Сокеты

Наумов Д.А., доц. каф. КТ

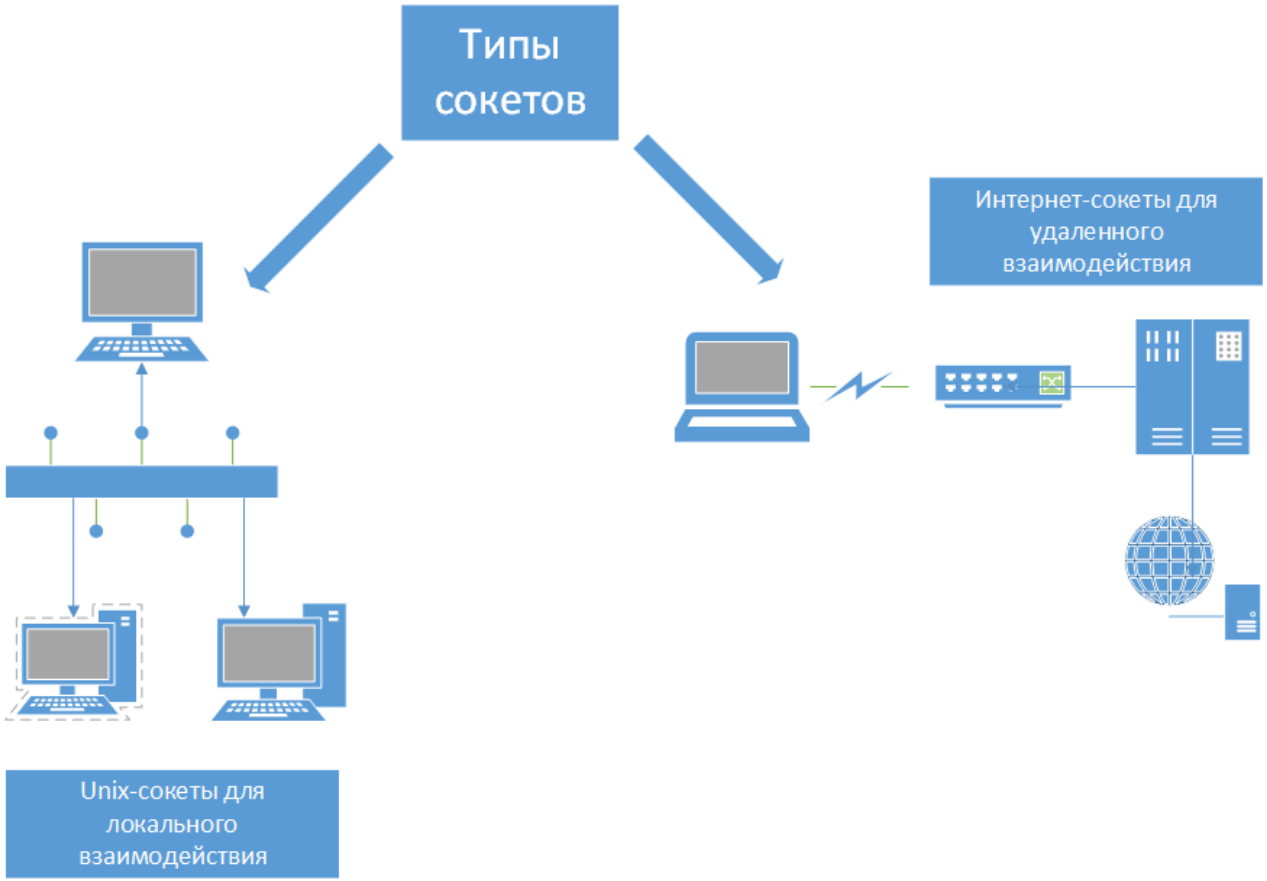
Операционные системы и системное программное обеспечение, 2019

Содержание лекции

Удаленное межпроцессное взаимодействие осуществляется *через сеть* посредством *сокетов*.

