СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ

ВВЕДЕНИЕ

Лектор

Дубовик Марина Владимировна

старший преподаватель кафедры ИСиТ

ауд. 115-1

dubovik@belstu.by

https://vk.com/marina_dubovik

Учебный план

Семестр	ЛК (час)	ЛР (час)	КП (час)	Итого (час)	Форма контроля
6	32	32	-	64	зачет

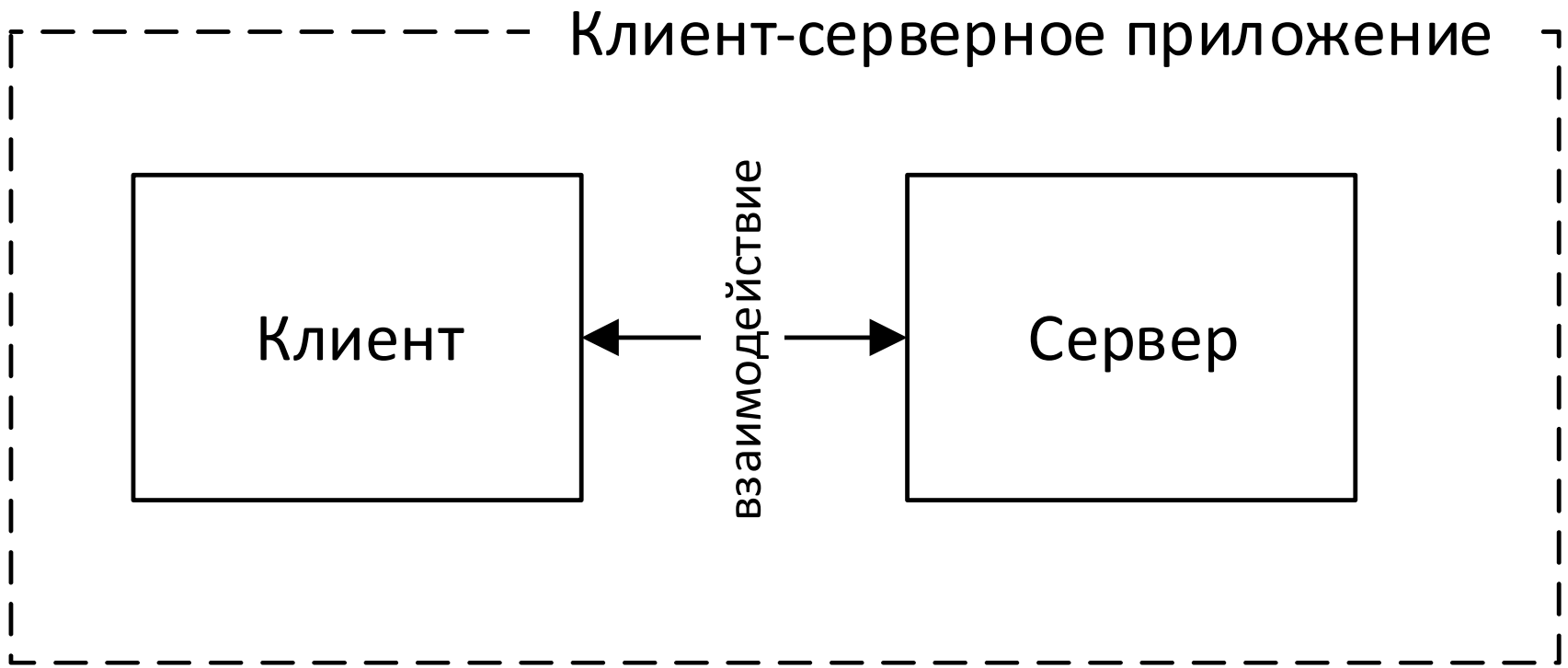
План лекций


Swagger
Авторизация
ORM
(sequelize, prisma)
Websocket-сервер
REST
GraphQL
HTTPS
HTTP-клиент
(axios)
Таймеры
EventEmitter
Асинхронное
программирование
FS
Express
Аутентификация
Глобальные
объекты
Модули, пакеты, npm
Введение в Node.js

Клиент-серверное приложение =

приложение (программа) с клиент-серверной архитектурой:

- состоит из двух компонент – клиента и сервера;
- клиент и сервер взаимодействуют между собой в соответствии с заданными правилами (протоколами);
- для взаимодействия между клиентом и сервером в соответствии с правилами (протоколом) должно быть установлено соединение;
- инициатором соединения – клиент.

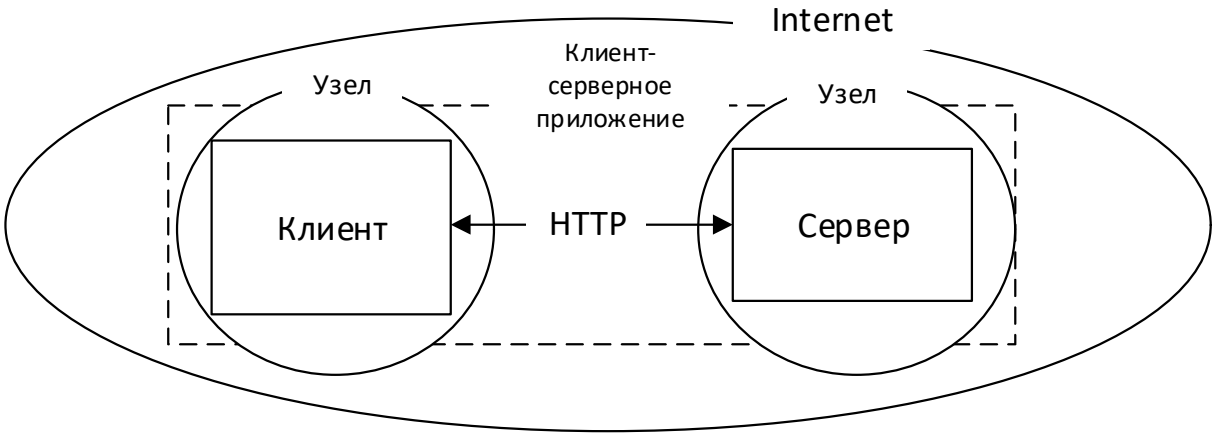

Клиент-серверное приложение
Клиент
взаимодействие
Сервер

Web-приложение =

клиент-серверное приложение, у которого клиент и сервер взаимодействуют по некоторому протоколу прикладного уровня.

Когда говорят о разработке web-приложения, говорят о разработке frontend (клиента) и backend (сервера)

Курс посвящен разработке серверной части web-приложения или иначе разработке web-сервера (backend).



Узел сети Интернет

=

устройство, имеющее IP-адрес и подключенное к сети Интернет (обычно к сети Интернет-провайдера).

Каждый узел характеризуется своей программно-аппаратной платформой – аппаратурой и операционной системой.

Сеть Интернет =

1) сеть на основе TCP/IP;
2) стандарты Internet (RFC, STD);
3) службы Интернет (DNS, SMTP/POP3/IMAP, WWW, FTP, Telnet, SSH,…);
4) организации, управляющие сетью Internet (ISOC, IETF, W3C, ICANN, IANA, …).

