1. Основы (как работает интернет):

Как работает Интернет?

Центр обработки данных (например, гугловый) м. б. в тысячах км от хоста – на нем хранятся данные.
Сеть волоконно-оптических кабелей охватывает весь мир и соединяют центр обработки данных и конкретный хост.
Мобильная сеть или маршрутизатор вай-фай – посредники между хостом и оптоволоконной сетью.
Конкретный файл, который необходимо передать хранится в центре обработки данных, а точнее на твердотельном накопителе (SSD), который выполняет функции внутренней памяти сервера.
Каждое устройство, подключенное к сети, имеет IP-адрес (его присваивает устройству интернет-провайдер). Кстати, сервер в ЦОД также имеет IP-адрес (если его узнать, сайты, которые на нем хранятся станут доступны).
IP-адресу соответствует доменное имя для удобства запоминания пользователем.
Сервер хранит несколько сайтов одновременно. однако они не могут быть привязаны к одному IP-адресу. Для разрешения этого неудобства используются заголовки хоста.
Чтобы серверу легче было соотнести IP-адрес с его доменным именем существует DNS-сервер.
DNS-сервером могут управлять интернет провайдеры и др. компании.
Сайт состоит из файлов, которые хранятся на сервере.
Как проходит поиск сайта:

1. Вводим в адресной строке домен
2. Браузер отправляет запрос на DNS-сервер
3. DNS-сервер ищет соответствующий IP-адрес и отдает его браузеру
4. Браузер перенаправляет запрос с IP-адресом в ЦОД (т.е. к серверу, на котором хранится сайт)
5. Сервер получает запрос на доступ к сайту, и начинается поток данных. Они передаются в цифровом формате (последовательности 0 и 1, которые разделены на пакеты, каждый по 6 битов, в каждом из которых также есть порядковый номер и IP-адрес хоста) через оптоволоконный кабель в виде световых импульсов.
6. Маршрутизатор (модем, например) преобразует световые сигналы оптоволоконного кабеля в электрические. А для передачи эл. сигналов на хост. ноутбук, в частности, используется кабель Ethernet, а с мобильной связью сигналы из оптоволокна направляются на вышку сотовой связи, с которой сигнал в виде электромагнитных волн передается на смартфон.
7. Браузер (он же клиент) получает данные и отрисовывает веб-страницу

Организация ICANN занимается регистрацией доменных имен, контролем IP-адресов и т.д.
Для скорой передачи данных каждый пакет выбирает кратчайший доступный маршрут, а там собираются в соответствии с порядковыми номерами, если часть информации не достигла хоста, с него оправляется повторный запрос на отправку данных.
Для управления потоками данных существуют протоколы (http/https – веб доступ, TCP/IP – передача данных, RTP – медиа, т.е. онлайн-видео, онлайн стримы, IP-телефония). Для разных приложений протоколы различны (исходя из нужд).
Frontend - клиентская часть сайта (та, которую мы видим, браузер, для ее создания используются CSS – каскад стилей для HTML, HTML - разметка, JavaScript). Backend – часть, связанная с сервером БД (общение происходит посредством серверных языков программирования – JavaScript, Python, Ruby, PHP, SQL)
MAC-адрес (media access control) – правила именования устройств на канальном уровне (для устройств: сетевых плат и т.д.). Называет их компания-изготовитель.
К IP-адресу добавили битовую маску. Есть биты, логические операции (пример & - если и то и то True, т.е. 1). IP-адрес состоит из 4 байт (IPv4), каждый из 8 бит. После применения операции – другое число, чтобы определить, какие биты исп. для нумерации сети и узлов внутри сети. Пр.: число до – к сети, после – к узлу.

У узла есть ИП и фикс. маска подсети. Для него внутренние адреса – номер сети ИП совпадает – узлы локальной сети, непосредственно доступны. Внешние узлы – через шлюз/маршрутизатор (у него один адрес в одной сети, а другой – в другой).
Таблица маршрутизации: номер сети назначения (IP через маску), IP-адрес шлюза, метрика.

Как работает ОС

Процессор:

- Читает инфу из памяти
- Выполняет операцию по инструкции
- Кладет в память результат

BIOS – basic input-output system - устройство ввода/вывода - первый интерфейс. Т.е. у компа теперь 2 режима: устройство вычислений и устройство ввода/вывода (делает удобней).

ОС отделяет приложения друг от друга и от себя, чб при поломке одного остальные не страдали.

Позволяет получить абстракции, облегчающие работу.

Абстракции:

- процессы и потоки
- файлы и файловые системы
- адресное пространство и память
- сокеты, протоколы, устройства и т.д.

Интерфейс системных вызовов

Kernal Space (OS) (– вызовы выполняются внутри ядра) и User Space (- а вызываются снаружи) – разграничивает процессор.

BASH – интерпретатор командной строки.

Ядро:

- обрабатывает запросы приложений
- обрабатывает запросы оборудования
- обеспечивает диспетчеризацию процессов (scheduling)
- обрабатывает исключительные ситуации

Сервисы, предоставляемые ОС:

- управление процессами
- управление памятью
- файлы, их содержимое, каталоги и директории
- модель безопасности
- межпроцессное взаимодействие
- прочее: терминалы, сети, таймеры, периферия

Драйвер – превращение интерфейса программы в общий интерфейс

Процесс – абстракция, которая предоставляет иллюзию ПК (как будто все ресурсы направлены на пользователя или процесс)

- адресное пространство (страницы, таблицы)
- CPU
- файлы и др. абстракции ОС
- состояние (состояние в памяти и набор регистров внутри процессора)

- стек

Поток – поток исполнения инструкций, процесса или его части

TGID – про поток

PID – про процесс

Вытесняющая многозадачность

Прерывание – ситуация остановки процессором последовательного выполнения программы для выполнения запроса или реакции на событие.

Системный вызов – специальное программное прерывание, соответствующее запросу сервера у ядра.

Исключение – неверное действие программы, приводящее к генерации прерывания.

Системные вызовы и драйверы

System call	Description
fork()	Create process
exit()	Terminate current process
wait()	Wait for a child process to exit
kill(pid)	Terminate process pid
getpid()	Return current process's id
sleep(n)	Sleep for n seconds
exec(filename, *argv)	Load a file and execute it
sbrk(n)	Grow process's memory by n bytes
open(filename, flags)	Open a file; flags indicate read/write
read(fd, buf, n)	Read n byes from an open file into buf
write(fd, buf, n)	Write n bytes to an open file
close(fd)	Release open file fd
dup(fd)	Duplicate fd
pipe(p)	Create a pipe and return fd's in p
chdir(dirname)	Change the current directory
mkdir(dirname)	Create a new directory
mknod(name, major, minor)	Create a device file
fstat(fd)	Return info about an open file
link(f1, f2)	Create another name (f2) for the file f1
unlink(filename)	Remove a file

Многозадачность:

- прерывание таймера
- смена контекста
- план блокировок (при наличии нескольких CPU)
- освобождение ресурсов при завершении процесса

состояние процессора хранится в регистрах

машинные коды – атомарные, их не видно среднестатистическому прогеру

deadlock – когда есть 2 критических процесса, которые происходят в критической секции и задействуют критические ресурсы

Active – состояние, когда процесс выполняется на процессоре

Waiting – состояние, когда процесс вызывает операцию ввода-вывода, которое не мб сразу выполнено

Ready - когда операция завершилась, готова к исполнению, но не исполнится

Физическая память

Виртуальная память (пейджинг, страничная организация памяти)

Процессор:

x86 – архитектура в большинстве компьютеров (32 и 64 бита)

ARM - архитектура в мобильных устройствах

AVR – авто, телевизоры и т.д.

Для каждой архитектуры пишется ПО.

Разрядность – величина, которая определяет размерность машинного слова (макс. кол-во бит, к-ми может оперировать процессор за раз)

Команду процессор считывает из памяти и кладет куда-то. Регистровая память (временные результаты вычислений, быстрая) и кэш память. Максимальный размер регистра = разрядности.

Регистр д.б. кратен ячейке ОЗУ. Байтовая адресация – каждая ячейка = 1 байту и каждый байт имеет свой адрес, по которому процессор может к нему обратиться. Словесная адресация – то же самое, но размер ячейки = машинному слову, и процессор обр-ся к машинному слову.

ASCII-символ = 7 бит

Регистры специального назначения – предназначаются для конкретного содержимого. В сегментных регистрах хранятся адреса памяти; регистры для работы со стеком указывают на каналы фрейма и верхушку стека; флаговые регистры содержат различные биты, которые отражают состояние результата предыдущей операции; указатель команд (instruction pointer) указывает адрес команды, которую нужно выполнить следующей и др.

Регистры общего назначения – могут быть использованы на усмотрение прогера, однако есть опр. соглашения по работе компиляторов. Выступают в роли переменных, параметров, временного хранилища для результатов вычислений.

Язык ассемблера – низкоуровневый язык программирования, состоящий из символического обозначения машинных команд.

Ассемблер – программа-транслятор этого языка в машинный код.

При включении компьютер запускает биос, его задачи: найти первое дисковое устройство, которое было указано, взять оттуда первый сектор, загрузить его в память и передать на него управление (указать процессору на то, чтобы теперь он начал выполнять команды по адресу этой загруженной программы, а именно - ОС). ОС переводит процессор в безопасный режим. ОС выставляет специальный флаг системного регистра, устанавливает разрешения и условия для других программ, распределяет таблицы дескрипторов и прерываний. Т.о. ОС становится полноценным хозяином компа. Далее реализуется работа с памятью, выставляются различные ограничения и т.д.

Когда мы запускаем программу, ОС выделяет под нее место, загружает ее в ОЗУ (например, жесткого диска), после чего передает ей управление, т.е. говорит процессору, с какого места из памяти ему нужно выполнить следующую команду. На этом моменте используется регистр IP. После процессор обновляет значение в этом регистре, чб он указывал на следующую инструкцию в памяти. Т.е. он определяет размер инструкции и прибавляет его к значению в IP.

Режимы работы процессора:

- реальных адресов – 16-битный режим, в который процессор переходит сразу после включения компьютера (адрес, сформированный программами, я-ся реальными не требует преобразований)
- защищенный – 32-битный режим, в который можно перейти только из режима реальных адресов, при нем обеспечивается защита данных, ОС, прикладных программ и данных этих программ друг от друга благодаря разделению приложений на разные уровни привилегий
- 64 разрядный режим – в него можно перейти только из защищенного режима

В оф. док-ии:

- режим реальных адресов + защищенный – legacy mode
- 64 разрядный + режим совместимости (поддержка 32 и 16-разрядного кода) – long mode

Уровни привилегий – доступ к использованию ресурсов процессора (начинаются с защищенного режима).

0 уровень – полный доступ к процессору (на нем ОС)

- 1 уровень – на нем дей-т запреты с 0 уровня
- 2 уровень – запреты с 0 и 1 ур.
- 3 уровень – (пользовательский, на нем прикладные проги) запреты с 0, 1 и 2 ур.

Устройство адресации. Вся оперативная память поделена на сегменты. Их размер зависит от режима, в котором работает процессор. Ячейки представляются в специальном формате: логический адрес = адрес начала сегмента (16 бит) : смещение в сегменте (16 бит).

В режиме реальных адресов адрес делится на сегменты по 64 кб.

Логический адрес преобразуется в 20 битный адрес ячейки памяти: физический адрес = адрес начала сегмента << 4 + смещение сегмента.

Максимальный размер адресов – 1мб физической памяти.

В защищенном режиме память делится на сегменты от 0 до 4 гб. ОС создает иллюзию того, что каждой программе доступно все пространство памяти. Она активирует механизм трансляции виртуальных адресов в физические, т.е. программы начинают работать в виртуальном адресном пространстве, про которое она сами не догадывается. Они продолжают формировать логические адреса, как и раньше, предполагая, что обращаются к реальной памяти.

Логический адрес, созданный программой в этом режиме, преобразуется не в физический, а в виртуальный: логический адрес = селектор дескриптора (16 бит) + смещение в сегменте (32 бита).

Селектор дескриптора – индекс дескриптора: уровень привилегий + таблица + индекс. Таблица и индекс показывают GDT/LDT (начальный адрес сегмента, уровень привилегий, размер сегмента).

Виртуальный адрес (32 бита) = адрес начала сегмента + смещение в сегменте.

Страничная организация памяти – структура, представляющая собой виртуальную память. Все адресное пространство разбивается на страницы, чей размер зависит от режима работы процессора и режима трансляции адресов. Каждая страница описывается структурой, которая объединяется в таблицу страниц, а таблицы страниц объединяются в каталог страниц. Виртуальный адрес = индекс в каталоге страниц + индекс в таблице страниц + смещение в странице.

При помощи механизма трансляции адресов виртуальный адрес преобразуется в физический (32 бита), максимальное число – 4 гб физической памяти.

В 64-битном режиме в основном так же, но есть различия, например, практически полностью отключена сегментация. В физической памяти доступно до 2^{52} байт, в виртуальной – до 2^{48} в виртуальной.

Ограничения обусловлены архитектурой процессора и ОС.

Как внешние устройства взаимодействуют с процессором, если он всегда занят своей работой? Например, нам нужно, чтобы процессор обработал запрос пользователя с клавиатуры прямо во время работы программы и без задержек. Внешнее устройство (клавиатура) через контроллер прерываний передает процессору сигнал о прерывании, чтобы он прервал выполнение текущей программы и передал управление специальной функции – обработчику прерываний. Конечно, сначала сохраняется состояние текущей программы (регистры, адрес сл. инструкции и т.д.). После совершения работы обработчик передает управление программе, восстанавливая ее последнее сохраненное состояние, и она продолжает работу с того места, на котором была прервана.

Прерывания находятся в специальной таблице дескрипторов прерываний, у которых есть уровни привилегий, которые определяет уровень привилегий программы, которая определяет прерывание.

Виды прерываний:

- программные – генерирует сам программист-оператор в ряде ОС (в винде нельзя, есть спец. проги)
- аппаратные – генерируются контроллером прерываний, которые выполняются согласно приоритетам
- исключения – их генерирует сам процессор при попытке программы нарушить ограничения защиты или при возникновении ошибок в ходе ее выполнения

Механизм многозадачности основан на прерывании – одновременная работа многих программ достигается путем попеременного их переключения.

При реализации ОС программы разделяются на процессы, которые разделяются на потоки/задачи (задача – самостоятельная последовательность команд, которая выполняется в своем окружении).

Многопроцессорность. Каждый процессор оснащен несколькими ядрами, а серверы оснащены несколькими процессорами.