Сокет

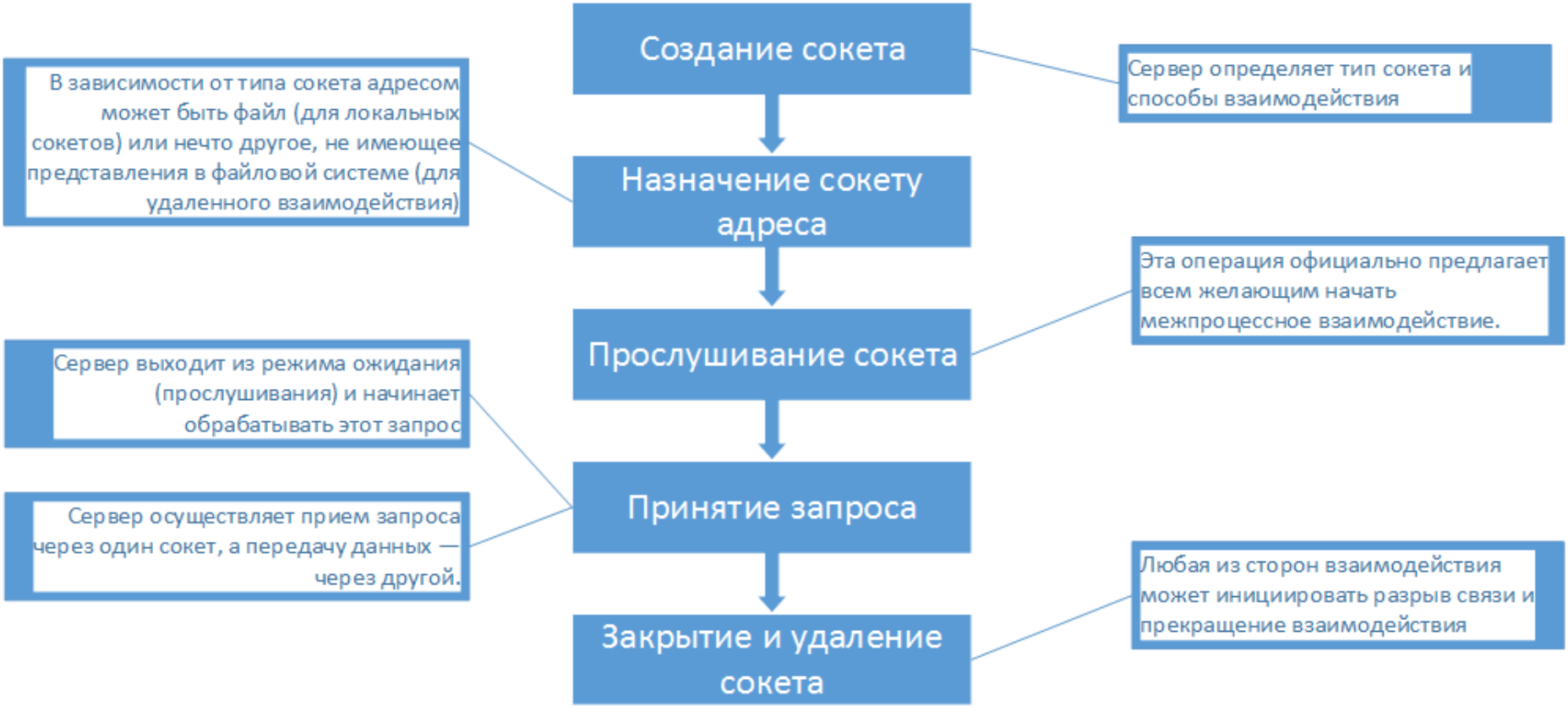
комплексное понятие, которое условно можно назвать «точкой соединения процессов».