Кроссплатформенное приложение =

приложение, способное работать на более чем одной программно-аппаратной (аппаратура + операционная система) платформе.



Кроссплатформенность может быть достигнута различными способами:

1) на уровне компилятора (C, C++);

2) на уровне среды (или фреймворка) исполнения (Java/JVM, C#/.NET CORE/CLR, JS/Node, …)

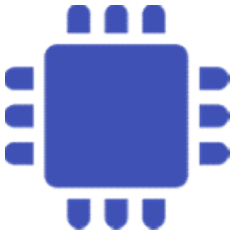
C++

Java/C#

Технологии для разработки кроссплатформенных web-серверов

- PHP / Apache, LAMP;
- Java / JVM / Application Server;
- C# / ASP.NET CORE;
- Python / Django;
- Ruby on Rails;
- JS / Node.js, ….

Ресурсы, потребляемые web-сервером



Процессор

Оперативная память

Жесткий диск

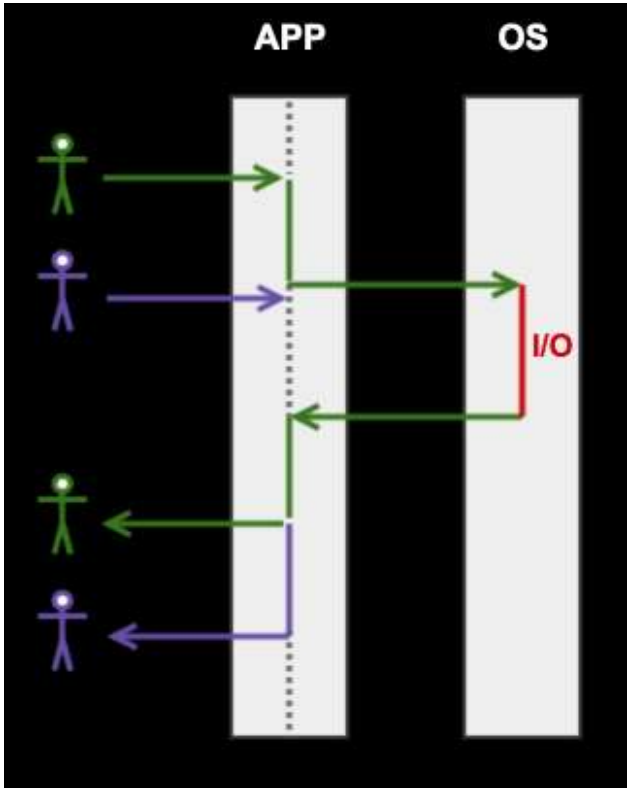
Сетевой интерфейс

Вычислительные задачи можно разделить на две категории:

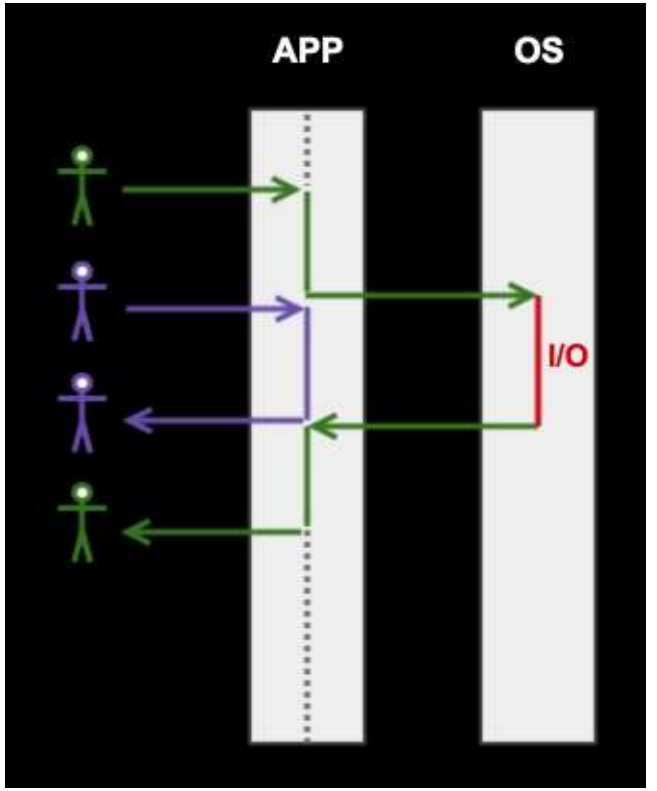
- **CPU bound** – нагружающие процессор (*вычисление факториала, вычисление хэша, процесс шифрования…*). Такие операции нагружают вычислительные мощности текущего устройства.
- **I/O bound** – требующие ввода/вывода (*поиск по файлу, подсчет числа строк в файле, копирование директории…*). В своей базовой форме Node.js лучше всего подходит для этого типа вычислений.

IO-операции

блокирующие



неблокирующие

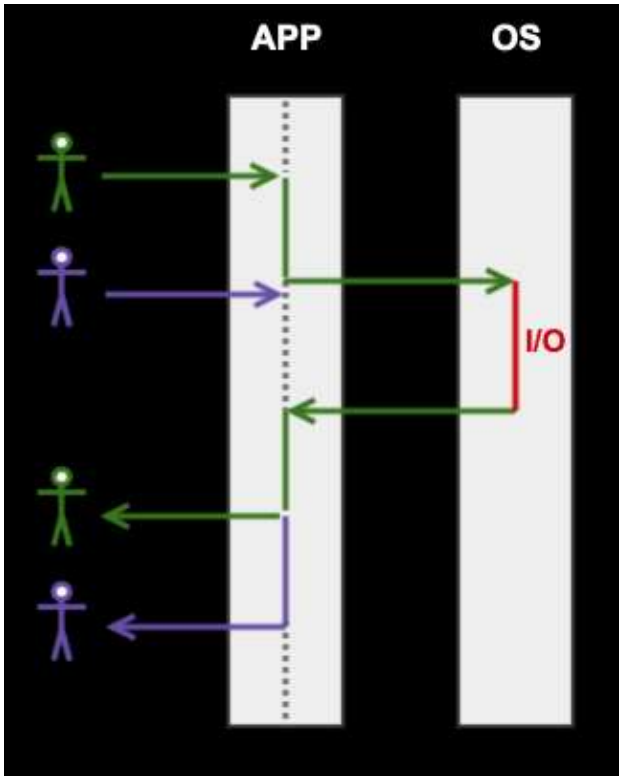


Подходы для решения проблем блокирующего ввода/вывода:

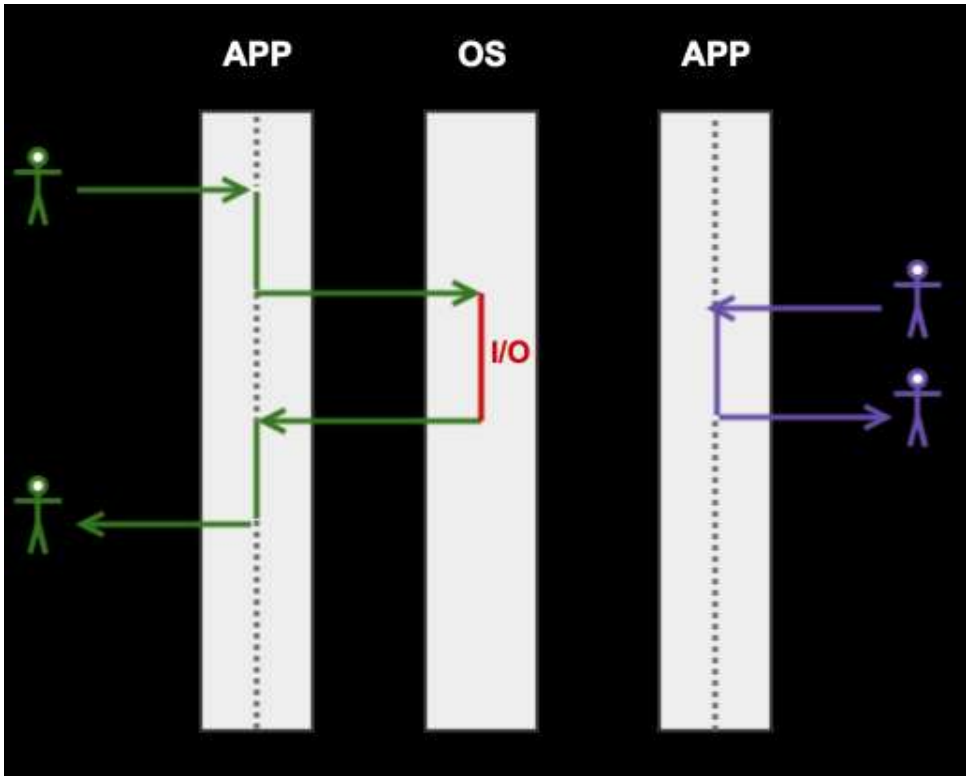
1. Применение многопоточности (ограничение по количеству потоков, каждый поток требует дополнительной памяти, синхронизация, отладка).
2. Применение паттерна Reactor.

Apache – многопоточность, Nginx – Reactor.

Однопоточность
(единственная задача в один момент времени)



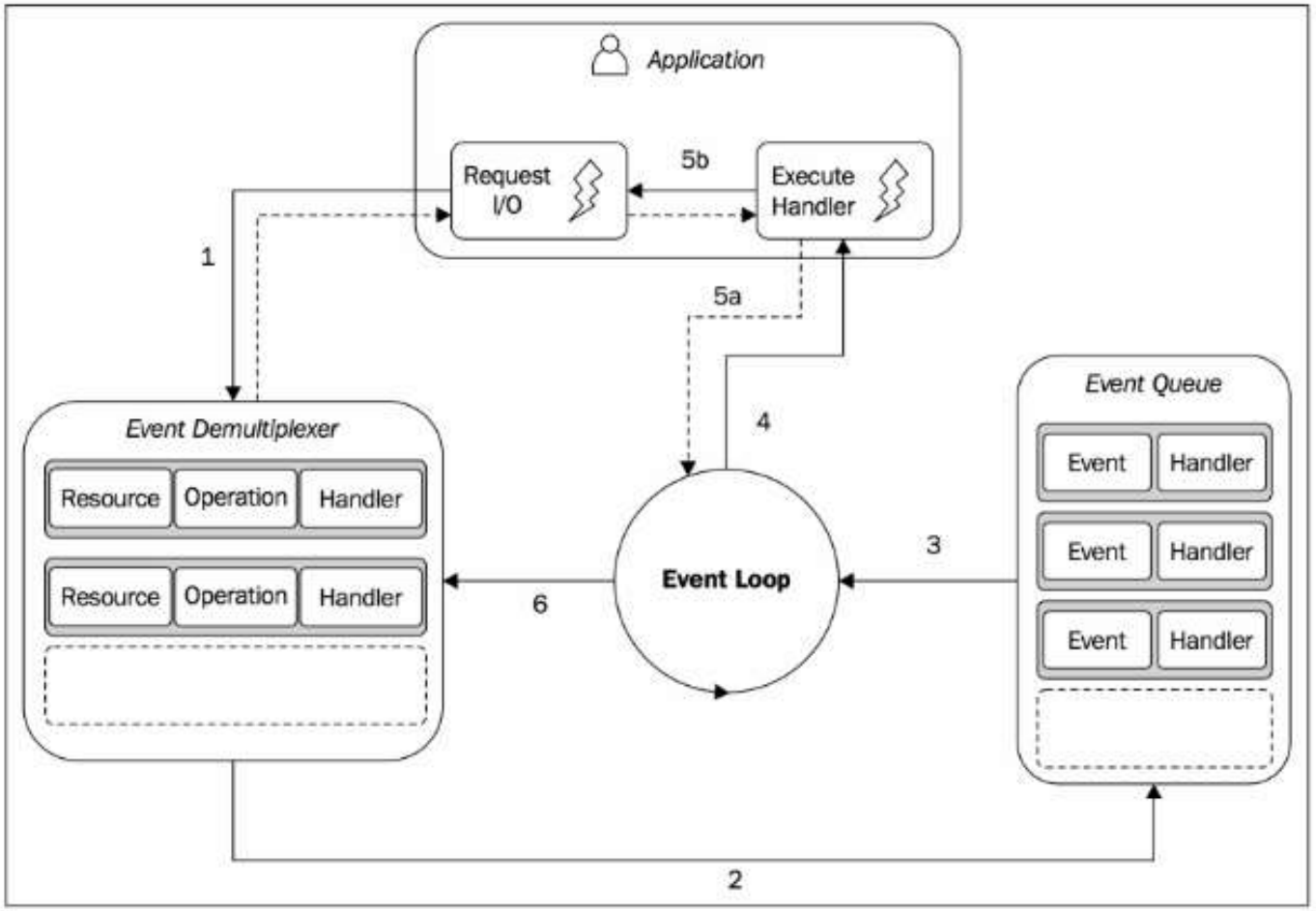
Многопоточность
(различные задачи в один момент времени)



Паттерн Reactor

Используется при обработке параллельных запросов к сервису.

Демультиплексор разбирает прибывшие запросы и синхронно перенаправляет их на соответствующие обработчики запросов.

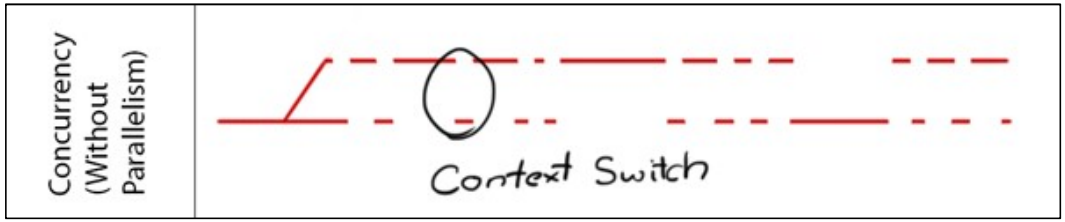


Какой подход используется в Node.js?

