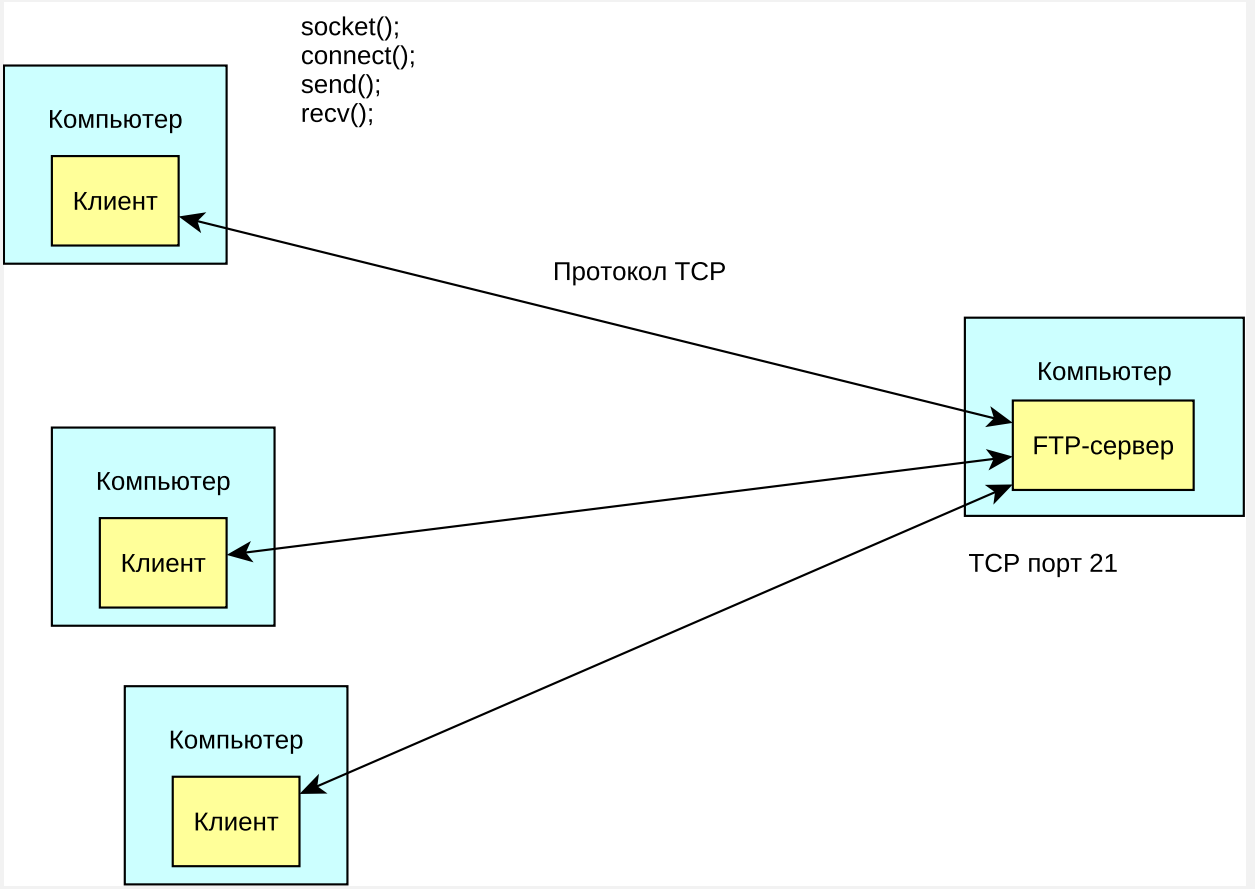
ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

