведения об адресе сервера;
- addrlen - размер адресной структуры.

Пример getwwwpage.c

```
$ gcc -o getwwwpage getwwwpage.c
$ ./getwwwpage rsreu.ru > index.html
```

Ожидание входящих соединений

Системный вызов `listen()` делает потоковый сокет `sockfd` пассивным. Впоследствии этот сокет будет использоваться для приема соединений от других (активных) сокетов.

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
// Возвращает 0 при успешном завершении или -1 при ошибке
```

- Вызов `listen()` нельзя применять к подключенным сокетам, для которых уже была успешно выполнена операция connect(), или к возвращенным вызовами `accept()`.

Прослушивание сокета

Чтобы понять назначение аргумента backlog, следует отметить:

- Клиент может вызвать connect() до того, как сервер выполнит вызов accept().
- Данная ситуация приводит к возникновению отложенного соединения. Ядро должно записывать сведения о каждом отложенном запросе на подключение, чтобы впоследствии выполнить необходимые вызовы accept().
- Аргумент backlog позволяет ограничить количество таких отложенных соединений.

```
if (listen (sock, backlog) == -1) {
    fprintf (stderr, "listen() error");
    return 0;
}
```

Прием соединения

- Системный вызов listen() блокирует сервер до тех пор, пока какой-нибудь клиент не выдаст запрос на подключение.
- Как только запрос поступил, сервер «просыпается».
- Если есть возможность обслужить запрос, то сервер вызывает системный вызов accept():

Системный вызов `accept()` принимает входящее соединение на слушающем потоковом сокете `sockfd`.

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr,
           socklen_t *addrlen);
// Возвращает файловый дескриптор или -1, если произошла ошибка
```

- sockfd - это дескриптор сокета;
- По адресу addr расположена структура, в которую помещаются адресные данные созданного соединения;
- addrlen — адрес переменной, в которую помещается размер адресной структуры.

Прием соединения

- Вызов `accept()` создает новый сокет, который затем подключается к удаленному сокету, выполнившему вызов `connect()`. Слушающий сокет остается открытым и может использоваться для приема последующих соединений.
- Аргумент `addr` указывает на структуру, применяемую для возвращения адреса сокета.
- Аргумент `addrlen` указывает на целое число. Перед выполнением вызова оно должно быть инициализировано с помощью размера буфера, на который указывает `addr`.

Пример: socket2-server.c, socket2-client.c

Вскоре после запуска сервер переходит в режим прослушивания сокета:

```
$ gcc -o socket2-server socket2-server.c
$ ./socket2-server
```

Откроем теперь другое терминальное окно и начнем передавать серверу запросы:

```
$ gcc -o socket2-client socket2-client.c
$ ./socket2-client Hello
$ ./socket2-client World
$ ./socket2-client Linux
```

В исходном терминальном окне сервера будут выводиться соответствующие сообщения:

```
>> Hello
>> World
>> Linux
```

Сервер будет выводить сообщения, пока клиент не пошлет «exit».

Соединение с удаленным сокетом

Системный вызов connect() соединяет активный сокет sockfd, со слушающим сокетом, чей адрес задан в виде аргументов addr и addrlen.