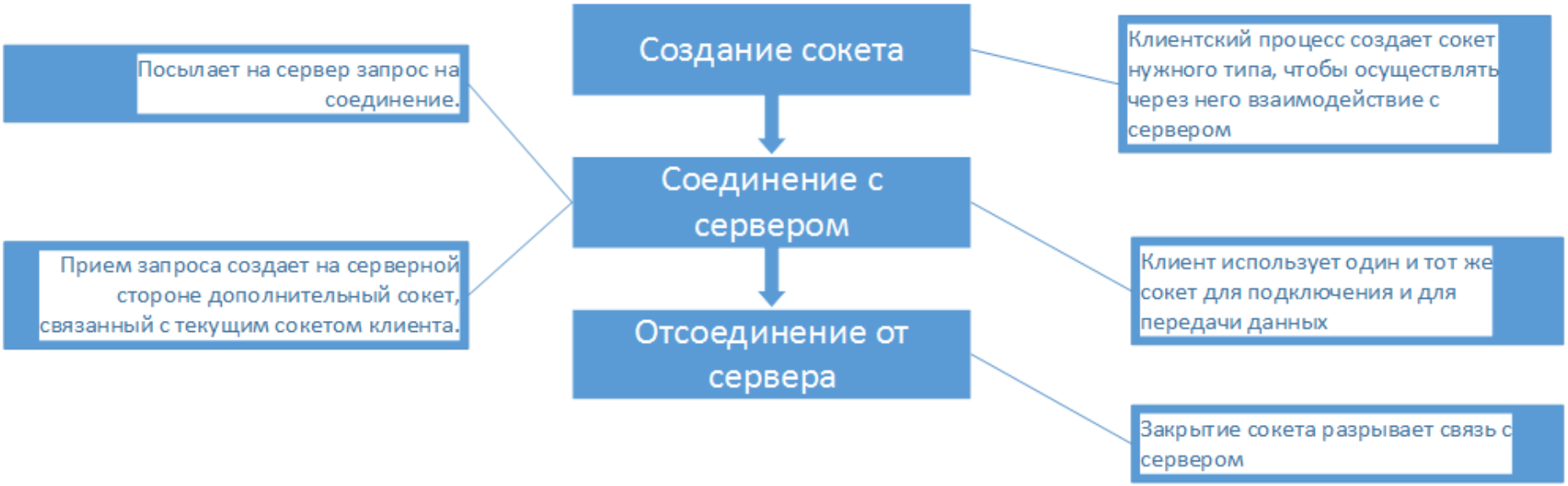
Осбенности использования сокетов:

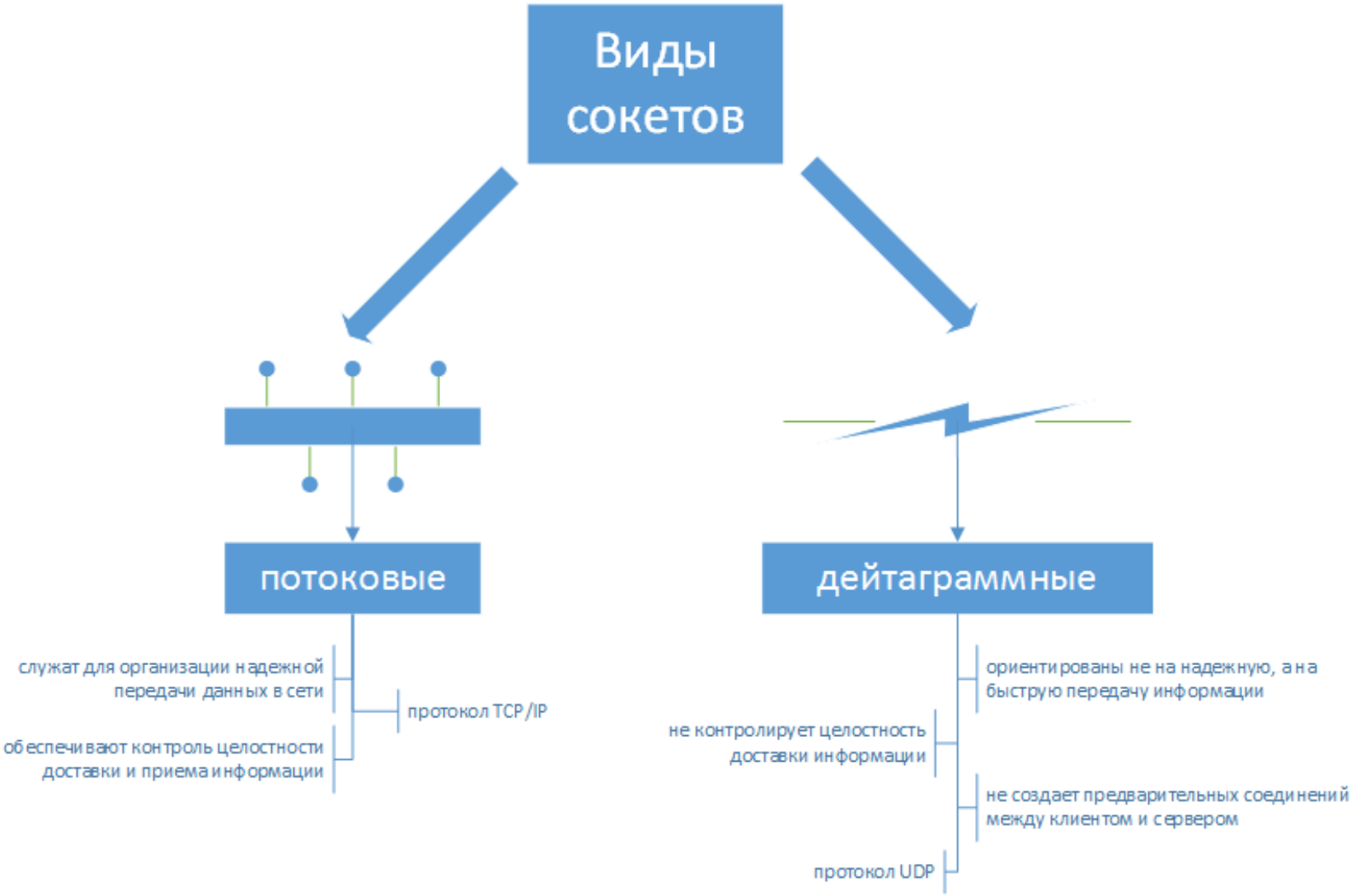
- сокеты фигурируют в виде *файловых дескрипторов*, над которыми (во многих случаях) можно осуществлять обычные операции чтения-записи (read(), write() и т. д.);
- набор действий, подготавливающих файловый дескриптор к использованию, несколько сложнее, чем в случае с обычными файлами;
- при взаимодействии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

Работа с сокетами на стороне сервера



Работа с сокетами на стороне клиента




Виды сокетов
потоковые
дейтаграммные
служат для организации надежной передачи данных в сети
обеспечивают контроль целостности доставки и приема информации
протокол TCP/IP
ориентированы не на надежную, а на быструю передачу информации
не контролирует целостность доставки информации
не создает предварительных соединений между клиентом и сервером
протокол UDP

Создание сокетов

Для создания сокетов предназначен системный вызов socket():

```
int socket (int PF, int SOCK_TYPE, int PROTOCOL)
```

- при успешном завершении возвращает дескриптор сокета (признак ошибки -1);
- PF - семейство протоколов сокета;
  - PF_LOCAL, PF_UNIX - локальные сокеты (Unix-сокеты);
  - PF_FILE - нестандартный синоним PF_LOCAL;
  - PF_INET - интернет-сокеты, основанные на IP версии 4;
  - PF_INET6 - интернет-сокеты, основанные на IP версии 6;
  - PF_BLUETOOTH - bluetooth-сокеты.
- SOCK_TYPE - способ взаимодействия;
  - SOCK_STREAM - потоковые сокеты;
  - SOCK_DGRAM - дейтаграммные сокеты
- PROTOCOL задание конкретного протокола передачи данных (0 - автоматический выбор).