28.04.20

Клиенты и сервер



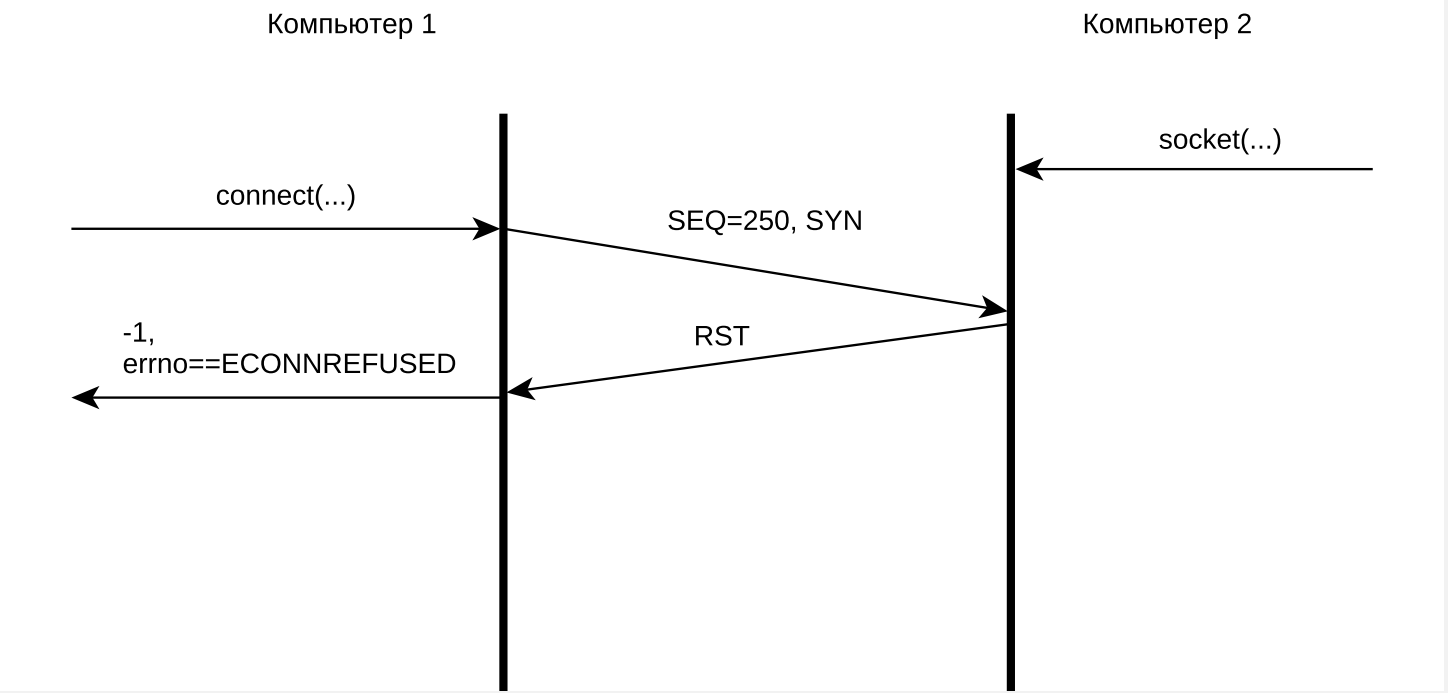
Вопросы

- Как указать номер TCP-порта в серверном приложении?
- Как выбирается номер TCP-порта в клиентском приложении?
- Как серверное приложение определяет источник (от какого клиента) принимаемых данных?

Слушающие и передающие сокеты

- В серверном приложении сетевое взаимодействие организовано с помощью 2 типов сокетов - *слушающего* сокет и *передающих* сокетов.
- Слушающий сокет отвечает за установку TCP-соединений со всеми клиентами.
- Слушающий сокет *привязывается* к определенному адресу и порту.
- Передающий сокет отвечает за передачу данных между 1 клиентом и сервером.
- Передающий сокет создается из слушающего сокета.
- Номер порта передающего сокета выделяется ядром динамически.

