

# Лекции по Операционным системам

Сверстал: Кузякин Никита Александрович

По лекциям ИТМО

Плейлист с лекциями — [тут](#)

# СОДЕРЖАНИЕ

## 1 Архитектура компьютерных систем

Первоначальными двумя архитектурами компьютерных систем являются Гарвардская и Неймановская архитектуры.

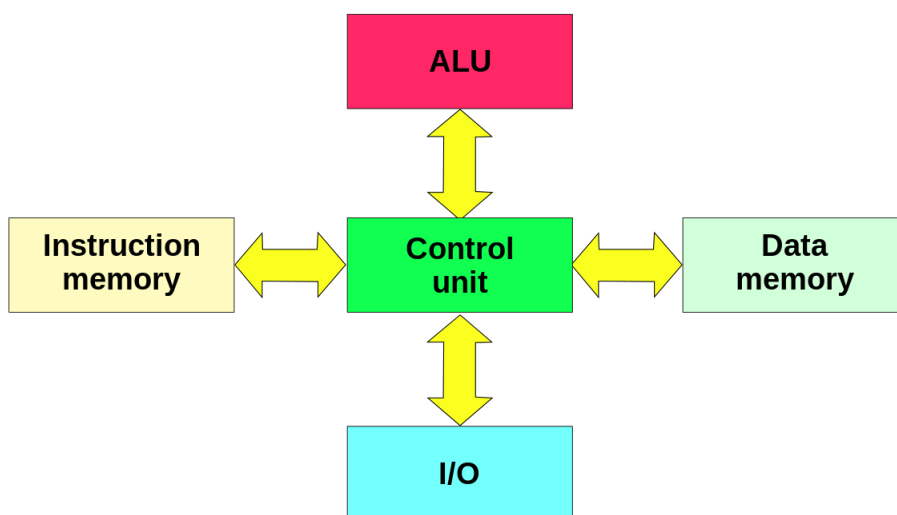
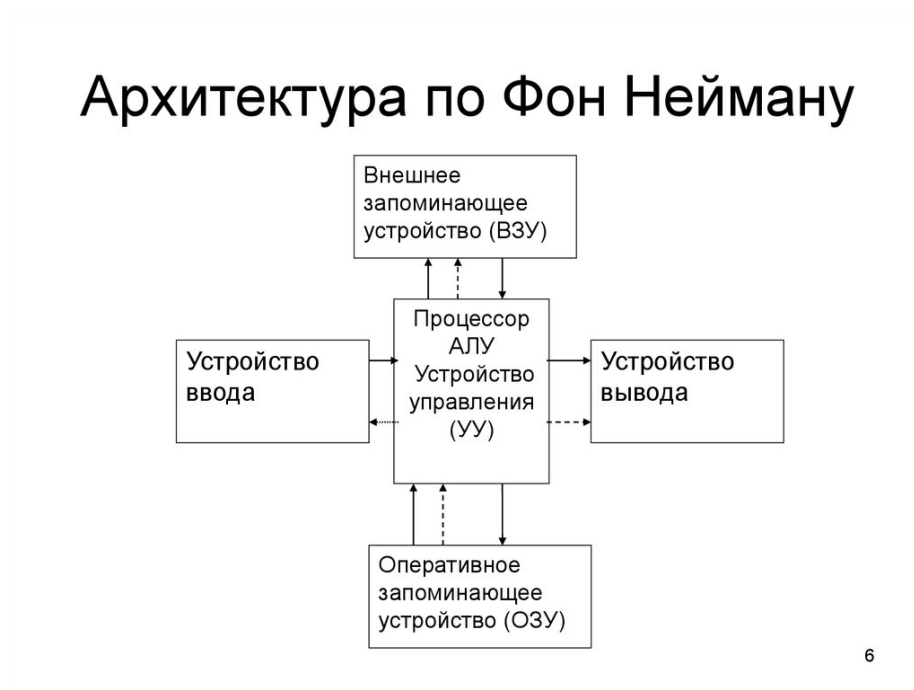


Рисунок 1 – Гарвардская архитектура ЭВМ

Любая вычислительная машины состоит из управляющего устройства (организует вычисления) и арифметико - логического устройства (производит вычисление арифметических операций), а также различных видов памяти.

