Направления подготовки: «Информатика и вычислительная техника» и «Информационные системы и технологии»

Профили образовательных программ:
«Системотехника и автоматизация проектирования в строительстве»
«Системотехника и информационные технологии управления в строительстве»

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Тема 4. Архитектура современных операционных систем

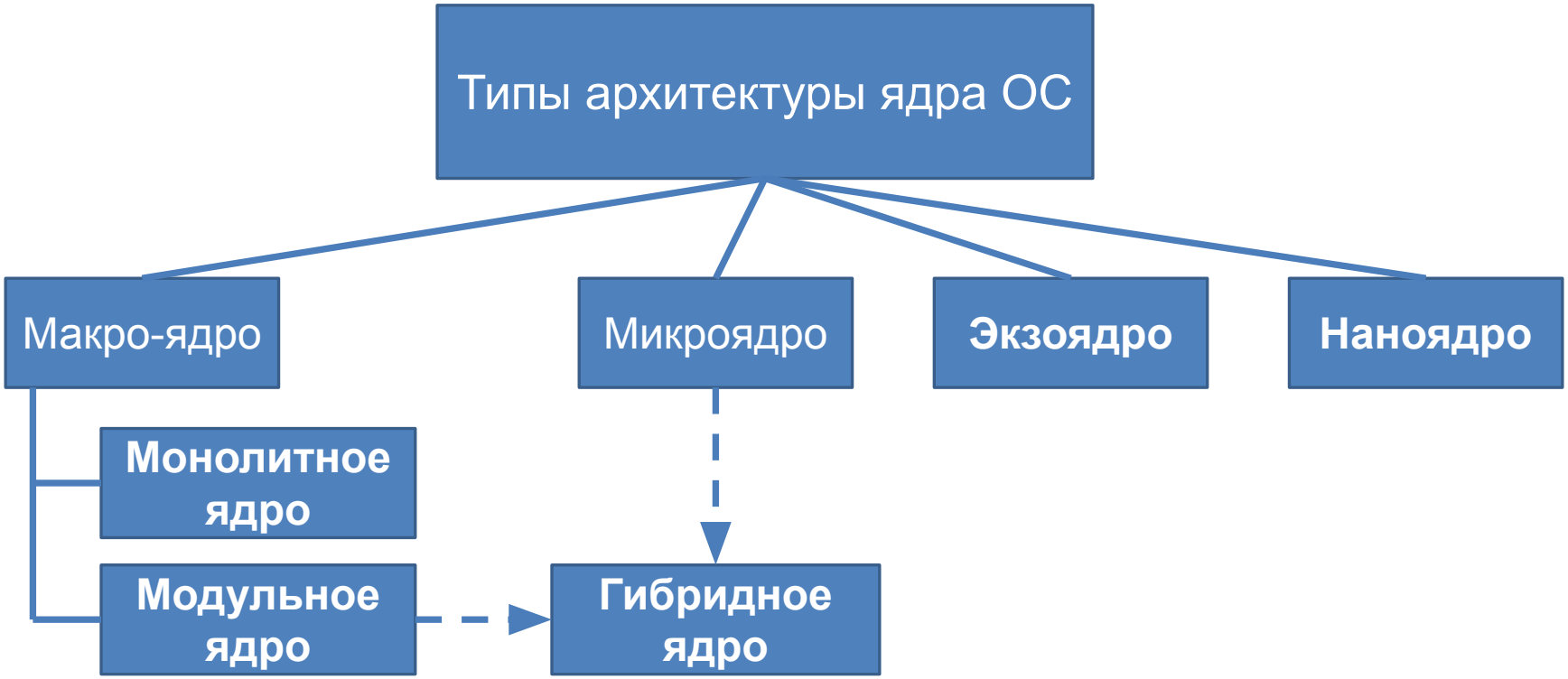
Кафедра «Информационных систем, технологий и автоматизации в строительстве» (ИСТАС)
Составитель: доцент, доц., к.т.н. Иванов Н.А.

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ СТРОИТЕЛЬНЫЙ УНИВЕРСИТЕТ
Москва 2021

Содержание разделов курса

№	Наименование раздела дисциплины	Тема и содержание лекций
1	Основные понятия	Тема 1. Основные сведения об операционных системах. История развития операционных систем. Доступ к ЭВМ: локальный непосредственный, через оператора, удаленный. Режимы решения задач в ЭВМ: пакетный, индивидуальный, разделение времени, реального времени. Операционная система как часть вычислительной системы. Принципы построения вычислительных систем.
		Тема 2. Назначение и функции операционной системы. Требования к современным операционным системам. Основные функциональные компоненты ОС. Требования, предъявляемые к современным ОС. Расширяемость. Переносимость. Совместимость и множественные прикладные среды. Безопасность.
		Тема 3. Классификация операционных систем. Особенности областей использования. Особенности алгоритмов управления ресурсами. Особенности аппаратных платформ. Особенности методов построения ОС. Сетевые операционные системы.
		Тема 4. Архитектура современных операционных систем. Модульная структура построения ОС. Ядро и модули расширения ядра. Режимы работы аппаратуры. Многослойная структура ядра операционной системы. Микроядерная архитектура.

Классификация по типу архитектуры ядра ОС



Архитектура ОС

Любая сложная система должна иметь понятную структуру,
то есть разделяться на *модули — части,*
имеющие вполне законченное функциональное назначение с четко оговоренными правилами взаимодействия.

Функциональная сложность операционной системы неизбежно приводит к сложности ее **архитектуры**, под которой понимают **структурную организацию ОС на основе различных программных модулей**.

Большинство современных операционных систем представляют собой хорошо структурированные *модульные системы*, способные к развитию, расширению и переносу на новые платформы.

Ядро и вспомогательные модули ОС

Какой-либо ***единой структуры*** для операционных систем ***не существует***, но *существуют универсальные подходы к их структурированию*.

Наиболее общим подходом к структуризации операционной системы является разделение всех ее модулей на две группы:
ядро — модули, выполняющие основные функции ОС;
модули расширения — модули, выполняющие вспомогательные функции ОС.

Модули ядра ОС

Модули ядра выполняют *базовые функции ОС*, решающие внутрисистемные задачи организации вычислительного процесса, такие как *переключение контекстов*, *загрузка/выгрузка страниц*, *обработка прерываний*.
Эти функции недоступны для приложений

Другой класс функций ядра служит для поддержки приложений, создавая для них так называемую *прикладную программную среду*.
Приложения могут обращаться к ядру с запросами — системными вызовами — для выполнения тех или иных действий, например для открытия и чтения файла, вывода графической информации на дисплей, получения системного времени и т. д.
Функции ядра, которые могут вызываться приложениями, образуют *интерфейс прикладного программирования — API*.

Модули ядра ОС

Функции, выполняемые модулями ядра, являются наиболее часто используемыми функциями операционной системы, поэтому скорость их выполнения определяет производительность всей системы в целом.

Для обеспечения высокой скорости работы ОС все модули ядра или большая их часть постоянно находятся в оперативной памяти, то есть являются *резидентными*.

Обычно ядро оформляется в виде программного модуля некоторого *специального формата*, отличающегося от формата пользовательских приложений.

Модули расширения ОС

Модули расширения ОС выполняют весьма полезные, но *менее обязательные* функции.
Например, к таким вспомогательным модулям могут быть отнесены программы архивирования данных на магнитной ленте, дефрагментации диска, вывода сведений о системе.

Вспомогательные модули оформляются либо в виде приложений, либо в виде библиотек процедур.

Поскольку некоторые компоненты ОС оформлены как обычные приложения, то есть в виде исполняемых модулей стандартного для данной ОС формата, то часто бывает очень сложно провести четкую грань между операционной системой и приложениями.

Решение о том, является ли какая-либо программа частью ОС или нет, принимает фирма-производитель ОС.

Группы модулей расширения ОС

Модули расширения ОС

утилиты — программы, решающие отдельные задачи управления и сопровождения компьютерной системы, такие, например, как программы сжатия дисков, архивирования данных на магнитную ленту;

программы предоставления пользователю дополнительных услуг — программы, предоставляющие пользователю альтернативный способ работы с файлами и каталогами, устанавливающие специальный пользовательский интерфейс, обеспечивающие антивирусную безопасность ОС и защиту системы от несанкционированного доступа и т.д.;

библиотеки процедур различного назначения

Для выполнения своих задач модули расширения ОС обращаются к функциям ядра посредством системных вызовов.

Преимущества деления модулей ОС на ядро и модули расширения

Легкая расширяемость ОС.

Чтобы добавить новую высокоуровневую функцию достаточно разработать новое приложение, и при этом не требуется модифицировать часть кода, образующую ядро системы.
Внесение изменений в функции ядра обычно происходит гораздо сложнее, и сложность эта зависит от структурной организации самого ядра.
В некоторых случаях исправление ядра может потребовать его полной перекомпиляции.

Экономия оперативной памяти компьютера.

Модули ОС, оформленные в виде утилит, системных обрабатывающих программ и библиотек, обычно загружаются в оперативную память только на время выполнения своих функций, то есть являются транзитными.

Ядро и режимы работы аппаратуры

Для надежного управления ходом выполнения приложений ***операционная система должна иметь*** *по отношению к приложениям определенные* ***привилегии***.
Иначе некорректно работающее приложение может вмешаться в работу ОС и, например, разрушить часть ее кодов.

Операционная система ***должна обладать исключительными полномочиями*** также для того, чтобы играть **роль арбитра** в споре приложений за ресурсы компьютера в мультипрограммном режиме.

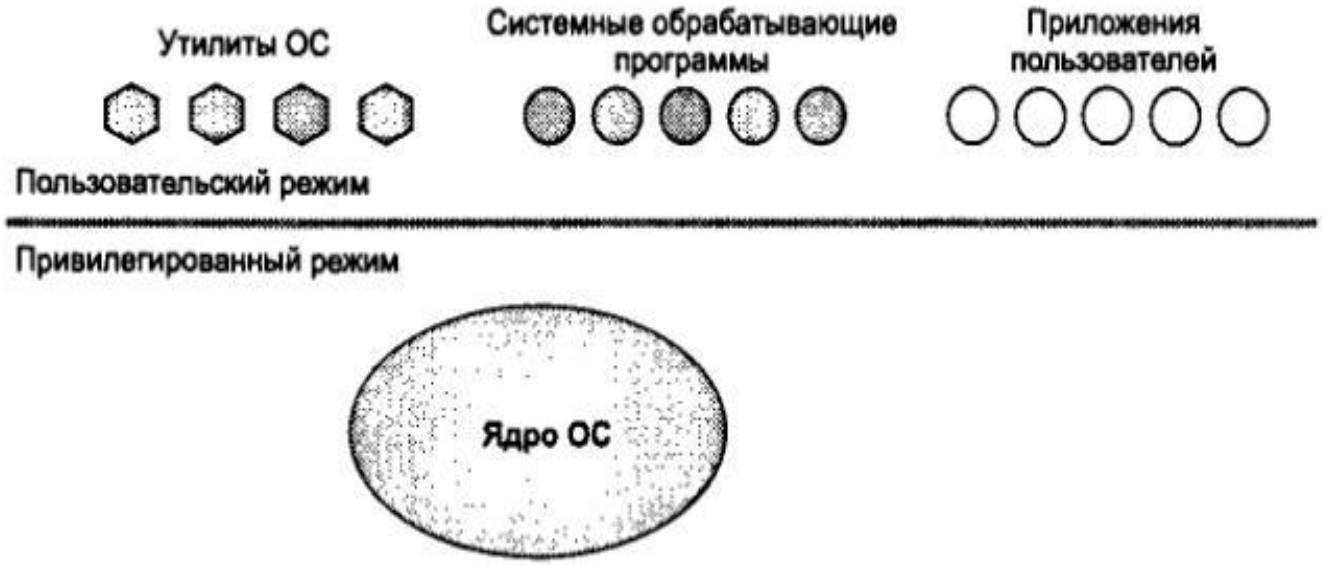
Ни одно ***приложение не должно иметь возможности*** без ведома ОС получать дополнительную область памяти,
занимать процессор дольше разрешенного ОС периода времени,
непосредственно управлять совместно используемыми внешними устройствами.

Ядро и режимы работы аппаратуры

Чтобы обеспечить привилегии операционной системе требуются специальные средства аппаратной поддержки.

Аппаратура компьютера должна поддерживать как минимум два режима работы — *пользовательский режим (user mode)* и *привилегированный режим*, который также называют *режимом ядра (kernel mode)* или *режимом супервизора (supervisor mode)*.

Подразумевается, что операционная система или некоторые ее части работают в привилегированном режиме, а приложения — в пользовательском режиме



Аппаратная и программная поддержка уровней привилегий

Возможность защиты кодов и данных операционной системы за счет выполнения функций ядра в привилегированном режиме является важным свойством архитектуры ОС, основанной на ядре.

Между количеством уровней привилегий, реализуемых аппаратно, и количеством уровней привилегий, поддерживаемых ОС, нет прямого соответствия.

Так, на базе четырех уровней, обеспечиваемых процессорами компании Intel, операционная система OS/2 строила *трехуровневую* систему привилегий, а операционные системы Windows NT, UNIX и некоторые другие ограничиваются *двухуровневой* системой.

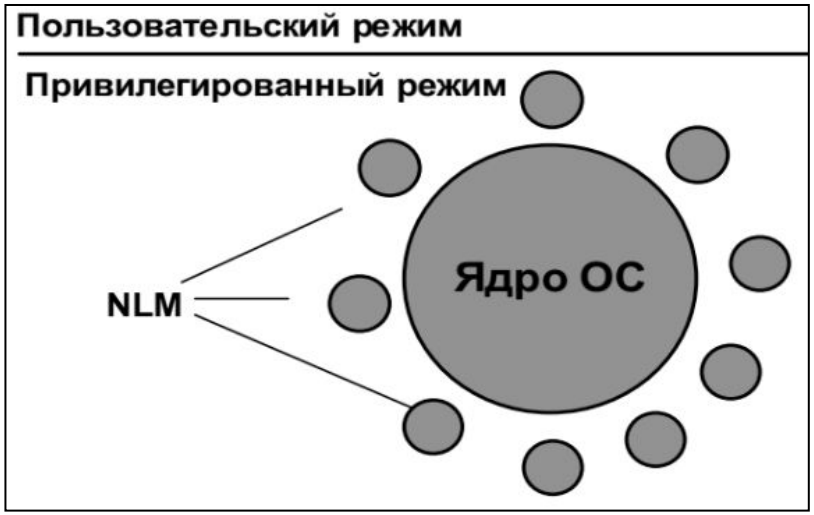
Архитектура ОС, основанная на привилегированном ядре и приложениях пользовательского режима, является, по существу, классической. Ее использовали и используют многие популярные операционные системы, в том числе многочисленные версии UNIX, VAX VMS, IBM OS/390, OS/2, Windows NT.

Смена режимов при выполнении системного вызова к привилегированному ядру

Системный вызов привилегированного ядра инициирует *переключение процессора из пользовательского режима в привилегированный*, а при возврате к приложению — переключение из привилегированного режима в пользовательский.

Во всех типах процессоров из-за дополнительной двукратной задержки переключения переход на процедуру со **сменой режима** выполняется медленнее, чем вызов процедуры **без смены режима**.

Исключения из правил. ОС NetWare

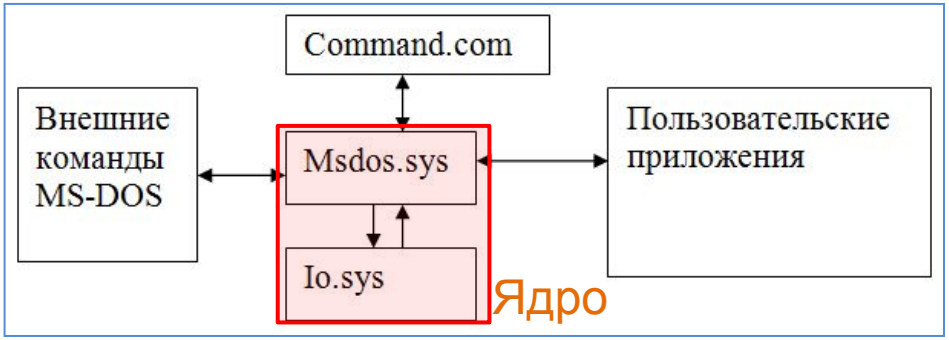


В некоторых случаях разработчики ОС отступают от классического варианта архитектуры, организуя работу ядра и приложений в одном и том же режиме. Так, *сетевая операционная система NetWare* компании Novell использовала привилегированный режим процессоров Intel x86/ Pentium как для работы ядра, так и для работы своих специфических приложений — NLM-модулей (***N**etWare **L**oadable **M**odule* – загружаемый модуль NetWare).

При таком построении ОС обращения приложений к ядру выполнялись быстрее, так как нет переключения режимов, однако при этом *отсутствовала надежная аппаратная защита* памяти, занимаемой модулями ОС, от некорректно работающего приложения.

Разработчики NetWare пошли на такое потенциальное снижение надежности своей операционной системы, поскольку ограниченный набор ее специализированных приложений позволял компенсировать этот архитектурный недостаток за счет тщательной отладки каждого приложения.

Исключения из правил. MS-DOS



В одном режиме работали также ядро и приложения тех операционных систем, которые были разработаны *для процессоров, вообще не поддерживающих привилегированного режима работы*. Наиболее популярным процессором такого типа был процессор Intel 8088/86, послуживший основой для персональных компьютеров IBM.

Операционная системы MS-DOS, разработанная для этих компьютеров фирмой Microsoft, состояла из двух модулей msdos.sys и io.sys, составлявших по своей сути ядро системы, к которым с системными вызовами обращались командный интерпретатор command.com, системные утилиты и приложения.