Типы многопроцессорных систем:

- на материнской плате несколько сокетов для процессоров
- когда процессор имеет несколько ядер
- когда процессор имеет виртуальные ядра

Frontend

клиентская часть

Пользовательский интерфейс для общения с сервером. UI/UX
Пул компетенций: HTML, CSS, JS, Git, WebPack, Gulp, SASS, LESS, JQuery, React, View, базовые навыки PS, Figma, Avacode и т.д.
Верстальщик – макеты дизайна, HTML, CSS (логика, JS и прочее).

Backend

программно-аппаратная часть

Хостинг

Это услуга, предоставляемая компаниями, заключающаяся в предоставлении определенному домену доступа к серверу, на котором будет запущено ПО, необходимое для обработки запросов к хранимым файлам (вэб-сервер). Как правило, в услугу входит предоставление места для постовой корреспонденции, баз данных, DNS, файлового хранилища на специально выделенном файл-сервере и т.п., а также поддержка функционирования соответствующих серверов.

Один из главных критериев выбора хостинга – операционная система, т.к. от нее зависит ПО, которое будет поддерживать функциональность сервисов.

Прочие критерии:

- поддержка CGI (стандарт интерфейса, используемого внешней программой для связи с веб-сервисом, или по-другому «скрипт»): Python, Ruby, PHP, Perl, ASP, JSP, Java.
- поддержка .htaccess/ .htpasswd – для HTTP-сервиса Apache
- поддержка баз данных, а т.ж. установленные модули и фреймворки для каждой из возможностей

Как услуга хостинг имеет такие критерии, как:

- размер дискового пространства под файлы пользователя (память)
- количество месячного трафика
- количество сайтов, которые можно разместить с одной учетки
- кол-во FTP пользователей
- кол-во e-mail ящиков, пространство под почту
- кол-во баз данных и пространство под них
- кол-во одновременных процессов на пользователя
- кол-во ОЗУ и максимальное время исполнения на каждый процесс пользователя

Качественные показатели:

- свободные ресурсы CPU (центральный процессор), оперативной памяти, которые влияют на быстродействие сервера
- пропускная способность каналов, от которой зависит загрузка информации
- удаленность оборудования хостера от ЦА, которая влияет на скорость загрузки информации

Виды хостинга:

- виртуальный хостинг – сервер с множеством сайтов, владельцы которых имеют одинаковые права и обязанности
- виртуальный выделенный сервер (VPS/VDS) – автономная (выделенная часть дискового пространства на сервере и фиксированные ресурсы. Владелец получает права админа и может сам устанавливать и настраивать программы
- выделенный сервер - полное владение сервером с отдельной ОС, ПО
- colocation – размещение сервера. которым владеет отдельный человек или компания, в дата-центре хостинговой компании.

Сервер (аппаратное обеспечение)

Это выделенный или специализированный компьютер (ПК или рабочая станция) для выполнения сервисного ПО (в т.ч. серверов тех или иных задач).

Это компьютер, выполняющий серверные задачи, или компьютер (или иное аппаратное обеспечение), специализированный (по форм-фактору и/или ресурсам) для использования в качестве аппаратной базы для серверов услуг (иногда — услуг определённого направления), разделяя ресурсы компьютера с программами, запускаемыми пользователем. Такой режим работы называется «невыделенным», в отличие от «выделенного» (англ. dedicated), когда компьютер выполняет только сервисные функции. Строго говоря, на рабочей станции (для примера, под управлением Windows XP) и без того всегда работает несколько серверов — сервер удалённого доступа (терминальный сервер), сервер удалённого доступа к файловой системе и системе печати и прочие удалённые и внутренние серверы.

Сервер (ПО)

Это программный компонент вычислительной системы, выполняющий сервисные функции по запросу клиента, предоставляя ему доступ к определенным ресурсам и услугам.

Для взаимодействия с клиентом (-ами) сервер выделяет необходимые ресурсы межпроцессного взаимодействия и ожидает запросы на открытие соединения. В зависимости от процесса сервер может обслуживать запросы одной системы или на других машинах через каналы передачи данных (пр.: COM-порт) или сетевые соединения.

Формат запросов клиента и ответов сервера определяется протоколом. Спецификации открытых протоколов описываются открытыми стандартами (пр.: документы RFC определяют протоколы Интернета).

Классификация стандартных серверов по типу услуг:

- Универсальные – не выполняют услуг самостоятельно, они предоставляют серверам услуг упрощенный интерфейс к ресурсам межпроцессного взаимодействия и/или унифицированный доступ клиентов к услугам.
    - inetd (internet super-server daemon – демон сервисов IP) – стандартное средство UNIX-систем – программа, позволяющая писать серверы TCP/IP (и сетевых протоколов других семейств), работающие с клиентом через перенаправленные inetd потоки стандартного ввода и вывода (stdin и stdout).
    - RPC (Remote Procedure Call – удаленный вызов процедур) – система интеграции серверов в виде процедур, доступных для вызова удаленным пользователем через унифицированный интерфейс. Сейчас в большинстве UNIX-систем и Windows используется унифицированный интерфейс от Sun Microsystems. изобретенный для их ОС (SunOS, Solaris, Unix-система).
    - Прикладные клиент-серверные технологии Windows:
        - (D-)COM – (Distributed) Component Object Model – модель составных объектов – позволяет одним программам совершать операции над объектами данных, используя процедуры других программ. Изначально предназначена для внедрения и связывания объектов (OLE – Object Linking and Embedding), но в общем позволяет писать широкий спектр различных прикладных серверов. COM – работает в пределах одного компьютера, DCOM – доступна удаленно через RPC.
        - Active-X – расширение COM и DCOM для создания мультимедийных приложений.

Универсальные серверы часто используются для написания информационных серверов – тех, что не нуждаются в специфической работе с сетью и не имеющих никаких задач, кроме обслуживания клиентов (пр.: обычные консольные программы и скрипты могут выступать в роли серверов для inetd).

Большинство внутренних и сетевых специфических серверов Windows работают через универсальные серверы (RPC, (D-)Com).

- Маршрутизация – не совсем сервер, скорее базовая функция поддержки сети операционной системой.

  Для TCP/IP маршрутизация является базовой функцией стека IP (кода поддержки TCP/IP) Каждая система в сети выполняет маршрутизацию своих пакетов к месту назначения. Маршрутизаторы (роутеры или шлюзы) выполняют маршрутизацию чужих пакетов (форвардинг), управляются через таблицы маршрутов и правила, их задачи:

    - принять пакет
    - найти машину, которой предназначался пакет, или следующий в цепи маршрутизатор
    - передать пакет или вернуть ICMP-сообщение о невозможности доставки пакета по причинам:
        - назначение недостижимо (destination unreachable) – у пакета кончилось «время жизни» прежде, чем он был доставлен
        - хост недостижим (host un.) – компьютер или сл. маршрутизатор выключен или не существует

        - сеть недостижима (network un.) – маршрутизатор не имеет маршрута в сеть назначения.
    - если пакет не может быть доставлен по причине чрезмерной загрузки маршрутизатора или сети – отбросить пакет без уведомлений.
- Динамическая маршрутизация – имеет своей задачей сбор информации о текущем состоянии сложной сети и поддержание таблицы маршрутов через эту сеть, чтобы обеспечить доставку пакета по кратчайшему и самом эффективному маршруту.

  Из решений динамической маршрутизации клиент-серверную модель использует только BGP (Border Gateway Protocol – протокол пограничного шлюза), применяемый для глобальной маршрутизации.

  Локальные решения (RIP (протокол маршрутной информации – простейший протокол, позволяющий маршрутизаторам динамически обновлять маршрутную информацию по данным от соседних маршрутизаторов небольшой сети), OSPF (ПДМ. позволяющий отслеживать состояние канала и использующий для нахождения кратчайшего пути алгоритм Дейкстры)) используют в своей работе бродкастовые (широковещательный канал - поток данных предназначен для всех участников сети) и мультикастовые (мультивещание, много адресное – адрес сетевого назначения – мультикастная группа) рассылки.

- Сетевые службы – обеспечивают функционирование сети (пр.: серверы DHCP (прикладной протокол, позволяющий сетевым устройствам автоматически получать IP-адрес и др. параметры, необходимые для работы в сети TCP/IP) и BOOTP (прикладной протокол, позволяющий клиенту автоматически получать IP-адрес, обычно при загрузке компьютера) обеспечивают стартовую инициализацию серверов и рабочих станций, а DNS – трансляцию имен в адреса и наоборот).

Серверы туннелирования (пр.: VPN) и прокси-серверы обеспечивают связь с сетью, недоступной роутингом.

Серверы AAA и Radius обеспечивают в сети единую аутентификацию, авторизацию и ведение логов доступа.

- Информационные службы – к ним относятся как простейшие серверы, сообщающие информацию о хосте (time, daytime, motd) и пользователях (finger, ident (протокол, описывающий способ идентификации пользователя для конкретного соединения TCP)), так и серверы для мониторинга (пр.: SNMP – стандартный интернет-протокол для управления устройствами в IP-сетях на основе архитектур TCP/UDP; поддерживаемые устройства – маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки и др.). Большинство информационных служб работают через универсальные серверы.

Особый вид информ. служб – сервисы синхронизации времени – NTP (сетевой протокол, используемый для синхронизации часов на компьютере с помощью сетей с переменной латентностью). Помимо установки времени на компьютере, NTP периодически опрашивает другие серверы для сверки точности собственного времени, а т.ж. скорости хода часов путем замедления или ускорения.

- Файловые серверы – обеспечивают доступ к файлам на диске сервера. Прежде всего они передают файлы по протоколам:
    - FTP -
    - TFTP – используется в основном при первоначальной загрузке бездисковых рабочих станций, в отличие от FTP не содержит возможности аутентификации и основан на транспортном протоколе UDP.
    - SFTP – протокол прикладного уровня, предназначенный для копирования и выполнения других операций с файлами поверх надежного и безопасного соединения.

- HTTP – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML (текстовые данные), в наст. вр. используется для передачи произвольных данных (веб-страницы, картинки, музыка и т.д.).

Другие серверы позволяют монтировать дисковые разделы сервера в дисковое пространство клиента и полноценно работать с файлами на них. Например, серверы протоколов NFS (протокол сетевого доступа к файловым системам, который позволяет подключать удаленные файловые системы через сеть) и SMB (сетевой протокол прикладного уровня для удаленного доступа к файлам, принтерам и др. сетевым ресурсам, а т.ж. межпроцессного взаимодействия), которые работаю через интерфейс RPC.

Недостатки файл-серверной системы:

- большая нагрузка на сеть, высокие требования к пропускной способности (делает практически невозможной одновременную работу большого числа пользователей с большим объемом данных)
- обработка данных осуществляется на компьютере пользователя (требования к уровню их аппаратного обеспечения)
- блокировка данных при работе с ними одним пользователем делает невозможной работу с ними других пользователей
- безопасность, т.к. для работы с файлом придется дать пользователю полный доступ к нему, когда необходима всего часть файла

- Серверы доступа к данным – обслуживают БД и отдают данные по запросу. Один из простейших – LDAP (Lightweight Directory Access Protocol – облегченный протокол доступа к спискам).

  Для БД единого протокола нет, однако ряд БД отъединяет использование единых правил формирования запросов – языка SQL (Structured Query Language – язык структурированных запросов). Остальные БД – NoSQL.

- Медиа серверы – предоставляют сети доступ к мультимедийным источникам, от аудио/видео по запросу до стриминга в аудио/видео в реальном времени.

- VoIP/ IP-телефония – программные коммутаторы (софтсвитчи), IP-АТС (автоматическая телефонная станция на основе межсетевого протокола IP операторского уровня), виртуальные АТС и серверы ВКС (сервер многочастотной конференции для неск. польз.), а также специализированные серверы Интернет-сервисов (пр.: Skype) обеспечивают пользователей возможностью голосовой и видеосвязи в реальном времени посредством компьютерной сети. Кроме потоковой передачи медиа данных, сервер IP-телефонии подобно классической АТС реализует возможность регистрации оконечного терминала, маршрутизацию вызова и корректное установление соединения между пользователями и др.

  В отдельных случаях, в зависимости от реализуемой технологии и административных настроек, VoIP-сервер может обеспечивать только управление — регистрацию пользователя в сети и коммутацию поступающих вызовов, без непосредственного участия в передаче медиа-данных между клиентскими терминалами. В этом случае потоковые данные с полезной нагрузкой передаются напрямую между конечными пользователями (peer-to-peer) и / или некоторыми промежуточными устройствами, приложениями. Известно, что такой вариант прямой связи с управлением через сервер применяется в Skype, Viber, Telegram и WhatsApp. Также, подобный режим нередко применяется в корпоративных IP-АТС.

В качестве клиентских терминалов к VoIP-серверу могут выступать VoIP-телефоны, видеотелефоны, программные телефоны (софтфоны), а также обычные аналоговые телефонные аппараты подключенные через VoIP-шлюз. Сервер IP-телефонии может работать как самостоятельное устройство для обеспечения связи между внутренними пользователями или быть подключенным к какой-либо сторонней сети, в том числе к телефонной сети общего пользования, через Интернет или через сеть оператора телефонной связи.

- Службы обмена сообщениями – эл. почты, работающие по протоколу SMTP. SMTP-сервер принимает сообщение и доставляет его в локальный почтовый ящик пользователя или на другой SMTP-сервер (сервер назначения или промежуточный). На многопользовательских компьютерах пользователи работают с почтой прямо на терминале или веб-интерфейсе. Для работы с почтой на ПК почта забирается из почтового ящика через серверы, работающие по протоколам POP3 и IMAP.

  Для организации конференций используются серверы новостей, работающие по протоколу NNTP.

  Для обмена сообщениями в реальном времени существуют серверы чатов по многочисленным протоколам (пр.: IRC, Jabber (XMPP), OSCAR).

- Серверы удаленного доступа – через соответствующую клиентскую программу обеспечивают пользователя аналогом локального терминала (текстового или графического) для работы на удаленной системе.

  Для обеспечения доступа к командной строке служат серверы telnet, RSH, SSH.

  В Unix-системах есть встроенный сервер удаленного доступа к графическому интерфейсу – X Window System. В Windows – терминальный сервер.

  SNMP-протокол позволяет удаленно производить мониторинг и конфигурацию, для этого на ПК или АУ должен быть SNMP-сервер.

- Серверы приложений – предоставляют сети прикладные сервисы.
- Игровые серверы - служат для одновременной игры нескольких пользователей в единой игровой ситуации. Некоторые игры имеют сервер в основной поставке и позволяют запускать его в невыделенном режиме (то есть позволяют играть на машине, на которой запущен сервер)
- Прочие серверы:
  - Принт-серверы позволяют пользователям сети совместно использовать общий принтер.
  - Факс-сервер позволяет пользователям сети отправлять факсимильные сообщения.