В архитектуре фон Неймана предполагается, что есть единое управляющее устройство, память при этом общая (и данная, и программа в одно блоке).

### Принципы архитектуры фон Неймана:

- Принцип однородности памяти — команды и данные хранятся в одной и той же памяти (внешне неразличимы).
- Принцип адресности — память состоит из пронумерованных ячеек, процессору доступна любая ячейка.
- Принцип программного управления — вычисления представлены в виде программы, состоящей из последовательности команд.
- Принцип двоичного кодирования — вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1.

## UMA / NUMA

В архитектуре **UUMA** подразумевается, что все устройства являются одноранговыми. Те у любого устройства в системе равные права на доступ к памяти и системные характеристики обращения к ней.

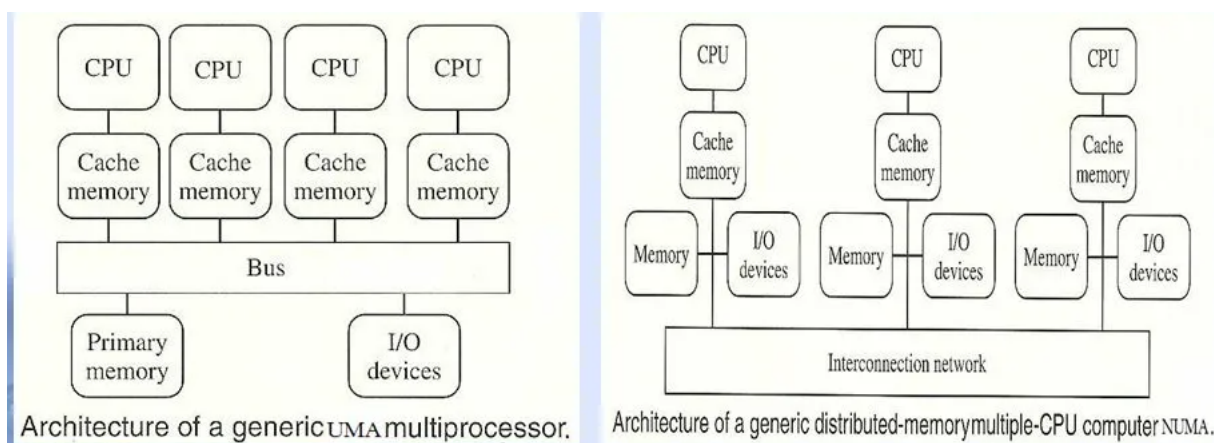


Рисунок 2 – Гарвардская архитектура ЭВМ

Минусом данной архитектуры является, то что тяжело организовать доступ к памяти для большого числа процессоров.

В архитектуре **NUMA** у нас есть память, которая находится ближе к какому-то процессору и память, которая доступна через коммутатор (передает данные через порты).