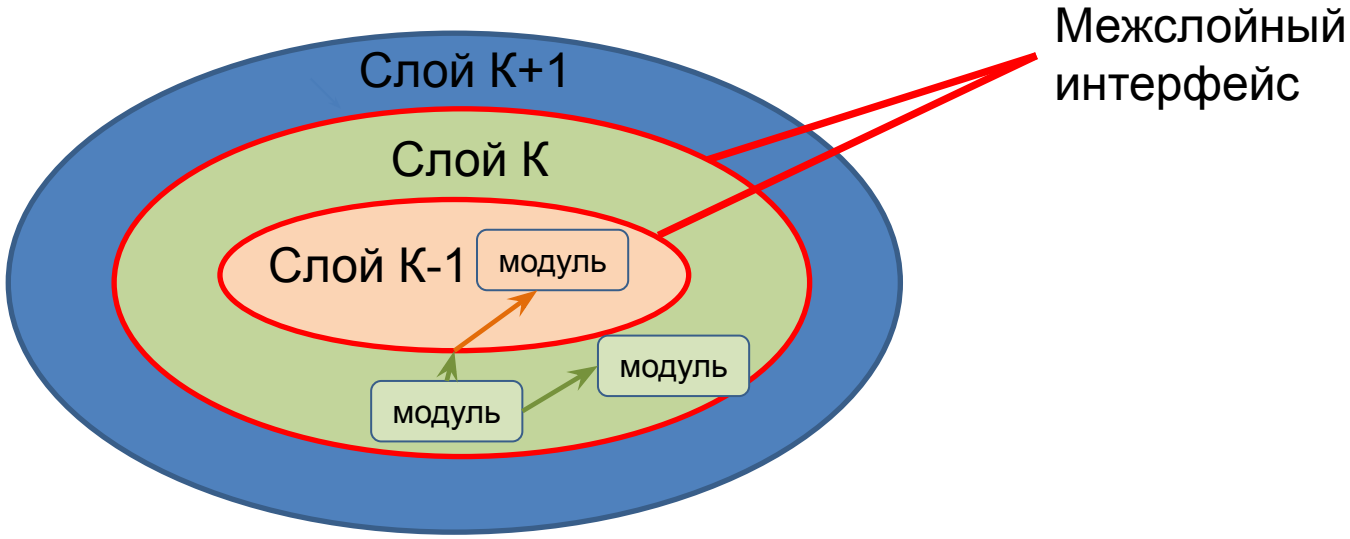
Некорректно написанные приложения вполне могли разрушить основные модули MS-DOS, что иногда и происходило.

Область использования MS-DOS и многих подобных ей ранних операционных систем для персональных компьютеров не предъявляла высоких требований к надежности ОС.

Как же устроено ядро?

Многослойные подход как инструмент декомпозиции сложных систем

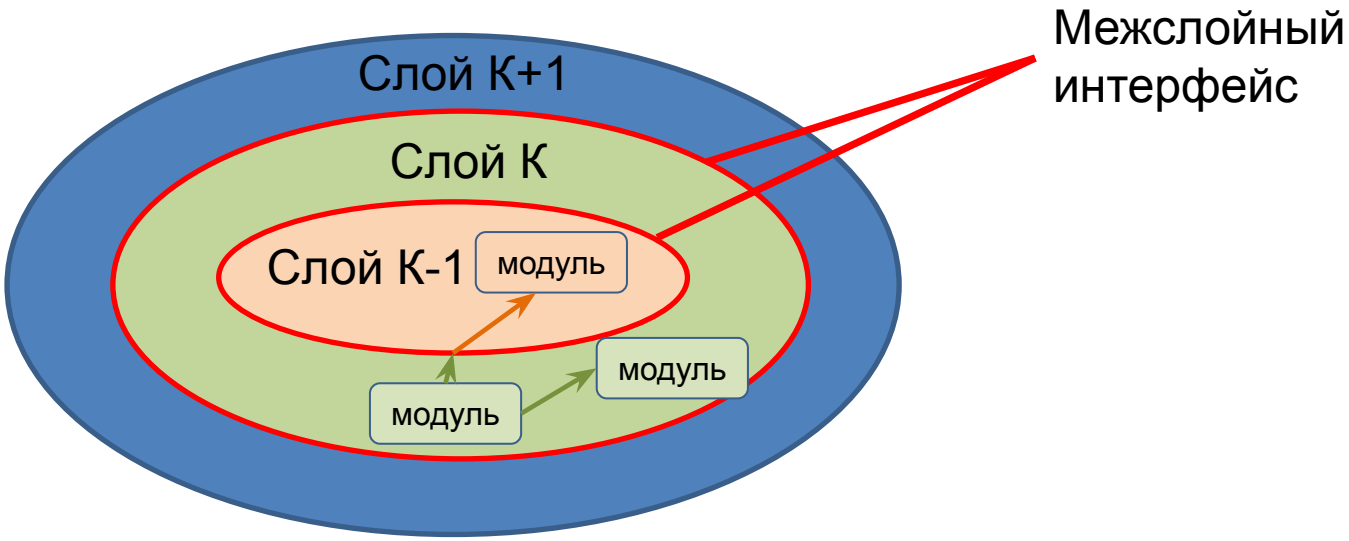
Универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе и вычислительных систем, является **многослойный подход**.
В соответствии с этим подходом *система состоит из иерархии слоев*.
Каждый слой обслуживает вышележащий слой, выполняя для него некоторый набор функций, которые образуют *межслойный интерфейс*.
На основе функций нижележащего слоя следующий (вверх по иерархии) слой строит свои функции — более сложные и более мощные, которые, в свою очередь, оказываются примитивами для создания еще более мощных функций вышележащего слоя.
Строгие правила касаются *только взаимодействия между слоями системы*, а *между модулями внутри слоя связи могут быть произвольными*.
Отдельный модуль может выполнить свою работу либо самостоятельно, либо обратиться к *другому модулю своего слоя*, либо обратиться за помощью *к нижележащему слою* через межслойный интерфейс.



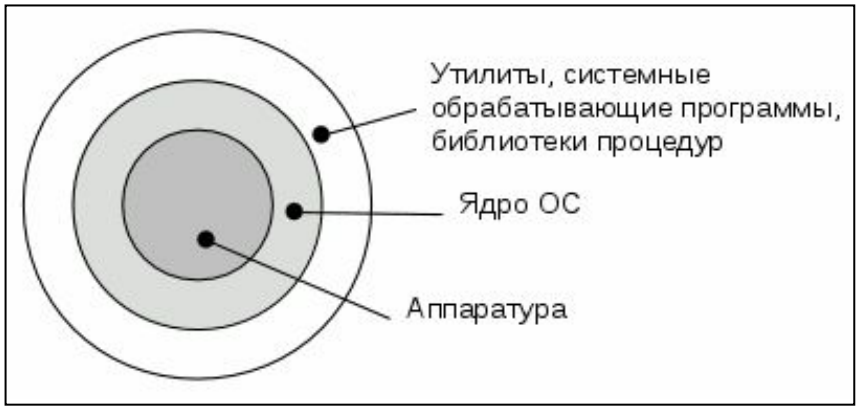
Достоинства многослойные подхода декомпозиции сложных систем

Существенно упрощается разработка системы, так как можно первоначально определить «сверху вниз» функции слоев и межслойные интерфейсы, а затем при детальной реализации постепенно наращивать мощность функций слоев, двигаясь «снизу вверх».

Кроме того, при модернизации системы можно изменять модули внутри слоя без необходимости производить какие-либо изменения в остальных слоях, если при этих внутренних изменениях межслойные интерфейсы остаются в силе.



Многослойные модели вычислительной и операционной систем



***Трехслойная** структура*
вычислительной системы

Вычислительную систему, работающую под управлением ОС на базе ядра, можно рассматривать как *систему из 3-х иерархически упорядоченных слоев*.

При такой организации ВС **модули расширения ОС** и **приложения** могут взаимодействовать с аппаратурой **только** через слой ядра ОС.

***6-слойная** структура*
аппаратной части вычислительной системы и ядра операционной системы

Ядро ОС, представленное 4-мя слоями, взаимодействует с аппаратурой только через слой средств аппаратной поддержки ОС, относящийся к аппаратной части ВС!!!

Назначения слоев структуры. Типовые средства аппаратной поддержки

6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

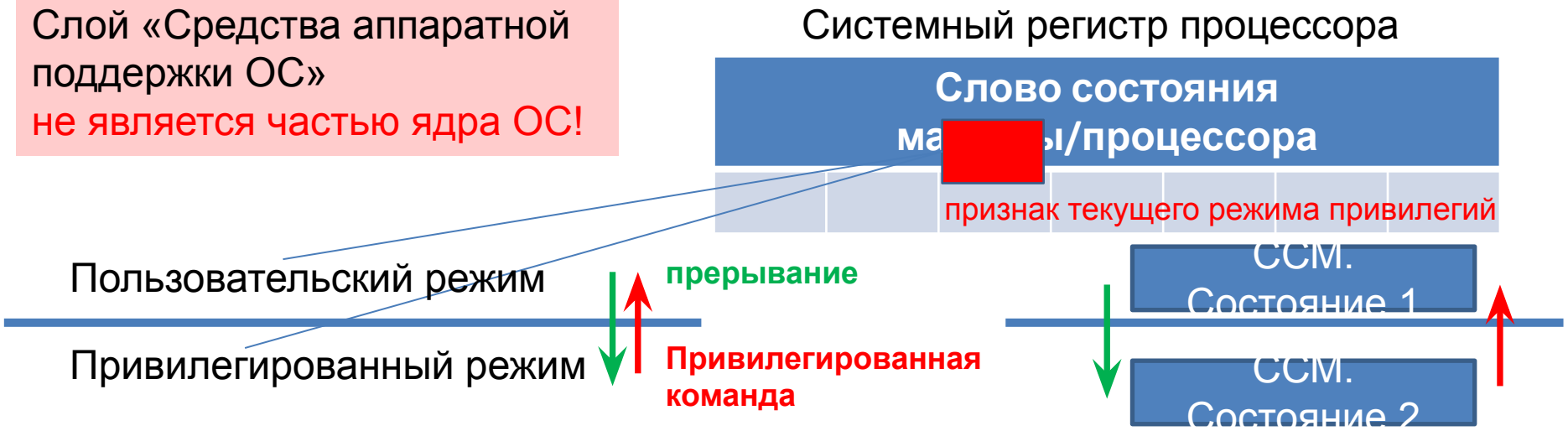
Слой «Средства аппаратной поддержки ОС» не является частью ядра ОС!

Он включает в себя ***средства аппаратной поддержки*** ОС, которые прямо участвуют в организации вычислительных процессов.

- средства поддержки привилегированного режима
- средства трансляции адресов
- средства переключения контекстов процессов
- система прерываний
- Устройства службы времени
- средства защиты областей памяти

Средства поддержки привилегированного режима

Слой «Средства аппаратной поддержки ОС»
не является частью ядра ОС!



Смена режима привилегий – это всегда переключение состояния процессора

Число уровней привилегий может быть *разным* у разных типов процессоров.
Наиболее часто используются:
2 уровня (*ядро*-*пользователь*)
4 (ядро- супервизор- выполнение- пользователь у платформы VAX
или
0-1-2-3 у процессоров Intel x86/Pentium).

В обязанности средств поддержки привилегированного режима входит выполнение *проверки допустимости выполнения* активной программой инструкций процессора при *текущем уровне привилегий.*

Средства трансляции адресов

Средства трансляции адресов выполняют операции *преобразования виртуальных адресов*, которые содержатся в кодах процесса, в *адреса физической памяти.*

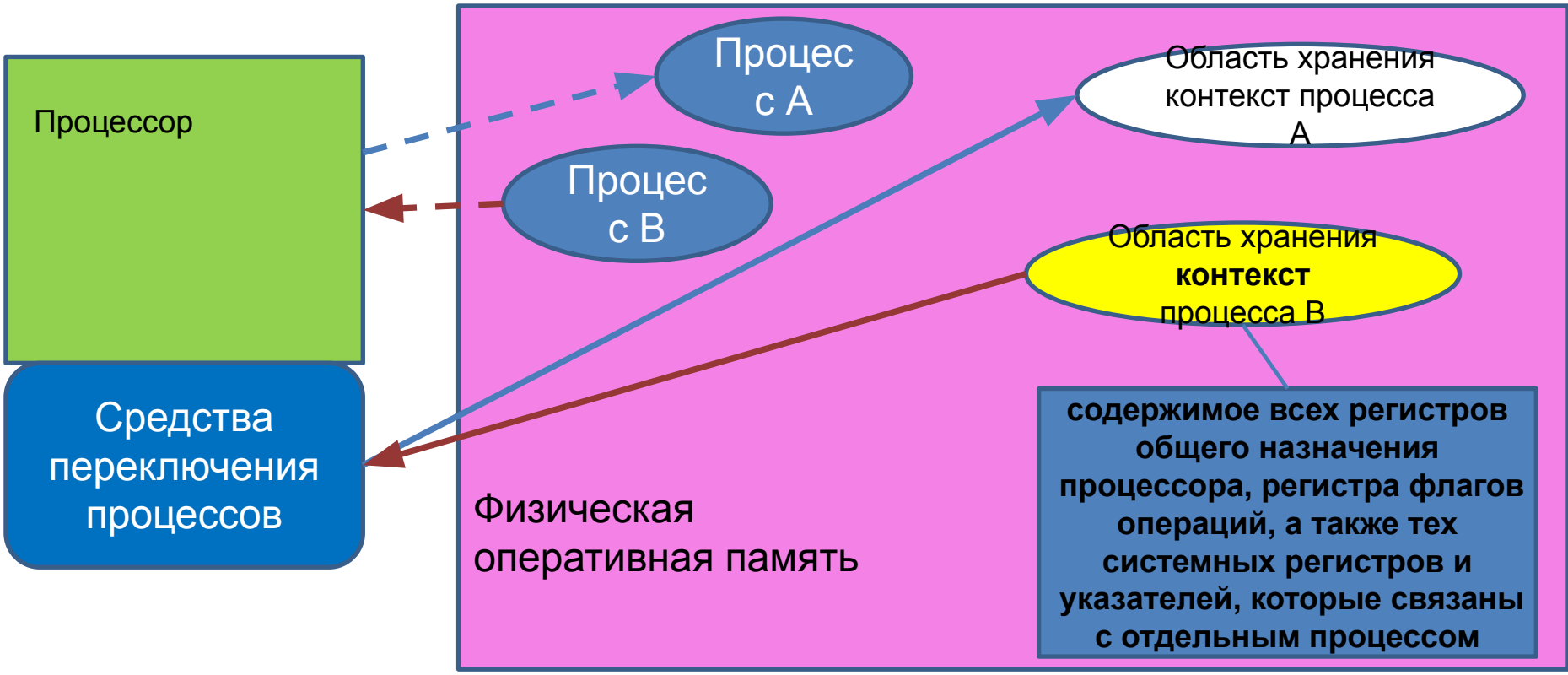
Таблицы, предназначенные для трансляции адресов, обычно имеют большой объем, поэтому для их хранения используются области оперативной памяти, а аппаратура процессора содержит только *указатели* на эти области.

Средства трансляции адресов *используют указатели для доступа к элементам таблиц* и аппаратного выполнения алгоритма преобразования адреса, что значительно ускоряет процедуру трансляции по сравнению с ее чисто программной реализацией.

Средства переключения процессов

Слой «Средства аппаратной поддержки ОС»
не является частью ядра ОС!

Средства переключения процессов предназначены для быстрого *сохранения контекста приостанавливаемого процесса* и *восстановления контекста процесса, который становится активным*.



Для хранения контекстов приостановленных процессов обычно используются области оперативной памяти, которые адресуются указателями процессора.

Система прерываний

Система прерываний позволяет компьютеру реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода-вывода, быстро переходить с одной программы на другую.

Механизм прерываний нужен для того, чтобы оповестить процессор о возникновении в вычислительной системе *некоторого непредсказуемого события или события, которое не синхронизировано с циклом работы процессора*.

В большинстве моделей процессоров отрабатываемый аппаратурой переход на процедуру обработки прерываний сопровождается заменой слова состояния машины (или даже всего контекста процесса), что позволяет одновременно с переходом по нужному адресу выполнить переход в привилегированный режим.
После завершения обработки прерывания обычно происходит возврат к исполнению прерванного кода.

Устройства службы времени

Системные часы — работающая на батареях электронная схема, — которые ведут независимый отсчет времени и календарной даты

Системный таймер — быстродействующий регистр-счетчик, необходим операционной системе для выдержки интервалов времени

В регистр таймера программно загружается значение требуемого интервала в условных единицах, из которого затем автоматически с определенной частотой начинает вычитаться по единице.
Частота «тиков» таймера, как правило, тесно связана с частотой тактового генератора процессора.
При достижении нулевого значения счетчика таймер инициирует прерывание, которое обрабатывается процедурой операционной системы.

Прерывания от системного таймера используются ОС в первую очередь для слежения за тем, как отдельные процессы расходуют время процессора.

Средства защиты областей памяти

Средства защиты областей памяти обеспечивают на аппаратном уровне проверку возможности программного кода осуществлять с данными определенной области памяти такие операции, как чтение, запись или выполнение (при передачах управления).

Функции аппаратуры по защите памяти состоят в сравнении уровней привилегий текущего кода процессора и сегмента памяти, к которому производится обращение.

Если аппаратура компьютера поддерживает механизм трансляции адресов, то средства защиты областей памяти встраиваются в этот механизм.

Описание назначения слоев структуры

6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

Машинно-зависимые компоненты ОС. Этот слой образуют программные модули ядра ОС, в которых отражается специфика аппаратной платформы компьютера.

В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры.

Это позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов аппаратных платформ, поддерживаемых данной ОС.

Примером экранирующего слоя может служить слой HAL операционной системы Windows NT.

Описание назначения слоев структуры

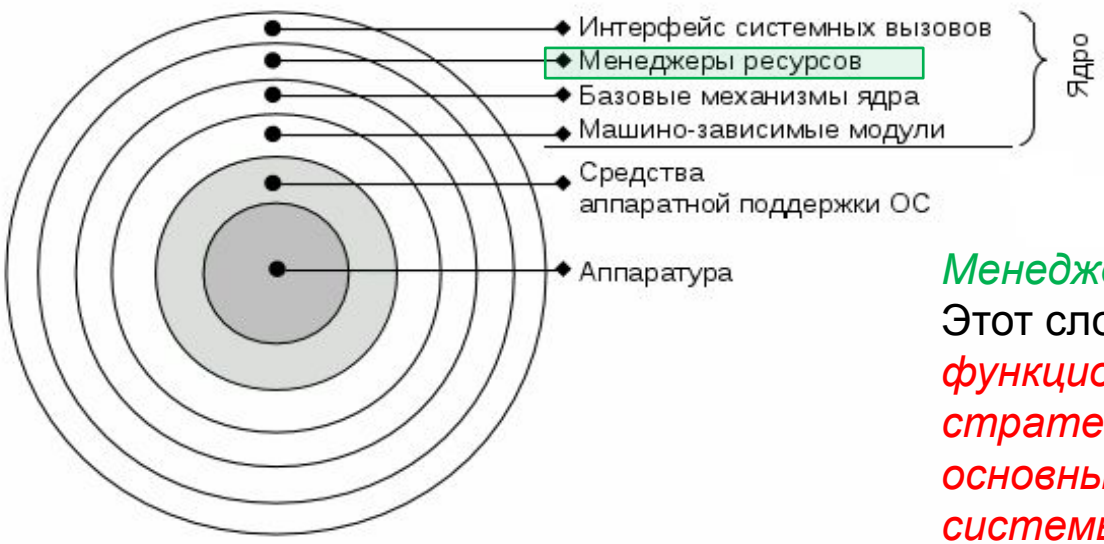
6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

Базовые механизмы ядра.
Этот слой выполняет наиболее примитивные операции ядра, такие как программное переключение контекстов процессов, диспетчеризацию прерываний, перемещение страниц из памяти на диск и обратно и т.п.