Серверные решения – ОС и/или пакеты программ, оптимизированные под выполнение компьютером функций сервера и/или содержание в своем составе комплект программ для реализации типичного набора серверов (пр.: Unix-системы, изначально предназначенные для реализации серверной инфраструктуры).

Также необходимо выделить пакеты серверов и сопутствующих программ (например комплект веб-сервер/PHP/MySQL для быстрого развёртывания хостинга) для установки под Windows (для Unix свойственна модульная или «пакетная» установка каждого компонента, поэтому такие решения редки, но они существуют. Наиболее известное — LAMP).

В интегрированных серверных решениях установка всех компонентов выполняется единовременно, все компоненты в той или иной мере тесно интегрированы и предварительно настроены друг на друга. Однако в этом случае замена одного из серверов или вторичных приложений (если их возможности не удовлетворяют потребностям) может представлять проблему.

Серверные решения служат для упрощения организации базовой ИТ-инфраструктуры компаний, то есть для оперативного построения полноценной сети в компании, в том числе и «с нуля». Компоновка отдельных серверных приложений в решение подразумевает, что решение предназначено для

выполнения большинства типичных задач; при этом значительно снижается сложность развёртывания и общая стоимость владения ИТ-инфраструктурой, построенной на таких решениях.

Клиент-сервер

Это вычислительная или сетевая архитектура (фактически ПО), в которой задания или сетевая нагрузка распределены между поставщиками услуг (серверы) и заказчиками услуг (клиентами).

Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов (но м.б. расположены и на одном компьютере).

Программы-серверы ожидают от клиентских программ запросы и предоставляют им ресурсы в виде данных (пр.: загрузка файлов через HTTP, FTP, Bit Torrent, потоковые мультимедиа, а также – работа с БД) или в виде сервисных функций (пр.: работа с эл. почтой, общение в мессенджерах, просмотр web-страниц).

В дополнение к клиент-серверной модели распределенные вычислительные приложения часто используют peer-to-peer архитектуру (одноранговая, децентрализованная или пиринговая сеть – оверлейная комп. сеть, основанная на равноправии участников; часто в ней отсутствуют выделенные серверы, а каждый узел (peer) как является клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Участниками сети являются все узлы.

Преимущества peer-to-peer:

- нагрузка и объем данных распределяются между несколькими компьютерами
- если один из них недоступен, общедоступные файлы разделяются между другими компьютерами

Преимущества клиент-сервер:

- отсутствие дублирования кода программы-сервера программами-клиентами
- ниже требования к компьютерам-клиентам
- сервер, как правило, защищен лучше большинства клиентов
- проще сформировать иерархию доступа

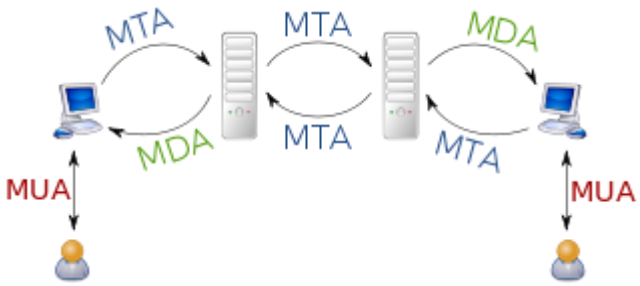
Недостатки клиент-сервер:

- неработоспособность сервера – проблема
- поддержка работы – нужен сис. админ
- дорогое железо, высокие требования к оборудованию, которые растут по мере прогнозируемого роста нагрузки

Многоуровневая архитектура «клиент — сервер» — разновидность архитектуры «клиент — сервер», в которой функция обработки данных вынесена на несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

Принцип работы эл. почты

Наиболее распространенные почтовые сервера: gmail, Sendmail, Exim, Postfix.



- DNS

Domain Name System – система доменных имен – компьютерная распределенная система для получения информации о доменах.

Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получении информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене (SRV-запись).

Распределенная БД DNS поддерживается с помощью иерархии DNS-серверов, взаимодействующих по определенному протоколу.

Основа DNS – представление о иерархической структуре доменных имен и зонах. Каждый сервер, отвечающий за имя, может передать ответственность за дальнейшую часть домена другому серверу (с административной т.з. – другой организации или человеку), что позволяет возложить ответственность за актуальность информации на серверы различных организаций (людей), отвечающих только за «свою» часть доменного имени.

DNS Security Extensions (DNSSEC) – средства проверки целостности передаваемых данных.

DANE – набор спецификаций IETF, обеспечивающий аутентификацию объектов адресации (сертификатов, обеспечивающих безопасное и защищенное соединение транспортного и прикладного уровней) и предоставляемых сервисов с помощью DNS.

Ключевые характеристики DNS:

- Распределенность администрирования – ответственность за разные части иерархической структуры несут разные люди и организации.
- Распределенность хранения информации – каждый узел сети обязан хранить только те данные, которые входят в его зону ответственности, и (возможно) адреса корневых DNS-серверов.
- Кэширование информации – узел может хранить некоторое количество данных не из своей зоны ответственности для уменьшения нагрузки на сеть.
- Иерархическая структура –все узлы объединены в дерево, каждый узел может или самостоятельно определять работу нижестоящих узлов, или делегировать их другим узлам.
- Резервирование – за хранение и обслуживание своих узлов (зон) отвечают (обычно) несколько серверов, разделенные как физически. так и логически, что обеспечивает сохранность данных и продолжение работы даже в случае сбоя в одном из узлов.

Дополнительные возможности:

- поддержка динамических обновлений
- защита данных (DNSSEC) и транзакций (TSIG)
- поддержка различных типов информации

Терминология:

- Домен – узел в дереве имен вместе со всеми подчиненными ему узлами, т.е. именованная ветвь или поддерево в дереве имен. Его структура отражает порядок следования узлов в иерархии,

читая слева направо от младших доменов до доменов высшего уровня: вверху – корневой домен (имеет идентификатор «.», обслуживается корневыми серверами DNS), ниже – домены первого уровня (доменные зоны), затем – второго уровня и т.д. DNS позволяет не указывать точку корневого домена (в конце).

- Поддомен – подчиненный домен, является частью домена более высокого уровня.
- Ресурсная запись – единица хранения и передачи информации в DNS. Имеет имя (т.е. привязана к определенному доменному имени), тип и поле данных, формат и содержание которого зависят от типа.
- Зона – часть дерева доменных имен (включая ресурсные записи), размещаемая как единое целое на некотором сервере доменных имен (DNS-сервере), чаще – одновременно на нескольких серверах. Цель отделения – передача ответственности за соответствующий домен другому лицу или организации (делегирование). Являясь связной частью дерева, зона тоже является деревом с дочерними зонами.
- Делегирование – операция передачи ответственности за часть дерева доменных имен отдельному лицу или организации. Технически это выделение части дерева в отдельную зону и ее размещение на DNS-сервере под управлением лица или организации. При этом в родительскую зону включаются «склеивающие» ресурсные записи (NS и A), содержащие указатели на DNS-сервера дочерней записи.
- DNS-сервер – специализированное ПО для обслуживания DNS, а т.ж. компьютер, на котором оно выполняется. М.б. ответственным за некоторые зоны и/или может перенаправлять запросы вышестоящим серверам.
- DNS-клиент – специализированная библиотека (или программа) для работы с DNS (DNS-сервер может выступать в этой роли).
- Авторитетность – признак размещения зоны на DNS-сервере. Ответы DNS-сервера м.б. двух типов: авторитетные (сервер заявляет, что сам отвечает за зону) и неавторитетные (сервер обрабатывает запрос и возвращает ответ других серверов). Иногда вместо передачи запроса дальше DNS-сервер может вернуть уже известное ему значение (режим кэширования).
- DNS-запрос – DNS query – запрос от клиента (или сервера) к серверу. М.б. рекурсивным или не рекурсивным.

Принцип работы DNS:

Имя не тождественно IP-адресу: один IP-адрес может иметь несколько имен, что позволяет поддерживать на одном компьютере несколько веб-сайтов (виртуальный хостинг). Но и имя может иметь множество IP-адресов, что позволяет создавать балансировку нагрузки.

Для повышения устойчивости системы используется множество серверов, содержащих идентичную информацию, а в протоколе есть средства, позволяющие поддерживать синхронность информации, расположенной на разных серверах. Существует 13 корневых серверов, их адреса практически не изменяются.

Протокол DNS использует для работы TCP- или UDP-порт 53 для ответов на запросы. Традиционно запросы и ответы отправляются в виде одной UDP-датаграммы. TCP используется, когда размер данных ответа превышает 512 байт, и для AXFR-запросов.

Рекурсия:

В DNS это алгоритм поведения DNS-сервера: выполнить от имени клиента полный поиск нужной информации по всей системе DNS, при необходимости обращаясь к другим DNS-серверам.

Рекурсивный DNS-запрос требует полного поиска, нерекурсивный (итеративный) – нет.

Сам DNS-сервер м.б. рекурсивным (умеющим выполнять полный поиск) и нерекурсивным. Некоторые программы DNS-серверов, например, BIND, можно сконфигурировать так, чтобы запросы одних клиентов выполнялись рекурсивно, а запросы других — нерекурсивно.

При нерекурсивном запросе, неумении или запрете сервера выполнять рекурсивные запросы, он либо возвращает данные о зоне, за которую ответственен, либо ошибку.

Нерекурсивный сервер, выдающий информацию о серверах с большим кол-вом информации, м.б. использован для DoS-атак.

Как работает:

Предположим, мы набрали в браузере адрес ru.wikipedia.org. Браузер ищет соответствие этого адреса IP-адресу в файле hosts. Если файл не содержит соответствия, то далее браузер спрашивает у сервера DNS: «какой IP-адрес у ru.wikipedia.org»? Однако сервер DNS может ничего не знать не только о запрошенном имени, но и даже обо всём домене wikipedia.org. В этом случае сервер обращается к корневому серверу — например, 198.41.0.4. Этот сервер сообщает — «У меня нет информации о данном адресе, но я знаю, что 204.74.112.1 является ответственным за зону org.» Тогда сервер DNS направляет свой запрос к 204.74.112.1, но тот отвечает «У меня нет информации о данном сервере, но я знаю, что 207.142.131.234 является ответственным за зону wikipedia.org.» Наконец, тот же запрос отправляется к третьему DNS-серверу и получает ответ — IP-адрес, который и передаётся клиенту — браузеру.

В данном случае при разрешении имени, то есть в процессе поиска IP по имени:

- браузер отправил известному ему DNS-серверу рекурсивный запрос — в ответ на такой тип запроса сервер обязан вернуть «готовый результат», то есть IP-адрес, либо пустой ответ и код ошибки NXDOMAIN;
- DNS-сервер, получивший запрос от браузера, последовательно отправлял нерекурсивные запросы, на которые получал от других DNS-серверов ответы, пока не получил ответ от сервера, ответственного за запрошенную зону;
- остальные упоминавшиеся DNS-серверы обрабатывали запросы нерекурсивно (и, скорее всего, не стали бы обрабатывать запросы рекурсивно, даже если бы такое требование стояло в запросе).

DNS-сервер браузера кэширует результат и в следующий раз не производятся опросы прочих серверов.

Обратный DNS-запрос:

С записью DNS могут быть сопоставлены различные данные, в том числе и какое-либо символьное имя. Существует специальный домен in-addr.arpa, записи в котором используются для преобразования IP-адресов в символьные имена. Например, для получения DNS-имени для адреса 11.22.33.44 можно запросить у DNS-сервера запись 44.33.22.11.in-addr.arpa, и тот вернёт соответствующее символьное имя. Обратный порядок записи частей IP-адреса объясняется тем, что в IP-адресах старшие биты расположены в начале, а в символьных DNS-именах старшие (находящиеся ближе к корню) части расположены в конце.

Записи DNS, или ресурсные записи (resource records, RR), — единицы хранения и передачи информации в DNS. Каждая ресурсная запись состоит из следующих полей:

- имя (NAME) — доменное имя, к которому привязана или которому «принадлежит» данная ресурсная запись,
- тип (TYPE) ресурсной записи — определяет формат и назначение данной ресурсной записи,
- класс (CLASS) ресурсной записи; теоретически считается, что DNS может использоваться не только с TCP/IP, но и с другими типами сетей, код в поле класс определяет тип сети,
- TTL (Time To Live) — допустимое время хранения данной ресурсной записи в кэше неответственного DNS-сервера,
- длина поля данных (RDLEN),
- поле данных (RDATA), формат и содержание которого зависит от типа записи.

Наиболее важные типы DNS-записей:

- Запись A (address record) или запись адреса связывает имя хоста с адресом протокола IPv4. Например, запрос A-записи на имя referrals.icann.org вернёт его IPv4-адрес — 192.0.34.164.
- Запись AAAA (IPv6 address record) связывает имя хоста с адресом протокола IPv6. Например, запрос AAAA-записи на имя K.ROOT-SERVERS.NET вернёт его IPv6-адрес — 2001:7fd::1.
- Запись CNAME (canonical name record) или каноническая запись имени (псевдоним) используется для перенаправления на другое имя.
- Запись MX (mail exchange) или почтовый обменник указывает сервер(ы) обмена почтой для данного домена.
- Запись NS (name server) указывает на DNS-сервер для данного домена.
- Запись PTR (pointer) обратная DNS-запись или запись указателя связывает IP-адрес хоста с его каноническим именем. Запрос в домене in-addr.arpa на IP-адрес хоста в reverse-форме вернёт имя (FQDN) данного хоста. Например, для IP-адреса 192.0.34.164 запрос записи PTR 164.34.0.192.in-addr.arpa вернёт его каноническое имя referrals.icann.org. В целях уменьшения объёма спама многие серверы-получатели электронной почты могут проверять наличие PTR-записи для хоста, с которого происходит отправка. В этом случае PTR-запись для IP-адреса должна соответствовать имени отправляющего почтового сервера, которым он представляется в процессе SMTP-сессии.
- Запись SOA (Start of Authority) или начальная запись зоны указывает, на каком сервере хранится эталонная информация о данном домене, содержит контактную информацию лица, ответственного за данную зону, тайминги (параметры времени) кеширования зонной информации и взаимодействия DNS-серверов.
- SRV-запись (server selection) указывает на серверы для сервисов, используется, в частности, для Jabber (XMPP) и Active Directory.

Зарезервированные доменные имена – здесь. Можно использовать в кач-ве примеров в документации.

Доменное имя может состоять только из ограниченного набора ASCII-символов, позволяя набрать адрес домена независимо от языка пользователя. ICANN утвердил основанную на Punycode систему IDNA, преобразующую любую строку в кодировке Unicode в допустимый DNS набор символов.