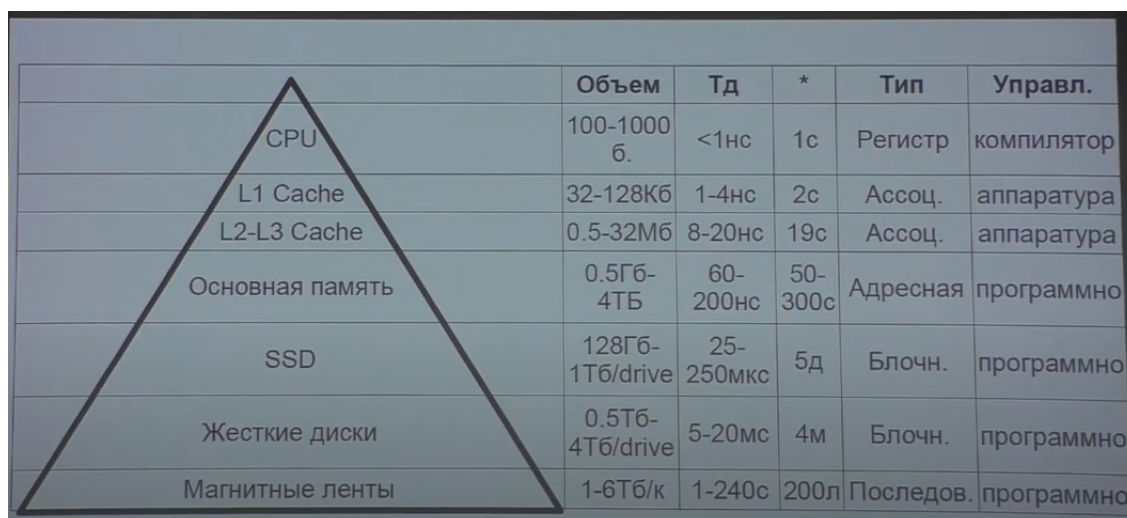
Адресное пространство для данной архитектуры является общим.

Огромным плюсом является, что можно заменять ее части прямо во время работы, что сильно повышает надежность системы.



Важно помнить, что процессор выполняет команды последовательно. Пока один компонент выполняет одно действие, другой выполняет другое (они не останавливаются пока одни данные пройдут от начала до конца).

*Определение 1. Виртуальная память* — это подход к управлению памятью компьютером, который скрывает физическую память (в различных формах, таких как: оперативная память, ПЗУ или жесткие диски) за единым интерфейсом, позволяя создавать программы, которые работают с ними как с единым непрерывным массивом памяти с произвольным доступом.



The diagram shows a pyramid representing the memory hierarchy, with levels from top to bottom: CPU, L1 Cache, L2-L3 Cache, Основная память (Main memory), SSD, Жесткие диски (Hard drives), and Магнитные ленты (Magnetic tapes). To the right of the pyramid is a table with 6 columns: Объем (Volume), Тд (Access time), \*, Тип (Type), and Управл. (Management).

	Объем	Тд	*	Тип	Управл.
CPU	100-1000 б.	<1нс	1с	Регистр	компилятор
L1 Cache	32-128Кб	1-4нс	2с	Ассоц.	аппаратура
L2-L3 Cache	0.5-32Мб	8-20нс	19с	Ассоц.	аппаратура
Основная память	0.5Гб-4ТБ	60-200нс	50-300с	Адресная	программно
SSD	128Гб-1Тб/drive	25-250мкс	5д	Блочн.	программно
Жесткие диски	0.5Тб-4Тб/drive	5-20мс	4м	Блочн.	программно
Магнитные ленты	1-6Тб/к	1-240с	200л	Последов.	программно

Управляется компилятором — означает, что именно компилятор определяет, как именно ваша программа будет взаимодействовать с данным блоком памяти, те что в какие регистры запишется и тд.

## **3 Общие сведения об операционных системах**

### **3.1 Функции OS**

- Разработка программ.
- Выполнение программ.
- Доступ к устройствам ввода / вывода.
- Контролируемый доступ к файлам.
- Доступ к системе и системным ресурсам.
- Обнаружение и обработка ошибок.
- Учет пользования и диспетчеризация ресурсов.
- Предоставление ключевых интерфейсов (ISA — набор команд, ABI — бинарный интерфейс приложения, API — интерфейс прикладных программ).

### **3.2 Оператор ЭВМ**

Что должен делать оператор?

1. получить программу с данными от программиста.
2. подготовить программу к загрузке.
3. загрузить программу и компилятор.
4. запустить программу на вычисление.
5. распечатку с результатом передать программисту.

Минусами оператора ЭВМ является: наличие расписания машинного времени и долгое время подготовки к работе.

### **3.3 Пакетная обработка**

В следствие минусов оператора ЭВМ появилась пакетная обработка.