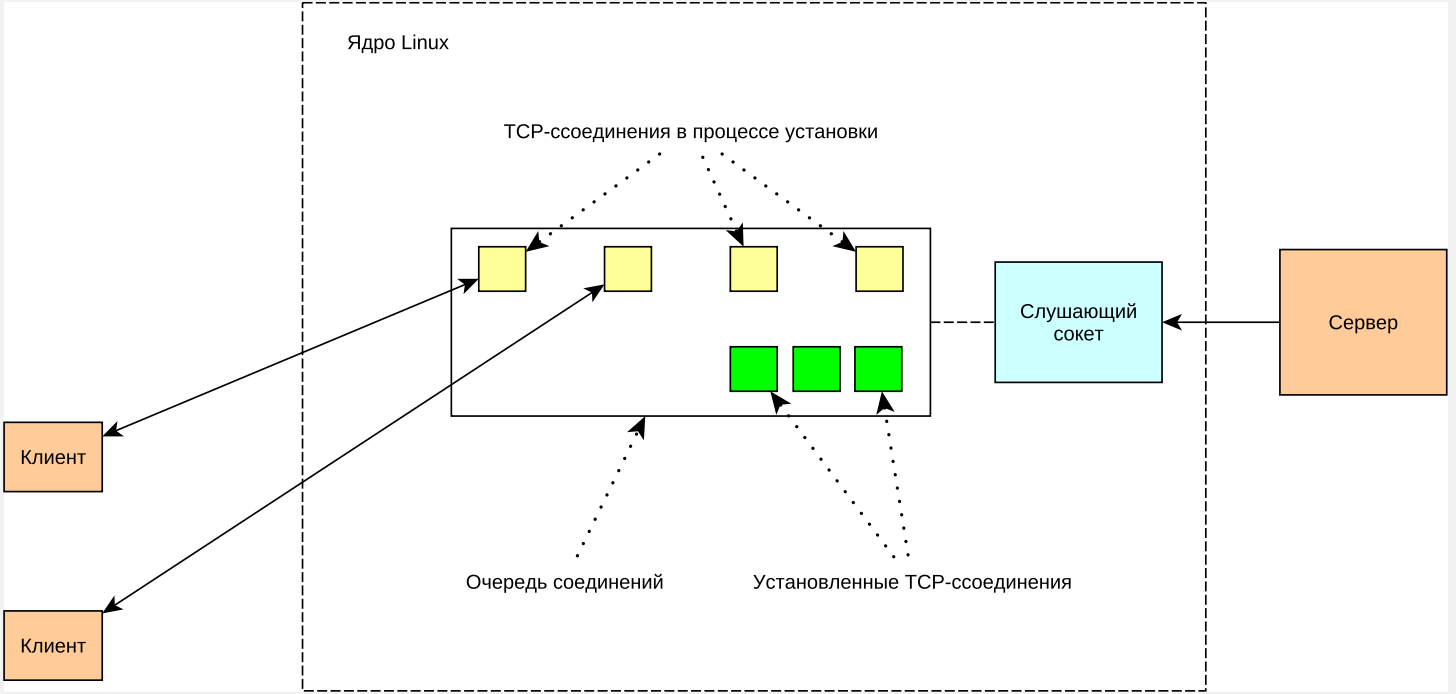
Серверный сокет до привязки



Привязка сокета, API

```
int sock;
struct sockaddr_in addr;
int ret;

sock = socket(AF_INET, SOCK_STREAM, 0);

addr.sin_family = AF_INET;
addr.sin_port = htons(port);
addr.sin_addr = {INADDR_ANY}; /* Привязка ко всем системным IP-адресам. */

/* int bind(int socket, const struct sockaddr *address, socklen_t addr_len); */
ret = bind(sock, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0) {
       fprintf(stderr, "Failed to bind to server: %s\n", strerror(errno));
       goto on_error;
}
```