Серверы имен:

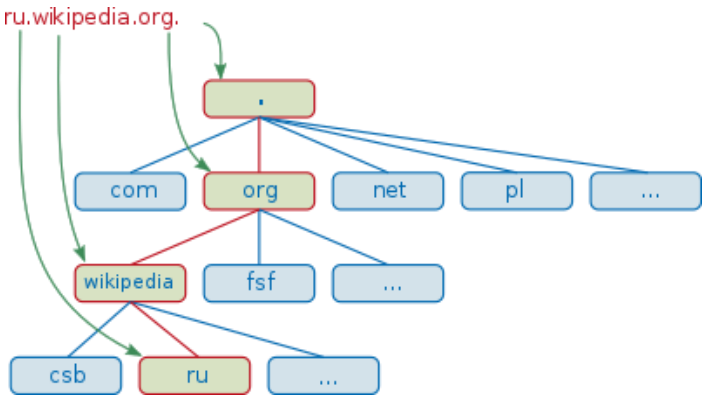
- BIND (Berkeley Internet Name Domain)
- djbdns (Daniel J. Bernstein's DNS)
- Dnsmasq
- MaraDNS
- NSD (Name Server Daemon)
- PowerDNS
- OpenDNS
- Microsoft DNS Server (в серверных версиях операционных систем Windows NT)
- MyDNS

- HTTP

- браузеры

- доменные имена

Пример структуры доменного имени



- архитектура сети и место бэка в ней

2. Знать особенности Python, где и как применяется в бэке, почему именно он, плюсы и минусы

4. Редактор IDE (PyCharm, VS Code, Visual Studio, Intellij Idea):

-какие есть

- почему PyCharm, его + и -

5. Терминал (PowerShell, Bash, Zsh) - то же, что и про редактор

6. Фреймворк (Django): плюсы и минусы, на что способен.

7. Базы данных:

- реляционные (PostgreSQL, MySQL, MS SQL, Azure Cloud SQL)

- нереляционные (NoSQL)(MongoDB, Redis, Cassandra, AWS, Firebase)

8. API (RESTful Api, Swagger)

Программный интерфейс приложения – это описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой.

Обычно входит в описание к-л интернет-протокола (пр.: RFC – Recall For Comments, заявка, содержит спецификации и стандарты, широко применяемые во всемирной сети), программного класса (фреймворка) или стандарта вызова функций ОС.

Часто реализуется отдельной программной библиотекой или сервисом ОС.

Облегчает процесс написания приложений, так как содержит набор действий, исключающих необходимость углубленного знания ряда областей. Так, например, разработчику не нужно знать операций, которые происходят в глубинах файловой системы, для того чтобы копировать файл, так как API предоставит функцию для данного действия.

Если рассмотреть программу (библиотеку, модуль) как черный ящик, API поможет им управлять. Программный компоненты иерархично взаимодействуют посредством API (высокоуровневые компоненты используют API низкоуровневых и т.д. по нисходящей).

По такому принципу построены протоколы передачи данных по интернету. Стандартный стек протоколов (сетевая модель OSI) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам HTTP и IMAP). Каждый уровень пользуется функциональностью нижестоящего уровня и предоставляет функциональность вышестоящему.

Понятие протокола близко по смыслу к понятию API: оба являются абстракцией функциональности, только первое связанно с передачей данных, а второе – со взаимодействием приложений.

API библиотеки функций и классов включают в себя описание сигнатур и семантики функций.

- Сигнатура функции – часть общего объявления функции, позволяющая средствам трансляции идентифицировать функцию среди других.

  Т.к. в разных языках разные представления о сигнатуре функций, есть возможность пользоваться перезагрузкой функции, т.е. использовать функции с одинаковыми названиями.

  Также различают сигнатуру вызова (составляется по синтаксической конструкции вызова функции с учетом ее области видимости, ее имени, последовательности фактических типов аргументов в вызове и типе результата) и сигнатуру реализации функции (в ней участвуют некоторые элементы из синтаксической конструкции объявления функции: спецификатор области видимости функции, ее имя и последовательность формальных типов аргументов).

- Семантика функции – описание того, что эта функция делает. Включает в себя описание того, что является результатом вычисления функции, как и от чего этот результат зависит. Он может зависеть от аргументов, а иногда и от состояния. Логика этих зависимостей и изменений относится к семантике функции.

У каждой ОС имеют API, которое позволяет создавать приложения для данной ОС. Главный API ОС – множество системных вызовов (обращений прикладной программы к ядру ОС для выполнения к-л операции).

Единые стандарты API внутри одной ОС гарантируют исправную и схожую работу программ, написанных в рамках этого API.

Однако различия API разных ОС приводит к затруднению переноса приложений между платформами. Возможные пути решения:

- написание «промежуточных» API (API графических интерфейсов wxWidgets, GTK и т. п.)
- написание библиотек, которые отображают системные вызовы одной ОС в системные вызовы другой (Wine, cygwin и т. п.)
- введение стандартов кодирования в языках программирования (пр.: стандартная библиотека языка C)
- написание интерпретируемых языков, реализуемых на разных платформах (sh, Python, Perl, PHP, Tcl, JavaScript, Ruby и т. д.)

Также в распоряжении программиста часто находится несколько различных API, позволяющих добиться одного и того же результата.

Основными сложностями существующих многоуровневых систем API, таким образом, являются:

- Сложность портирования программного кода с одной системы API на другую (например, при смене ОС);
- Потеря функциональности при переходе с более низкого уровня на более высокий. Грубо говоря, каждый «слой» API создаётся для облегчения выполнения некоторого стандартного набора

операций. Но при этом реально затрудняется либо становится принципиально невозможным выполнение некоторых других операций, которые предоставляет более низкий уровень API.

Наиболее известные API:

- ОС: Amiga ROM Kernel, Cocoa, Linux Kernel API, OS/2 API, POSIX, Windows API
- Графических интерфейсов: DirectDraw/Direct3D (часть DirectX), GDI, GDI+, GTK, SFML, Motif, OpenGL, OpenVG, Qt, SDL, Vulkan, Tk, wxWidgets, X11, Zune
- Звуковых интерфейсов: DirectMusic/DirectSound (часть DirectX), OpenAL
- Аутентификационных систем: BioAPI, PAM

Web API – используется в веб-разработке, содержит определенный набор HTTP-запросов, а т.ж. определение структуры HTTP-ответов, для выражения которых используют XML – или JSON-формат.

Сейчас перешли от SOAP к REST типу коммуникации.

Веб-интерфейсы, обеспечивающие сочетание нескольких сервисов в новых приложениях, известны как гибридные.

Структуры данных

Статический массив

Динамический массив

Ассоциативный массив (HASHTABLE)

9. Аутентификация (OAuth 2.0, JWT):

- Провайдеры (Auth0, Firebase)

10. Паттерны проектирования (чит. кн. “Банда 4х”):

- порождающие

- структурные

- поведенческие

11. Тесты (+изучить фреймворки для написания этих тестов):

- Unit

- интеграционные

12. Доп. инструменты разработки

- менеджеры пакетов (NuGet, nmp, yarn)

- системы контроля версий (git, GitHub, TFS, BitBucket)

Также:

-умение декомпозировать предметную область (понять уже написанный код, легко в нем ориентироваться)

- знание вэб-сервисов, умение их создавать

- сетевые протоколы (как открыть порт, к нему приконнектиться, отрыть приложение вызовов)

- ui фреймфорки

Unix- подобная ОС

Это свободные/открытые ОС, созданные по подобию Unix (пр.: Linux, FreeBSD).

Операционные системы семейства Unix характеризуются модульным дизайном, в котором каждая задача выполняется отдельной утилитой, взаимодействие осуществляется через единую файловую систему, а для работы с утилитами используется командная оболочка.

Межпроцессное взаимодействие (англ. inter-process communication, IPC)

Это обмен данными между потоками одного или разных процессов. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Может осуществляться как на одном компьютере, так и между несколькими компьютерами сети.

Из механизмов, предоставляемых ОС и используемых для IPC, можно выделить:

- механизмы обмена сообщениями;
- механизмы синхронизации;
- механизмы разделения памяти;
- механизмы удалённых вызовов (RPC).

Для оценки производительности различных механизмов IPC используют следующие параметры:

- пропускная способность (количество сообщений в единицу времени, которое ядро ОС или процесс способно обработать);
- задержки (время между отправкой сообщения одним потоком и его получением другим потоком).

Именованный канал

В программировании именованный канал или именованный конвейер (англ. named pipe) — один из методов межпроцессного взаимодействия, расширение понятия конвейера в Unix и подобных ОС.

Именованный канал позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны для взаимодействия с другими программами.

Это понятие также существует и в Microsoft Windows, хотя там его семантика существенно отличается. Традиционный канал — «безымянен», потому что существует анонимно и только во время выполнения процесса.

Именованный канал — существует в системе и после завершения процесса. Он должен быть «отсоединён» или удалён, когда уже не используется. Процессы обычно подсоединяются к каналу для осуществления взаимодействия между ними.

Сокет (программный интерфейс)

Это название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.

Следует различать клиентские и серверные сокеты. Клиентские сокеты грубо можно сравнить с конечными аппаратами телефонной сети, а серверные — с коммутаторами. Клиентское приложение (пр.: браузер) использует только клиентские сокеты, а серверное (пр.: веб-сервер, которому браузер посылает запросы) — как клиентские, так и серверные сокеты.

Мейнфре́йм (также мэйнфрейм, от англ. mainframe) — большой универсальный высокопроизводительный отказоустойчивый сервер со значительными ресурсами ввода-вывода, большим объёмом оперативной и внешней памяти, предназначенный для использования в критически важных системах (англ. mission-critical) с интенсивной пакетной и оперативной транзакционной обработкой.

Файловый сервер

Выделенный сервер, предназначенный для файловых операций ввода-вывода и хранящий файлы любого типа. Как правило, обладает большим объемом дискового пространства, реализованным в формате RAID-массива для обеспечения бесперебойной работы и повышенной скорости записи и чтения данных. Есть также файл-серверные приложения.

Двоичный интерфейс приложений

Он же бинарный, он же Application Binary Interface (ABI) – набор соглашений для доступа приложения к ОС и другим низкоуровневым сервисам, спроектированный для переносимости исполняемого кода между машинами, имеющими совместимые ABI.

В отличие от API, который регламентирует совместимость на уровне исходного кода, ABI можно рассматривать как совокупность правил, позволяющих компоновщику объединять откомпилированные модули компонента без перекомпиляции всего кода, в то же время определяя двоичный интерфейс.

Двоичный интерфейс регламентирует:

- использование регистров процессора
- состав и формат системных вызовов и вызова одного модуля другим
- формат передачи аргументов и возвращаемого значения при вызове функции

Двоичный интерфейс приложений описывает функциональность, предоставляемую рядом ОС и архитектурой набора команд (без привилегированных команд).

Если интерфейс программирования разных платформ совпадает (API), код для этих платформ можно компилировать без изменений.

Если совпадают и API, и ABI, исполняемые файлы можно переносить на эти платформы без изменений.

Если API или ABI платформ различаются, код требует изменений и повторной компиляции.

API не обеспечивает совместимости среды исполнения программы – это задача двоичного интерфейса.

Есть также бинарный интерфейс встраиваемых приложений (Embedded ABI, EABI) – набор соглашений для использования во встраиваемом ПО, описывающий:

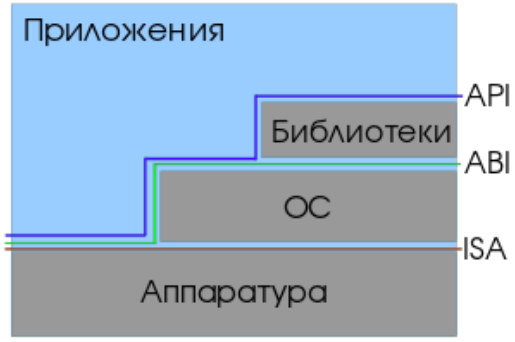
- форматы файлов

- типы данных
- способы использования регистров
- организацию стека
- соглашение о вызове функции.

Если объектный файл был создан компилятором, поддерживающим EABI, становится возможной компоновка этого объектного файла любым компоновщиком, поддерживающим тот же EABI.

Основное отличие EABI от ABI в ОС общего назначения заключается в том, что в коде приложения допускаются привилегированные команды, а динамическое связывание (компоновка) не требуется (а иногда и полностью запрещена), а также, в целях экономии памяти, используется более компактная организация стека.

Уровни и интерфейсы между ними. API, ABI и архитектура набора команд (ISA)



IP-адрес – Internet Protocol – уникальный числовой идентификатор устройства в компьютерной сети, работающий по протоколу TCP/IP. Состоит из двух частей: номера сети и номера узла.

Хост – host – любое устройство, предоставляющее сервисы формата «клиент-сервер» в режиме сервера по каким-либо интерфейсам и уникально определенное на этих интерфейсах. Глобально – любой компьютер, подключенный к локальной или глобальной сети.

Чаще всего, под «хостом» без дополнительных комментариев подразумевается хост протокола TCP/IP, то есть сетевой интерфейс устройства, подключённого к IP-сети. Как и всякий другой хост, этот имеет уникальное определение в среде сервисов TCP/IP (IP-адрес). С хостом протокола TCP/IP может быть также связана необязательная текстовая характеристика — доменное имя.

Служебная запись (SRV-запись) – стандарт в DNS, определяющий местоположение, т.е. имя хоста и номер порта серверов для определенных служб.

Некоторые Интернет-протоколы, такие как SIP и XMPP, часто требуют поддержки SRV-записей.

SRV-запись имеет такой формат:

_service._proto.name TTL class SRV priority weight port target

- service: символьное имя сервиса.
- proto: транспортный протокол, используемый сервисом, как правило TCP или UDP.
- name: доменное имя, для которого эта запись действует.
- TTL: стандарт DNS, время жизни.
- class: стандарт DNS, поле класса (это всегда IN).
- priority: приоритет целевого хоста, более низкое значение означает более предпочтительный.
- weight: относительный вес для записей с одинаковым приоритетом.

- port: Порт TCP или UDP, на котором работает сервис.
- target: канонические имя машины, предоставляющей сервис.

Распределенная БД (параллельная) – Distributed Database, DDB – единая БД, составные части которой размещаются в различных узлах компьютерной сети в соответствии с к-л критерием.

Данные я-ся DDB только если они связаны в соответствии с некоторым структурным формализмом, реляционной моделью, а доступ к ним обеспечивается единым высокоуровневым интерфейсом.

Могут иметь разный уровень реплицированности – от полного отсутствия дублирования информации до дублирования всей информации во всех распределенных копиях (пр.: блокчейн).

Прозрачность – распределение БД по множеству узлов невидимо для пользователей. Полная прозрачность достигается за счет: прозрачности сети, распределения, репликации, фрагментации, доступа.