Появился первый Системный монитор, который включал в себя: обработчик прерываний, драйверы устройств, планировщик заданий, интерпретатор командного языка и было отведено пространство под пользовательские программы и данные.

### **3.4 Многозадачность**

Одним из главных минусов первых ЭВМ было то, что вовремя вывода, ввода или работы других устройств процессор простаивал. В следствие этого появилась концепция многозадачности.



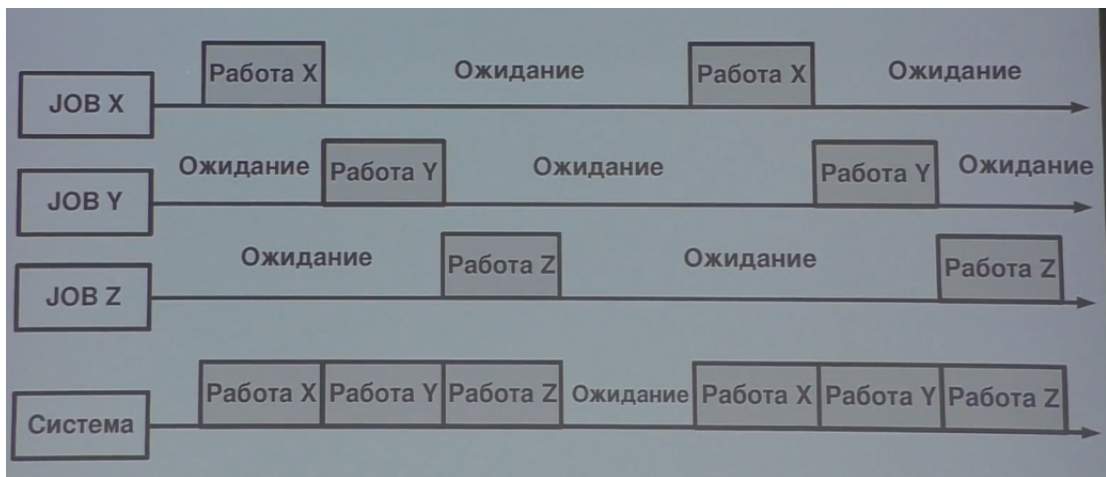


Рисунок 3 – Схема многозадачности первых ЭВМ

### 3.5 Разделение времени

Следующим нововведением в ЭВМ стало исключение оператора и добавление пользователей. Каждому пользователю выдавалось часть времени процессора с использованием квантового времени. В следствие этого появились проблемы разделения ресурсов и защита одних программ от других.

## 4 Основные задачи OS

### 4.1 Управление процессами

*Определение 2. Процесс* (с точки зрения обывателя) — экземпляр программы во время ее исполнения.

*Определение 3. Процесс* (с точки зрения OS) — единица потребления ресурсов OS, в которой существует последовательность действий, текущее состояние и набор связанных ресурсов.