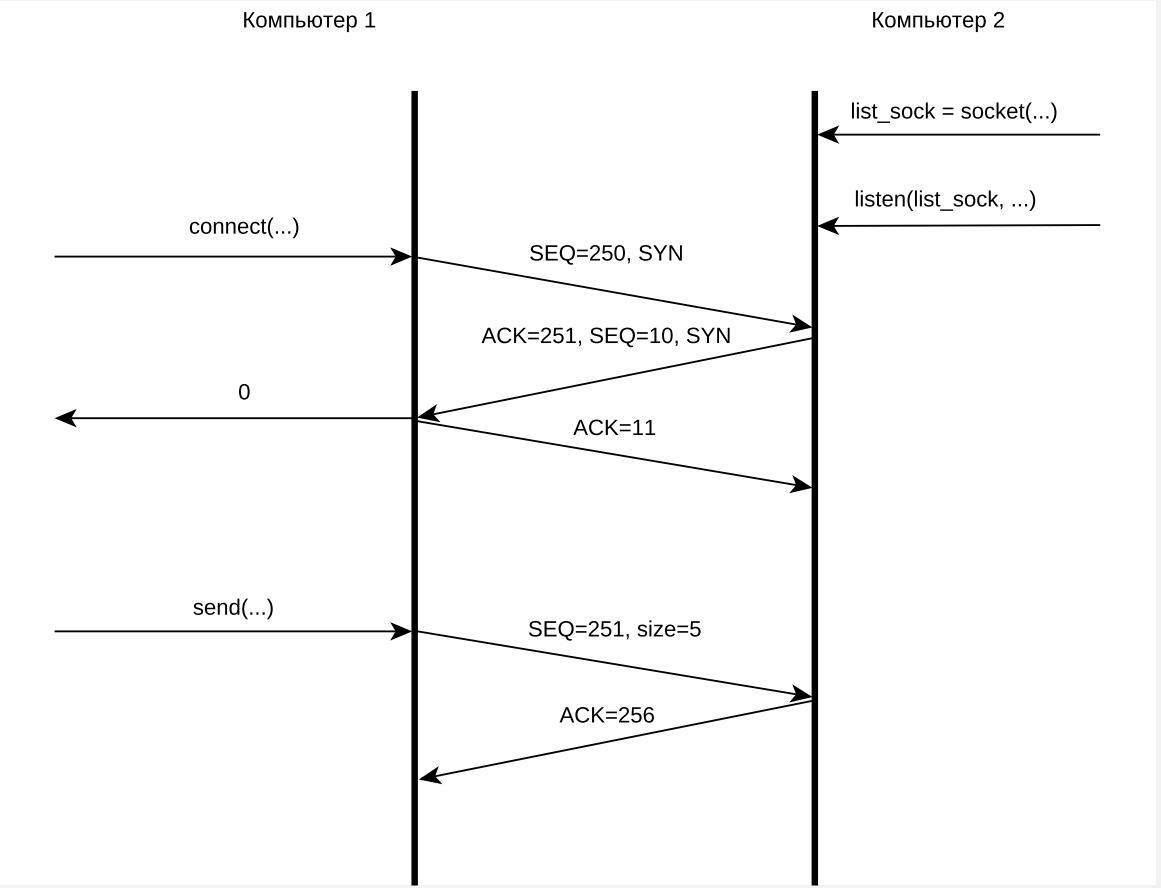
Очередь входящих соединений



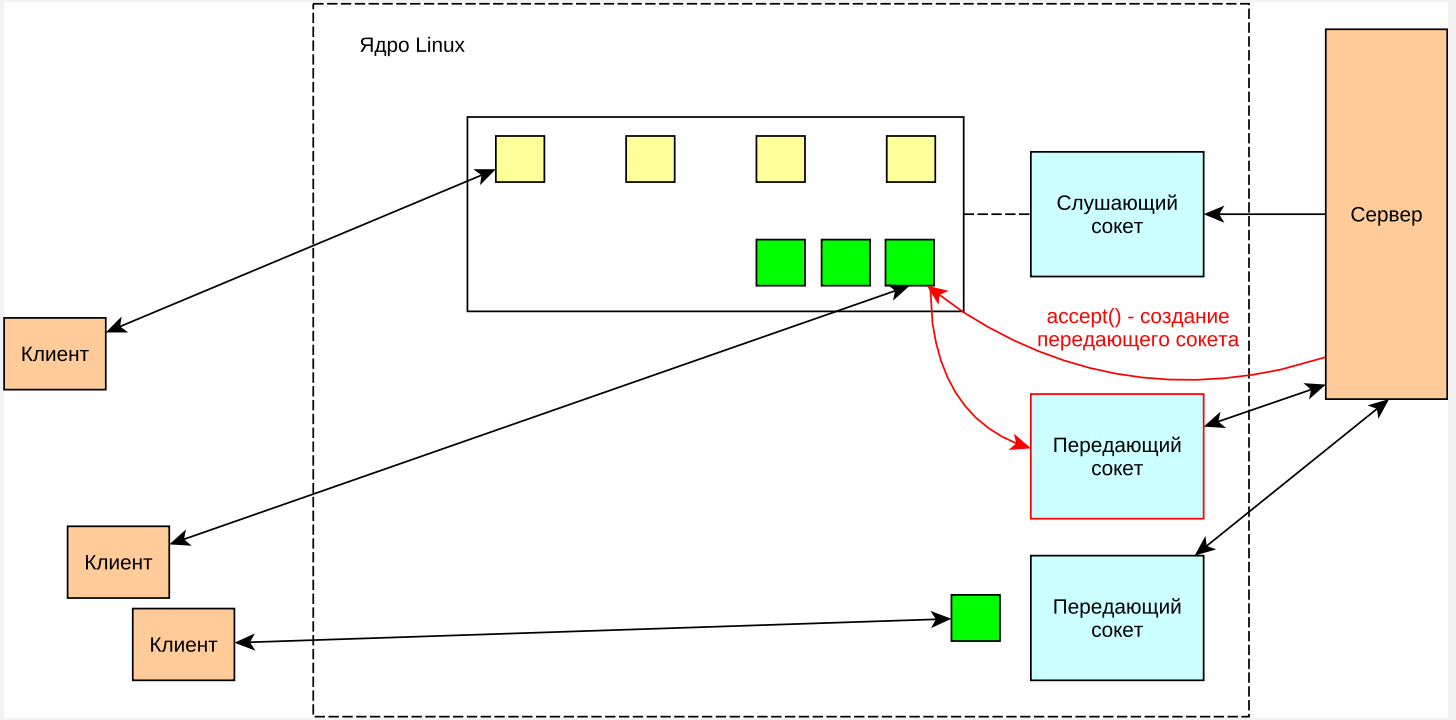
Перевод сокета в слушающий режим, API

```
int sock;
struct sockaddr_in addr;
int ret;

sock = socket(AF_INET, SOCK_STREAM, 0);

/* ... */
bind(sock, (struct sockaddr *)&addr, sizeof(addr));

/* int listen(int socket, int backlog); */
ret = listen(sock, 5);
if (ret < 0) {
       fprintf(stderr, "Failed to put socket into listening state: %s\n",
                     strerror(errno));
       goto on_error;
}
```