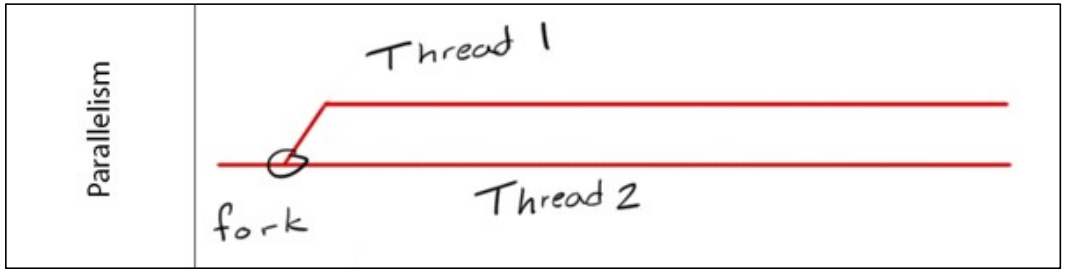
Многопоточность

Паттерн Reactor

Конкурентность и параллельность



- Работа с несколькими потоками управления.
- Каждый поток получает квант времени на исполнение.



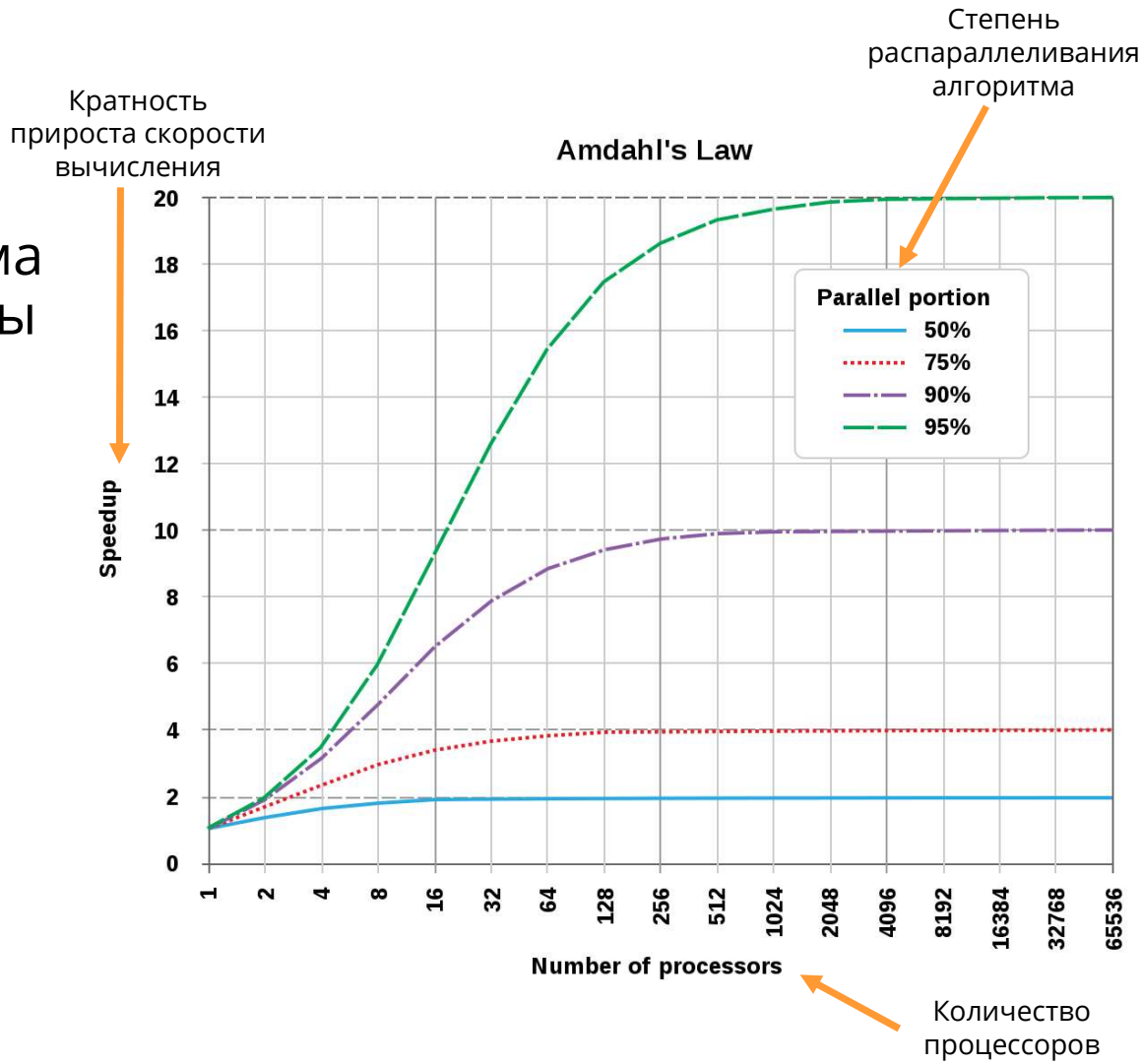
- Необходима аппаратная составляющая.
- Наличие потоков необязательно.

Закон Амдала

Идеальный случай: система из n процессоров могла бы ускорить вычисления в n раз.

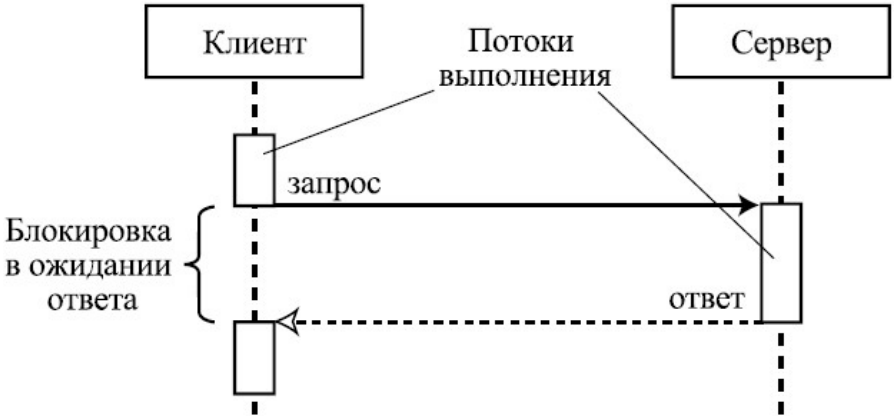
Проблемы:

- невозможность полного распараллеливания ни одной из задач;
- невозможность одинаковой загрузки каждого процессора.