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
// Возвращает 0 при успешном завершении или -1 при ошибке
```

Если вызов connect() завершается неудачей и мы хотим повторить попытку соединения, то:

- нужно закрыть имеющийся сокет,
- создать вместо него новый сокет
- с его помощью попытаться соединиться еще раз.

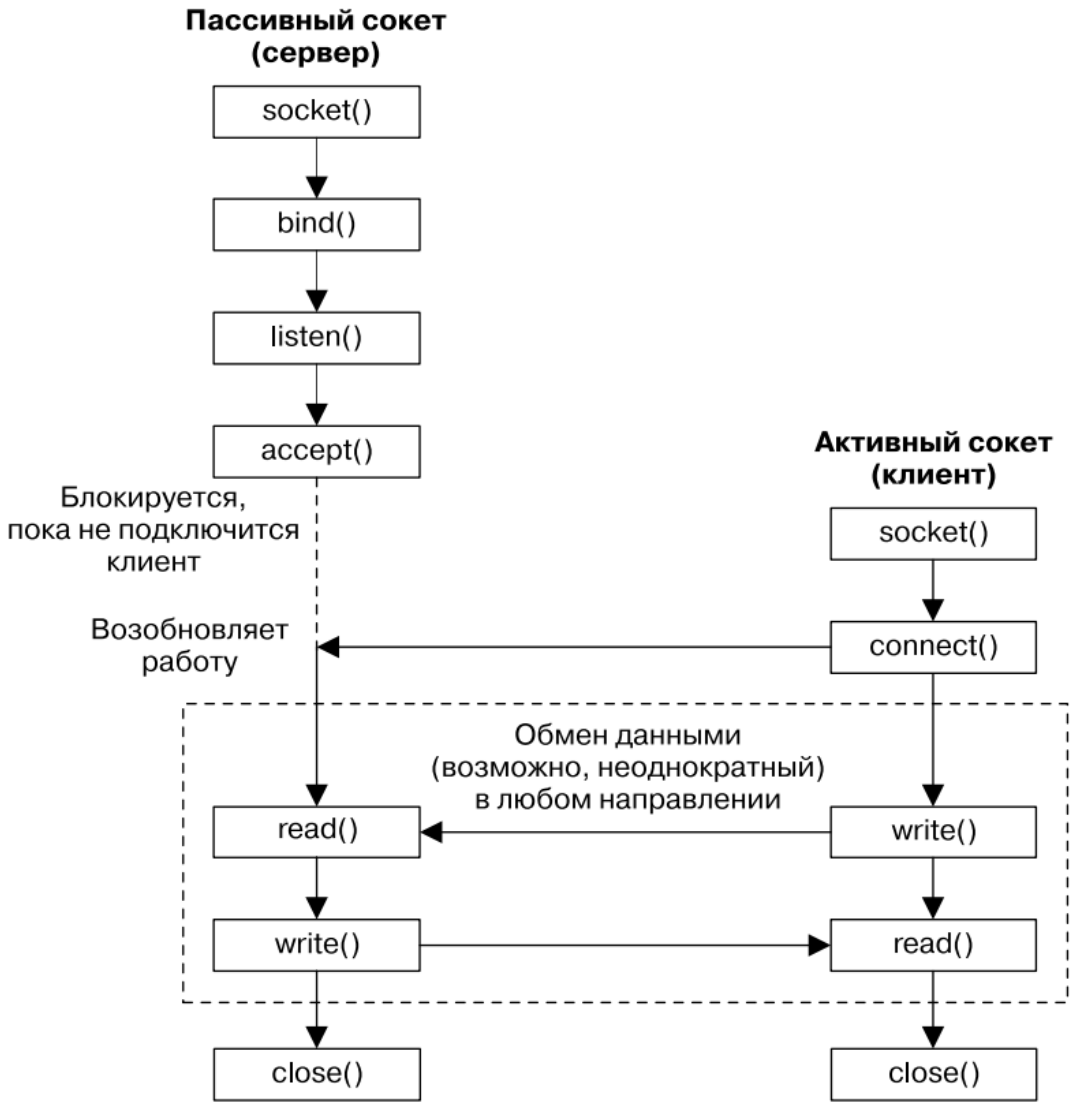
Закрытие соединения

Обычно для закрытия соединения на основе потоковых сокетов используется вызов close().

```
#include <unistd.h>

int close(int fd);
// Возвращает 0 при успешном завершении или -1 при ошибке
```

Взаимодействие TCP-сокетов



Содержание лекции

Принцип работы

Принцип работы датаграммных сокетов:

1. Любое приложение, которое хочет отправлять или получать датаграммы, должно создать датаграммный сокет с помощью вызова `socket()`.
2. Чтобы иметь возможность принимать датаграммы от других приложений, нужно привязать свой сокет к общеизвестному адресу, воспользовавшись вызовом `bind()`. Обычно это делает сервер, а клиент инициирует взаимодействие, отправляя по данному адресу свою датаграмму.
3. Для отправки датаграммы приложение делает вызов `sendto()`. Он в качестве одного из своих аргументов принимает адрес удаленного сокета, которому предназначено сообщение.
4. Чтобы получить датаграмму, приложение делает вызов `recvfrom()`, который может заблокироваться, если она еще не пришла. Данный вызов также позволяет определить адрес отправителя.
5. Когда сокет больше не нужен, приложение закрывает его с помощью вызова `close()`.

Прием датаграмм

Системный вызов `recvfrom()` принимает датаграммы на датаграммный сокет.