Модули данного слоя не принимают решений о распределении ресурсов, они только отрабатывают принятые «наверху» решения, что и дает повод называть их исполнительными механизмами для модулей верхних слоев.

Например, решение о том, что в данный момент нужно прервать выполнение текущего процесса А и начать выполнение процесса В, принимается менеджером процессов на вышележащем слое, а слою базовых механизмов передается только директива о том, что нужно выполнить переключение с контекста текущего процесса на контекст процесса В.

Описание назначения слоев структуры



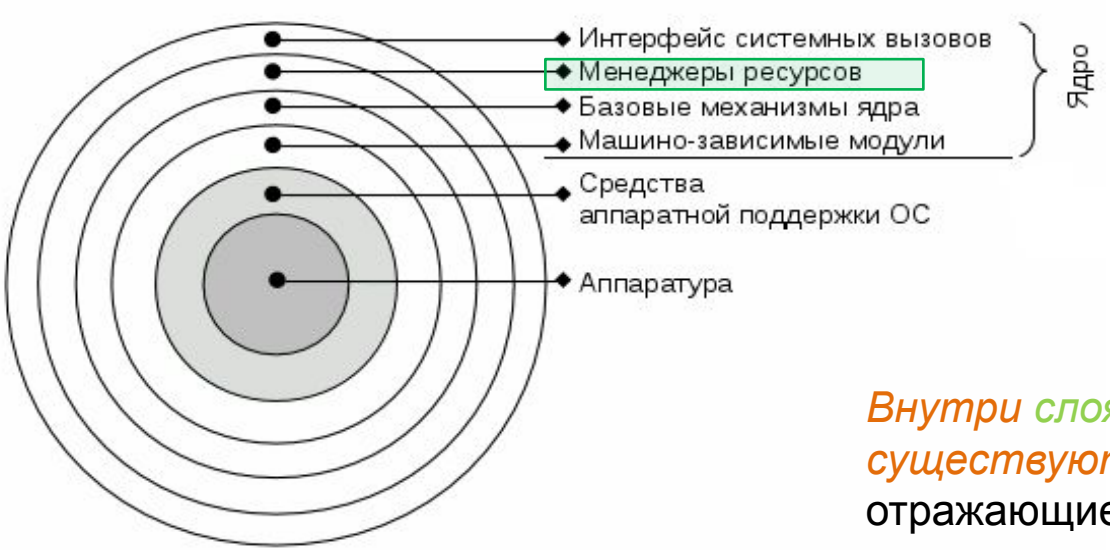
6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

Менеджеры (диспетчеры) ресурсов. Этот слой состоит из *мощных функциональных модулей*, *реализующих стратегические задачи по управлению основными ресурсами вычислительной системы*. Обычно на данном слое работают диспетчеры процессов, ввода-вывода, файловой системы и оперативной памяти.

Модули данного слоя ведут учет свободных и используемых ресурсов определенного типа и планируют их распределение в соответствии с запросами приложений. Например, *диспетчер виртуальной памяти* управляет перемещением страниц из оперативной памяти на диск и обратно. Он должен отслеживать интенсивность обращений к страницам, время пребывания их в памяти, состояния процессов, использующих данные, и многие другие параметры, на основании которых время от времени диспетчером принимаются решения о том, какие страницы необходимо выгрузить и какие — загрузить.

Для исполнения принятых решений *диспетчер виртуальной памяти* обращается к нижележащему *слою базовых механизмов* с запросами о загрузке (выгрузке) конкретных страниц.

Описание назначения слоев структуры

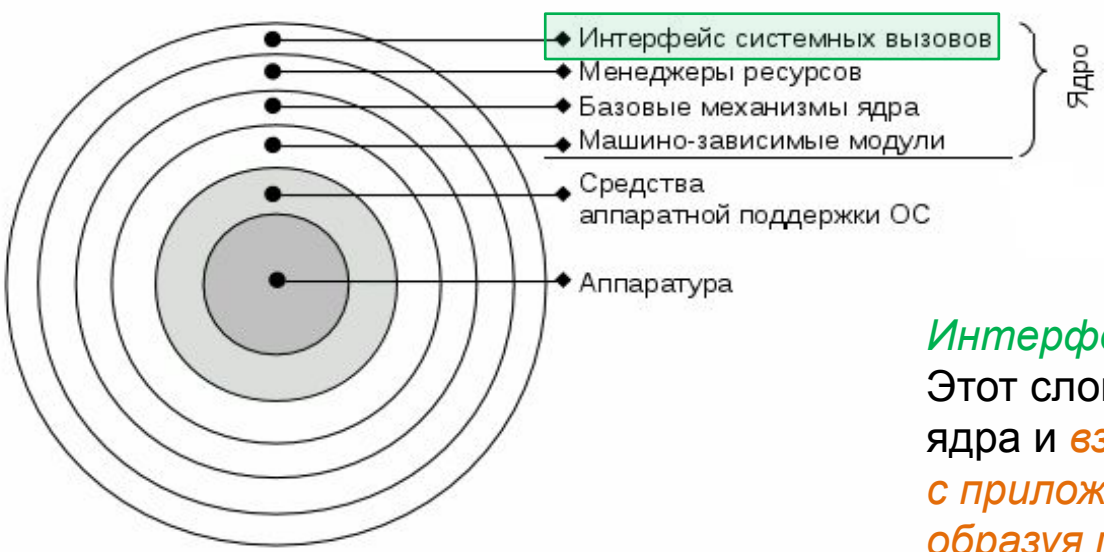


6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

Внутри слоя диспетчеров ресурсов существуют тесные взаимные связи, отражающие тот факт, что *для выполнения процессу нужен доступ одновременно к нескольким ресурсам* — процессору, области памяти, возможно, к определенному файлу или устройству ввода-вывода.

Например, при создании процесса *диспетчер процессов* обращается к *диспетчеру виртуальной памяти*, который должен выделить процессу определенную область памяти для его кодов и данных.

Описание назначения слоев структуры



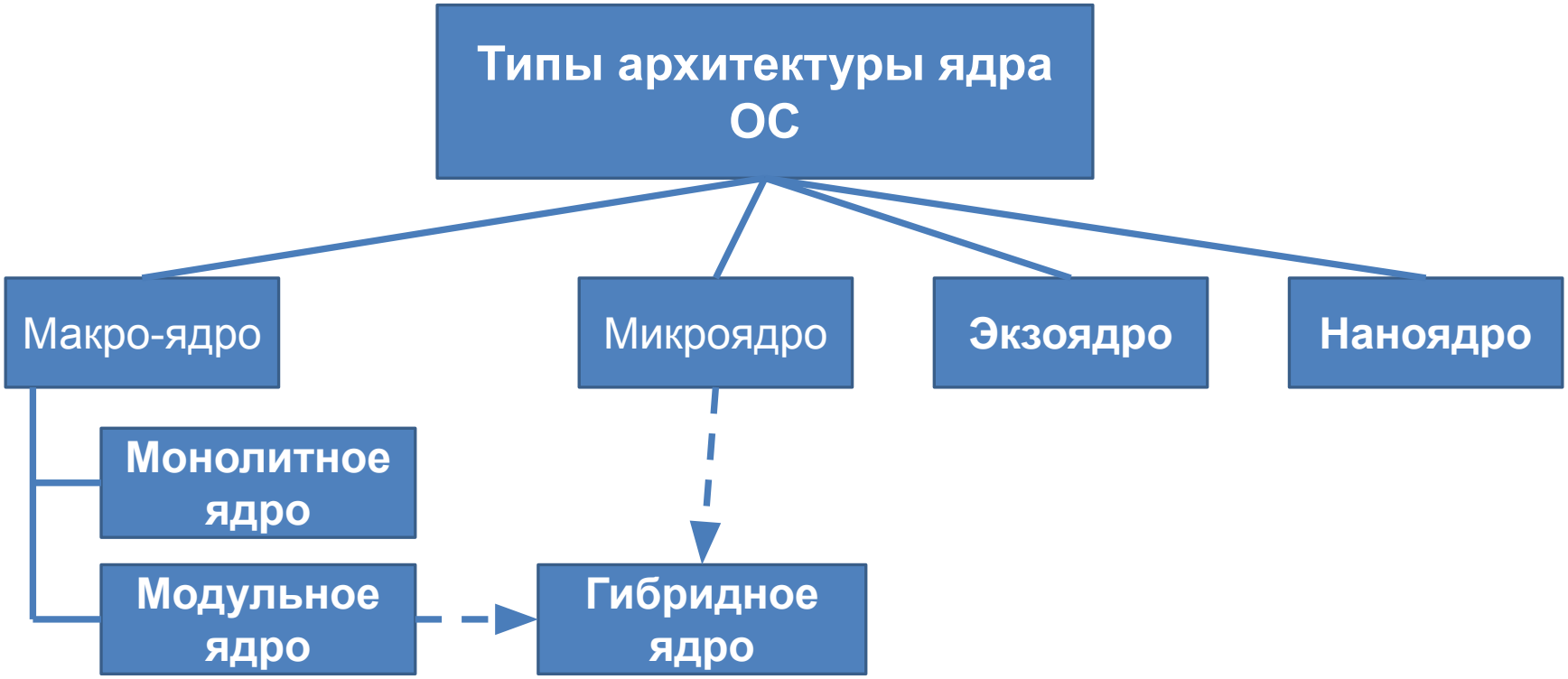
6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

Интерфейс системных вызовов.
Этот слой является самым верхним слоем ядра и *взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс операционной системы*.

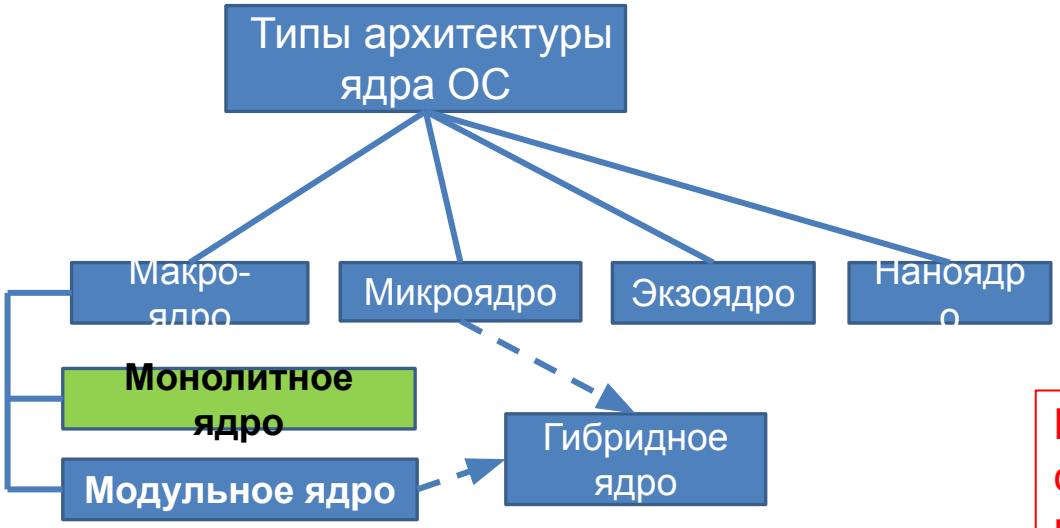
Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения.

Например, в операционной системе UNIX с помощью системного вызова *fd=open('/doc/a.txt',0-RDONLY)* приложение открывает файл *a.txt*, хранящийся в каталоге */doc*, а с помощью системного вызова *read(fd,buffer,count)* читает из этого файла в область своего адресного пространства, имеющую имя *buffer*, некоторое количество байт. Для осуществления таких комплексных действий системные вызовы обычно обращаются за помощью к функциям слоя диспетчеров ресурсов, причем для выполнения одного системного вызова может понадобиться несколько таких обращений.

Практические варианты реализации многослойной модели ядра ОС



Монолитное ядро



Монолитное ядро - схема, при которой все компоненты ядра ОС являются *составными частями одной программы*, используют *общие структуры данных* и взаимодействуют друг с другом путём *непосредственного вызова процедур*.

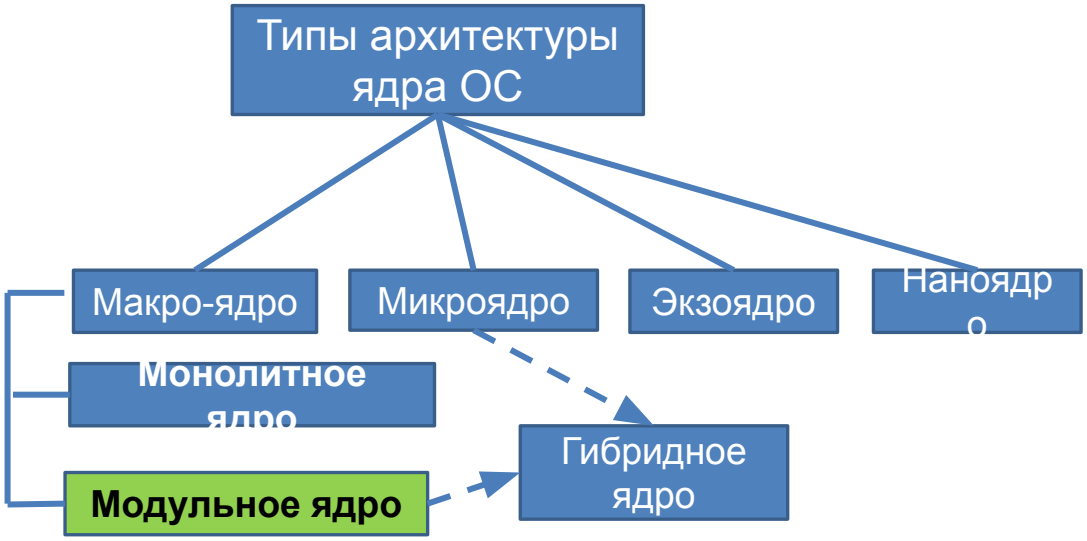
Все части монолитного ядра работают в одном адресном пространстве в привилегированном режиме!!!

Монолитное ядро — старейший способ организации операционных систем.
Достоинства: Скорость работы, упрощённая разработка модулей.
Недостатки: Поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы.

Примеры: Традиционные ядра UNIX (такие как BSD), Linux; ядро MS-DOS, ядро KolibriOS.

Некоторые старые монолитные ядра, в особенности систем класса UNIX/Linux, требовали *перекомпиляции* при любом изменении состава оборудования.
Большинство современных ядер позволяют во время работы подгружать *модули*, выполняющие часть функций ядра.
В этом случае компоненты операционной системы являются не самостоятельными модулями, а составными частями одной большой программы, называемой ***монолитным ядром*** (monolithic kernel), которое представляет собой набор процедур, каждая из которых может вызвать каждую.

Модульное ядро



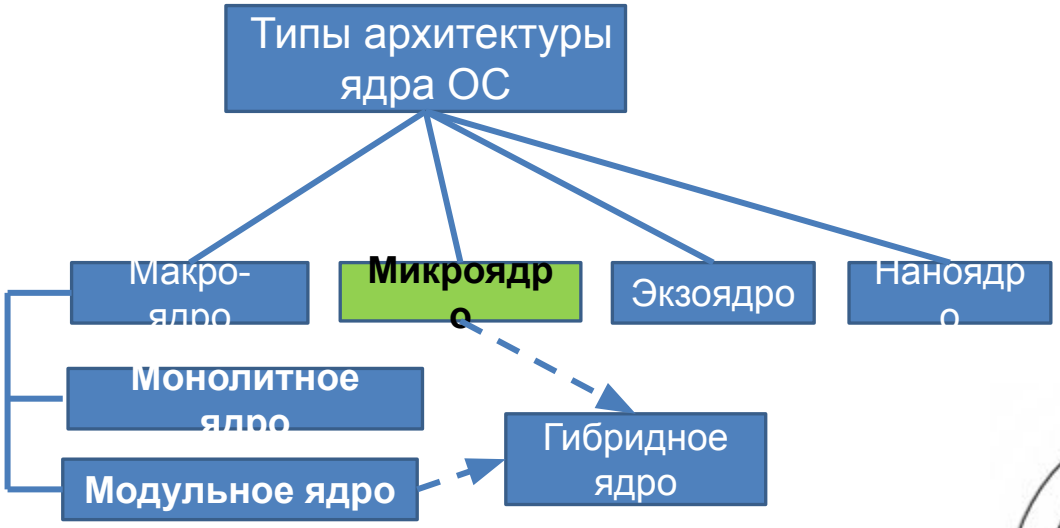
Модульное ядро - схема, при которой ядро делится на *отдельные загружаемые модули ядра*, причём в оперативную память они могут загружаться как по отдельности, так и все вместе, что *зависит от особенностей системы, от конфигурации аппаратных средств и настроек, установленных пользователем*.

Примеры: Linux и NetBSD.

Как и в случае монолитного ядра, все модули ядра по-прежнему находятся в общем адресном пространстве и исполняются в режиме ядра, то есть не вытесняются другими задачами

В отличие от «классических» монолитных ядер, *модульные ядра*, как правило, не требуют полной перекомпиляции ядра при изменении состава аппаратного обеспечения компьютера.
Вместо этого модульные ядра предоставляют тот или иной механизм *подгрузки* модулей ядра, поддерживающих то или иное аппаратное обеспечение (например, драйверов).
При этом подгрузка модулей может быть как *динамической* (выполняемой «на лету», без перезагрузки ОС, в работающей системе), так и *статической* (выполняемой при перезагрузке ОС после переконфигурирования системы на загрузку тех или иных модулей).