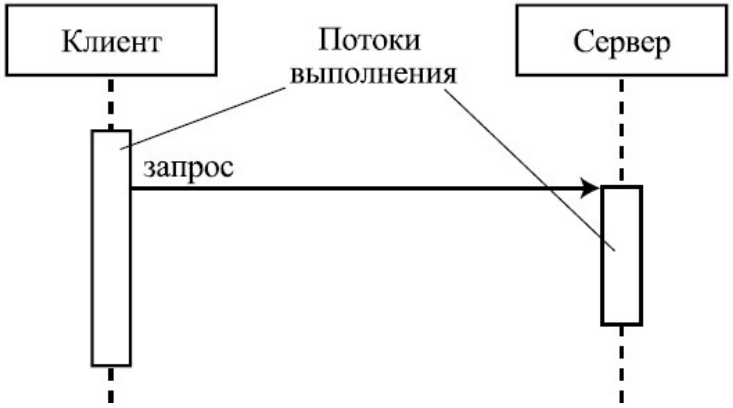
Синхронный запрос

запрос, при котором поток, выдавший http-запрос, блокируется до поступления ответа



Асинхронный запрос

запрос, при котором поток, выдавший http-запрос, не блокируется до поступления запроса; для обработки ответа применяется функция обратного вызова.



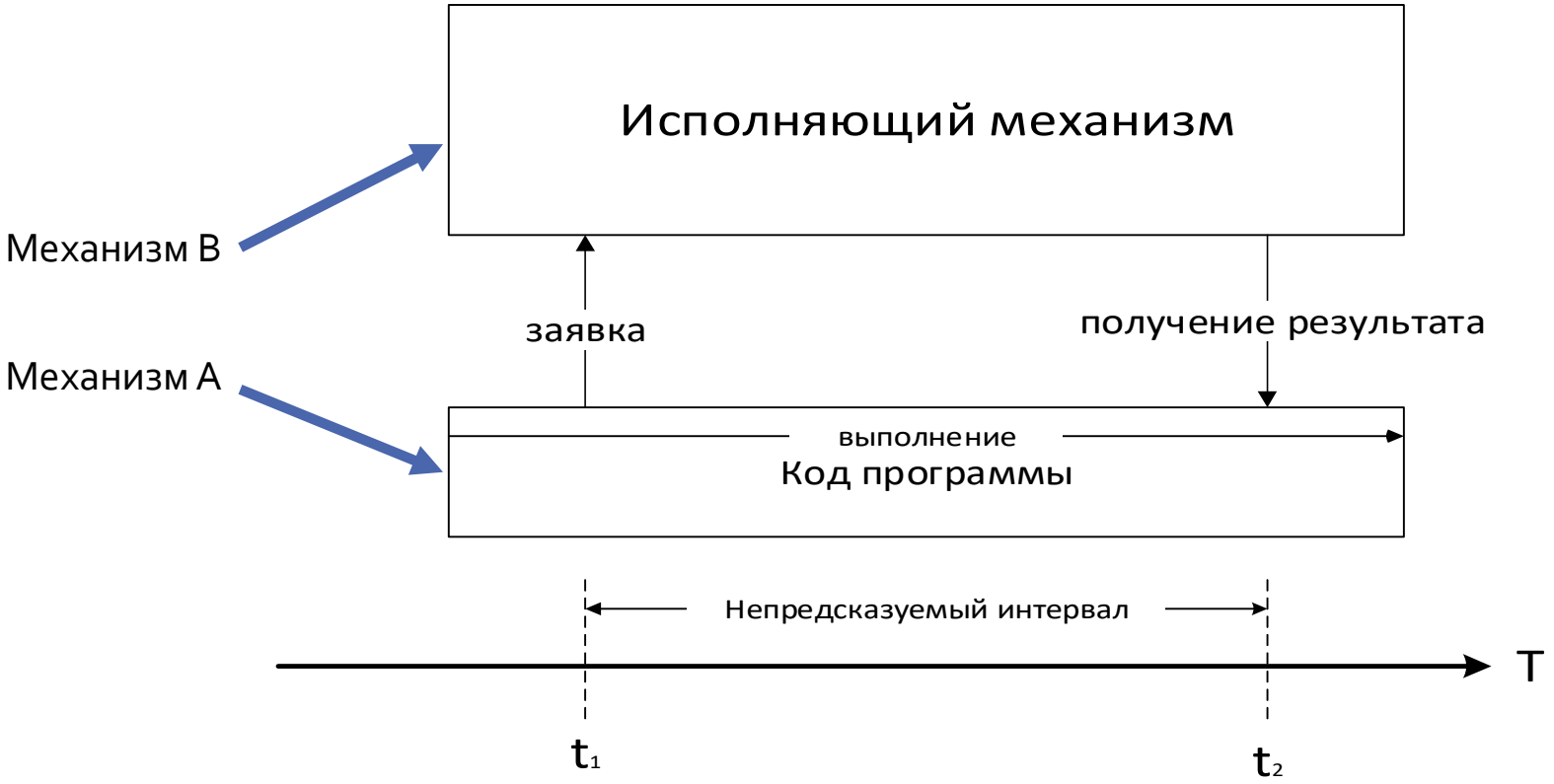
Асинхронность =

операция называется асинхронной, если ее выполнение осуществляется в 2 фазы:

1) заявка на исполнение;
2) получение результата; при этом участвуют два механизма: А-механизм, формирующий заявку и потом получающий результат; В-механизм, получающий заявку от А, исполняющий операцию и отправляющий результат А; продолжительность исполнения операции В-механизмом, как правило, непредсказуемо; в то время пока В-механизм исполняет операцию, А-механизм выполняет собственную работу.