Виртуальный хостинг - shared hosting – при нем множество веб-сайтов расположено на одном веб-сервере, каждый в своем разделе, по все используют одно ПО.

Существует два основных метода реализации доступа к веб-сайтам:

- по имени shared IP hosting), когда все веб-сайты используют один общий IP-адрес. Согласно протоколу HTTP/1.1, веб-браузер при запросе к веб-серверу указывает доменное имя веб-сайта в поле Host заголовка текущего запроса, и веб-сервер использует его для правильного выполнения запроса, а также копирует это имя в ячейку [HTTP_HOST] суперглобального массива $_SERVER.
- по IP-адресу (dedicated IP hosting), при котором у каждого веб-сайта есть собственный IP-адрес, а веб-сервер имеет несколько физических или виртуальных сетевых интерфейсов.

Сертификат открытого ключа (сертификат электронной подписи, сертификат ключа подписи, сертификат ключа проверки электронной подписи)— электронный или бумажный документ, содержащий открытый ключ, информацию о владельце ключа, области применения ключа, подписанный выдавшим его Удостоверяющим центром и подтверждающий принадлежность открытого ключа владельцу.

Протокол передачи данных — набор определённых правил или соглашений интерфейса логического уровня, который определяет обмен данными между различными программами. Эти правила задают единообразный способ передачи сообщений и обработки ошибок.

TCP/IP — набор протоколов передачи данных, получивший название от двух принадлежащих ему протоколов: TCP (Transmission Control Protocol) и IP (Internet Protocol)

HTTP (Hyper Text Transfer Protocol) — это протокол передачи гипертекста. Протокол HTTP используется при пересылке Web-страниц между компьютерами, подключёнными к одной сети.

FTP (File Transfer Protocol) — это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP даёт возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удалённым компьютером, пользователь может скопировать файл с удалённого компьютера на свой или скопировать файл со своего компьютера на удалённый.

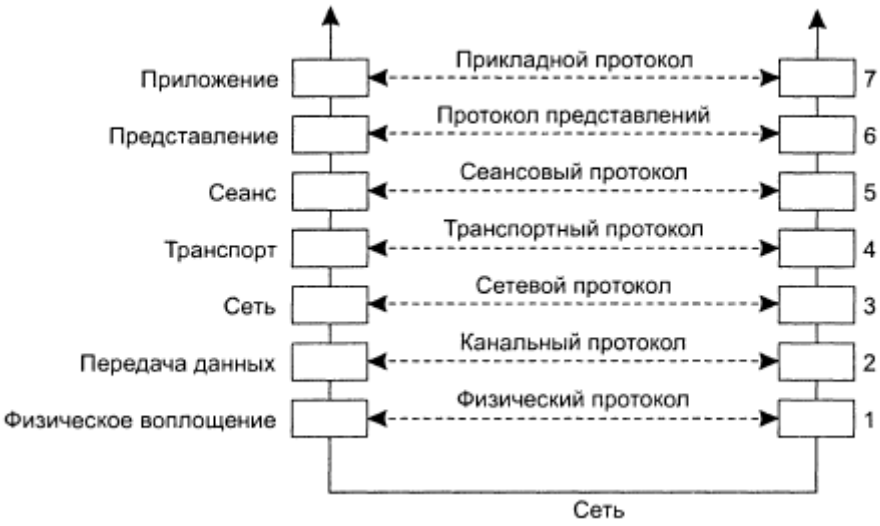
POP3 (Post Office Protocol) — это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.

SMTP (Simple Mail Transfer Protocol) — протокол, который задаёт набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приёме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.

TELNET — это протокол удалённого доступа. TELNET даёт возможность абоненту работать на любой ЭВМ, находящейся с ним в одной сети, как на своей собственной, то есть запускать программы, менять режим работы и так далее. На практике возможности ограничиваются тем уровнем доступа, который задан администратором удалённой машины.

Модель OSI — 7-уровневая логическая модель работы сети. Реализуется группой протоколов и правил связи, организованных в несколько уровней:

- на физическом уровне определяются физические (механические, электрические, оптические) характеристики линий связи;
- на канальном уровне определяются правила использования физического уровня узлами сети;
- сетевой уровень отвечает за адресацию и доставку сообщений;
- транспортный уровень контролирует очерёдность прохождения компонентов сообщения;
- сеансовый уровень координирует связь между двумя прикладными программами, работающими на разных рабочих станциях;
- уровень представления служит для преобразования данных из внутреннего формата компьютера в формат передачи;
- прикладной уровень является пограничным между прикладной программой и другими уровнями, обеспечивая удобный интерфейс связи для сетевых программ пользователя.



Модель — стек протоколов TCP/IP — содержит 4 уровня:

- канальный уровень (link layer),
- сетевой уровень (Internet layer),
- транспортный уровень (transport layer),
- прикладной уровень (application layer).

Демон (daemon) — компьютерная программа в UNIX-подобных системах, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем.

Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: серверы сетевых протоколов (HTTP, FTP, электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается процесс, который не имеет управляющего терминала. Традиционно названия демон-процессов заканчиваются на букву d.

В системах Windows аналогичный класс программ называется службой (Services).

TCP/IP – сетевая модель четырехуровневой передачи цифровых данных от источника к получателю.

Каждый уровень описан протоколом передачи. На стеке протоколов передачи данных базируется Интернет.

TCP – Transmission Control Protocol – протокол управления передачей данных (пакеты – сегменты).

IP – Internet Protocol – масштабируемый протокол сетевого уровня, который объединил отдельные компьютерные сети в глобальную сеть Интернет.

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься.

Уровни:

- Прикладной уровень (Application Layer) - обеспечивает обмен данными между процессами для приложений (пр.: HTTP, RTSP, FTP, DNS);

  На нем работает большинство сетевых приложений. Они имеют собственные протоколы обмена данными (пр.: интернет браузер для протокола HTTP, ftp-клиент для протокола FTP (передача файлов), почтовая программа для протокола SMTP (электронная почта), SSH (безопасное соединение с удалённой машиной), DNS (преобразование символьных имён в IP-адреса) и др.)

  В массе своей эти протоколы работают поверх TCP или UDP и привязаны к определённому порту, например:
  - HTTP на TCP-порт 80 или 8080,
  - FTP на TCP-порт 20 (для передачи данных) и 21 (для управляющих команд),
  - SSH на TCP-порт 22,
  - запросы DNS на порт UDP (реже TCP) 53,
  - обновление маршрутов по протоколу RIP на UDP-порт 520.

  К этому уровню относятся: Echo, Finger, Gopher, HTTP, HTTPS, IMAP, IMAPS, IRC, NNTP, NTP, POP3,

POPS, QOTD, RTSP, SNMP, SSH, Telnet, XDMCP.

- Транспортный уровень (Transport Layer) - обрабатывающий связь между хостами (пр.: TCP, UDP, SCTP, DCCP; RIP, протоколы маршрутизации, подобные OSPF, что работают поверх IP, являются частью сетевого уровня);

  Могут решать проблему негарантированной доставки сообщений, а т.ж. гарантировать правильную последовательность прихода данных. В стеке TCP/IP определяют, для какого именно приложения эти данные.

  Протоколы автоматической маршрутизации, логически представленные на этом уровне (поскольку работают поверх IP), на самом деле являются частью протоколов сетевого уровня; например OSPF (IP идентификатор 89).

  TCP (IP идентификатор 6) — «гарантированный» транспортный механизм с предварительным установлением соединения, предоставляющий приложению надёжный поток данных, дающий уверенность в безошибочности получаемых данных, перезапрашивающий данные в случае потери и устраняющий дублирование данных. TCP позволяет регулировать нагрузку на сеть, а также уменьшать время ожидания данных при передаче на большие расстояния. Более того, TCP гарантирует, что полученные данные были отправлены точно в такой же последовательности. В этом его главное отличие от UDP.

  UDP (IP идентификатор 17) протокол передачи датаграмм без установления соединения. Также его называют протоколом «ненадёжной» передачи, в смысле невозможности удостовериться в доставке сообщения адресату, а также возможного перемешивания пакетов. В приложениях, требующих гарантированной передачи данных, используется протокол TCP.

  И TCP, и UDP используют для определения протокола верхнего уровня число, называемое портом.

- Межсетевой уровень (Сетевой уровень) (Internet Layer) - обеспечивающий межсетевое взаимодействие между независимыми сетями (пр.: для TCP/IP это IP (вспомогательные протоколы, вроде ICMP и IGMP, работают поверх IP, но тоже относятся к сетевому уровню; протокол ARP является самостоятельным вспомогательным протоколом, работающим поверх канального уровня);

  Разработан для передачи данных из любой сети в любую сеть, независимо от протоколов нижнего уровня, а также может запрашивать данные от удалённой стороны, например в протоколе ICMP (используется для передачи диагностической информации IP-соединения) и IGMP (используется для управления multicast-потоками).

  На этом уровне работают маршрутизаторы, которые перенаправляют пакеты в нужную сеть путём расчёта адреса сети по маске сети. Примерами такого протокола является X.25 и IPC в сети ARPANET.

  ICMP и IGMP расположены над IP и должны попасть на следующий — транспортный — уровень, но функционально являются протоколами сетевого уровня, и поэтому их невозможно вписать в модель OSI.

  Пакеты сетевого протокола IP могут содержать код, указывающий, какой именно протокол следующего уровня нужно использовать, чтобы извлечь данные из пакета. Это число — уникальный *IP-номер протокола*. ICMP и IGMP имеют номера, соответственно, 1 и 2.

  К этому уровню относятся: DVMRP, ICMP, IGMP, MARS, PIM, RIP, RIP2, RSVP

- Канальный уровень (Network Access Layer, Link Layer) - содержащий методы связи для данных, которые остаются в пределах одного сегмента сети (пр.: Ethernet, IEEE 802.11 WLAN, SLIP, Token Ring, ATM и MPLS, физическая среда и принципы кодирования информации, T1, E1).

  Описывает способ кодирования данных для передачи пакета данных на физическом уровне (то есть специальные последовательности бит, определяющих начало и конец пакета данных, а также обеспечивающие помехоустойчивость). Ethernet, например, в полях заголовка пакета содержит указание того, какой машине или машинам в сети предназначен этот пакет.

  Примеры протоколов канального уровня — Ethernet, IEEE 802.11 WLAN, SLIP, Token Ring, ATM и MPLS.

  PPP не совсем вписывается в такое определение, поэтому обычно описывается в виде пары протоколов HDLC/SDLC.

  MPLS занимает промежуточное положение между канальным и сетевым уровнем и, строго говоря, его нельзя отнести ни к одному из них.

  Канальный уровень иногда разделяют на 2 подуровня — LLC и MAC.

  Кроме того, канальный уровень описывает среду передачи данных (будь то коаксиальный кабель, витая пара, оптическое волокно или радиоканал), физические характеристики такой среды и принцип передачи данных (разделение каналов, модуляцию, амплитуду сигналов, частоту сигналов, способ синхронизации передачи, время ожидания ответа и максимальное расстояние).

  При проектировании стека протоколов на канальном уровне рассматривают помехоустойчивое кодирование — позволяющие обнаруживать и исправлять ошибки в данных вследствие воздействия шумов и помех на канал связи.

Распределение протоколов по уровням модели OSI

	TCP/IP	OSI	
7	Прикладной	Прикладной	напр., HTTP, SMTP, SNMP, FTP, Telnet, SSH, SCP, SMB, NFS, RTSP, BGP
6		Представления	напр., XDR, AFP, TLS, SSL
5		Сеансовый	напр., ISO 8327 / CCITT X.225, RPC, NetBIOS, PPTP, L2TP, ASP
4	*Транспортный*	Транспортный	напр., TCP, UDP, SCTP, SPX, ATP, DCCP, GRE
3	*Сетевой*	Сетевой	напр., IP, ICMP, IGMP, CLNP, OSPF, RIP, IPX, DDP
2	*Канальный*	Канальный	напр., Ethernet, Token ring, HDLC, PPP, X.25, Frame relay, ISDN, ATM, SPB, MPLS, ARP
1		Физический	напр., электрические провода, радиосвязь, волоконно-оптические провода, инфракрасное излучение

Сетевая модель OSI (The Open System Interconnection model) – сетевая модель стека (магазина) сетевых протоколов OSI/ISO, посредством которой различные сетевые устройства могут взаимодействовать друг с другом; определяет различные уровни взаимодействия систем, у каждого из которых свои функции.

Протоколы связи позволяют структуре на одном хосте взаимодействовать с соответствующей структурой того же уровня на другом хосте.

На каждом уровне N два объекта обмениваются блоками данных (PDU) с помощью протокола данного уровня на соответствующих устройствах. Каждый PDU содержит блок служебных данных (SDU), связанный с верхним или нижним протоколом.

Обработка данных двумя взаимодействующими OSI-совместимыми устройствами происходит следующим образом:

1. Передаваемые данные составляются на самом верхнем уровне передающего устройства (уровень N) в протокольный блок данных (PDU).
2. PDU передается на уровень N-1, где он становится сервисным блоком данных (SDU).
3. На уровне N-1 SDU объединяется с верхним, нижним или обоими уровнями, создавая слой N-1 PDU. Затем он передается в слой N-2.
4. Процесс продолжается до достижения самого нижнего уровня, с которого данные передаются на принимающее устройство.
5. На приемном устройстве данные передаются от самого низкого уровня к самому высокому в виде серии SDU, последовательно удаляясь из верхнего или нижнего колонтитула каждого слоя до достижения самого верхнего уровня, где принимаются последние данные.

Модель					
Уровень (layer)		Тип данных (PDU[14])	Функции	Примеры	Оборудование
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3, WebSocket	Хосты (клиенты сети), Межсетевой экран
	6. Представления (presentation)		Представление и шифрование данных	ASCII, EBCDIC, JPEG, MIDI	
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP, L2TP, gRPC	
	4. Транспортный (transport)	Сегменты (segment) / Датаграммы (datagram)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, Порты	
Media[15] layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk, ICMP	Маршрутизатор, Сетевой шлюз, Межсетевой экран
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта.	Сетевой мост, Коммутатор, точка доступа
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, RJ («витая пара», коаксиальный, оптоволоконный), радиоканал	Концентратор, Повторитель (сетевое оборудование)

Web-сервер – сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров (обозначены URL-адресами), и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными.

Дополнительные функции:

- автоматизация работы веб-страниц
- ведение журнала обращений пользователя к ресурсам
- аутентификация и авторизация пользователей

- поддержка динамически генерируемых страниц
- поддержка HTTPS для защищенных соединений с клиентами.

65% рынка занимает web-сервер Apache – свободный веб-сервер, наиболее часто используемый в Unix-подобных ОС.