Микроядро



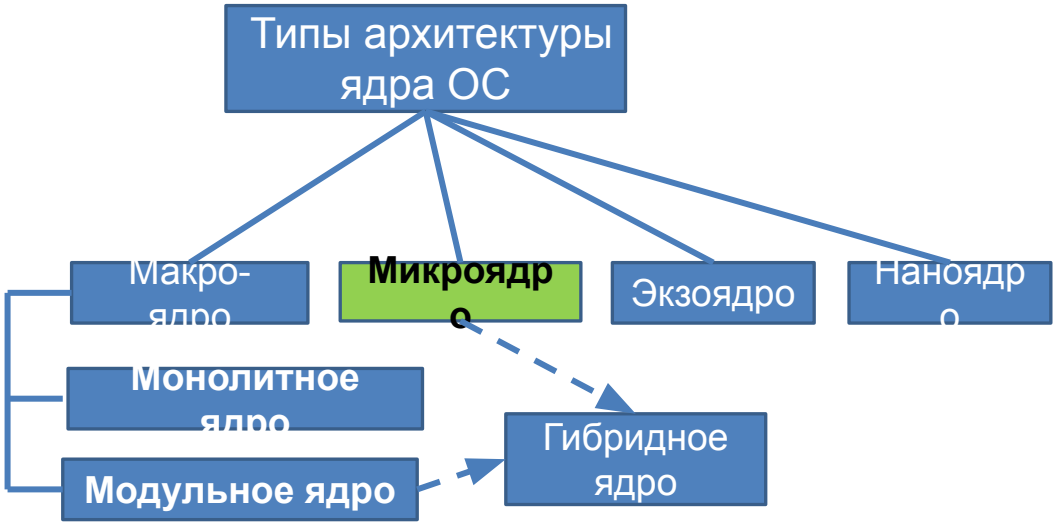
Микроядро - схема, при которой ядро предоставляет только базовые сервисы операционной системы.
Бóльшая часть работы осуществляется с помощью специальных работающих в пользовательском режиме процессов, называемых *сервисами*.

В состав микроядра обычно входят *машинно-зависимые модули*, а также *модули, выполняющие базовые функции ядра по управлению процессами, обработке прерываний, управлению виртуальной памятью, пересылке сообщений, управлению устройствами ввода-вывода, связанные с загрузкой или чтением регистров устройств*.

Набор функций микроядра обычно соответствует функциям слоя базовых механизмов обычного ядра.
Такие функции операционной системы практически невозможно, выполнить в пространстве пользователя.

Примеры: Symbian OS; Windows CE; OpenVMS; Mach, Mac OS X; QNX; AIX; Minix; ChorusOS; AmigaOS.

Микроядро



Микроядро - схема, при которой ядро предоставляет только базовые сервисы операционной системы.
Бóльшая часть работы осуществляется с помощью специальных работающих в пользовательском режиме процессов, называемых *сервисами*.

Решающим критерием «микроядерности» является размещение всех или почти всех драйверов и модулей в сервисных процессах, иногда с явной невозможностью загрузки любых модулей расширения в собственно микроядро.

Основное достоинство микроядерной архитектуры — высокая степень модульности ядра операционной системы. Это существенно упрощает добавление в него новых компонентов. При работе операционной системы с микроядром можно, не прерывая её работы, загружать и выгружать новые драйверы, файловые системы и т. д.

Существенно упрощается процесс отладки компонентов ядра, так как новая версия драйвера может загружаться без перезапуска всей операционной системы. Компоненты ядра операционной системы ничем принципиально не отличаются от пользовательских программ, поэтому для их отладки можно применять обычные средства.

Микроядерная архитектура повышает надежность системы, поскольку ошибка на уровне непривилегированной программы менее опасна, чем отказ на уровне режима ядра.

Обобщенная модель ОС с микроядром

В привилегированном режиме остается работать только очень небольшая часть ОС, называемая микроядром. Микроядро защищено от остальных частей ОС и приложений.

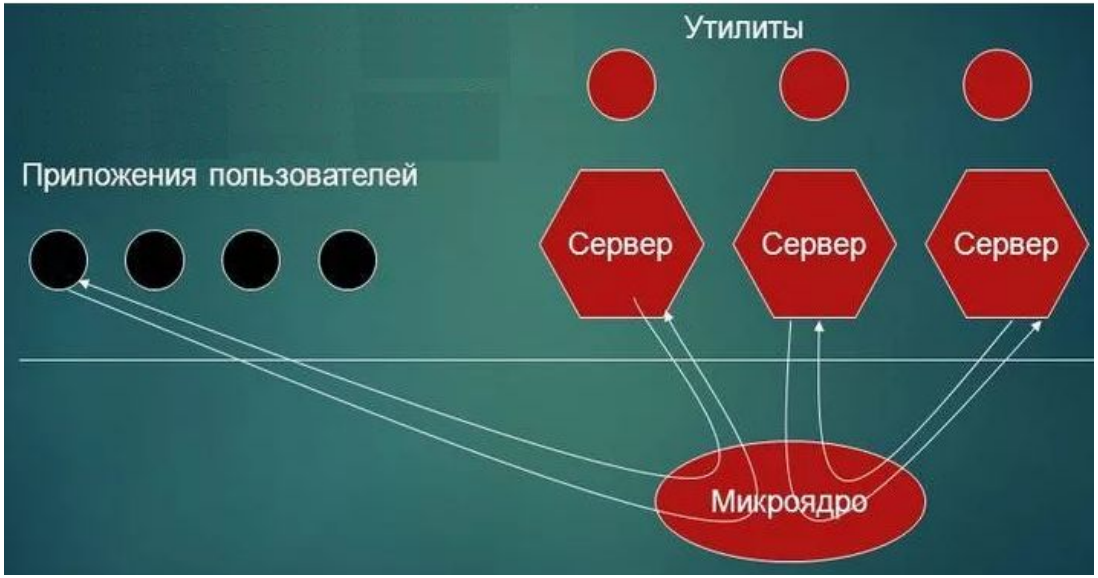
Высокоуровневые функции ядра оформляются в виде приложений, работающих в пользовательском режиме. Практически все диспетчеры устройств, являющиеся неотъемлемыми частями обычного ядра — файловая система, подсистемы управления виртуальной памятью и процессами, менеджер безопасности и т. п., — становятся «периферийными» модулями, работающими в пользовательском режиме.

Работа таких «периферийными» модулей имеет принципиальные отличия от работы традиционных утилит и обрабатывающих программ операционной системы.
Утилиты и обрабатывающие программы *вызываются в основном пользователями*.
Основным назначением «периферийными» модулей является *обслуживание запросов локальных приложений и других модулей ОС*, например создание процесса, выделение памяти, проверка прав доступа к ресурсу и т.д. Именно поэтому такие модули называются *серверами ОС*.

Очевидно, что для реализации микроядерной архитектуры необходимым условием является наличие в операционной системе удобного и эффективного способа вызова процедур одного процесса из другого.

Поддержка такого механизма и является одной из главных задач микроядра.

Обработка системного вызова в ОС с микроядром



Клиент, которым может быть либо прикладная программа, либо другой компонент ОС, запрашивает выполнение некоторой функции у соответствующего *сервера*, посылая ему сообщение. Непосредственная передача сообщений между приложениями невозможна, так как их адресные пространства изолированы друг от друга.

Микроядро, выполняющееся в привилегированном режиме, имеет доступ к адресным пространствам каждого из этих приложений и поэтому может работать в качестве посредника. Микроядро сначала передает сообщение, содержащее имя и параметры вызываемой процедуры нужному серверу, затем сервер выполняет запрошенную операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения.

Таким образом, работа микроядерной операционной системы соответствует известной *модели клиент-сервер*, в которой *роль транспортных средств выполняет микроядро*.

Преимущества микроядерной архитектуры

Переносимость

Весь машинно-зависимый код изолирован в микроядре, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.

Расширяемость

Ограниченный набор четко определенных интерфейсов микроядра открывает путь к упорядоченному росту и эволюции ОС.
Добавление новой подсистемы требует разработки нового приложения, что никак не затрагивает целостность микроядра.
Микроядерная структура позволяет не только добавлять, но и сокращать число компонентов операционной системы, что также бывает очень полезно.

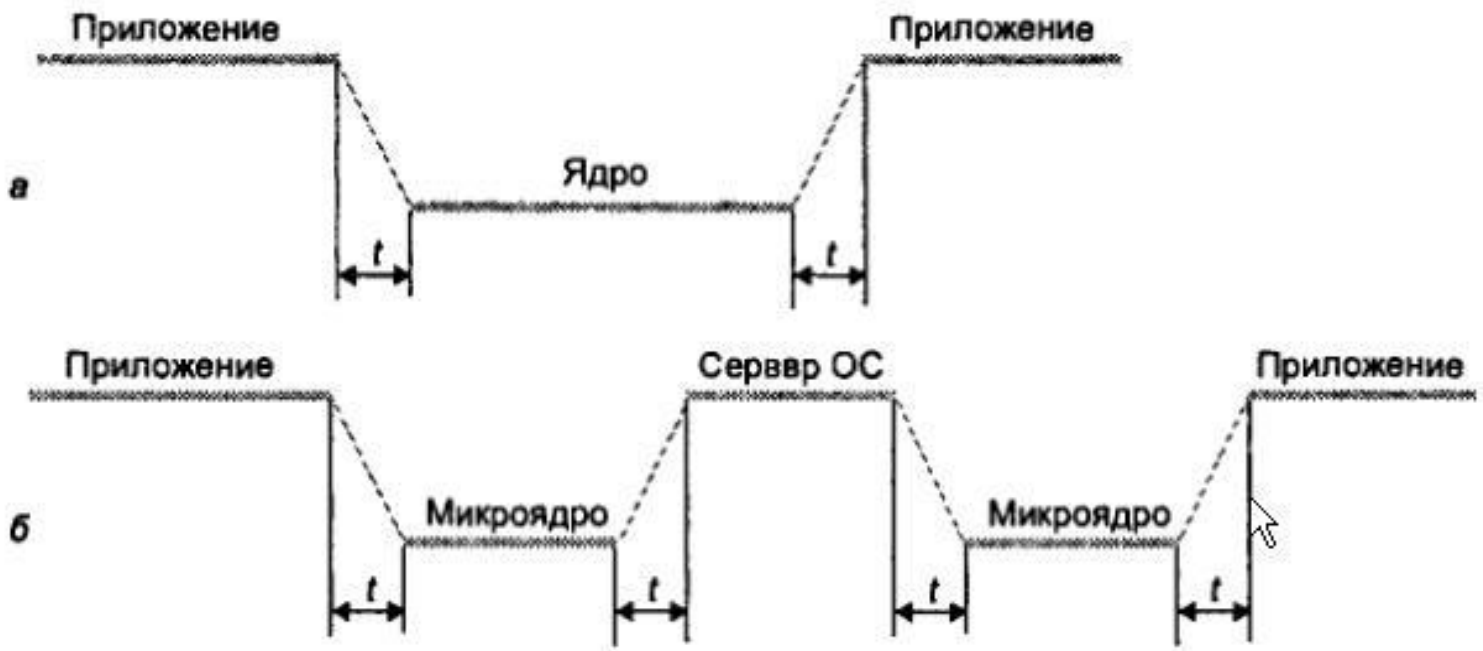
Надежность

Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти и, таким образом, защищен от других серверов операционной системы.
Поскольку серверы выполняются в пользовательском режиме, они не имеют непосредственного доступа к аппаратуре и не могут модифицировать память, в которой хранится и работает микроядро.
Уменьшенный объем кода микроядра по сравнению с традиционным ядром снижает вероятность появления ошибок программирования.

Поддержка распределенных вычислений

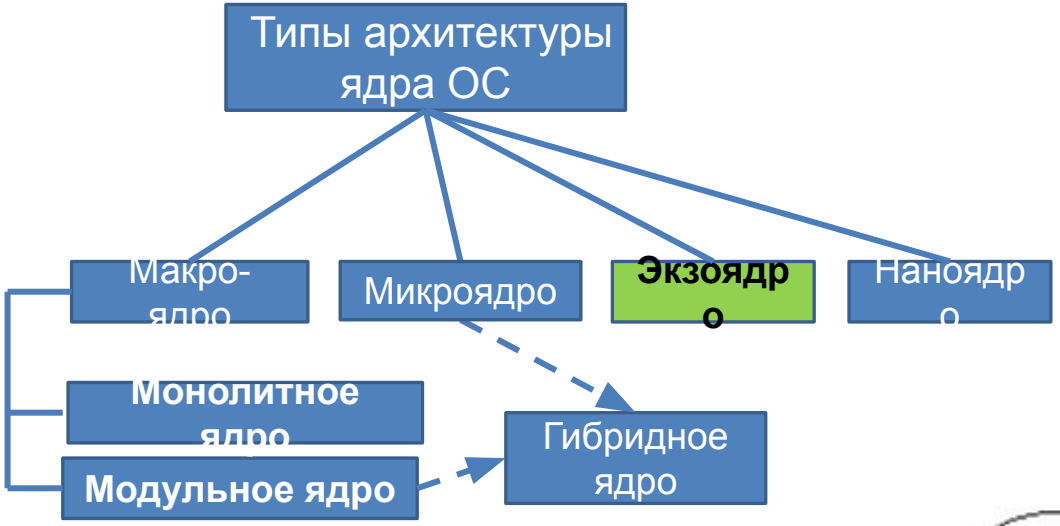
Серверы микроядерной ОС могут работать как на одном, так и на разных компьютерах. В этом случае при получении сообщения от приложения микроядро может обработать его самостоятельно и передать локальному серверу или же переслать по сети микроядру, работающему на другом компьютере. Переход к распределенной обработке требует минимальных изменений в работе операционной системы — просто локальный транспорт заменяется на сетевой.

Недостаток микроядерной архитектуры



Производительность. При классической организации ОС (а) выполнение системного вызова сопровождается двумя переключениями режимов, а при микроядерной (б) организации — четырьмя.
Таким образом, операционная система на основе микроядра при прочих равных условиях *всегда будет менее производительной*, чем ОС с классическим ядром. Именно по этой причине микроядерный подход не получил такого широкого распространения, которое ему предрекали.

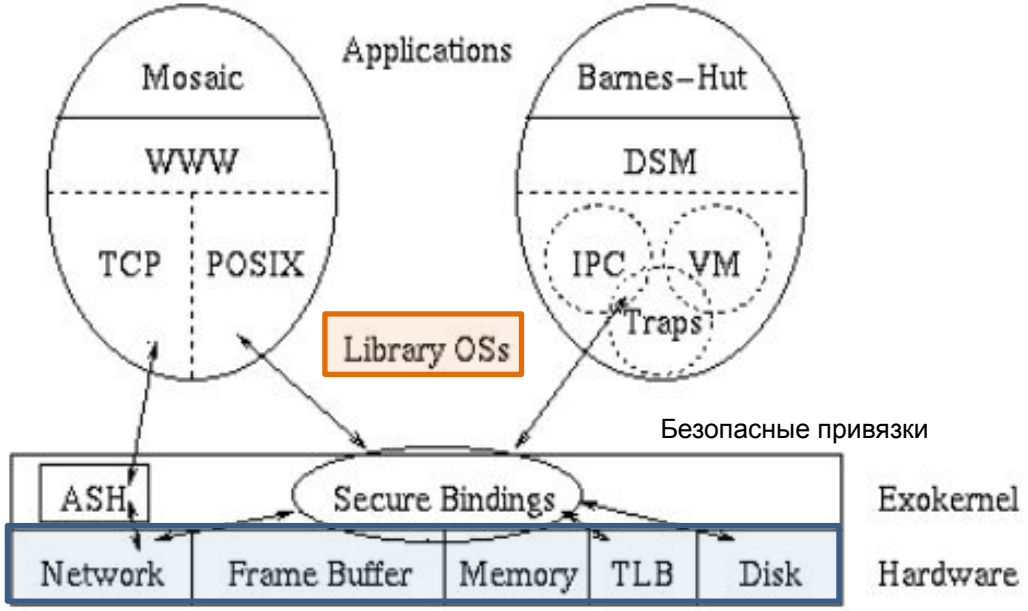
Экзоядро



Экзоядро - схема, при которой ядро представляет лишь функции для взаимодействия между процессами, безопасного выделения и освобождения ресурсов.

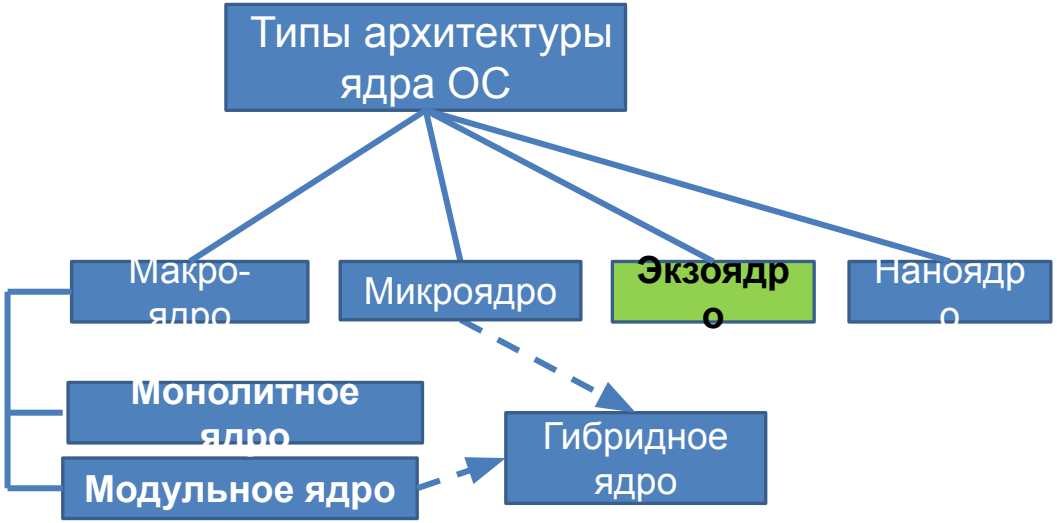
Предполагается, что API для прикладных программ предоставляются *внешними по отношению к ядру библиотеками пользовательского уровня (LibOs)*.

Библиотекам пользовательского уровня (LibOs), функционирующим как приложения, **лучше** чем операционной системе **известно**, каковы должны быть цели политик управления ресурсами, следовательно, LibOs нужно предоставить как можно больше контроля над этими политиками.



https://osdev.fandom.com/ru/wiki/Экзоядро: Архитектура Операционной Системы для Управления Ресурсами Прикладным Уровнем

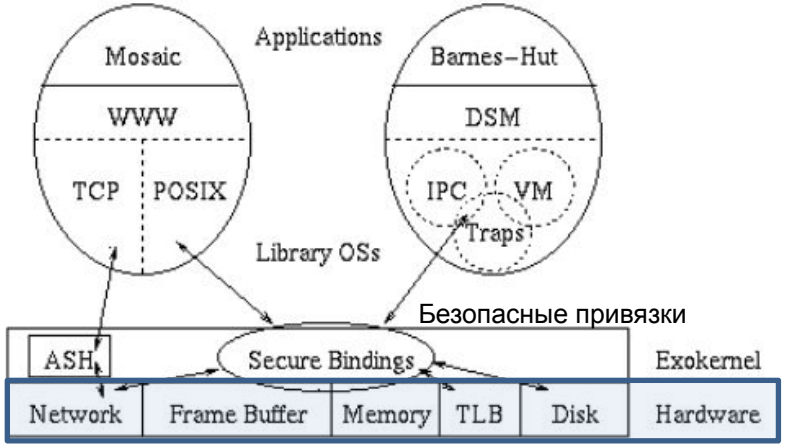
Экзоядро



Экзоядро - схема, при которой ядро представляет лишь функции для взаимодействия между процессами, безопасного выделения и освобождения ресурсов.