Работа сокета в слушающем режиме



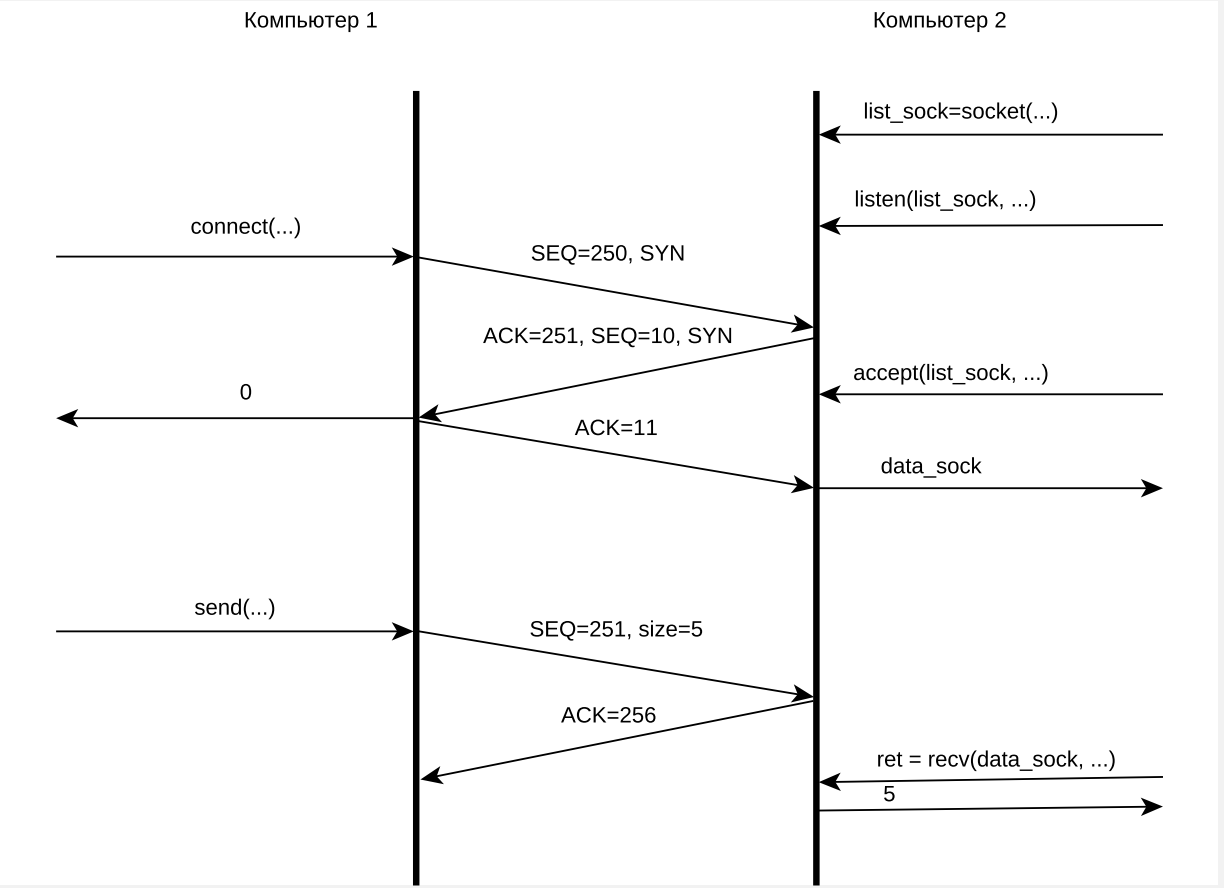
Передающий сокет



Создание передающего сокета, API

```
int listen_sock, data_sock;
struct sockaddr_in addr;
int ret;

listen_sock = socket(AF_INET, SOCK_STREAM, 0);

/* ... */
bind(listen_sock, (struct sockaddr *)&addr, sizeof(addr));
listen(listen_sock, 5);

/* int accept(int socket, struct sockaddr *address, socklen_t *addr_len); */
data_sock = accept(listen_sock, NULL, NULL);
if (data_sock < 0) {
       fprintf(stderr, "Failed to accept data connection: %s\n",
                     strerror(errno));
       goto on_error;
}

ret = recv(data_sock, buf, sizeof(buf), 0);
```

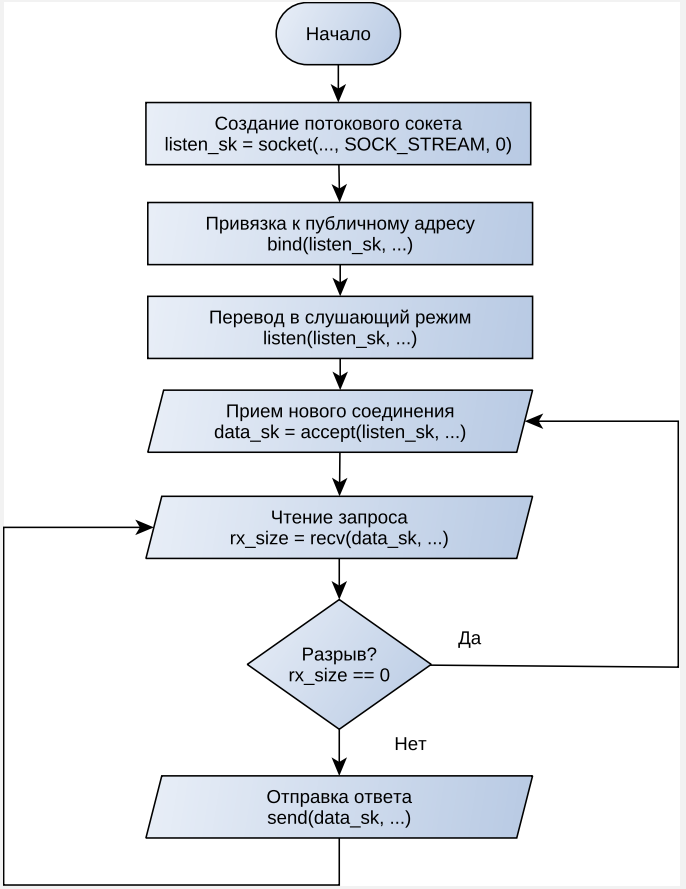
Работа сервера при создании передающих сокетов



Разрыв соединения со стороны клиента

```
	int listen_sock, data_sock;
	struct sockaddr_in addr;
	int ret;

wait_connection:
	data_sock = accept(listen_sock, NULL, NULL);
	if (data_sock < 0) {
		goto on_error;
	}

	while (1) {
		char rx_buf[100], tx_buf[100];

		ret = recv(data_sock, rx_buf, sizeof(rx_buf), 0);
		if (ret == 0) {
			printf("Client disconnected\n");
			goto wait_connection;
		}

		/* Обработка запроса. */

		send(data_sock, tx_buf, sizeof(tx_buf), 0);
	}
```