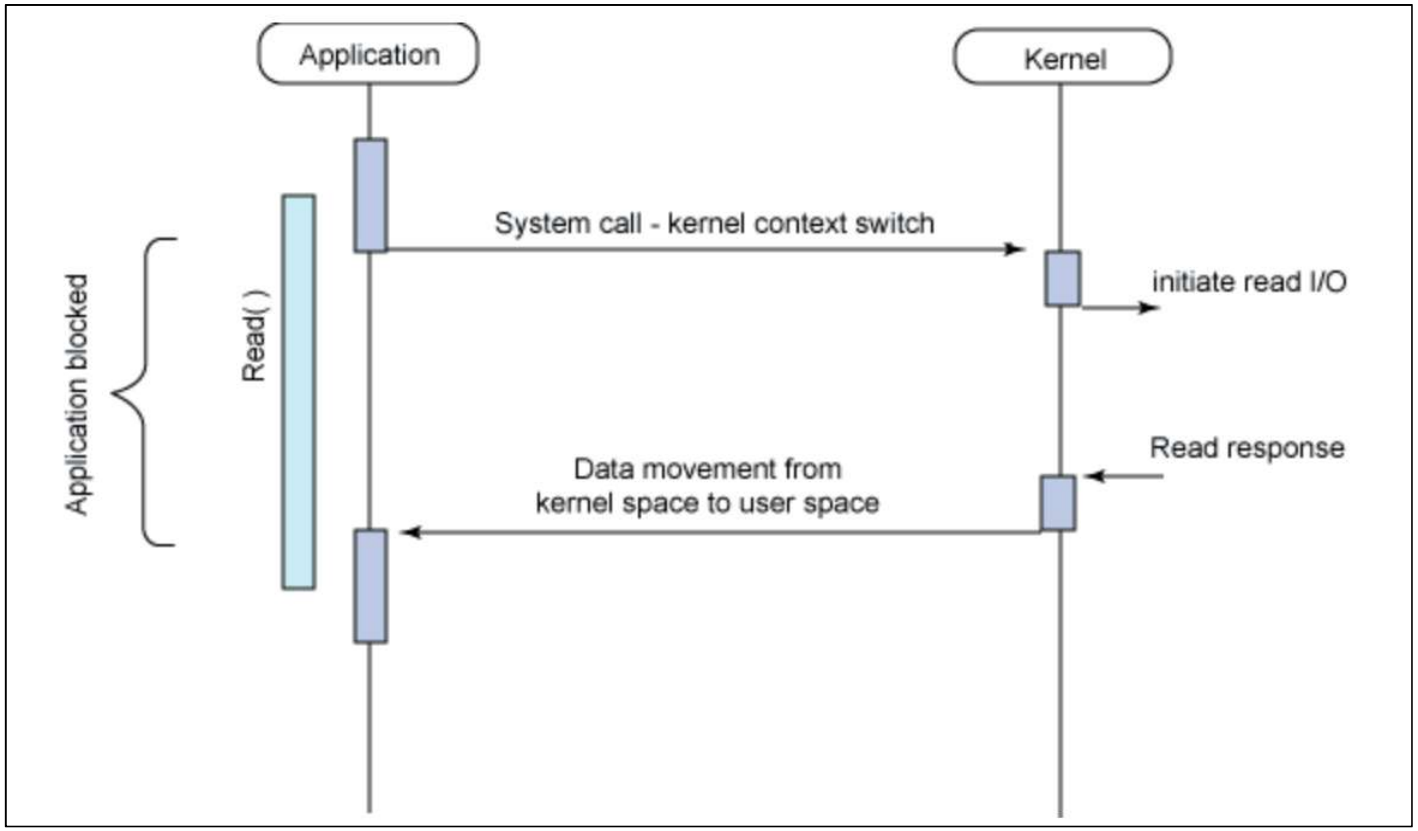
Применение асинхронности не противоречит применению многопоточности.

Асинхронность в программировании



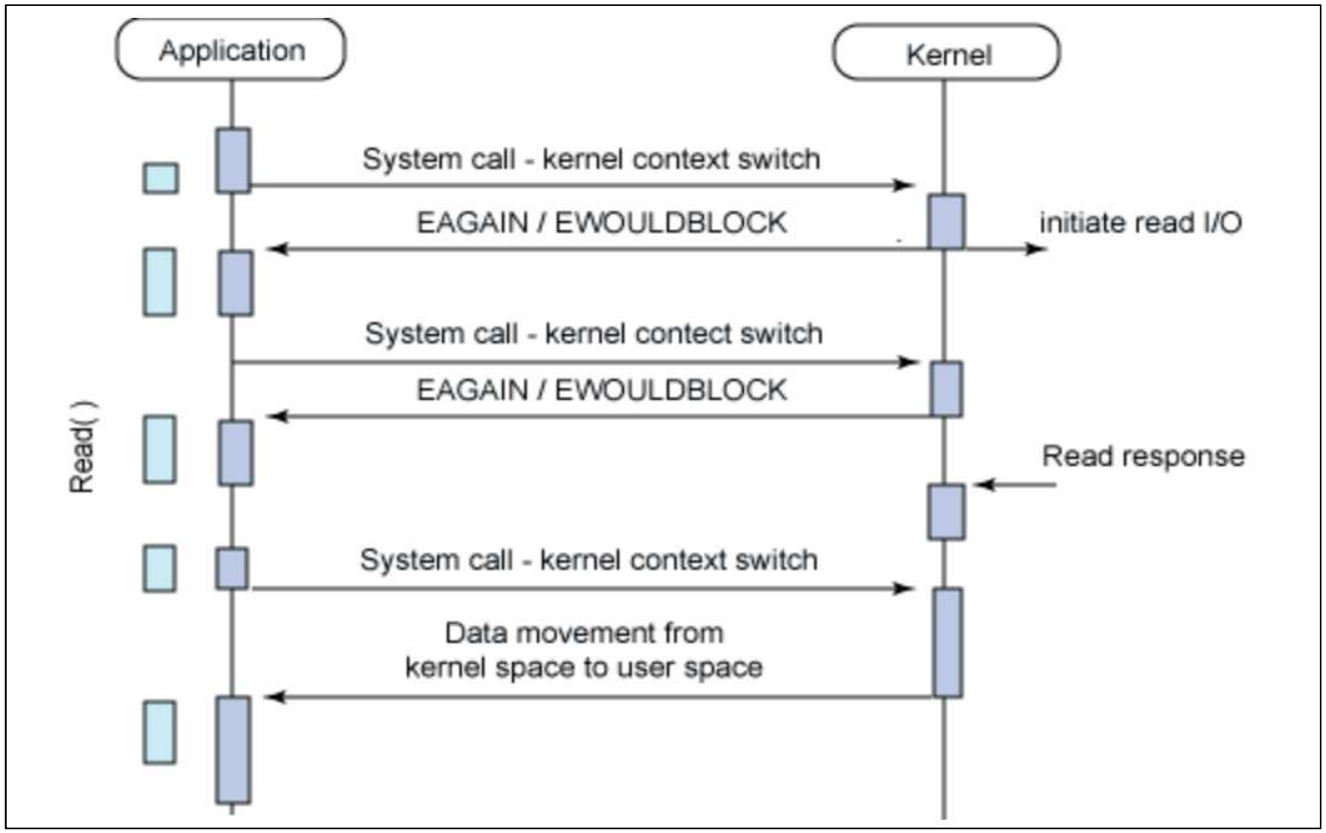
Синхронный блокирующий ввод/вывод

Приложение блокируется до тех пор, пока операция ввода/вывода не будет завершена.