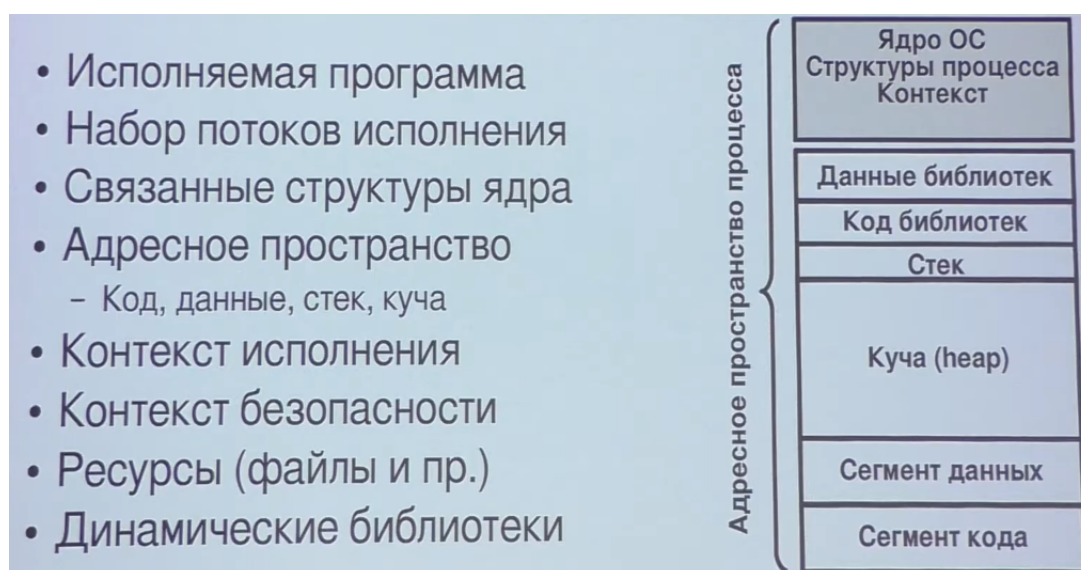


Рисунок 4 – Структура процесса

Для того, чтобы создать процесс, необходимо создать все части адресного пространства представленного на рисунке 4.

Процесс создается не так быстро, поэтому для вычислений на процессоре можно просто создать поток (по сути он будет представлять набор регистров) и с помощью него провести вычисления. Это все и является контекстом.

Когда создается процесс, ядро OS должно построить для ресурсов, которое он будет потреблять систему (описание ресурсов) (в линуксе task structure).

Проблемы современных процессов:

- Защита памяти процессов — недетерминированное поведение процесса, к примеру обращение не к своей памяти, может нарушить другие процессы.
- Взаимные блокировки — есть два процесса, один из них захватил один ресурс, другой другой, и они пытаются также добавить к себе захваченный другим процессом ресурс. (deadlock, livelock, starvation)

- Проблема синхронизации — тк у нас может быть несколько процессов, а адресное пространство для них одно.
- Взаимное исключение доступа ресурсов.

## 4.2 Виртуальная память

Для решения проблемы с единым адресным пространством была придумана виртуальная память.

Управление памятью:

- Изоляция процессов.
- Управление выделением и освобождением памяти (аллокаторы и мепинг памяти).
- Поддержка модулей (модульности) — динамическая загрузка и выгрузка модулей.
- Защита и контроль доступа — права на сегменты памяти.
- Долговременное хранение — запись информации на диск.
- Страничный обмен.

*Определение 4. Виртуальная память* — отдельное виртуальное адресное пространство для каждого процесса и ядра.

Также виртуальная память подразумевает, что некоторые страницы нельзя выгружать из памяти, к примеру если они используются в большом количестве процессов.

## 4.3 Безопасность

Также важный аспект OS это то, на сколько она безопасна, на сколько она обеспечивает безопасность данных.

Самым важным аспектом безопасности является протокол работы с информацией.

Что должна обеспечивать OS:

1. Безопасность доступа к системе — защита от несанкционированного доступа.
2. Конфиденциальность — невозможность неавторизованного доступа к данным.
3. Целостность данных — защита данных от неавторизованного и нецелостного изменения.
4. Аутентификация и авторизация.

#### **4.4 Диспетчеризация и планирование ресурсов**

Что важно учесть при планирование ресурсов (с точки зрения OS):

- Равноправие — пользователи, процессы и тд должны получать ресурсы равноправно. (Интересный факт: в UNIX приоритет процесса развернутого окна на 15 пунктов выше других).
- Дифференциация отклика — в некоторых задачах нужно понизить время отклика, к примеру в задачах выполняющихся в реальном времени.
- Общесистемная эффективность.
- Планировщик процессов, дисков и тд.