```
#include <sys/socket.h>
ssize_t recvfrom(int sockfd, void *buffer, size_t length,
                 int flags, struct sockaddr *src_addr,
                 socklen_t *addrlen);
// Возвращает количество полученных байтов,
// 0, если обнаружен конец файла, или -1 при ошибке
```

- Возвращаемое значение и первые три аргумента такие же, как у операции `read()`.
- Четвертый аргумент, `flags`, представляет собой битовую маску, которая определяет специальные возможности ввода/вывода, связанные с сокетами.
- Аргументы `src_addr` и `addrlen` возвращают адрес удаленного сокета, с помощью которого была отправлена датаграмма.

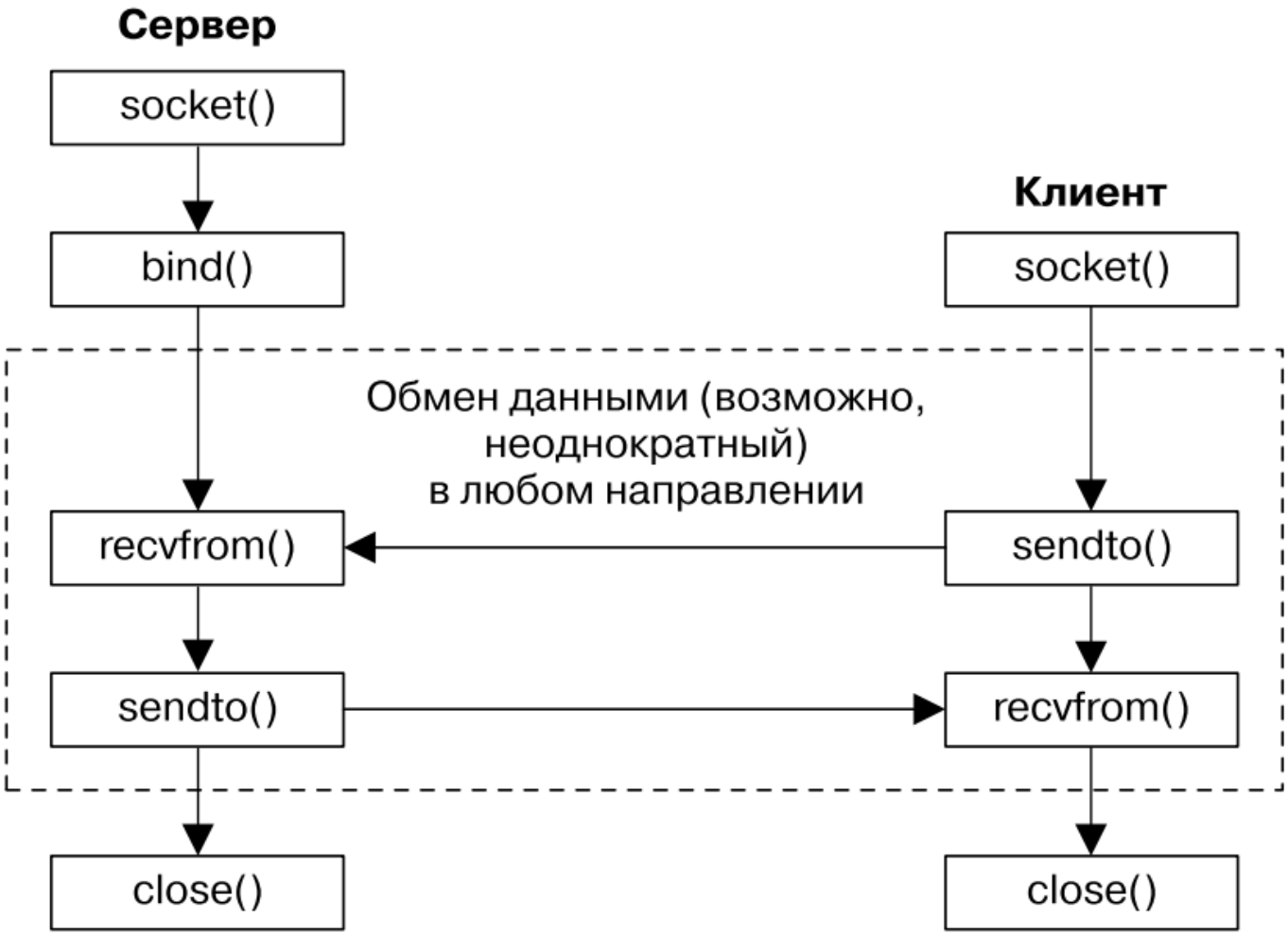
Отправка датаграмм

Системный вызов sendto() отправляет датаграммы на датаграммный сокет.

```
#include <sys/socket.h>
ssize_t sendto(int sockfd, const void *buffer, size_t length,
               int flags, const struct sockaddr *dest_addr,
               socklen_t addrlen);
// Возвращает количество отправленных байтов или -1 при ошибке
```

- Возвращаемое значение и первые три аргумента такие же, как у операции write().
- Четвертый аргумент, flags, представляет собой битовую маску, которая определяет специальные возможности ввода/вывода, связанные с сокетами.
- Аргументы dest_addr и addrlen описывают сокет, которому будет отправлена датаграмма.

Взаимодействие UDP-сокетов



Пример: *socket3-server.c*, *socket3-client.c*

Вызов connect() для UDP-сокета

Датаграммные сокеты не нуждаются в соединении, однако при работе с ними тоже можно применять системный вызов connect(). Он заставляет ядро записать адрес удаленного сокета. Такие сокеты называются **соединенными**.

После соединения датаграммного сокета:

- датаграммы, отправляемые с помощью вызова write() (или send()), автоматически передаются заданному удаленному сокету;
- мы можем читать только датаграммы, отправленные заданным удаленным сокетом.

Удаленный адрес соединенного датаграммного сокета можно изменить с помощью повторного вызова connect().