Прочие:

- IIS от компании *Microsoft*, распространяемый с ОС семейства *Windows*.
- *nginx* — свободный веб-сервер, разрабатываемый Игорем Сысоевым с 2002 года и пользующийся большой популярностью на крупных сайтах[2] [3]
- lighttpd — свободный веб-сервер.
- Google Web Server — веб-сервер разработанный компанией Google.
- Resin — свободный веб-сервер приложений.
- Cherokee — свободный веб-сервер, управляемый только через web-интерфейс.
- Rootage — веб-сервер, написанный на Java.
- THTTPD — простой, маленький, быстрый и безопасный веб-сервер.
- Open Server — бесплатная программа с графическим интерфейсом использует множество исключительно свободного программного комплекса.
- H2O — свободный быстрый веб-сервер, написанный на C.
- nghttp2 — веб-сервер, встроенный в Node.js.
- Go HTTP — веб-сервер, встроенный в Go.

Клиенты:

- веб-браузер, работающий на ПК или переносном устройстве (пр.: кПК)
- программы, самостоятельно обращающиеся к веб-серверам для получения обновлений или другой информации (пр.: антивирус запрашивает обновление БД)
- мобильный телефон, получающий доступ к ресурсам веб-сервера через WAP
- др. цифровые устройства и бытовая техника.

Отличие веб-сервера от сервера приложений: веб-сервер предназначен для обслуживания статических страниц (пр.: HTML, CSS), а сервер приложений отвечает за генерацию динамического содержимого путем выполнения кода на стороне сервера (пр.: JSP, EJB и др.).

Доменное имя – domain name – символьное имя, служащее для идентификации областей, которые я-ся единицами административной автономии в сети Интернет, в составе вышестоящей по иерархии такой области. Каждая такая область – домен. Общее пространство имен функционирует благодаря DNS. Доменные адреса дают возможность адресации Интернет-узлов и расположенным на них ресурсам (веб-сайтам, серверам, серверам эл. почты, другим службам) быть предоставленными в удобной для человека форме.

Структура полного доменного имени:

Непосредственное имя домена + имена всех доменов, в которые он входит, разделенные точками.

FQDN (fully qualified domain name) – полное доменное имя, т.е. не имеющее неоднозначностей в определении, включающее в себя имена всех родительских доменов иерархии DNS.

В DNS, а точнее в zone file, FQDN завершаются точкой, т.е. включают корневое имя «.», которое я-ся безымянным (на практике опускается).

Различия между FQDN и доменным именем проявляются при именовании доменов второго, третьего и т.д. уровней. Для FQDN обязательно указать домены более высокого уровня.

В DNS-записях доменов (для перенаправления (трансляция порт-адрес – технология трансляции порт-адреса в зависимости от TCP-UDP-порта получателя), почтовых серверов и т.д.) всегда используется FQDN.

Доменная зона – совокупность доменных имен определенного уровня, входящих в конкретный домен (термин применяется в тех. сфере при настройке DNS-серверов: поддержание, делегирование и трансфер зоны).

DNS-резолверы – отвечают на вопросы, касающиеся любых зон.

Служба whois – позволяет узнать владельца (админа) домена.

Дроп-домен – у него истек срок регистрации, и теперь он свободен.

Виды:

- Тематические домены (gTLD). Общие домены верхнего уровня (gTLD) управляются организацией ICANN.
- Интернационализованные домены (IDN). Доменные имена, которые содержат символы национальных алфавитов. IDN верхнего уровня управляются и находятся под контролем ICANN.
- Национальные домены (ccTLD).Национальные домены верхнего уровня (ccTLD) делегированы соответствующим национальным регистраторам, которые устанавливают правила регистрации в них либо сами, либо согласно указаниям правительства. Управляющей организацией является IANA.
- Зарезервированные доменные имена. Документ RFC 2606 (Reserved Top Level DNS Names — Зарезервированные имена доменов верхнего уровня) определяет названия доменов, которые следует использовать в качестве примеров (например, в документации), а также для тестирования. Кроме example.com, example.org и example.net, в эту группу также входят .test, .invalid и др.
- Длинные доменные имена. Размер доменного имени ограничивается по административным и техническим причинам. Обычно разрешается регистрация доменов длиной до 63 символов. В некоторых странах можно регистрировать домены длиной до 127 знаков.

Apache HTTP-сервер – апач – свободный веб-сервер.

Это кроссплатформенное ПО, поддерживает операционные системы Linux, BSD, Mac OS, Microsoft Windows, Novell NetWare, BeOS.

Его основные достоинства – надежность и гибкость конфигурации. Он позволяет переключать внешние модули для предоставления данных, использовать СУБД для аутентификации пользователей, модифицировать сообщения об ошибках и т.д. Поддерживает IPv4.

Архитектура:

- Ядро – включает в себя основные функциональные возможности (обработка конфигурационных файлов, протокол HTTP и система загрузки модулей). Язык – C.
- Система конфигурации – основана на текстовых конфигурационных файлах, имеет 3 уровня конфигурации:
    - К. сервера (httpd.conf) – директивы к. сгруппированы в 3 осн. раздела: 1) управляющие процессом Apache в целом (глобальное окружение); 2) определяющие параметры «главного» сервера, или сервера по умолчанию, который отвечает на запросы, которые не обрабатываются виртуальными хостами (определяют т.ж. установки по умолчанию для всех остальных виртуальных хостов); 3) установки для виртуальных хостов, позволяющие обрабатывать запросы Web одним едиными сервером Apache, но направлять по разным адресам IP или именам хостов.
    - К. виртуального хоста (httpd.conf с версии 2.2, extra/httpd-vhosts.conf).
    - К. уровня каталога (.htaccess).

Имеет собственный язык конфигурационных файлов, основанный на блоках директив. Практически все параметры ядра могут быть изменены через конфигурационные файлы, вплоть до управления MPM. Большая часть модулей имеет собственные параметры.

Часть модулей использует в своей работе конфигурационные файлы операционной системы (например /etc/passwd и /etc/hosts).

Помимо этого, параметры могут быть заданы через ключи командной строки.

- Многопроцессорные модели (MPM).
  Для веб-сервера Apache существует множество моделей симметричной многопроцессорности. Вот основные из них:

Название	Разработчик	Поддержи-ваемые ОС	Описание	Назначение	Статус
worker	Apache Software Foundation	Linux, FreeBSD	Гибридная многопроцессорно-многопоточная модель. Сохраняя стабильность многопроцессорных решений, она позволяет обслуживать большое число клиентов с минимальным использованием ресурсов.	Среднезагруженные веб-серверы.	Стабильный.
pre-fork	Apache Software Foundation	Linux, FreeBSD	MPM, основанная на предварительном создании отдельных процессов, не использующая механизм threads.	Большая безопасность и стабильность за счёт изоляции процессов друг от друга, сохранение совместимости со старыми библиотеками, не поддерживающими threads.	Стабильный.
perchild	Apache Software Foundation	Linux	Гибридная модель, с фиксированным количеством процессов.	Высоконагруженные серверы, возможность запуска дочерних процессов используя другое имя пользователя для повышения безопасности.	В разработке, нестабильный.
netware	Apache Software Foundation	Novell NetWare	Многопоточная модель, оптимизированная для работы в среде NetWare.	Серверы Novell NetWare	Стабильный.
winnt	Apache Software Foundation	Microsoft Windows	Многопоточная модель, созданная для операционной системы Microsoft Windows.	Серверы под управлением Windows Server.	Стабильный.
Apache-ITK	Steinar H. Gunderson	Linux, FreeBSD	MPM, основанная на модели prefork. Позволяет запуск каждого виртуального хоста под отдельными uid и gid.	Хостинговые серверы, серверы, критичные к изоляции пользователей и учёту ресурсов.	Стабильный.
peruser	Sean Gabriel Heacock	Linux, FreeBSD	Модель, созданная на базе MPM perchild. Позволяет запуск каждого виртуального хоста под отдельными uid и gid. Не использует потоки.	Обеспечение повышенной безопасности, работа с библиотеками, не поддерживающими threads.	Стабильная версия от 4 октября 2007 года, экспериментальная — от 10 сентября 2009 года.
event	Apache Software Foundation	Linux, FreeBSD	Модель использует threads и thread-safe polling основана на worker. предназначен для одновременного обслуживания большего количества запросов путем передачи некоторой обработки в потоки слушателей, освобождая рабочие потоки для обслуживания новых запросов.	Обеспечение повышенной производительности. не очень хорошо работает на старых платформах, в которых отсутствует хорошая многопоточность, но требование EPoll или KQueue делает это спорным.	Стабильный.

- Система модулей.
  Apache HTTP Server поддерживает модульность. Модули могут быть как включены в состав сервера в момент компиляции, так и загружены динамически, через директивы конфигурационного файла.
  В модулях реализуются такие вещи, как:
  - Поддержка языков программирования.
  - Добавление функций.
  - Исправление ошибок или модификация основных функций.
  - Усиление безопасности.

  Часть веб-приложений, например панели управления ISPmanager и VDSmanager реализованы в виде модуля Apache.
- Механизм виртуальных хостов.

  Apache имеет встроенный механизм виртуальных хостов. Он позволяет полноценно обслуживать на одном IP-адресе множество сайтов (доменных имён), отображая для каждого из них собственное содержимое.

Для каждого виртуального хоста можно указать собственные настройки ядра и модулей, ограничить доступ ко всему сайту или отдельным файлам. Некоторые MPM, например Apache-ITK, позволяют запускать процесс httpd для каждого виртуального хоста с отдельными идентификаторами uid и guid.

Также существуют модули, позволяющие учитывать и ограничивать ресурсы сервера (CPU, RAM, трафик) для каждого виртуального хоста.

Функциональные возможности:

- Интеграция с другим ПО и языками программирования

Существует множество модулей, добавляющих к Apache поддержку различных языков программирования и систем разработки.

К ним относятся:

- PHP (mod_php).
- Python (mod python, mod wsgi).
- Ruby (apache-ruby).
- Perl (mod perl).
- ASP (apache-asp).
- Tcl (rivet)

Кроме того, Apache поддерживает механизмы CGI и FastCGI, что позволяет исполнять программы на практически всех языках программирования, в том числе C, C++, Lua, sh, Java.

- Безопасность

Apache имеет различные механизмы обеспечения безопасности и разграничения доступа к данным. Основными являются:

- Ограничение доступа к определённым каталогам или файлам.
- Механизм авторизации пользователей для доступа к каталогу на основе HTTP-аутентификации (mod_auth_basic) и digest-аутентификации (mod_auth_digest).
- Ограничение доступа к определённым каталогам или всему серверу, основанное на IP-адресах пользователей.
- Запрет доступа к определённым типам файлов для всех или части пользователей, например запрет доступа к конфигурационным файлам и файлам баз данных.
- Существуют модули, реализующие авторизацию через СУБД или PAM.

В некоторых MPM-модулях присутствует возможность запуска каждого процесса Apache, используя различные uid и gid с соответствующими этим пользователям и группам пользователей.

Также существует механизм suexec, используемый для запуска скриптов и CGI-приложений с правами и идентификационными данными пользователя.

Для реализации шифрования данных, передающихся между клиентом и сервером, используется механизм SSL, реализованный через библиотеку OpenSSL. Для удостоверения подлинности веб-сервера используются сертификаты X.509.

Существуют внешние средства обеспечения безопасности, например mod_security.

- Интернационализация

Начиная с версии 2.0 появилась возможность определения сервером локали пользователя. Сообщения об ошибках и событиях, посылаемые браузеру, теперь представлены на нескольких языках и используют SSI-технологию.

Также, можно реализовать средствами сервера отображение различных страниц для пользователей с различными локалями. Apache поддерживает множество кодировок, в том числе Unicode, что позволяет использовать страницы, созданные в любых кодировках и на любых языках.

- Обработка событий

  Администратор может установить собственные страницы и обработчики для всех HTTP-ошибок и событий, таких как 404 (Not Found) или 403 (Forbidden). В том числе существует возможность запуска скриптов и отображения сообщений на разных языках.

SSI - В версиях 1.3 и старше был реализован механизм Server Side Includes, позволяющий динамически формировать HTML-документы на стороне сервера. Управлением SSI занимается модуль mod_include, включённый в базовую поставку Apache.

База данных – Data Base – совокупность данных, хранимых в соответствии со схемой данных, манипулирование с которыми осуществляют в соответствии с правилами средств модулирования данных.

База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Признаки:

- БД хранится и обрабатывается в вычислительной системе
- Данные БД логически структурированы для облегчения работы с ними (систематизация: выделение составных частей, построение связей между ними, типизация, семантика и допустимые операции)
- БД включает схему, или метаданные, описывающие ее логическую структуру в формальном виде (в соответствии с некоторой метамоделью)

Виды БД:

По модели данных:

- иерархические
- объектные или объектно-ориентированные
- объектно-реляционные
- реляционные
- сетевые
- функциональные

По системе хранения:

- традиционные – conventional DB – хранят данные во вторичной памяти
- резидентные – все данные на стадии исполнения находятся в оперативной памяти
- третичные – tertiary DB – хранят данные на отсоединяемых устройствах массового хранения

Для некоторых БД строятся специализированные СУБД, либо доп. возможности СУБД:

- пространственные (spatial DB) – базы с пространственными свойствами сущности предметной области
- временные – поддерживают к-л аспект времени (не считая времени, определяемого пользователем)

По степени распределенности:

- централизованные (сосредоточенные) – полностью поддерживаемые на одном оборудовании

- распределенные (distributed DB) – на разных узлах комп. сети по к-л критерию. Среди них выделяют:
    - сегментированные – разделенные на части под управлением различных экземпляров СУБД по к-л критерию
    - тиражированные (реплицированные) – в них одни и те же данные разнесены под управление разных экземпляров СУБД
    - неоднородные (heterogeneous distributed DB) – фрагменты распределенной базы в разных узлах сети поддерживаются средствами более одной СУБД

По способам организации хранения:

- циклические – записывают новые данные заместо устаревших
- потоковые

SOLID

Хороший код:

- Масштабируемый, легко вносить изменения
- Порог вхождения в проект – новому человеку легко разобраться
- Простой, без усложнений

- **S** – *Single Responsibility Principle* – один объект – одна задача, одна зона ответственности

Декомпозиция на модули. Разделяем модель данных и поведение.

Иначе – путаница, трудно вносить изменения. Декомпозированный код легче читать, править, тестировать, мержить.

God-object – делает много задач.

- **O** – *Open/Closed Principle* – программные сущности открыты для расширения и закрыты для изменения.

Новый функционал вводим за счет создания новых сущностей путем наследования, композиции, а не изменением старых.

Если и изменяем код, необходимо регрессионное тестирование.