Предполагается, что API для прикладных программ предоставляются *внешними по отношению к ядру библиотеками пользовательского уровня (LibOs)*.

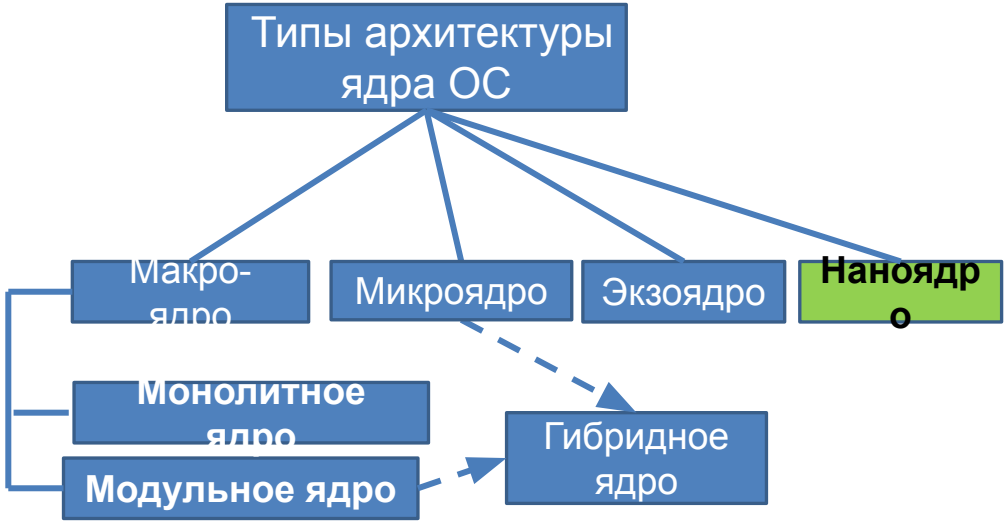
Разделяя защиту и управление, экзоядро выполняет три важные задачи:
(1) *отслеживает чьей собственностью является любой ресурс*, *гарантирует стабильность*, (2) *охраняя использование ресурсов*, (3) *отменяя доступ к ресурсам*.
Чтобы выполнить эти задачи, ядро использует три метода.
Во-первых, используя *безопасные связи*, библиотеки пользовательского уровня могут безопасно соединяться с ресурсами компьютера.
Во-вторых, видимые ревокации* позволяют библиотекам пользовательского уровня участвовать в процессе ревокации ресурсов.
В-третьих, протокол отмены позволяет экзоядру *насильно нарушать безопасные связи несговорчивых приложений*.



Ревокация – возможность отказаться от ранее запрошенного

https://osdev.fandom.com/ru/wiki/Экзоядро: Архитектура Операционной Системы для Управления Ресурсами Прикладным Уровнем

Наноядро



Наноядро - схема, при которой крайне упрощённое и минималистичное ядро выполняет лишь одну задачу — обработку аппаратных прерываний, генерируемых устройствами компьютера.

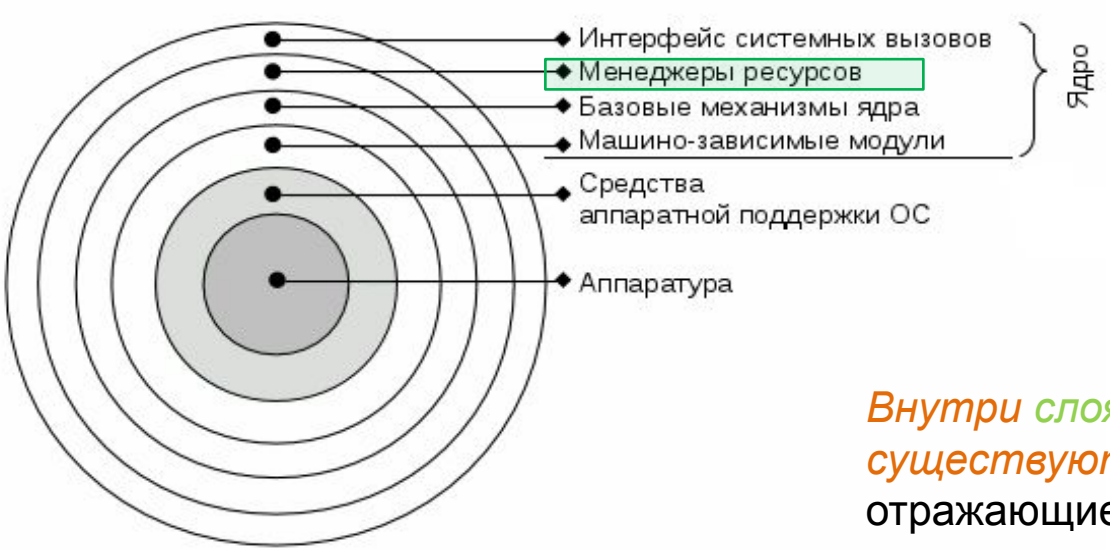
После обработки прерываний от аппаратуры наноядро, в свою очередь, посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему программному обеспечению при помощи того же механизма прерываний.

Примером является KeyKOS — самая первая ОС на наноядре.

Наиболее часто в современных компьютерах наноядра используются для виртуализации аппаратного обеспечения реальных компьютеров или для реализации механизма гипервизора, с целью позволить нескольким или многим различным операционным системам работать одновременно и параллельно на одном и том же компьютере.

http://www.cis.upenn.edu/~KeyKOS/NanoKernel/NanoKernel.html
https://ru.bmstu.wiki/index.php?title=%D0%9D%D0%B0%D0%BD%D0%BE%D1%8F%D0%B4%D1%80%D0%BE&mobileaction=toggle_view_desktop

Описание назначения слоев структуры

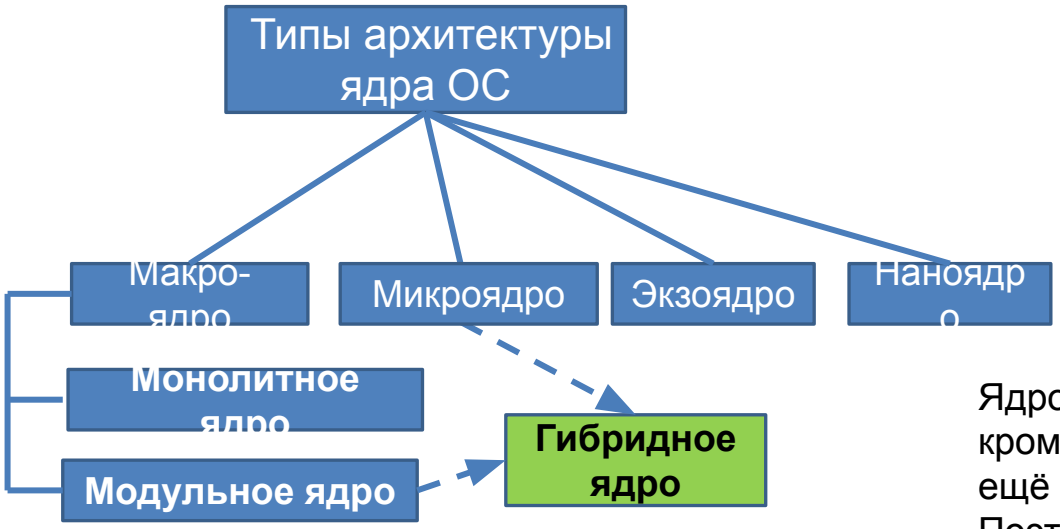


6-слойная структура аппаратной части вычислительной системы и ядра операционной системы

Внутри слоя диспетчеров ресурсов существуют тесные взаимные связи, отражающие тот факт, что *для выполнения процессу нужен доступ одновременно к нескольким ресурсам* — процессору, области памяти, возможно, к определенному файлу или устройству ввода-вывода.

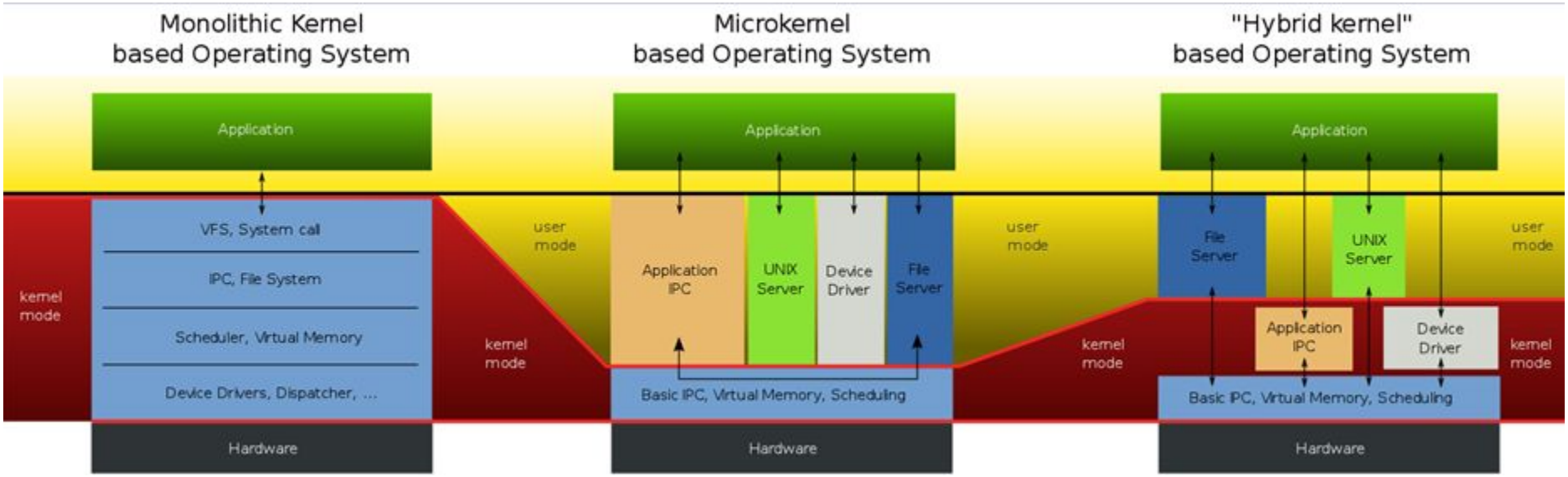
Например, при создании процесса *диспетчер процессов* обращается к *диспетчеру виртуальной памяти*, который должен выделить процессу определенную область памяти для его кодов и данных.

Гибридное ядро



Гибридные ядра — это модифицированные микроядра, позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра.
Пример: ядра семейства ОС Windows NT.

Ядро NT слишком велико (более 1 Мбайт), кроме того, в ядре системы находится, например, ещё и модуль графического интерфейса). Поэтому ядро NT не может носить приставку «микро».



IPC – средства межпроцессного взаимодействия

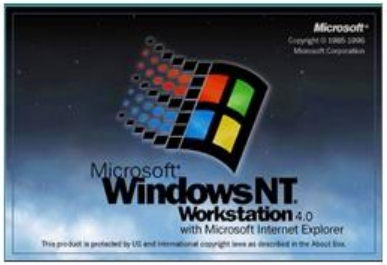
Ошибки классификации ядер ОС

Термин «***модульное ядро***» иногда адресуют и к различным «***гибридным***» ядрам от DragonFly BSD, Mac OS X до Windows NT, что **не является верным**, так как *большинство системных служб и драйверов данных систем реализуется в виде* ***процессов пользовательского режима***, то есть в них активно используется *архитектура микроядра*, и следовательно эти ОС относятся, как минимум, к ОС с гибридным ядром.

Проблема неверной классификации связана с тем, что данные гибридные ядра поддерживают архитектуру *загузочных модулей ядра*, СВОЙСТВЕННУЮ ***модульным ядрам***,
но здесь также есть ВАЖНОЕ ОТЛИЧИЕ - ***подгружаемые модули ядра*** и прочие компоненты ядер Windows NT и Mac OS X располагаются в вытесняемой памяти (памяти пользовательского режима) и взаимодействуют друг с другом путем передачи сообщений, как положено в микроядерных операционных системах.

Типы архитектуры ядра наиболее распространённых ОС для ПК

Семейство операционных систем Windows NT



22 октября 2009 года. Windows NT версии 6.1

17 февраля 2000г. Windows NT версии 5.0

26 октября 2012 года Windows NT 6.2.9200

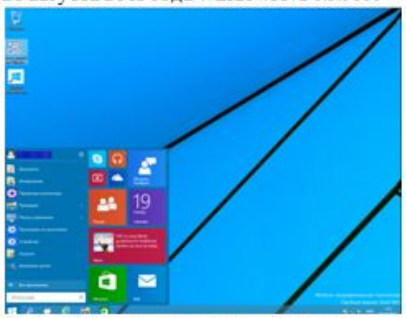
04 сентября 2012 года

24 августа 2001 года Windows NT версии 5.1

24 апреля 2003г. Windows NT версии 5.2

21 августа 2013 года Windows NT 6.3.9600

22 октября 2009 года. Windows NT версии 6.0

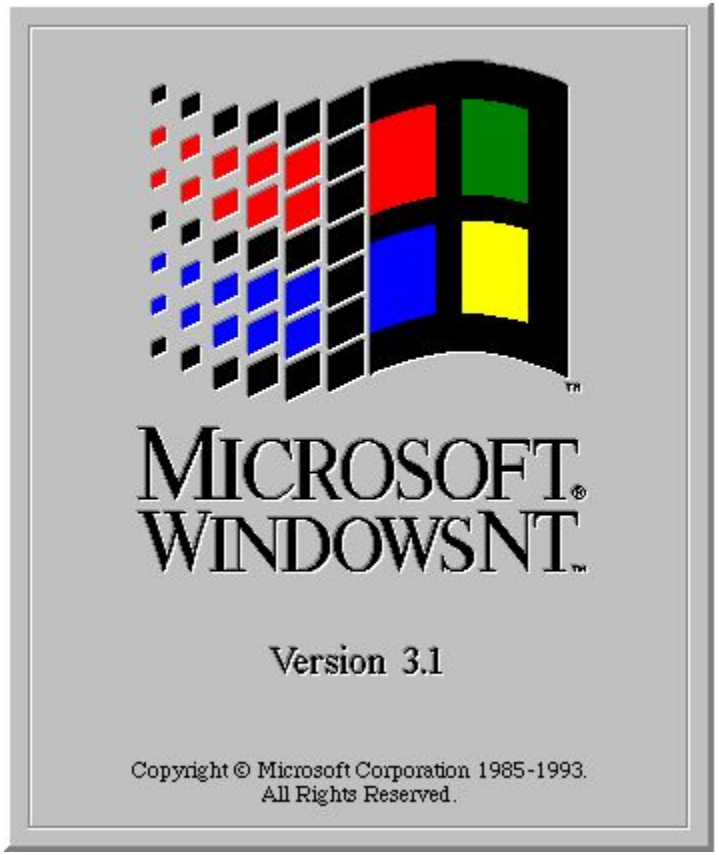


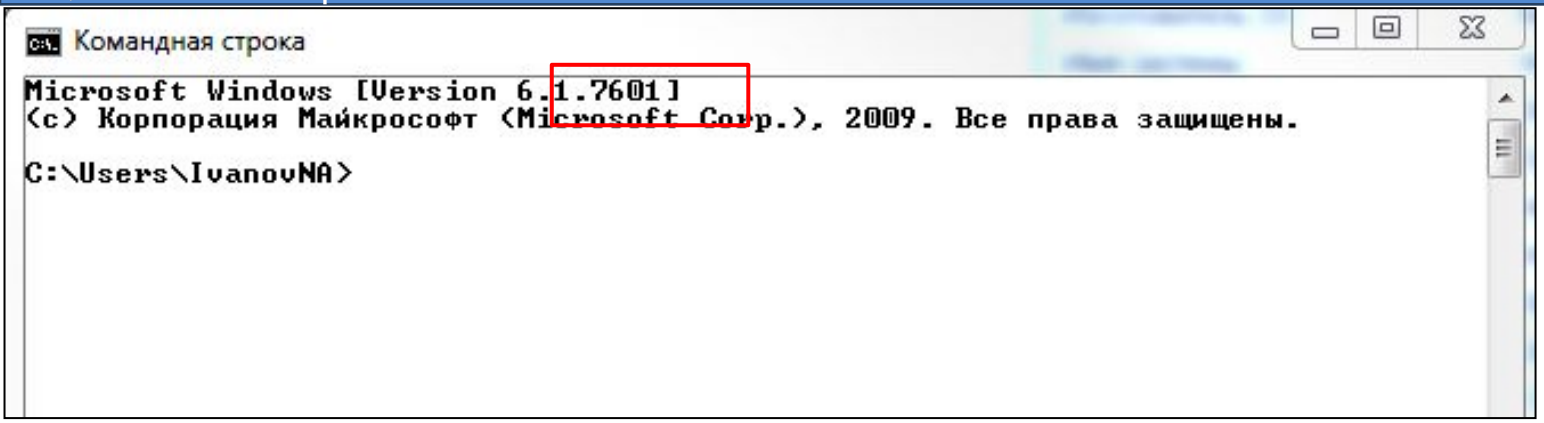
1 октября 2014г. Windows 10 Technical Preview. Windows NT 6.4.9841

Windows NT,
Windows 2000,
Windows XP, Windows Server 2003,
Windows Vista,
Windows Server 2008,
Windows 7, Windows Server 2008 R2,
Windows 8, Windows Server 2012,
Windows 8.1,
Windows 10,
Server 2016,
Server 2019

«Надежное решение для рабочих станций и серверов»

27 июля 1993 года Windows NT 3.1 - первая полностью 32-битная операционная система семейства Windows NT.