2 цикла обработки



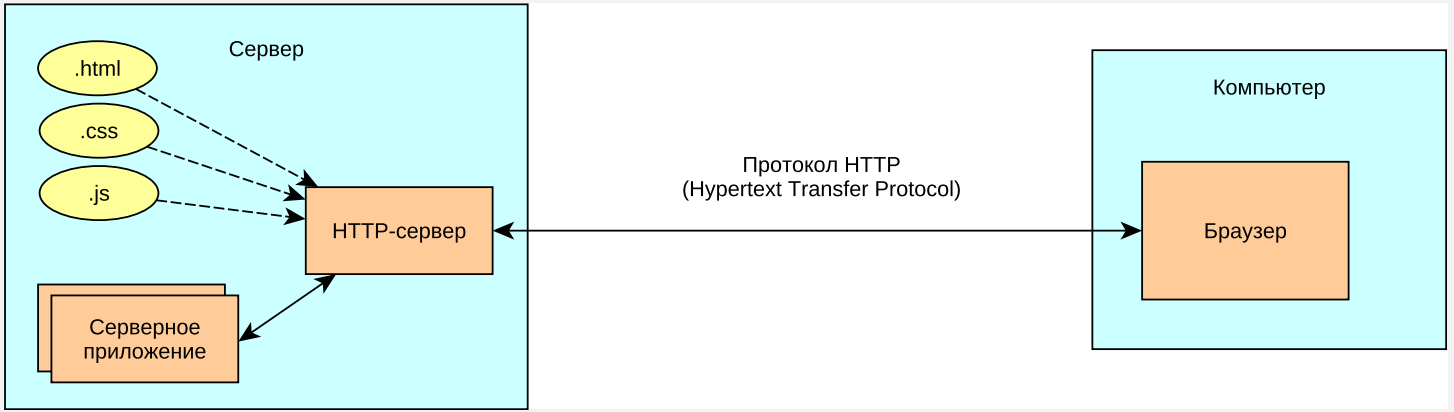
2 цикла обработки, пример

```
int listen_sock, data_sock;
struct sockaddr_in addr;
int ret;

listen_sock = socket(AF_INET, SOCK_STREAM, 0);
bind(listen_sock, (struct sockaddr *)&addr, sizeof(addr));
listen(listen_sock, 5);

while (1) {
	data_sock = accept(listen_sock, NULL, NULL);

	while (1) {
		char rx_buf[100], tx_buf[100];

		ret = recv(data_sock, rx_buf, sizeof(rx_buf), 0);
		if (ret == 0)
			break;

		/* Обработка запроса. */

		send(data_sock, tx_buf, sizeof(tx_buf), 0);
	}
}
```

Замечания

- Функции `accept()` и `recv()` являются блокирующими.
- Обмен данным в каждый момент времени производится максимум с одним клиентом. Соединения остальных ждут в очереди.

Web-технологии



Hyper-text transfer protocol (HTTP)

- Протокол обеспечивает удаленный доступ к сетевым *ресурсам*. Пример: чтение HTML-страницы.
- Ресурс идентифицируется с помощью URL - Uniform Resource Locator. Пример: http://google.com/index.html.
- Взаимодействие между клиентом и сервером по принципу запрос-ответ.
- К указанному ресурсу применяется *метод*: чтение, запись, изменение и др.
- HTTP использует TCP в качестве протокола транспортного уровня.
- Каждая пара запрос-ответ передается в отдельном соединении TCP.

Uniform Resource Locator (URL)

Синтаксис <схема>://<доменное имя>:<порт>/<путь>

Пример 1 http://angtel.ru/directory1/resource3

Пример 2 https://192.168.0.8/directory1/resource3

- Применение не ограничено протоколом HTTP. URL идентифицирует абстрактный сетевые ресурсы: файлы, люди, телефоны, почтовые ящики и др.
- Схема - задает протокол и определяет синтаксис остальной части URL. https- HTTP поверх протокола TLS.
- Доменные имена преобразуются в IP-адреса с помощью протокола DNS.
- Порт - опциональный TCP или UDP порт.
- Путь интерпретируется HTTP-сервером. Не обязательно путь к файлу в файловой системе.

Методы HTTP

- GET - чтение содержимого ресурса.
- PUT - задание нового содержимого ресурса.
- POST - передача данных ресурсу.
- DELETE - удаление ресурса.
- PATCH - частичное изменение ресурса.

Тип содержимого, MIME

Содержимое ресурса передается в *теле* запроса или ответа. В зависимости от ресурса формат данных в теле может быть разным. Этот формат указывается в сообщении в соответствии со стандартом MIME (Multipurpose Internet Mail Extensions).