- **L** – *Liskov's Substitution Principle* - функции, сущности, которые используют родительский тип, должны так же работать и с дочерними классами, при этом не должна нарушаться логика программы. Наследуемый класс должен дополнять, а не замещать поведение базового класса.
- **I** – *Interface Segregation Principle* – программные сущности не должны зависеть от методов, которые они не используют.

Код менее связанный, сущности не зависят от методов, которые к ним не относятся.

- **D** – *Dependency Inversion Principle* – модули высокого уровня не должны зависеть от модулей нижнего уровня, все они должны зависеть от абстракций, а абстракции не должны зависеть от деталей, наоборот – детали должны зависеть от абстракций.

Парадигма ООП

Процедурный подход – программе даются данные, она выполняет какие-то функции и возвращает какой-то результат.

В ООП (объектно-ориентированном программировании) есть класс, в нем есть объекты, у которых есть свойства. Объект – конкретный представитель класса со своими обозначенными свойствами. Объект может совершать некоторые действия – методы.

Объекты здесь не только классы (и объекты), но и сущности более высокого уровня – модули, пекеджи, неймспейсы, сабсистемы, система, сабпекеджи и т.д.

Основные концепции ООП (принципы):

1. **Инкапсуляция**

Сам класс – капсула, которая содержит свойства и методы для работы с этими свойствами.

В объект (класс) объединяются методы и данные.

Сокрытие – private (можно использовать только внутри класса, использовать извне нельзя) и public (модификаторы доступа) (одна из трактовок инкапсуляции).

Итого, это и объединение методов и данных в один объект, и сокрытие этих методов и данных от внешнего воздействия. Объект не должен изменяться ничем извне, кроме методов самого объекта (внутри).

Get, set – создаются, чтобы получать доступ к приватным свойствам (получать и изменять).

Пример: имя пользователя и пароль можно получать и менять, а ID – только получать.

Из инкапсуляции вытекает множество паттерном GRASP и GoF.

Lowcoplin

2. **Наследование**

Новые классы наследуют черты от родительского(-их) класса(-ов) и имеют собственные свойства.

Не можем унаследоваться от нескольких классов (в большинстве языков).

Конструктор т.ж. по умолчанию наследуется.

3. **Полиморфизм**

В общем, это способность функции работать с данными разных типов.

Есть два типа:

- Полиметрический (истинный)

  Когда одна и та же функция с одним и тем же телом способна принимать в качестве параметра данные разных классов.

  Пример: у нас есть несколько классов: родительский класс «человек», от него наследуется класс «работник», а уже от него – класс «программист». Всем классам нужно написать функцию приветствия, возвращающую «Привет» + «я» + название класса + имя объекта.

  Принимать данные от каждого массива – плохая идея, поэтому мы применяем полиморфизм, чтобы каждый объект, унаследованный от класса «человек» смог поздороваться.

  Создаем массив со всеми объектами, создаем функцию массового приветствия с аргументом массивом.

- ad-hoc (мнимый) – класс имеет несколько методов, например, работающих с разными типами данных, а так же класс, в котором мы делаем преобразование дочернего класса до родительского

Это приведение данных к тому типу, с которым работаем метод; перегрузка метода – когда он существует с одинаковыми названиями, но разными параметрами.

Есть высказывание, что все if можно заменить на полиморфизм. Он помогает в разы уменьшать размер программы.

Рефакторинг, coding by exception.

Еще сейчас выделяют 4 принцип – абстракция.

Помимо наследования существую еще два **способа взаимодействия классов**:

- Композиция

Пример: есть класс автомобиль, внутри которого есть объект класса двигатель, а также массив объектов класса колесо. Двигатель и колеса не могут существовать отдельно от автомобиля.

- Агрегация

Все тот же автомобиль. Однако добавляется класс «елочка-освежитель». Он может существовать отдельно от класса «автомобиль». Тогда елочка будет отдельным свойством у класса «автомобиль», обращающемуся к собственному классу – «освежитель». А в конструкторе объекта мы к нему обращаемся через свойство.

Абстрактные классы и интерфейсы

Интерфейс – как оглавление в учебнике. Он говорит, что нужно сделать, но не говорит как. Из него нельзя сделать объект.

Абстрактные классы похожи на интерфейс, кроме того, в них можно определять абстрактные методы (аналоги методов в интерфейсе, т.е. без реализации), но в них можно создавать и обычные методы с реализацией и логикой. Класс, образованный от абстрактного, наследует все обычные методы, а т.ж. абстрактные.

generic – обобщение <ожидаемый тип данных>

Пример для закрепления: Dependency Injection (внедрение зависимостей) (паттерн)

У нас есть два слоя приложения:

1) работа с БД (получить, записать, изменить, удалить из репозитория). Реализации – через MongoRepository и MySqlRepository.

2) БЛ. Здесь вопрос: какой репозиторий выбрать? Лучше, чб сервисный слой не знал, с каким репозиторием работает. Т.е. мы делаем имплементацию БД извне, пр. на уровне конфигурации. Так нам не надо менять сервисный слой, достаточно поправить конфигурацию.

interface UserRepo

    getUsers

class UserMongoDBRepo implements UserRepo

class UserService

    userRepo: UserRepo

    constructor (userRepo: UserRepo):

        this.userRepo = userRepo

    filterUserByAge(age: number)

```
        const users = this.UserRepo.getUsers
        // какая-то логика по фильтрации
        console.loge(users)
const userService = new UserService (new UserMongoDBRepo)
userService.filterUserByAge (age=13)
```

Мы можем передать в конструктор аргумент new UserMySqlDBRepo, то есть извне, а сам сервис не трогали.

Еще пример с паттерном Singleton (нужно, чб было только одно подключение к БД):

```
class Database
    url: string
    private static instance: Database
    constructor(url: string)
        if Database.instance
            return Database.instance
        this.url = Math.random()
        Database.instance = this
const db1 = new Database()
const db2 = new Database()
console.log(db1.url)
console.log(db2.url)
```

Выведет два одинаковых рандомных числа, т.е. вход один.

Паттерны проектирования

П. пр. – это часто встречающееся решение (концепция) определенной проблемы при проектировании архитектуры программ.

Польза:

- проверенные решения (не надо изобретать велосипед)
- стандартизация кода (проще для понимания, меньше просчетов, а значит, и костылей)
- общий программистский словарь (удобство понимания)

Критика:

- костыль для языка с недостаточным уровнем абстракции («слабого» в данной ситуации)
- могут быть неэффективны, если применяются без адаптации под конкретный проект
- новички начинают «пихать» паттерны везде, даже если можно сделать проще

Классификация:

- Самые простые и низкоуровневые – идиомы – реализуемы в рамках одного языка.

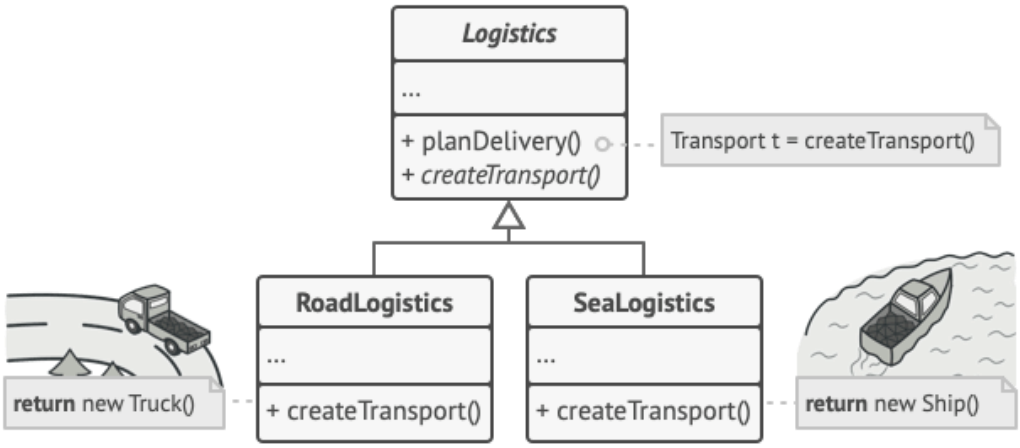
- Самые высокоуровневые – архитектурные паттерны – применимы для всех языков и служат для проектирования программ в целом, не отдельных элементов.
- Классификация по предназначению:
    - Порождающие – служат для быстрого создания объектов без внесения в программу лишних зависимостей
    - Структурные – показывают различные способы построения связей между объектами
    - Поведенческие – заботятся об эффективной коммуникации между объектами

Каталог паттернов:

- **Порождающие**
    - **Фабричный метод (Factory method)** – он же «виртуальный конструктор» - порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

      Решаемая проблема: если код задействует один класс (пр.: грузовик), а потом надо внедрить новый класс (пр.: пароход), то придется переписывать весь код, создавать условные операторы, которые работают в зависимости от класса.

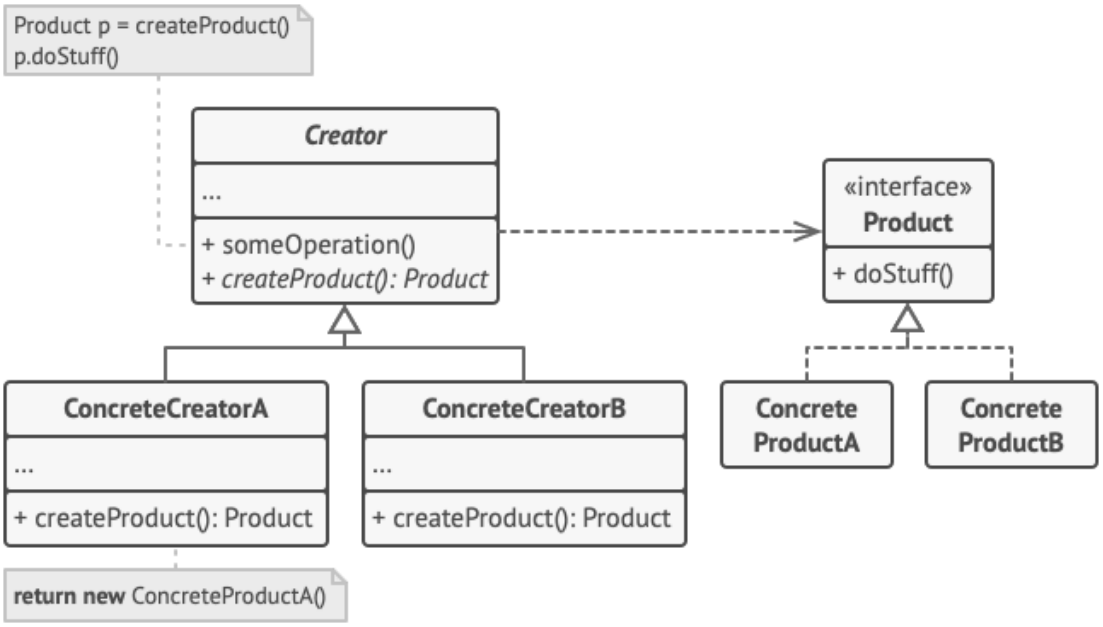
      Решение: объекты создаются не напрямую, а через вызов фабричного метода (при этом фабричный метод использует все те же операторы, например, new).



      Подклассы смогут производить объекты различных классов, следующих одному и тому же интерфейсу.

      Например, классы Грузовик и Судно реализуют интерфейс Транспорт с методом доставить. Каждый из этих классов реализует метод по-своему: грузовики везут грузы по земле, а суда — по морю. Фабричный метод в классе «Дорожной Логистики» вернёт объект-грузовик, а класс «Морской Логистики» — объект-судно.

      Для клиента фабричного метода нет разницы между этими объектами, так как он будет трактовать их как некий абстрактный Транспорт. Для него будет важно, чтобы объект имел метод доставить, а как конкретно он работает — не важно.



1. Продукт определяет общий интерфейс объектов, которые может произвести создатель и его подклассы.

2. Конкретные продукты содержат код различных продуктов, которые отличаются реализацией, но имеют общий интерфейс.

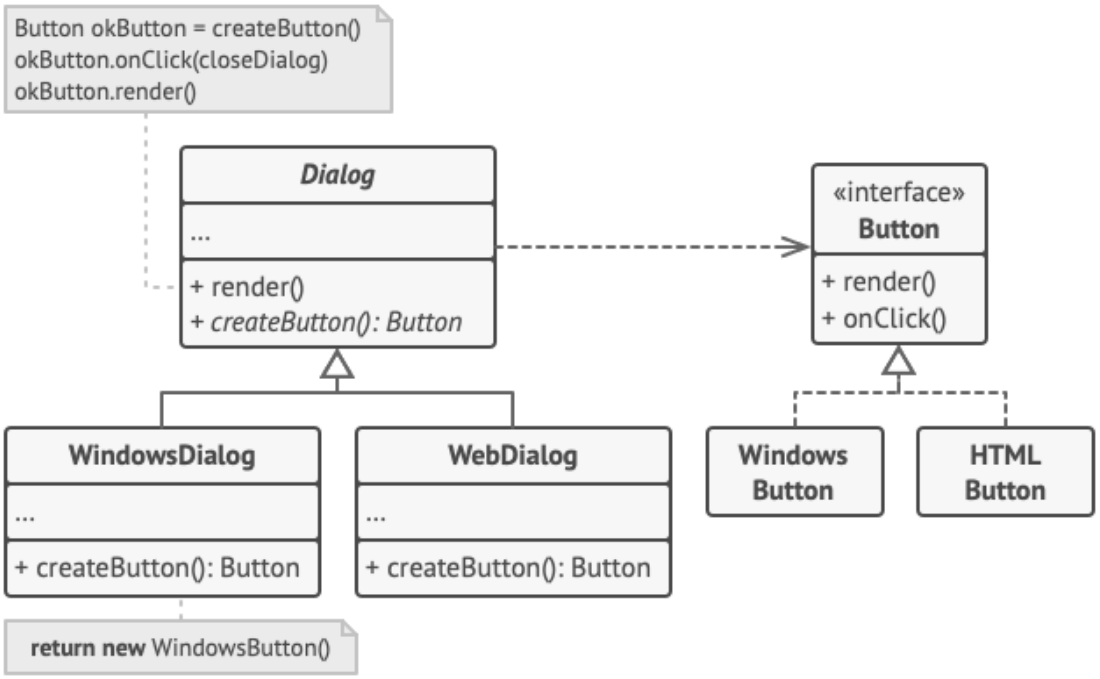
3. Создатель объявляет фабричный метод, который должен возвращать новые объекты продуктов. Важно, чтобы тип результата совпадал с общим интерфейсом продуктов. Обычно, ФМ – абстрактный, но может создавать и типовой продукт. Создатель, помимо создания продуктов, может выполнять и сторонние функции.

4. Конкретные создатели по-своему реализуют фабричный метод, производя те или иные конкретные продукты.