Стандартная нумерация ядра Windows NT, на которой основаны все сегодняшние ОС Microsoft, начиная с самой Windows NT и заканчивая Windows 8.1, **никогда не менялась**.
Фактически, за каждым выпуском Windows всегда закреплялся *номер версии* самой Windows NT (т. е. *версии исполняемого файла ядра* **ntoskrnl.exe**), например: Windows 2000 — NT 5.0, Windows Vista — NT 6.0, Windows 7 — NT 6.1, Windows 8 — NT 6.3.

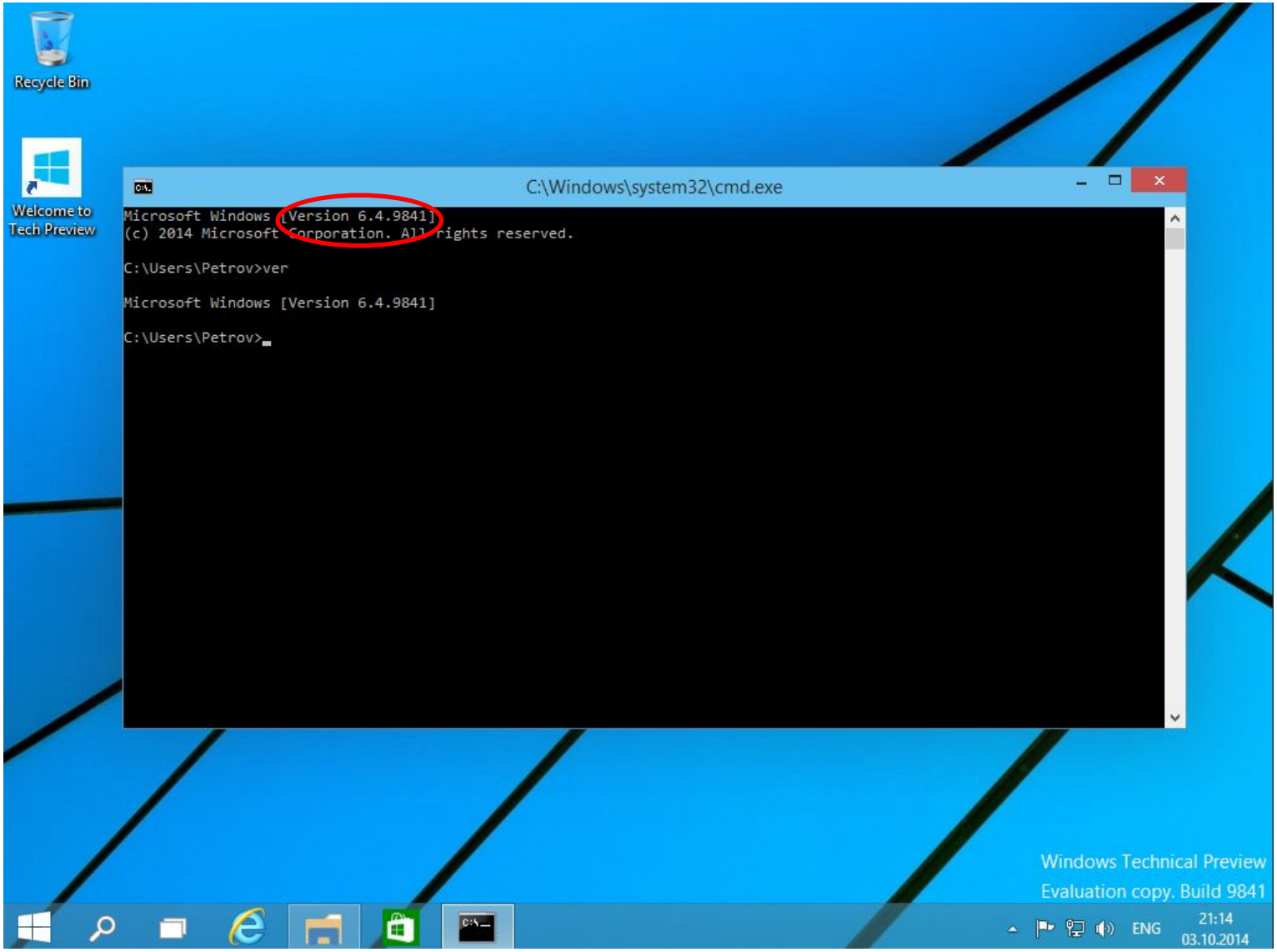
Это так называемые *старшее* (major) и *младшее* (minor) числа, определяющие *настоящую* версию Windows.

В зависимости от количества изменений, вносимых разработчиками в ядро, а также от масштаба этих изменений, для выпускаемой ОС эти числа увеличивались на единицу или оставались прежними

Название (*кодовое название*), варианты	номер версии	первый выпуск	последний выпуск / SP
Windows NT 3.1	3.1.528	27 июля 1993	SP3 (10 ноября 1994)
	Workstation, Advanced Server		
Windows NT 3.5 (*Daytona*)	3.5.807	21 сентября 1994	SP3 (21 июня 1995)
	Workstation, Server		
Windows NT 3.51 (*Tukwila*)	3.51.1057	30 мая 1995	SP5 (19 сентября 1996)
	Workstation, Server		
Windows NT 4.0 (*Indy*)	4.0.1381	29 июля 1996	SP6a (30 ноября 1999)
	Workstation, Server, Server Enterprise (*Granite*), Terminal Server (*Hydra*), Embedded (*Impala*)		
Windows 2000 (*Cairo*)	5.0.2195	17 февраля 2000	SP4 (26 июня 2003)
	Professional, Server, Advanced Server, Datacenter Server		
Windows XP (*Whistler*)	5.1.2600	25 октября 2001	SP3 (6 мая 2008)
	Home, Professional, Media Center (*eHome*), Tablet PC, Starter, Embedded (*Mantis*), N; Windows Fundamentals for Legacy PCs (*Eiger*)		
Windows Server 2003 (*Whistler Server*, *Windows .NET Server*)	5.2.3790	24 апреля 2003	SP2 (13 мая 2007)
	Standard, Enterprise, Datacenter, Web, Small Business Server (*Bobcat*), Compute Cluster Server, Storage Server; Windows XP Professional x64		
Windows Vista (*Longhorn*)	6.0.6000	30 января 2007	SP2 (25 мая 2009)
	Starter, Home Basic, Home Premium, Business, Enterprise, Ultimate, N Home Basic, N Business; x64-варианты всех, кроме Starter		
Windows Home Server	6.0.2423	16 июля 2007	Power Pack 3 (24 ноября 2009)
Windows Server 2008 (*Longhorn Server*)	6.0.6001	27 февраля 2008	SP2 (27 мая 2009)
	Standard, Enterprise, Datacenter, HPC, Web, Storage, Small Business (*Cougar*), Essential Business (*Centro*), Itanium; x64-варианты всех, кроме HPC		
Windows 7 (*Blackcomb*, *Vienna*)	6.1.7600	22 октября 2009	SP1 (KB976932) (22 февраля 2011)
	Начальная, Домашняя базовая, Домашняя расширенная, Профессиональная, Корпоративная, Максимальная, Windows 7 N, Windows 7 E; x64-варианты всех, кроме Начальной		
Windows Server 2008 R2	6.1.7600	22 октября 2009	SP1 (KB976932) (22 февраля 2011)
	Standard, Enterprise, Datacenter, HPC, Web, Storage, Small Business , Itanium; все версии — только 64-разрядные		
Windows Home Server 2011	6.1.7657	6 апреля 2011	
Windows 8	6.2.9200	26 октября 2012	Pro (26 октября 2012)
	Windows 8, Windows 8 RT, Профессиональная, Профессиональная N, Профессиональная WMC, Корпоративная, Корпоративная N; x64-варианты всех, кроме Windows RT		
Windows Server 2012	6.2.9200	26 октября 2012	RTM (1 августа 2012)
	Foundation, Essentials, Standard, Datacenter; все версии — только 64-разрядные		
Windows 8.1	6.3.9600	18 октября 2013	
	Windows 8.1, Windows 8.1 RT, Профессиональная, Профессиональная N, Профессиональная WMC, Корпоративная, Корпоративная N; x64-варианты всех, кроме Windows RT		
Windows Server 2012 R2	6.3.9600	18 октября 2013	
	Foundation, Essentials, Standard, Datacenter; все версии — только 64-разрядные		

Новая точка отсчета …

За Windows 10 должен был быть закреплен номер **NT 6.4**, как это было видно в первой версии Windows 10 TP.



Новая нумерация ядер ОС Windows

Со времени выпуска первых версий Windows NT в начале 90-х, эти данные никогда не претерпевали маркетинговых или иных изменений.

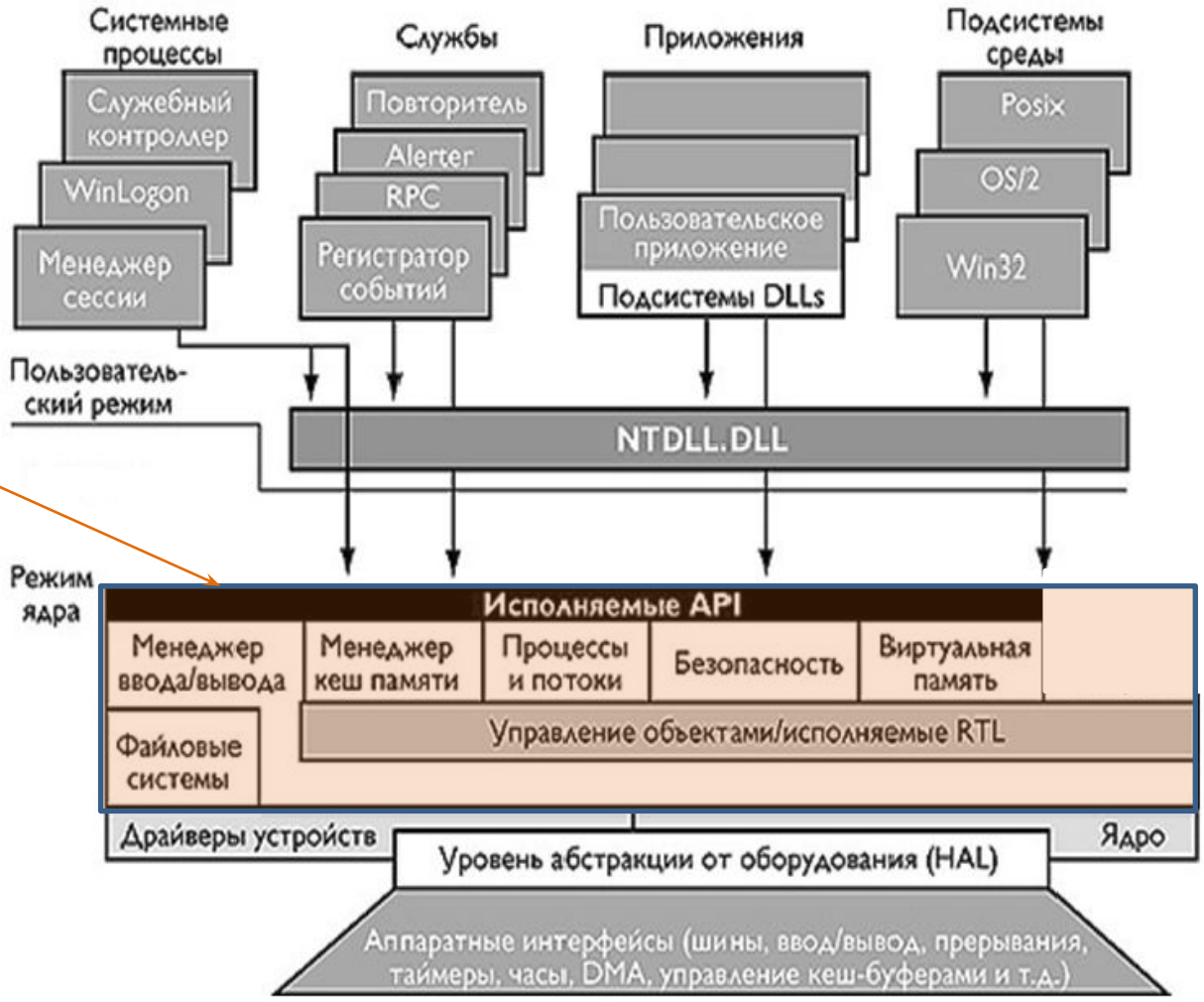
MS *отказалась от привычной нумерации версий ядра* Windows NT, перескочив с версии 6.4 (NT 6.4, настоящий номер версии ядра Windows 10) сразу на номер 10.

Модель ядра Windows NT

В Windows NT использована *модифицированная модель микроядра*, в котором в режиме ядра над модулями **HAL** и **Ядро** появляется дополнительный уровень - ***службы исполнения***.

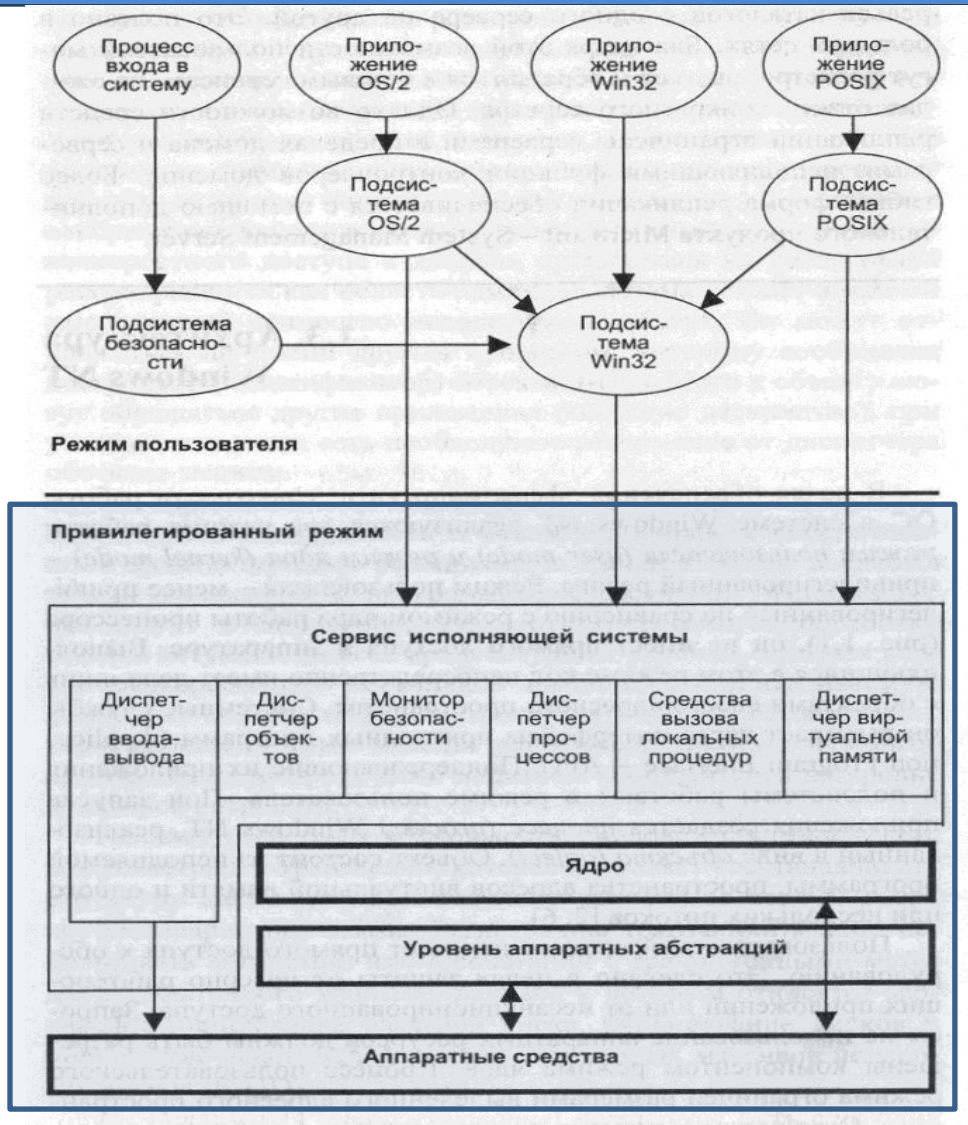
Термином «службы исполнения» обозначают все новые *модули, которые работают в режиме ядра*.

В пользовательском режиме работают подсистемы OS/2, POSIX, DOS/Windows 3.1 и клиент/серверная Win32.



Укрупненная схема функционирования Windows NT

Режим ядра - это *привилегированный режим* работы процессора, в котором код получает прямой доступ ко всем аппаратным ресурсам и всей памяти, включая адресные пространства всех процессов режима пользователя.



Укрупненная схема функционирования Windows NT

Режим пользователя (пользовательский) - менее привилегированный по сравнению с режимом ядра режим работы процессора.