Синхронный неблокирующий ввод/вывод

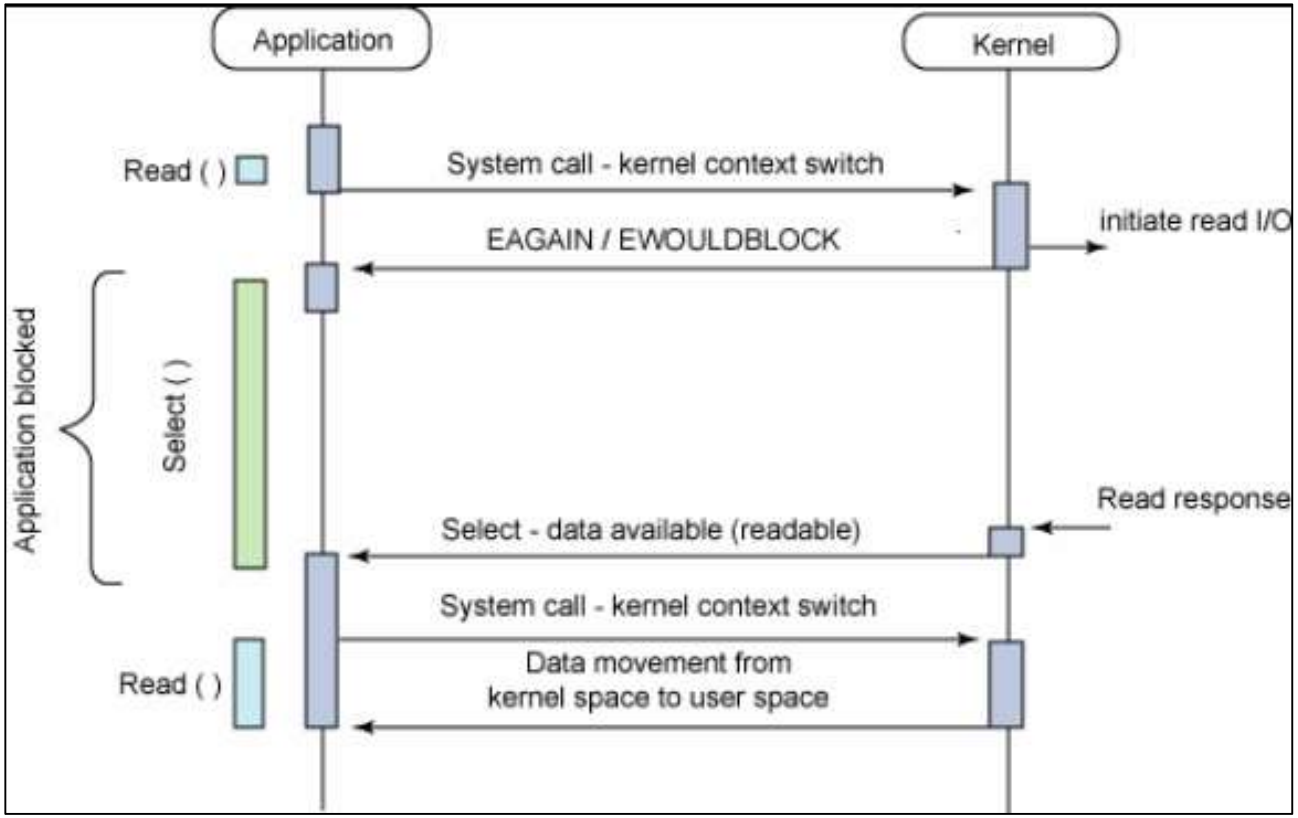
Приложение может сделать что-то еще вместо ожидания ввода/вывода, однако происходит увеличение количества переключений контекста между ядром и пользовательским пространством.



Асинхронный блокирующий ввод/вывод

Ввод-вывод инициируется как неблокирующий, но уведомление принимается в режиме блокировки с помощью системного вызова select ().

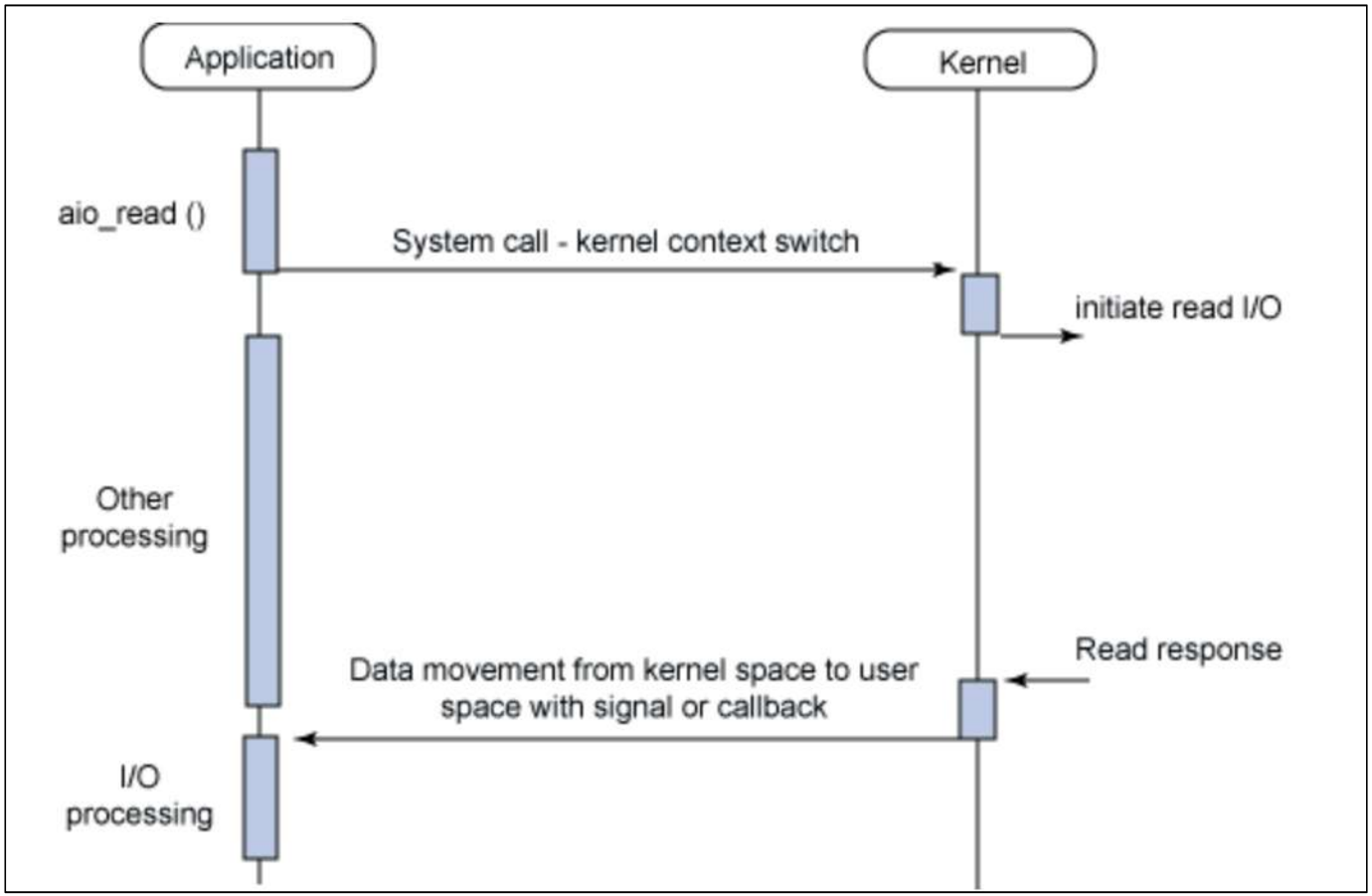
Как только файловый дескриптор готов к работе, вызов read() разблокируется, и приложение сможет извлечь и обработать данные.



Асинхронный неблокирующий ввод/вывод

Приложение может выполнять вычисления, пока чтение выполняется в фоне ядром.

При готовности ответа генерируется сигнал или обратный вызов для завершения ввода-вывода.