Фабричный метод можно и переписать так, чтобы возвращать существующие объекты из какого-то хранилища или кэша.

Пример:

В этом примере Фабричный метод помогает создавать кросс-платформенные элементы интерфейса, не привязывая основной код программы к конкретным классам элементов.



Фабричный метод объявлен в классе диалогов. Его подклассы относятся к различным операционным системам. Благодаря фабричному методу, вам не нужно переписывать логику диалогов под каждую систему. Подклассы могут наследовать почти весь код из базового диалога, изменяя типы кнопок и других элементов, из которых базовый код строит окна графического пользовательского интерфейса.

Базовый класс диалогов работает с кнопками через их общий программный интерфейс. Поэтому, какую вариацию кнопок ни вернул бы фабричный метод, диалог останется рабочим. Базовый класс не зависит от конкретных классов кнопок, оставляя подклассам решение о том, какой тип кнопок создавать.

Стоит применять:

- когда заранее не известны типы и зависимости объектов, с которыми должен работать код (код производства продуктов идет отдельно от остального кода, так что его можно расширять, не трогая основной код; например, чтобы добавить поддержку нового продукта, вам нужно создать новый подкласс и определить в нём фабричный метод, возвращая оттуда экземпляр нового продукта)

- когда вы хотите дать возможность пользователям расширять части вашего фреймворка или библиотеки (через наследование, создание и использование подклассов)

- когда вы хотите экономить системные ресурсы, повторно используя уже созданные объекты, вместо создания новых (т.к. в таком случае нужен метод, который будет и отдавать существующие объекты, и новые)

Шаги реализации:

1. Приведите все создаваемые продукты к общему интерфейсу.

2. В классе, который производит продукты, создайте пустой фабричный метод. В качестве возвращаемого типа укажите общий интерфейс продукта.

3. Затем пройдитесь по коду класса и найдите все участки, создающие продукты. Поочерёдно замените эти участки вызовами фабричного метода, перенося в него код создания различных продуктов.

4. В фабричный метод, возможно, придётся добавить несколько параметров, контролирующих, какой из продуктов нужно создать.

На этом этапе фабричный метод, скорее всего, будет выглядеть удручающе. В нём будет жить большой условный оператор, выбирающий класс создаваемого продукта. Но не волнуйтесь, мы вот-вот исправим это.

5. Для каждого типа продуктов заведите подкласс и переопределите в нём фабричный метод. Переместите туда код создания соответствующего продукта из суперкласса.

6. Если создаваемых продуктов слишком много для существующих подклассов создателя, вы можете подумать о введении параметров в фабричный метод, которые позволят возвращать различные продукты в пределах одного подкласса.

Например, у вас есть класс Почта с подклассами АвиаПочта и НаземнаяПочта, а также классы продуктов Самолёт, Грузовик и Поезд. Авиа соответствует Самолётам, но для НаземнойПочты есть сразу два продукта. Вы могли бы создать новый подкласс почты для поездов, но проблему можно решить и по-другому. Клиентский код может передавать в фабричный метод НаземнойПочты аргумент, контролирующий тип создаваемого продукта.

7. Если после всех перемещений фабричный метод стал пустым, можете сделать его абстрактным. Если в нём что-то осталось — не беда, это будет его реализацией по умолчанию.

Плюсы и минусы:

+ Избавляет класс от привязки к конкретным классам продуктов.

+ Выделяет код производства продуктов в одно место, упрощая поддержку кода.

+ Упрощает добавление новых продуктов в программу.

+ Реализует принцип открытости/закрытости.

- Может привести к созданию больших параллельных иерархий классов, так как для каждого класса продукта надо создать свой подкласс создателя.

Отношения с другими паттернами:

- Многие архитектуры начинаются с применения Фабричного метода (более простого и расширяемого через подклассы) и эволюционируют в сторону Абстрактной фабрики, Прототипа или Строителя (более гибких, но и более сложных).
- Классы Абстрактной фабрики чаще всего реализуются с помощью Фабричного метода, хотя они могут быть построены и на основе Прототипа.
- Фабричный метод можно использовать вместе с Итератором, чтобы подклассы коллекций могли создавать подходящие им итераторы.
- Прототип не опирается на наследование, но ему нужна сложная операция инициализации. Фабричный метод, наоборот, построен на наследовании, но не требует сложной инициализации.
- Фабричный метод можно рассматривать как частный случай Шаблонного метода. Кроме того, Фабричный метод нередко бывает частью большого класса с Шаблонными методами.

Примеры реализации - см. в источнике.

```
from __future__ import annotations
from abc import ABC, abstractmethod


class Creator(ABC):
    """
    Класс Создатель объявляет фабричный метод, который должен
возвращать объект
    класса Продукт. Подклассы Создателя обычно предоставляют
реализацию этого
    метода.
    """

    @abstractmethod
    def factory_method(self):
        """
        Обратите внимание, что Создатель может также обеспечить
реализацию
        фабричного метода по умолчанию.
        """
        pass

    def some_operation(self) -> str:
        """
        Также заметьте, что, несмотря на название, основная обязанность
        Создателя не заключается в создании продуктов. Обычно он
содержит
        некоторую базовую бизнес-логику, которая основана на объектах
Продуктов,
        возвращаемых фабричным методом. Подклассы могут косвенно
изменять эту
        бизнес-логику, переопределяя фабричный метод и возвращая из
него другой
        тип продукта.
        """

        # Вызываем фабричный метод, чтобы получить объект-продукт.
        product = self.factory_method()

        # Далее, работаем с этим продуктом.
        result = f"Creator: The same creator's code has just worked with {product.operation()}"

        return result


"""
Конкретные Создатели переопределяют фабричный метод для того,
чтобы изменить тип
результирующего продукта.
"""


class ConcreteCreator1(Creator):
    """
    Обратите внимание, что сигнатура метода по-прежнему использует
тип
    абстрактного продукта, хотя фактически из метода возвращается
конкретный
    продукт. Таким образом, Создатель может оставаться независимым от
конкретных
    классов продуктов.
    """

    def factory_method(self) -> Product:
        return ConcreteProduct1()


class ConcreteCreator2(Creator):
    def factory_method(self) -> Product:
        return ConcreteProduct2()
```

```
class Product(ABC):
    """
    Интерфейс Продукта объявляет операции, которые должны выполнять все
    конкретные продукты.
    """

    @abstractmethod
    def operation(self) -> str:
        pass


"""
Конкретные Продукты предоставляют различные реализации интерфейса Продукта.
"""


class ConcreteProduct1(Product):
    def operation(self) -> str:
        return "{Result of the ConcreteProduct1}"


class ConcreteProduct2(Product):
    def operation(self) -> str:
        return "{Result of the ConcreteProduct2}"


def client_code(creator: Creator) -> None:
    """
    Клиентский код работает с экземпляром конкретного создателя, хотя и через
    его базовый интерфейс. Пока клиент продолжает работать с создателем через
    базовый интерфейс, вы можете передать ему любой подкласс создателя.
    """

    print(f"Client: I'm not aware of the creator's class, but it still works.¥n"
          f"{creator.some_operation()}", end="")


if __name__ == "__main__":
    print("App: Launched with the ConcreteCreator1.")
    client_code(ConcreteCreator1())
    print("¥n")

    print("App: Launched with the ConcreteCreator2.")
    client_code(ConcreteCreator2())
```

Output.txt: Результат выполнения

```
App: Launched with the ConcreteCreator1.
Client: I'm not aware of the creator's class, but it still works.
Creator: The same creator's code has just worked with {Result of the ConcreteProduct1}

App: Launched with the ConcreteCreator2.
Client: I'm not aware of the creator's class, but it still works.
Creator: The same creator's code has just worked with {Result of the ConcreteProduct2}
```

- **Абстрактная фабрика (Abstract factory)**

Это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

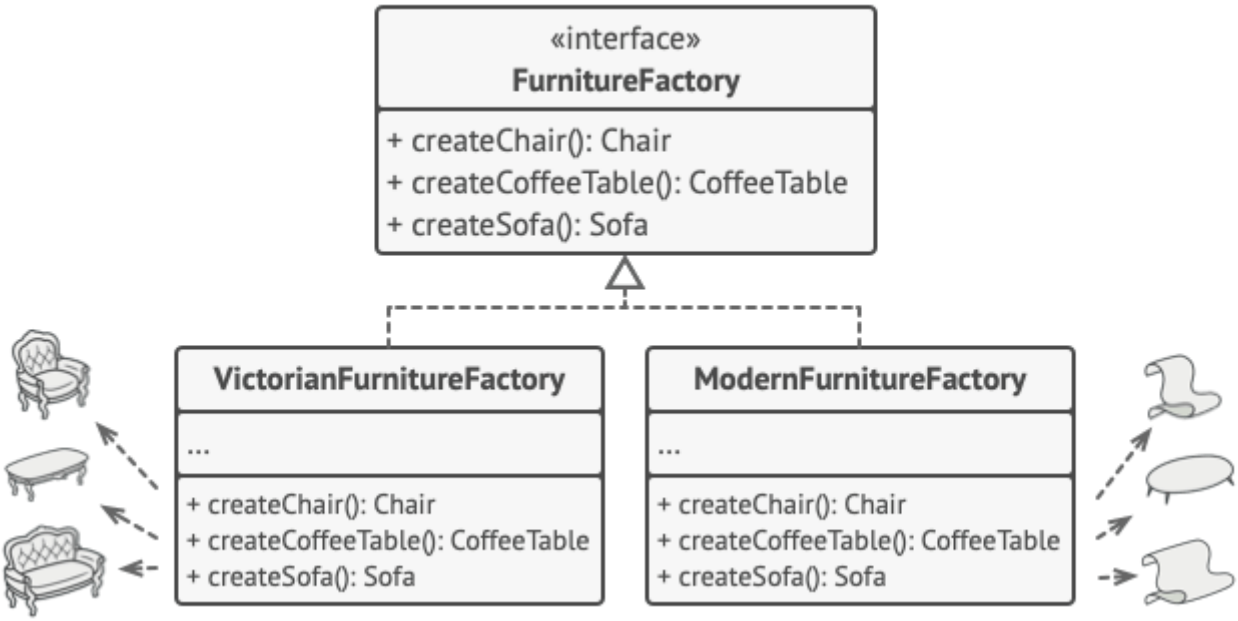
Решаемая проблема: пример – мебельный магазин. Есть:

- семейство зависимых продуктов (кресло+диван+столик)
- несколько вариаций этого семейства (стили модерн, классика и т.д.)

Нужен способ создавать объекты продуктов, чтобы они сочетались с другими продуктами того же семейства, не внося изменений в существующий код при создании новых продуктов или семейств.

Решение: выделяем общие интерфейсы для отдельных продуктов, составляющих семейства (у кресел свой одинаковый интерфейс и т.д.).

Создаем абстрактную фабрику – общий интерфейс, который содержит методы создания всех продуктов семейства. Эти операции должны возвращать абстрактные типы продуктов, представленные интерфейсами (кресла, диваны и т.д.)



Для каждой вариации продуктов мы должны создать свою собственную фабрику, реализовав абстрактный интерфейс. Фабрики создают продукты одной вариации (ФабрикаМодерн – КреслаМодерн, ДиванМодерн и т.д.).

Клиентский код должен работать как с фабриками, так и с продуктами только через их общие интерфейсы, что позволяет подавать в классы любой тип фабрики и производить любые продукты.

Кто же создает объекты конкретных фабрик, если клиентский код создает только интерфейсы самих фабрик? Обычно программа создает конкретный объект фабрики при запуске, причем тип фабрики выбирается, исходя из параметров окружения или конфигурации.

    - Строитель (Builder)
    - Прототип (Prototype)
    - Одиночка (Singleton)

- Структурные
    - Адаптер (Adapter)
    - Мост (Bridge)
    - Компоновщик (Composite)
    - Декоратор (Decorator)
    - Фасад (Façade)
    - Легковес (Flyweight)
    - Заместитель (Proxy)

- Поведенческие
    - Цепочка обязанностей (Chain of Responsibility)
    - Команда (Command)
    - Итератор (Iterator)

- Посредник (Mediator)
- Снимок (Memento)
- Наблюдатель (Observer)
- Состояние (State)
- Стратегия (Strategy)
- Шаблонный метод (Template Method)
- Посетитель (Visitor)

JSON

JavaScript Object Notation – текстовый формат обмена данными, основанный на JavaScript. Используется в REST.API. Альтернатива – XML.

Значения: числа, строки, массивы, JSON-объекты, литералы (лог. зн. true, false, null).

JSON-объект – неупорядоченное множество пар {“ключ” : значение}, взаимодействие с которыми происходит, как со словарями.

Ключ – название параметра, который мы передаем серверу. Он служит маркером для принимающей стороны запрос системы, чтобы она поняла, что мы ей отправили.

Ключ всегда строка. Строки в кавычках. Порядок – любой.

Обратиться к ключу можно через:

- get() – если ключа нет, то пустое значение (NoneType), также имеет второй передаваемый аргумент (возвращается, если ключа нет)
- dict[key] – если ключа нет, то KeyError

JSON-массив: [“MALE”, “FEMALE”].

Нет ключей, набор значений, обращение по номеру элемента. Нельзя менять местами данные – упорядоченное множество.

Well Formed JSON – синтаксически правильный.

Правила:

1. Данные написаны в виде пар «ключ: значение»
2. Данные разделены запятыми
3. Объект находится внутри {фигурных скобок}
4. Массив – внутри [квадратных]

Для проверки синтаксиса - JSON Validator, JSON Formatter (он еще и форматирует).

Бизнес-логика

Это правила работы программы, взаимодействия ее внутренних составляющих, а также взаимодействие со внешним элементами.

---------- не по теме, но интересно----------

- [ ] How does the Internet work?
- [ ] How does a computer work?
- [ ] What is “back-end”?
- [ ] Hosting
- [ ] API
- [ ] Server
- [ ] DNS

- [ ] HTTP
- [ ] Browser
- [ ] Domain name
- [ ] About Python
- [ ] IDE (e.g. PyCharm, VS Code, Visual Studio, Intellij Idea)
- [ ] Terminal (PowerShell, Bash, Zsh)
- [ ] Framework (Django)
- [ ] Data Bases (PostgreSQL, MySQL, MS SQL, Azure Cloud, SQL; MongoDB, Redis, Cassandra, AWS, Firebase)
- [ ] Data Structures
- [ ] Authentication (OAuth 2.0, JWT; providers - Auth0, Firebase)
- [ ] Patterns of Programming
- [ ] Testing, QA (frameworks)
- [ ] Packet Managers (NuGet, nmp, yarn)
- [ ] Version Control Systems (git, GitHub, TFS, BitBucket)
- [ ] Web-services
- [ ] Network Protocols
- [ ] UI-frameworks
- [ ] OOP