В пользовательском режиме процесс:

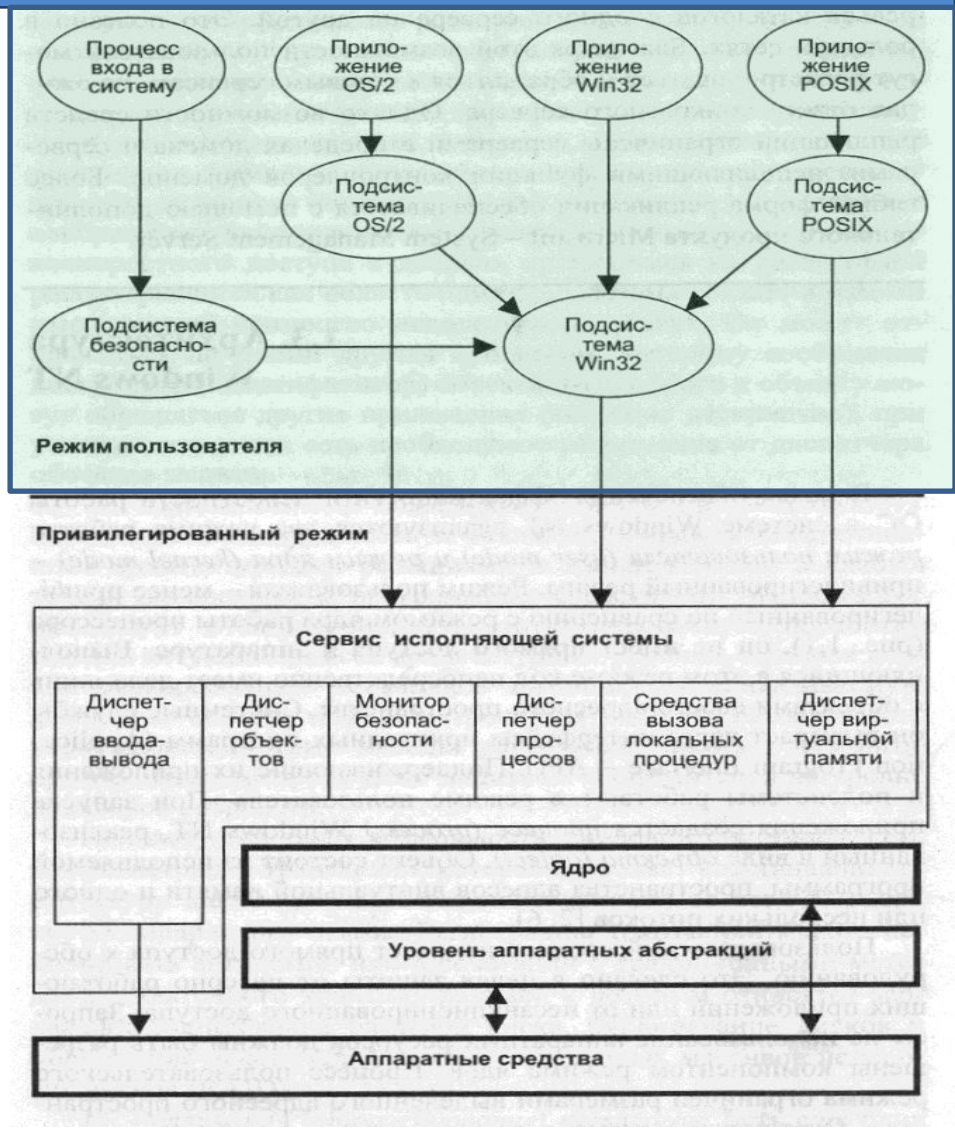
· *не имеет прямого доступа к аппаратуре*. Это сделано в целях защиты от неверно работающих приложений или от несанкционированного доступа. Запросы на использование аппаратных ресурсов должны быть разрешены компонентом режима ядра;

• *ограничен размерами выделенного адресного пространства*, что позволяет обеспечить дополнительную защиту ОС;

• *может быть выгружен из физической памяти в виртуальную память (virtual memory)* на жестком диске;

• *приоритет любого пользовательского процесса ниже*, чем у процессов режима ядра;

• *пользовательскому процессу*, как правило, предоставляется меньше процессорного времени, чем процессу режима ядра.



Типы процессов пользовательского режима в Windows NT

Специальные процессы поддержки системы, например, процесс регистрации пользователя и менеджер сессий, которые не являются службами NT.

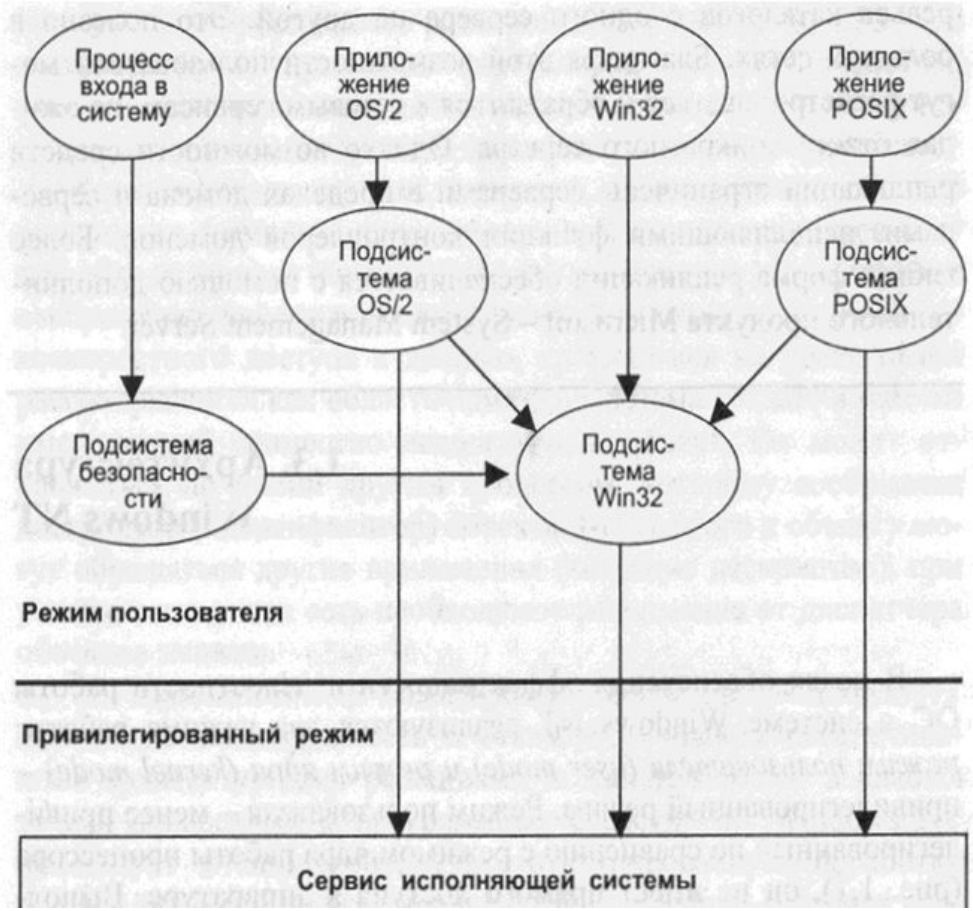
Процессы сервера или службы Windows NT (аналог демонов в ОС Unix). Примером может быть регистратор событий (*Event Logger*).
Многие дополнительно устанавливаемые приложения, такие как Microsoft SQL Server, также включают компоненты, работающие как службы NT.

Подсистемы среды, которые обеспечивают пользовательским приложениям среду выполнения других операционных систем.

Windows NT поставлялась с тремя подсистемами: Win32, Posix и OS/2 2.1.

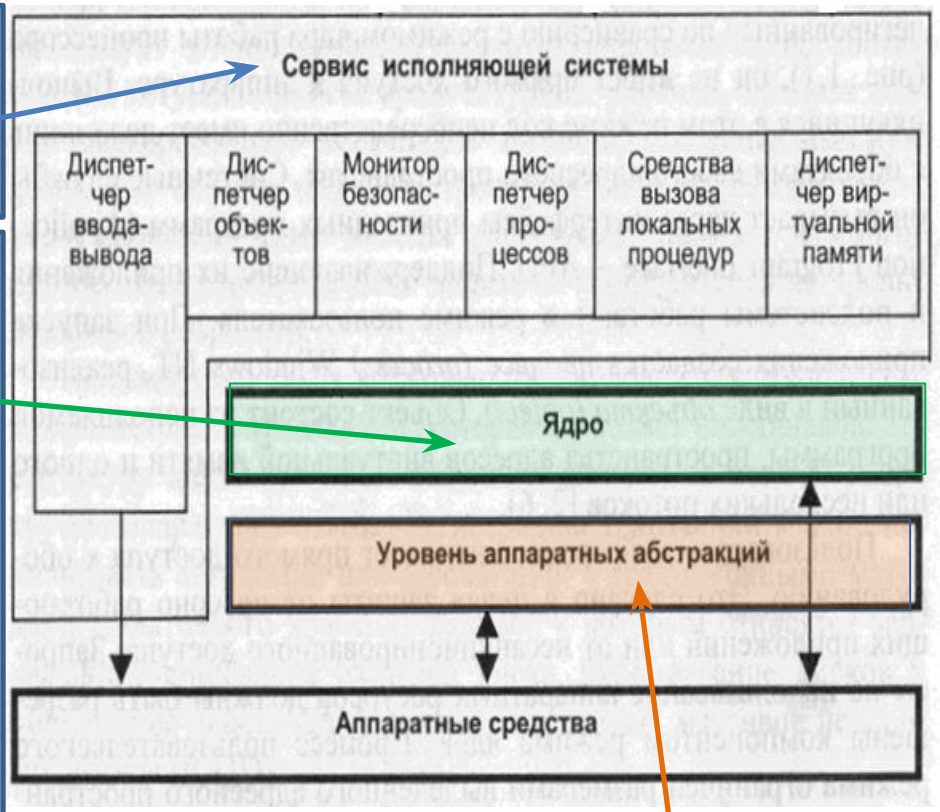
Пользовательские приложения одного из пяти типов: Win32, Windows 3.1,
MS-DOS, Posix или OS/2 1.2.

По мере развития семейства добавлена поддержка приложений Win64.



Исполняющая система Windows NT (Windows NT Executive) – основа функционирования режима ядра

· **сервис исполняющей системы** - управление памятью, процессами, потоками, безопасностью, вводом/выводом, межпроцессорными обменами;



· **ядро Windows NT** (ntoskrnl.exe) выполняет низкоуровневые функции операционной системы:
- диспетчеризация потоков,
- обслуживание прерываний и исключений,
- синхронизация процессов,
- отложенный вызов процедур.

Ядро никогда не выгружается из оперативной памяти, выполнение его модулей никогда не прерывается другими потоками.

Код ядра написан в основном на С, а части, дающие наибольшую нагрузку на процессор, на языке ассемблера;

· **слой аппаратных абстракций (HAL - Hardware Abstraction Layer)** изолирует ядро, драйверы устройств и исполняемую часть NT от аппаратных платформ, на которых должна работать ОС.

Этот программный слой позволяет скрыть особенности аппаратных платформ, предоставив ОС стандартные точки входа в процедуры, благодаря чему для нее исчезают различия между платформами и архитектурами.

Модули исполняемой части Windows NT

Часть модулей исполняемой части Windows NT соответствует *слою диспетчеров (менеджеров) ресурсов* классической модели монолитного ядра:

Менеджер ввода/вывода
Обеспечивает независимый от физических устройств ввод/вывод и отвечает за пересылку данных соответствующим драйверам для дальнейшей обработки.

Менеджер кэш-памяти
Улучшает производительность системы ввода/вывода данных, размещая читаемые с диска данные в основной памяти для ускорения доступа к ним, а также откладывая на короткое время запись измененных данных на диск

Менеджер процессов и потоков
Управляет процессами и потоками, осуществляет распределение ресурсов, отличных от оперативной памяти и времени работы процессора, между всеми процессами и потоками, предотвращая тупиковые ситуации и синхронизируя выполнения потоков.

Монитор безопасности
Проводит политику обеспечения мер безопасности на локальном компьютере, охраняя системные ресурсы и выполняя процедуры аудита и защиты объектов.

Менеджер виртуальной памяти
Использует схему управления, при которой каждый процесс получает собственное достаточно большое адресное пространство, защищенное от воздействия других процессов. Менеджер памяти также обеспечивает низкоуровневую поддержку для менеджера кэш-памяти.

Модули, обеспечивающие работу менеджеров/диспетчеров исполняемой части Windows NT

Работа каждого из перечисленных модулей базируется на использовании 4-х групп функций, объединенных в блок «*Управление объектами/Исполняемые RTL*», так же входящих в исполняемую часть:

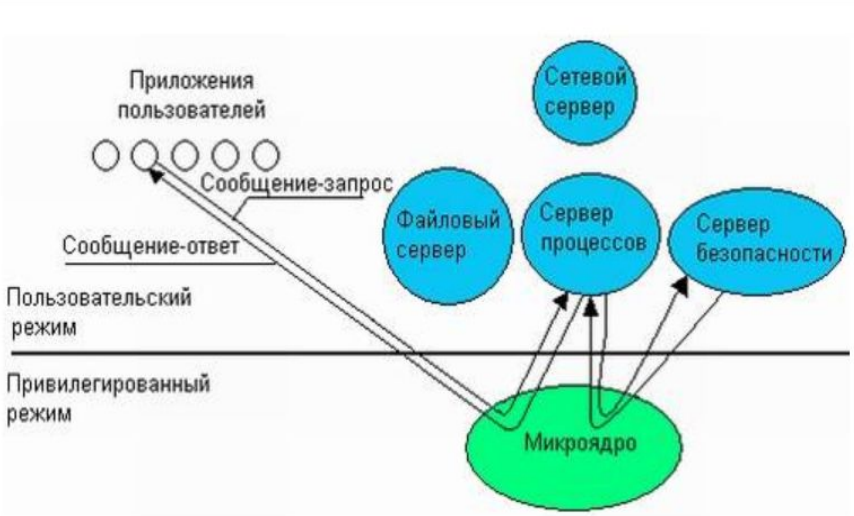
Менеджер объектов
Создает, удаляет объекты и абстрактные типы данных, а также управляет ими.
Объекты используются в Windows NT для представления таких ресурсов операционной системы, как процессы, потоки и объекты синхронизации

Механизм LPC (Local Procedure Call, локальный вызов процедуры). Используется операционной системой для передачи сообщения между клиентским процессом и процессом сервера *на том же самом компьютере* через специальные объекты – *порты*.

Набор библиотечных функций общего типа: обработка строк, арифметические операции, преобразование типов данных, обработка структур.

Процедуры распределения памяти обеспечивают взаимообмен между процессами через память с использованием двух специальных типа объектов синхронизации – *ресурс* и объект *fast mutex*.

Микроядерная архитектура ОС. Цена перехода на модель клиент/сервер

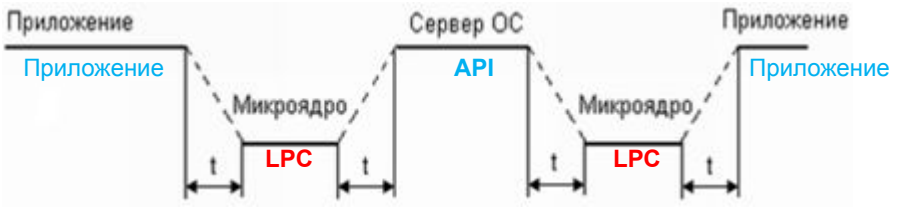


Для переключения между режимом ядра и пользовательским режимом процессору требуется несколько десятков микросекунд. Процессор 3 ГГц успевает выполнить за 10 микросекунд 30 000 инструкций.
Поэтому постоянное переключение режимов существенно снижает производительность системы.

Модули пользовательского режима не видят друг друга и не могут напрямую взаимодействовать между собой.
Для обмена данными им требуется смена режима.

Допустим, приложение хочет прочитать клавиатурный буфер.
Для этого оно вызывает функцию API, поскольку это единственный разрешенный метод запроса к ОС.

Вместо прямого вызова API приложение обращается к модулю микроядра, который обслуживает пересылку сообщений (LPC - Local Procedure Call, *запрос локальной процедуры*).
В API запрос поступает из вспомогательного модуля, а не от приложения.



Сообщение отправляется из приложения, пересекает границу User/Kernel и попадает в модуль микрояра LPC. Откуда через границу User/Kernel возвращается в API.
Функция API запрашивает у ОС выполнение операции, что еще раз приведет к переключению режима.

Таким образом, доступ к сети, файловой системе, экрану или пользовательскому интерфейсу предполагает **многократное переключение режимов работы процессора**.

Компромиссное решение в архитектуре Windows NT

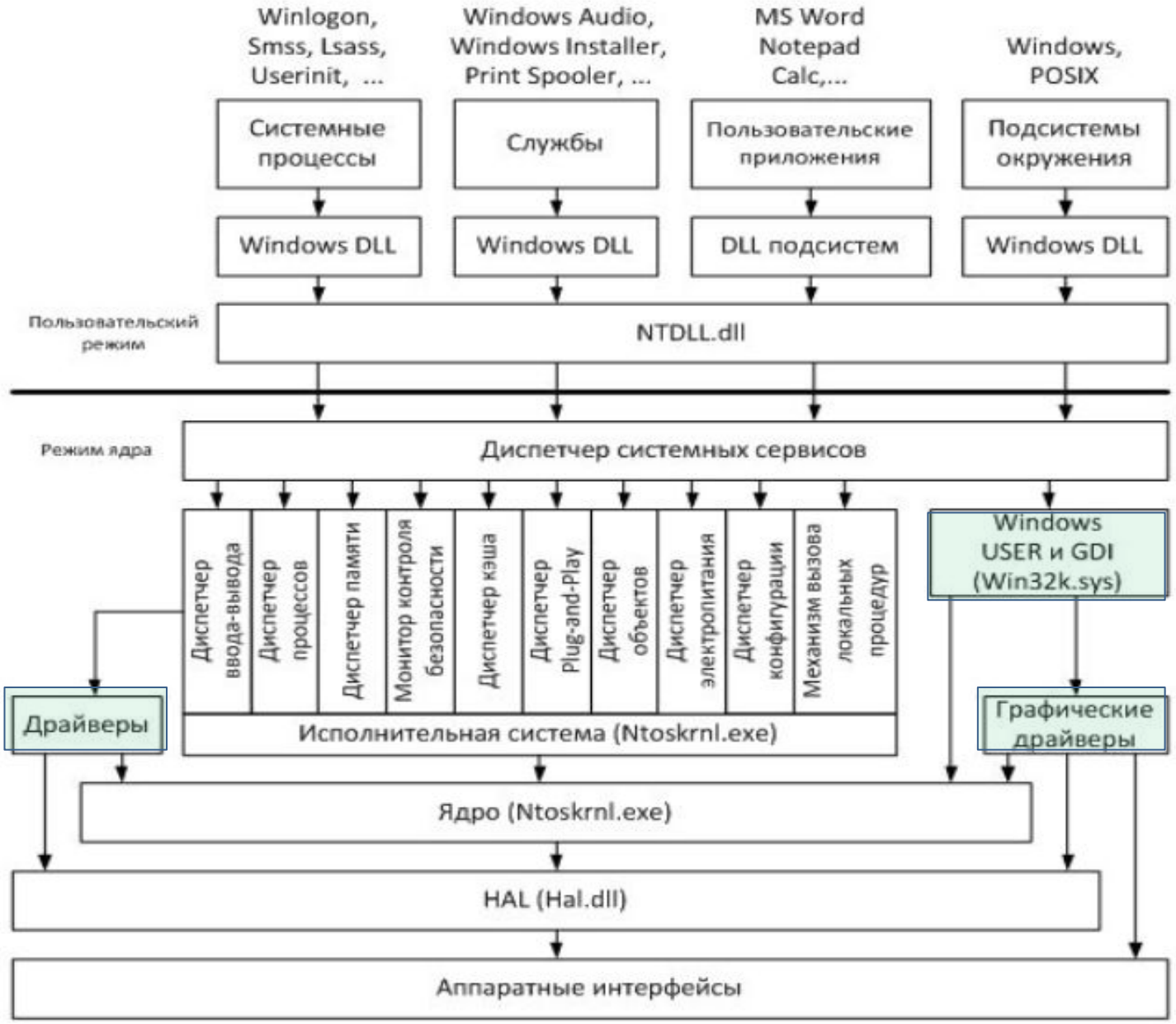
Начиная с версии 4.0 "родная" программная среда (**Win32 API**) *была перенесена из пользовательского режима в режим ядра*.