## 5 Современные архитектурные концепции OS

### 5.1 Архитектура ядер

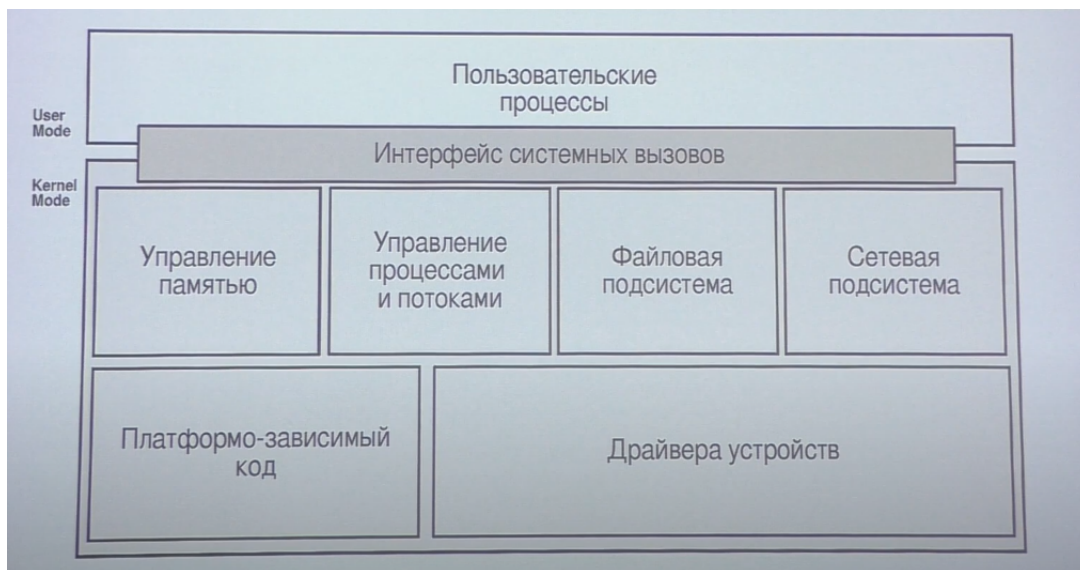


Рисунок 5 – Схема ядра OS

Управление памятью, процессами и потоками, файловая подсистема и сетевая подсистема работают на основе драйверов и платформено-зависимого кода.

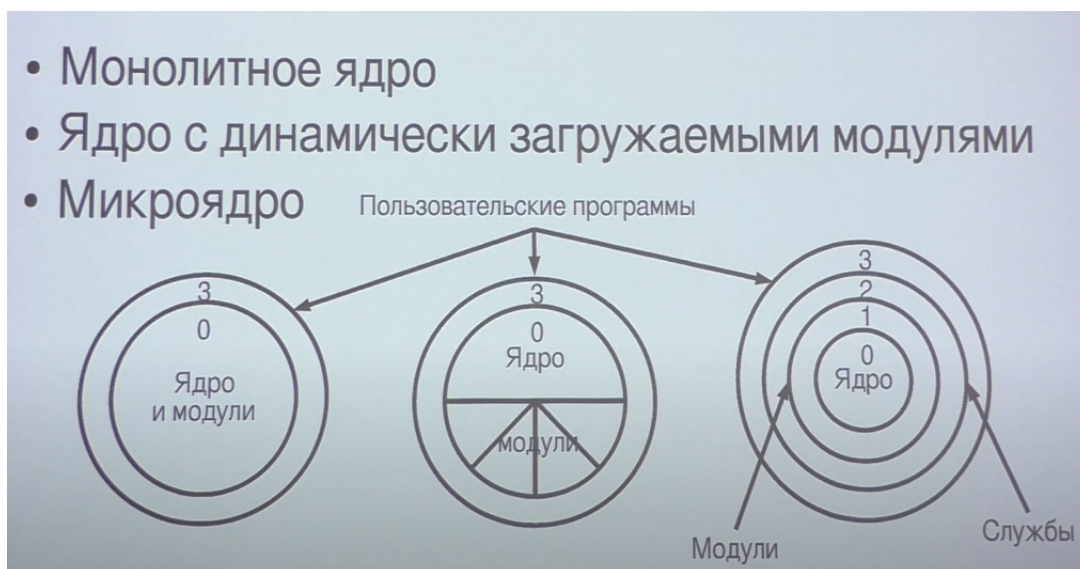


Рисунок 6 – Виды архитектур ядер OS

Монолитное ядро — подразумевает, что для изменение чего-то в ядре придется перекомпилировать OS. Подходит для систем где набор устройств

определен и не будет изменяться. 0 уровень — ядро и встроенные в него модули, 3 уровень — пользовательские программы. 1 и 2 не используются.