Назначение адресов

Чтобы сервер и клиент могли взаимодействовать, сокету нужно назначить адрес (имя). В зависимости от типа сокета адресом может являться:

- имя файла (локальное взаимодействие);
- сетевой адрес и порт (удаленное взаимодействие).

Адрес сокета назначается на стороне сервера. Клиентский процесс может использовать этот адрес для подсоединения к серверу или для передачи данных.

```
int bind (int FD, const struct sockaddr * ADDRESS,
          socklen_t LEN);
```

- FD - это дескриптор сокета;
- ADDRESS - структура типа sockaddr задает адресное пространство и, собственно, сам адрес;
- LEN - указывает размер адресной структуры во втором аргументе.

Назначение адресов: 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

Чтобы сервер и клиент могли взаимодействовать, сокету нужно назначить адрес (имя). В зависимости от типа сокета адресом может являться:

- имя файла (локальное взаимодействие);
- сетевой адрес и порт (удаленное взаимодействие).

Адрес сокета назначается на стороне сервера. Клиентский процесс может использовать этот адрес для подсоединения к серверу или для передачи данных.

```
int bind (int FD, const struct sockaddr * ADDRESS,
          socklen_t LEN);
```

- FD - это дескриптор сокета;
- ADDRESS - структура типа sockaddr задает адресное пространство и, собственно, сам адрес;
- LEN - указывает размер адресной структуры во втором аргументе.

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.

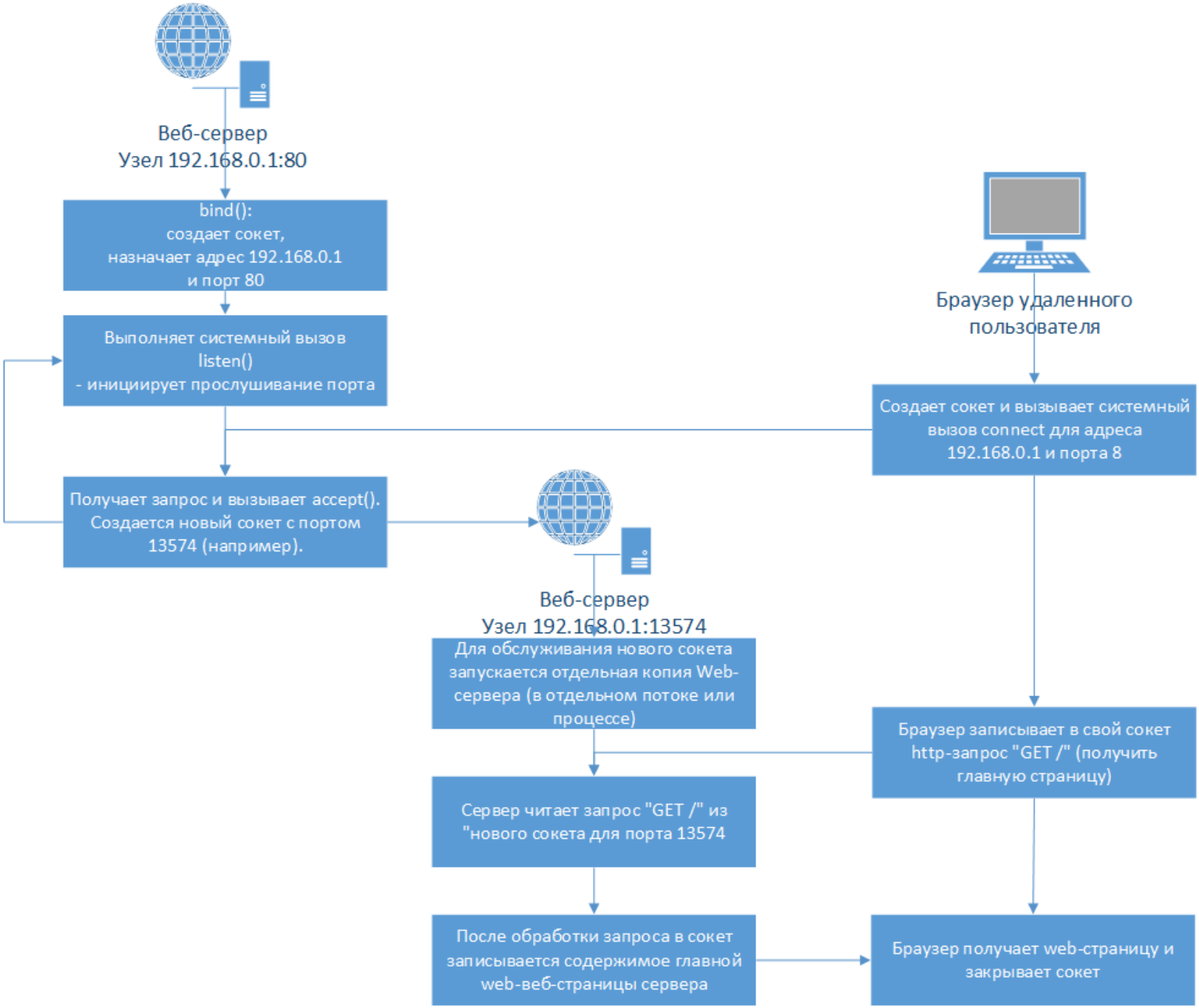
```
host = gethostbyname (argv[1]);
if (host == NULL) {
    fprintf (stderr, "Unknown server");
    return 1;
}
```