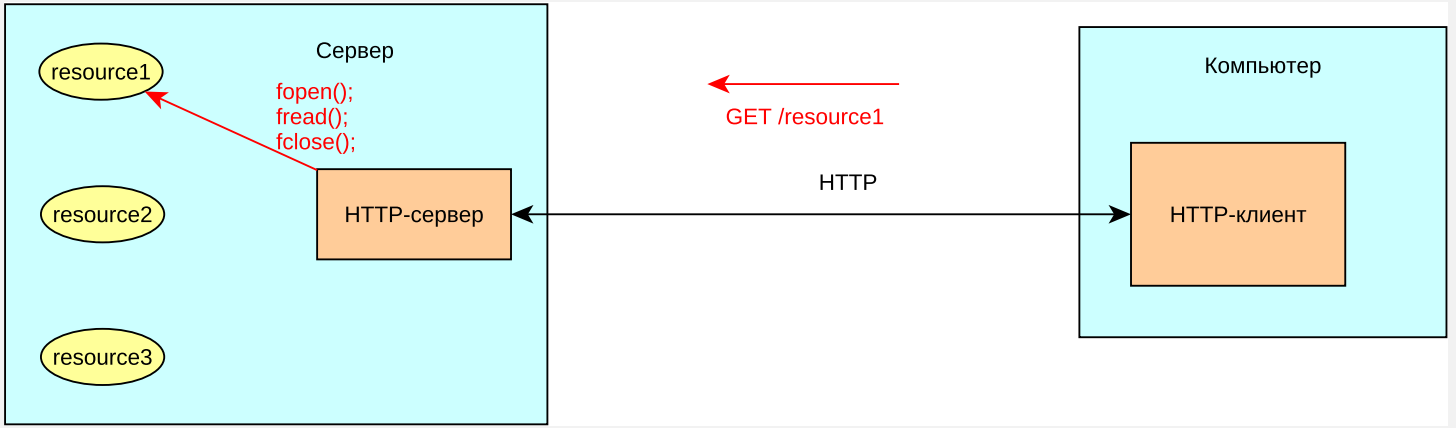
Синтаксис: <тип>/<подтип>

Распространенные типы: image, text, audio, application.

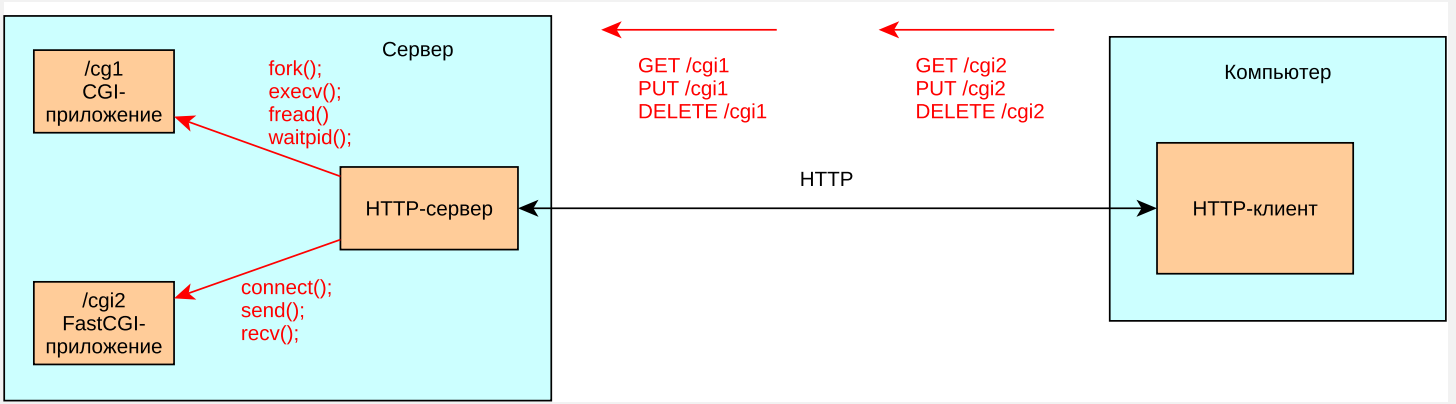
Примеры: text/html, image/jpeg, application/pdf, application/json.

Тип содержимого указывается в *заголовке* Content-Type в HTTP сообщении.

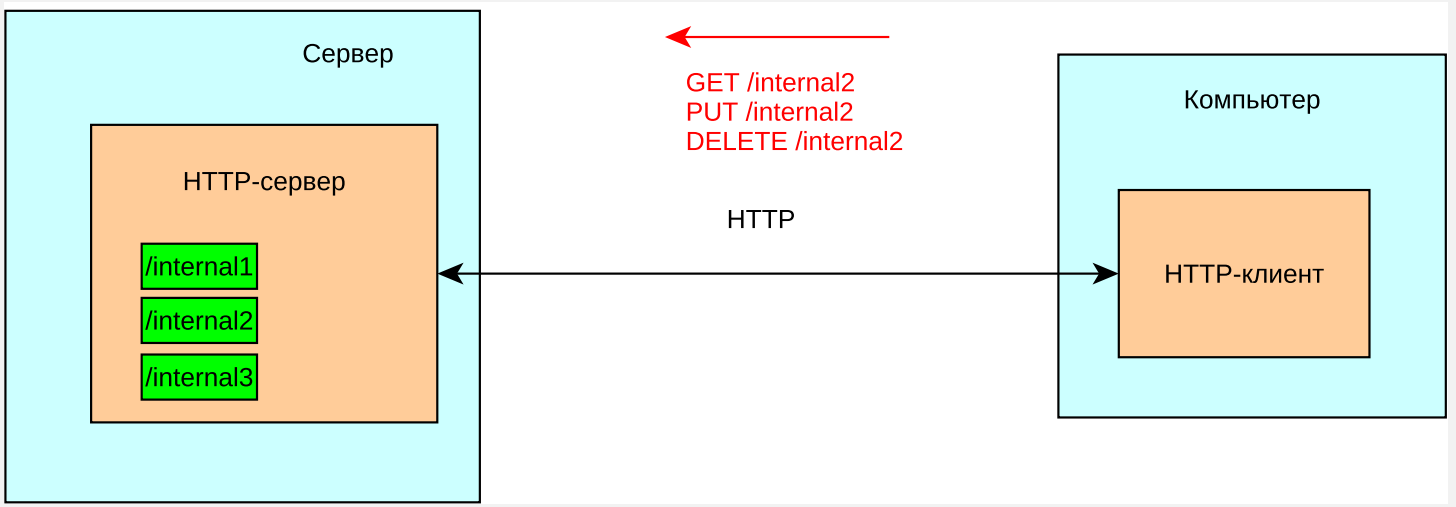
Доступ к страницам HTML



Доступ к ресурсам других приложений



Доступ к внутренним ресурсам сервера



Программируемая сеть

- Протокол HTTP можно использовать отдельно от технологий HTML, CSS, JavaScript и др. Клиентом в таком случае является не браузер, а некоторое неитерактивное управляющее приложение.
- REST (Representational State Transfer) - подход к построению API системы на основе сетевых ресурсов и протокола HTTP.
- REST - возможная основа программируемой сети устройств IoT.