Ядро с динамически загружаемыми модулями имеет возможность загрузить модули во время выполнения операционной системы.

Микроядро — концепция, в которой само ядро занимается базовыми задачами: диспетчеризация процессов и выделение памяти. 1 и 2 уровни занимают остальные задачи, реализованные в виде сервисов. Пользовательские приложения работают на 3 уровне. Из-за частого переключения контекстов это работает очень медленно.

## 5.2 Многопоточность

Из-за сложности создания процесса была придумана концепция реализации внутри процесса потоков. Tread — нить / поток.

Библиотека порождающая потоки на UNIX системах Posix Treads.

Существует множество концепций реализации потоков, они будут рассмотрены в следующих параграфах.

## 5.3 SMP и ASMP

**Symmetric multiprocessing** — процессы равны, процесс выполняется на нескольких процессорах одновременно. Это дает следующие плюсы: простота разработки и производительность, более высокая надежность (при отказе выполнить процесс, его могут выполнить другие), масштабируемость приложений, динамическое добавление ресурсов процессора.

**Asymmetric multiprocessing** — в системе с асимметричной многопроцессорностью не все процессоры играют одинаковую роль. Например, система может использовать (либо на аппаратном, либо на уровне операционной системы) только один процессор для выполнения кода операционной системы, или поручать только одному процессору выполнение операций ввода-вывода. В других АМР-системах все процессоры могут выполнять код операционной системы и операции ввода-вывода, так что с этой стороны они ведут себя как симметричная многопроцессорная система, но определенная периферийная аппаратура может быть подсоединена только к одному процессору, так что со стороны работы с этой аппаратурой система предстаёт асимметричной. Более дешевая альтернатива в системах, которые поддерживали SMP.

**Многопоточность ! = Многопроцессорность**

## 5.4 Виртуализация

**Виртуальные машины (интерпретаторы)** — по сути программы, которые работают под выполнением другой программы. Как примеры: JS в браузере, python, JAVA VM. Это позволяет поднять уровень абстракции.

*Определение 5. Интерпретация* — посстрочный анализ, обработка и выполнение исходного кода программы или запроса, в отличие от компиляции, где весь текст программы, перед запуском анализируется и транслируется в машинный или байт-код без её выполнения.

**Контейнеры приложений** — позволяет писать приложения один раз и запускать их где угодно. Разработчики могут создавать и развертывать приложения быстрее и безопаснее, чем при традиционном подходе к написанию кода — когда он разрабатывается в определенной вычислительной среде, а его перенос в новое место, например из тестовой среды в продуктивную, часто приводит к ошибкам выполнения кода.

*Определение 6. Контейнер приложения* — экземпляр исполняемого программного обеспечения (ПО), который объединяет двоичный код приложения вместе со всеми связанными файлами конфигурации, библиотеками, зависимостями и средой выполнения.

Смысл и главное преимущество технологии в том, что контейнер абстрагирует приложение от операционной системы хоста, то есть остается автономным, благодаря чему становится легко переносимым — способным работать на любой платформе.

Примеры: Docker, Solaris containers, Linux containers.

**Аппаратурная виртуализация** — виртуализация с поддержкой специальной процессорной архитектуры. В отличие от программной виртуализации с помощью данной техники возможно использование изолированных "гостевых" операционных систем.

Примеры: Virtual BOX, KVM.

**Облачные технологии** — по сути облачная виртуализация, главным плюсом является, что в случае сбоя одной физической системы, данные иммигрируют на другую систему и продолжают выполняться. Данные технологии построены на базе аппаратурной виртуализации.

## **6 Полезные утилиты**

Google Test Framework — <https://google.github.io/googletest/primer.html>