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().


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 FD, const struct sockaddr * SADDR,
                     socklen_t LEN);
```

- FD - это дескриптор сокета;
- SADDR - указатель на структуру, содержащую сведения об адресе сервера;
- LEN - размер адресной структуры.

Пример getwwwpage.c

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

Прослушивание сокета

При взаимодействии через потоковые сокеты сервер должен включить прослушивание, т. е. перейти в режим ожидания запросов на подключение при помощи системного вызова listen():

```
int listen (int FD, int QUEUE_LEN);
```

- FD - это дескриптор сокета;
- QUEUE_LEN определяет максимальный размер очереди запросов на подключение.

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

Принятие запроса на подключение

- 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():

```
int accept (int FD, struct sockaddr * ADDRESS,
            socklen_t * LENP);
```

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

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».

Прием и передача данных через сокеты

- **Дейтаграммные сокеты** не предназначены для установки соединений посредством системных вызовов connect(), listen() и accept().
- Для передачи информации без использования соединений нужны системные вызовы, которые посылают и принимают данные через непосредственные адреса.

Для передачи данных на сервер через дейтаграммный сокет предусмотрен системный вызов sendto():

```
ssize_t sendto (int FD, const void * BUFFER,
                size_t BUF_SIZE, int FLAGS,
                const struct sockaddr * SADDR,
                socklen_t * LEN);
```

- возвращаемое значение и первые три аргумента полностью идентичны тем, что используются в системном вызове write();
- FLAGS позволяет передавать дополнительные флаги; если

На серверной стороне данные принимаются при помощи системного вызова recvfrom():

```
ssize_t recvfrom (int FD, void * BUFFER,
                  size_t BUF_SIZE, int FLAGS,
                  struct sockaddr * CADDR,
                  socklen_t * LENP)
```

- системный вызов, кроме прочего, позволяет получить адрес отправителя, что важно в тех случаях, когда клиент ожидает ответ от сервера;
- возвращаемое значение и первые три аргумента полностью идентичны тем, что используются в системном вызове read();
- FLAGS позволяет передавать дополнительные флаги.
- Через CADDR сервер может получить адресную структуру клиента.
- LENP - это размер этой адресной структуры.

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