AJAX
Asynchronous JavaScript and XML

=

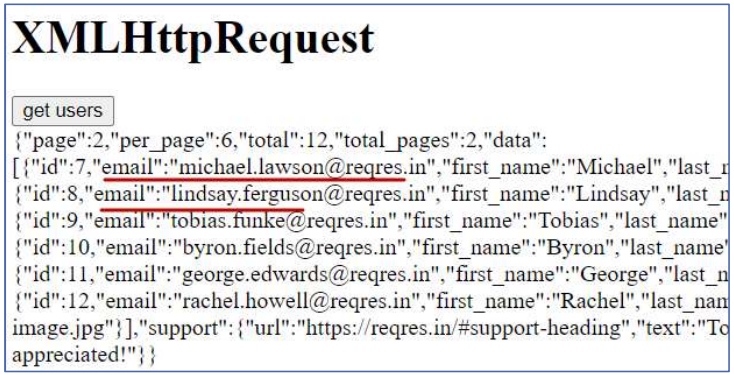
методология (подход) построения динамических приложений, в которых не осуществляется полная перезагрузка HTML-страниц.

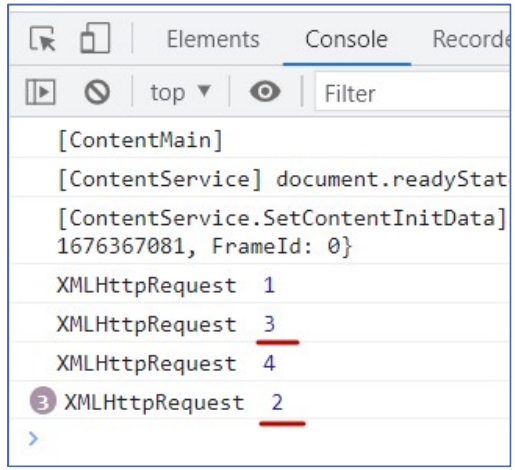
Пример асинхронных запросов

```html
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <title>Async requests</title>
</head>

<body>
    <h1>XMLHttpRequest</h1>
    <input type="submit" value="get users" onclick="getUsers()">
    <div id="divResult1"></div>

    <h1>Fetch</h1>
    <h2>Login</h2>
    <form id="myForm">
        Email: <br>
        <input type="text" name="email" required>
        <br> <br>
        Password: <br>
        <input type="password" name="password" required>
        <br> <br>
        <input type="submit" value="Login"></button>
    </form>
    <div id="divResult2"></div>

    <h1>JQuery</h1>
    <input type="submit" value="get user by ID" onclick="getUser()">
    <div id="divResult3"></div>

    <script>
      ...
    </script>
</body>

</html>
```

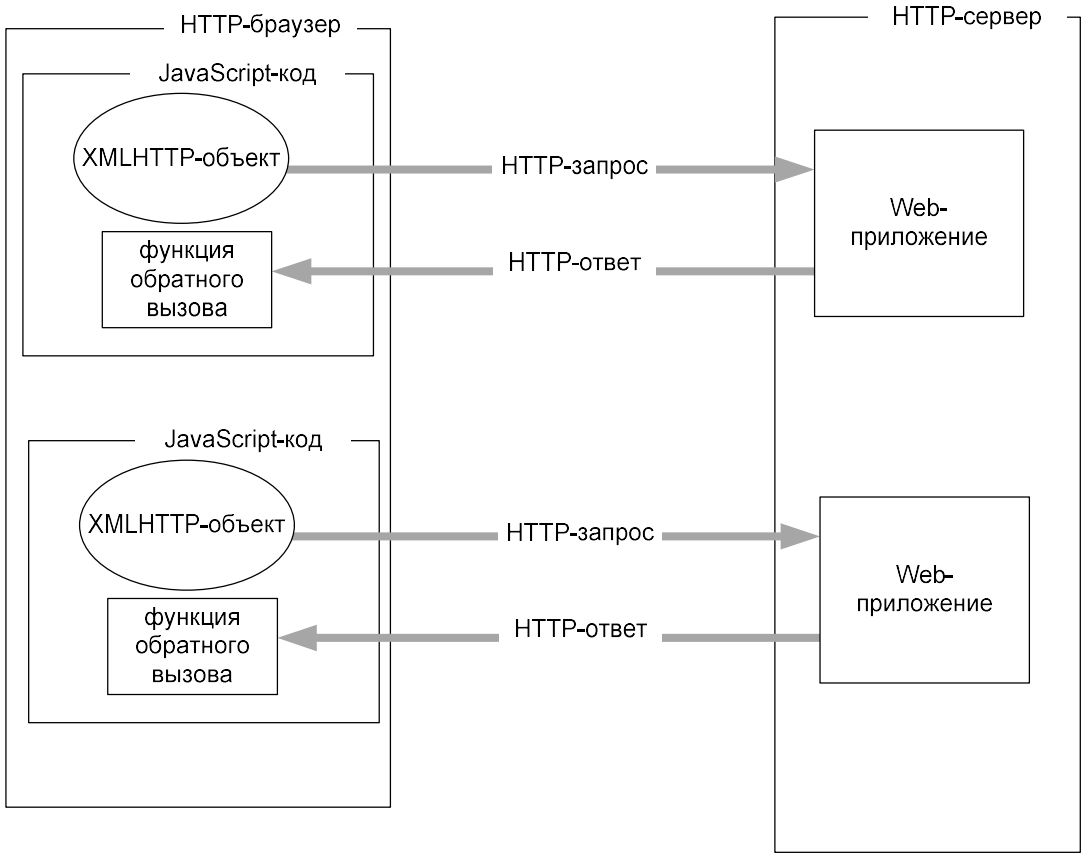
XMLHttpRequest

```
<script>
        function getUsers() {
            const xhr = new XMLHttpRequest();

            xhr.open("GET", "https://reqres.in/api/users?page=2", true);

            console.log("XMLHttpRequest ", 1);

            xhr.onreadystatechange = () => {
                console.log("XMLHttpRequest ", 2);
                if (xhr.readyState == 4) {
                    if (xhr.status = 200)
                        document.getElementById("divResult1").innerText = xhr.responseText;
                }

            };
            xhr.onerror = (e) => {
                console.log("XMLHttpRequest.onerror ", e)
            };

            console.log("XMLHttpRequest ", 3);

            xhr.send();

            console.log("XMLHttpRequest ", 4);
        };

        //...

    </script>
```





Асинхронный XMLHttpRequest запрос



fetch

```
<script>
    //...
    document.querySelector("#myForm").addEventListener("submit", (event) => {
            event.preventDefault();

            let formData = new FormData(event.target);
            let data = Object.fromEntries(formData);

            fetch("https://reqres.in/api/register", {
                method: "POST",
                mode: "cors",
                headers: {
                    "Accept": "application/json",
                    "Content-Type": "application/json"
                },
                body: JSON.stringify(data)
            })
                .then(resp => resp.text())
                .then(respText => document.getElementById("divResult2").innerText = respText)
                .catch(err => console.log(err))
        });

</script>
```

JQuery

```
<script>
        //...
        function getUser() {
            $.ajax({
                url: "https://reqres.in/api/users/4",
                type: "GET",
                dataType: "json",
                success: function (data) {
                    document.getElementById("divResult3").innerText = JSON.stringify(data.data);
                    console.log(data.data)
                },
                error: function (e) {
                    console.log(e);
                },
            });
        }

</script>
```