Пример - протокол RESTCONF:

- Внутренние объекты системы (сетевые интерфейсы, таблицы маршрутизации, датчики и т.д.) представляются в виде сетевых ресурсов.
- Задание конфигурации производится через применение методов PUT, PATCH, DELETE к соответствующим ресурсам.
- Чтение конфигурации и статуса производится через применение метода GET.

Синтаксис HTTP-запроса

```
<Имя метода> <Путь> HTTP/1.1 \r\n
<Имя заголовка>: <Значение> \r\n
...
<Имя заголовка>: <Значение> \r\n
\r\n
<Тело запроса>
```

Пример:

```
PUT /directory/resource2 HTTP1.1 \r\n
Host: angtel.ru \r\n
User-Agent: firefox \r\n
Content-Type: text/html \r\n
\r\n
<p> sdgsrthsrt <\p>
```

Синтаксис HTTP-ответа

```
HTTP/1.1 <Код> <Сообщение> \r\n
<Имя заголовка>: <Значение> \r\n
...
<Имя заголовка>: <Значение> \r\n
\r\n
<Тело ответа>
```

Пример:

```
HTTP/1.1 301 Moved Permanently\r\n
Location: http://www.google.com\r\n
\r\n
```

picohttprequest, API

```
const char *method_ptr, *path_ptr;
struct phr_header headers[100];
size_t method_len, path_len, num_headers;
int minor_version;
int headers_len;

num_headers = 100;
/* buf - буфер с запросом HTTP;
 * len - размер буфера;
 * method_ptr, path_ptr - указатели на метод и путь в HTTP-запросе,
 *                        задаются функцией phr_parse_request;
 * method_len, path_len - длина метода и пути в HTTP-запросе,
 *                        задаются функцией phr_parse_request;
 * headers - массив объектов, описывающих заголовки запроса,
 *           заполняются функцией;
 * num_headers - на входе определяет максимальное количество заголовков,
 *               на выходе - обнаруженное количество заголовков.
 */
headers_len = phr_parse_request(buf, len, &method_ptr, &method_len, &path_ptr, &path_len,
                                &minor_version, headers, &num_headers, 0);
if (headers_len <= 0)
       return -1;
```

picohttprequest, API 2

```
const char *method_ptr, *path_ptr;
struct phr_header headers[100];
size_t method_len, path_len, num_headers;
int minor_version;
int headers_len;

num_headers = 100;
headers_len = phr_parse_request(buf, len, &method_ptr, &method_len, &path_ptr, &path_len,
                                &minor_version, headers, &num_headers, 0);
if (headers_len <= 0)
       return -1;

/* После вызова функции path_ptr указывает на символ внутри буфера buf, на
первый символ пути. Этот путь не ограничен нулевым терминатором '\0', так как
phr_parse_request не меняет содержимое исходного буфера buf. Поэтому
использовать path_str как C-строку нельзя! Для получения C-строки с путем
необходимо скопировать путь в отдельный буфер. */
snprintf(path, PATH_SIZE, "%.*s", (int)path_len, path_ptr);

snprintf(body, BODY_SIZE, "%.*s", (int)(len - headers_len), buf + headers_len);
```

Формирование ответа, API

```
int status = 200;
char status_msg[] = "Status message";
char body[] = "Response body";

char response[1024];

snprintf(response, sizeof(response), "HTTP/1.1 %d %s\r\n"
                                     "Content-Type: text/plain\r\n"
                                     "\r\n"
                                     "%s",
                                     status, status_msg, body);
```

Работа с HTTP в shell

Отправка запроса GET для ресурса resource1 на сайте google.com и отображение тела ответа:

```
$ curl google.com/resource1
```

Отправка запроса GET и отображение запроса и ответа со всеми заголовками:

```
$ curl google.com/resource1 -v
```

Отправка запроса PUT с заданным телом:

```
$ curl google.com/resource2 -X PUT --data "New resource content"
```

Повторная привязка сокета после перезапуска приложения

```
int listen_sock = -1;
int option = 1;
struct sockaddr_in addr = {
       .sin_family = AF_INET,
       .sin_port = htons(port),
       .sin_addr = {INADDR_ANY},
};
int ret;

listen_sock = socket(AF_INET, SOCK_STREAM, 0);

/* Установка опции SO_REUSEADDR позволяет создавать и привязывать сокет
 * сразу после закрытия предыдущего сокета на этом порте. Ускоряет перезапуск
 * программы при отладке.
 */
ret = setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option));
if (ret < 0) {
       fprintf(stderr, "Failed to set socket option: %s\n", strerror(errno));
       goto on_error;
}
```