В режим ядра были перемещены три части ОС:
* **USER** (менеджер пользовательского интерфейса)
* **GDI** (менеджер графики)
* **Драйвер видео-платы**.

Модуль **USER** запускается в ответ на щелчок или перетаскивание управляющего элемента (окно, кнопка, бегунок, переключатель, флажок, список, раскрывающийся список или панель инструментов).
Модуль **GDI** обеспечивает низкоуровневые функции графического пользовательского интерфейса. В этом модуле обрабатываются растры, цвета, виды курсора, значки и шрифты. Когда текстовому процессору нужно вывести строку шрифтом Times Roman, то символы на экран помещает GDI.
USER и **GDI** являются важными и неотъемлемыми, а также хорошо отлаженными частями ОС, поэтому оправдано перемещение этих модулей в режим ядра.

Перенос драйверов графических плат и принтеров в режим ядра - неудачная затея, но, начиная с Windows XP Professional, Microsoft обеспечила несколько программных инструментов, помогающих во время разработки драйверов.
Поэтому в Windows XP Professional компоненты режима ядра стали стабильнее и сократилось число причин для появления «синего экран смерти» (BSOD).

Архитектура ОС Windows NT (XP – 10)

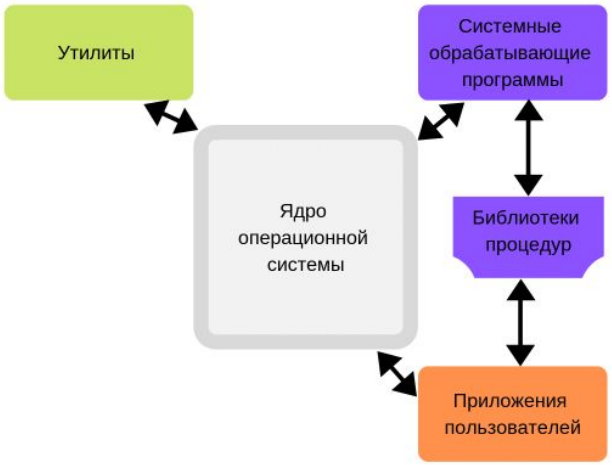


"Classic" Mac OS

https://en.wikipedia.org/wiki/Classic_Mac_OS

Аппаратная платформа	•Motorola 68k (1.0–8.1) •PowerPC (7.1.2–9.2.2)
Тип ядра	Монолитное для 68k, Наноядро для PowerP

Монолитное ядро



Монолитное ядро - схема, при которой все компоненты ядра ОС являются *составными частями одной программы*, используют *общие структуры данных* и взаимодействуют друг с другом путём *непосредственного вызова процедур*.

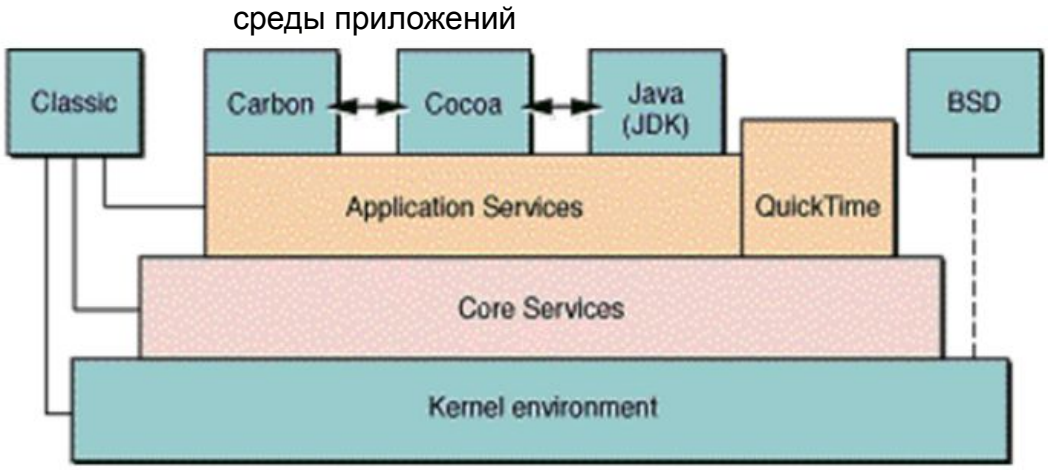
Все части монолитного ядра работают в одном адресном пространстве в ***привилегированном*** режиме!

Наноядро

Крайне упрощённое и минималистичное ядро выполняет лишь одну задачу — обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки прерываний от аппаратуры наноядро, в свою очередь, посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему программному обеспечению при помощи того же механизма прерываний.

Наноядро в версии Mac OS Classic для PowerPC использовалось для того, чтобы транслировать аппаратные прерывания, генерировавшиеся их компьютерами на базе процессоров PowerPC в форму, которая могла «пониматься» и распознаваться Mac OS для процессоров Motorola 680x0.
Таким образом, наноядро эмулировало для Mac OS «старое» 680x0 железо.

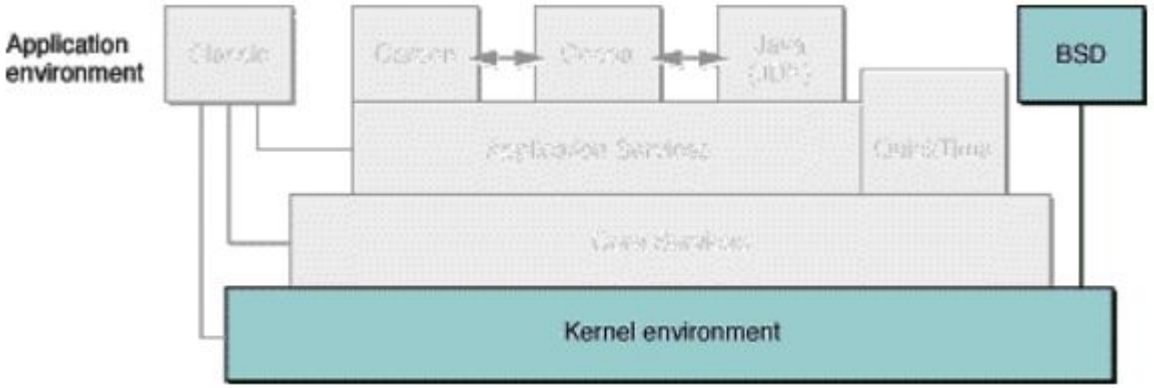
Архитектура OS X



Ядро вместе с другими основными частями OS X вместе именуются **ОС Darwin**.

Darwin - это полная операционная система, основанная на многих из тех же технологий, которые лежат в основе OS X.

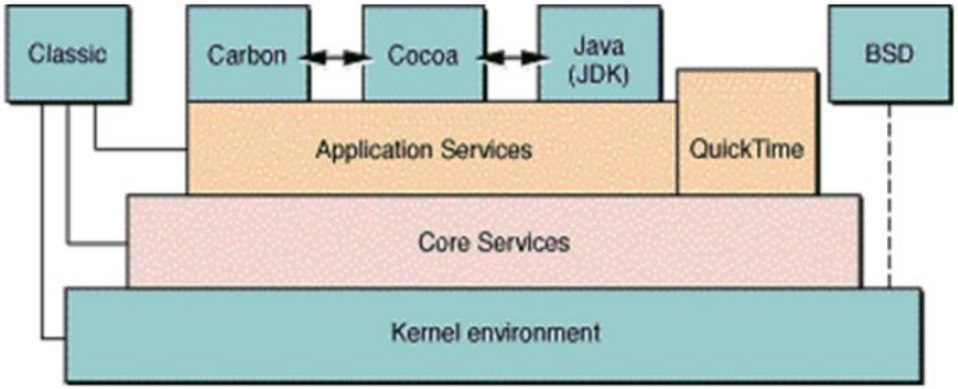
Однако **Darwin** не включает проприетарную графику или уровни приложений Apple, такие как Quartz, QuickTime, Cocoa, Carbon или OpenGL.



https://devyanibajadeja.wordpress.com/kernel-architecture/

https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/Architecture/Architecture.html#//apple_ref/doc/uid/TP30000905-CH1g-CACDAEDC

Архитектура OS X



Технология Darwin основана на технологиях *BSD* , Mach 3.0 и Apple.

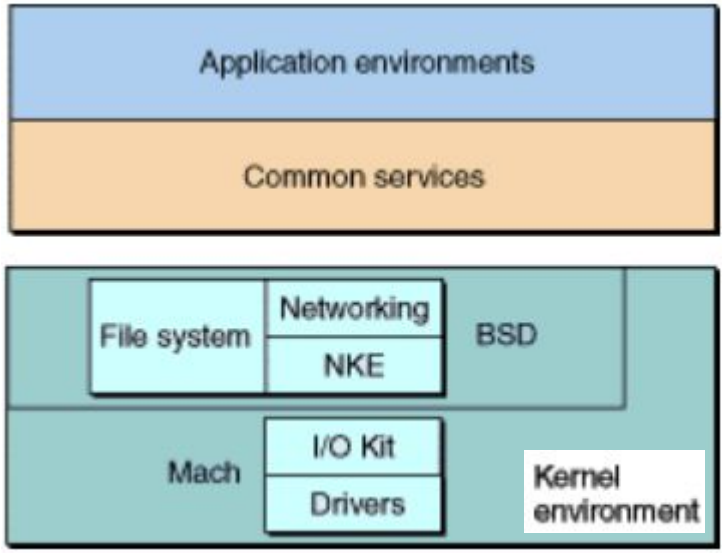
Mach предоставляет набор абстракций для работы с управлением памятью, межпроцессным (и межпроцессорным) взаимодействием (IPC) и другими низкоуровневыми функциями ОС.

В быстро меняющейся аппаратной среде это обеспечивает полезный *уровень изоляции* операционной системы от базового оборудования.

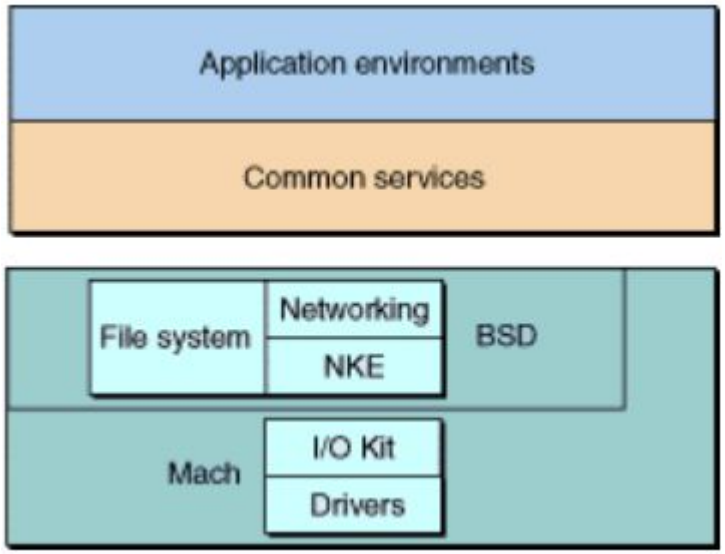
BSD - это тщательно спроектированная зрелая операционная система с множеством возможностей. Фактически, большинство современных коммерческих UNIX и UNIX-подобных операционных систем содержат большое количество кода BSD.
BSD также предоставляет *набор стандартных API*.

Новые технологии, такие как ***I / O Kit*** и ***Network Kernel Extensions (NKE)***, были спроектированы Apple, чтобы воспользоваться преимуществами расширенных возможностей, которые предоставляются моделью объектно-ориентированного программирования.

Архитектура ядра OS X



Архитектура OS X



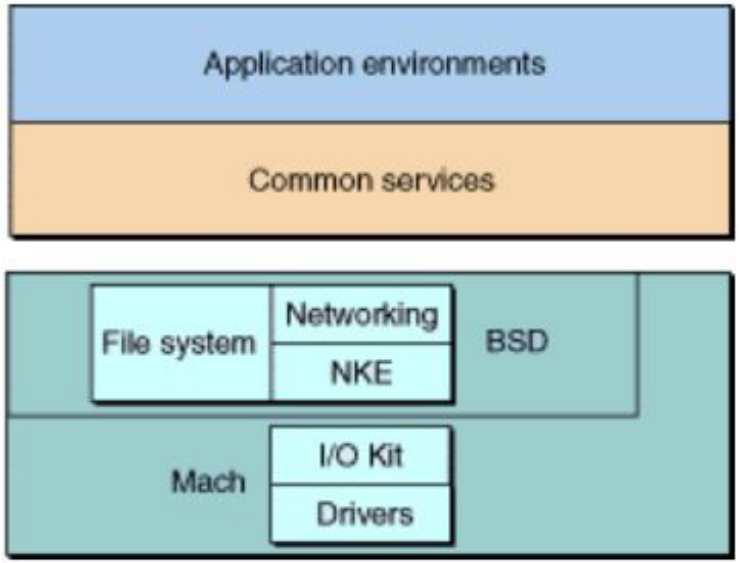
Среда ядра OS X включает микроядро Mach, BSD, I / O Kit, файловые системы и сетевые компоненты (NKE). Их и называют ядром OS X.

Mach управляет ресурсами процессора, такими как использование ЦП и память, управляет планированием, обеспечивает защиту памяти и предоставляет инфраструктуру, ориентированную на обмен сообщениями, для остальных уровней ОС.

Компонент Mach обеспечивает:

- нетипизированное межпроцессное взаимодействие (*IPC*);
- вызовы удаленных процедур (*RPC*);
- поддержка планировщика для симметричной многопроцессорной обработки (*SMP*);
- поддержка сервисов в *реальном времени;*
- поддержка виртуальной памяти;
- поддержка модульная архитектура.

Архитектура OS X

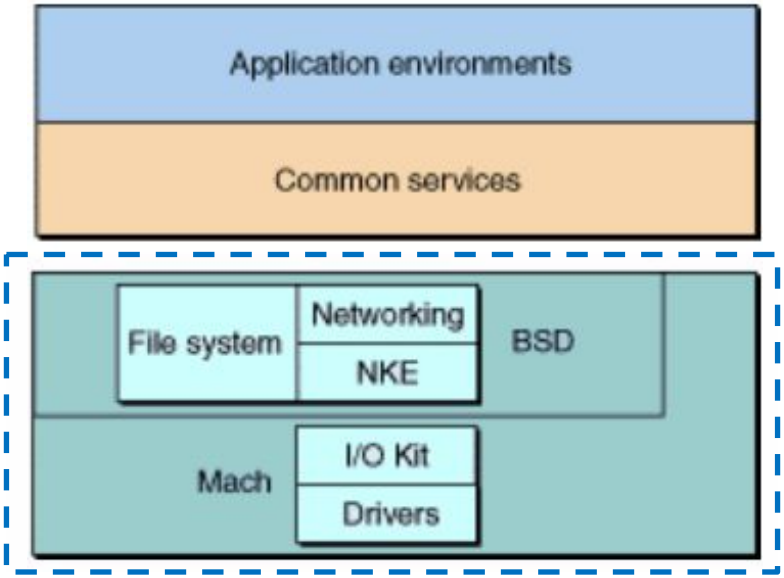


Компонент BSD предоставляет:
- файловые системы
- поддержку сетевых протоколов (кроме уровня аппаратного устройства)
- модель безопасности UNIX
- модель процесса BSD, включая идентификаторы процессов и сигналы API ядра FreeBSD
- поддержка ядра для модели потоков POSIX (*pthreads*)
- различные API POSIX

Сетевой компонент обеспечивает:
- 4.4BSD TCP / IP стек и API сокетов
- поддержка IP и DDP (транспорт AppleTalk)
- *множественная адресация*
- маршрутизация
- поддержка *многоадресной рассылки*
- настройка сервера
- фильтрация пакетов
- поддержка Mac OS Classic (через фильтры)

I / O Kit обеспечивает основу для упрощенной разработки драйверов, поддерживающую многие категории устройств.
I / O Kit представляет собой объектно-ориентированную архитектуру ввода-вывода.
Компонент I / O Kit обеспечивает:
- настоящий Plug & Play
- динамическое управление устройством
- динамическая загрузка драйверов
- управление питанием для настольных систем и портативных устройств
- возможности многопроцессорной работы.

Архитектура OS X. Гибридное ядро XNU

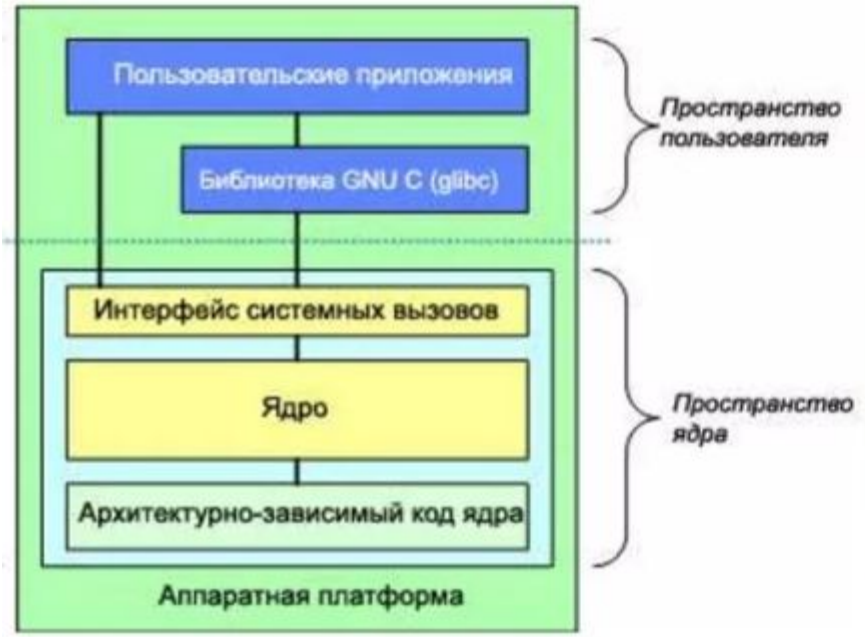


Гибридное ядро XNU

Ядро XNU является современным ***гибридным*** ядром, сочетающем в себе преимущества как монолитных, так и микроядер, в частности, возможности по передаче сообщений микроядер для повышения модульности системы и защиты памяти разных модулей и высокую скорость монолитных ядер в некоторых критичных задачах.

В настоящее время XNU может работать на процессорах с архитектурой ARM, x86, x86-64. Поддержка PowerPC закончилась начиная с версии Mac OS X 10.6. Поддерживаются как одноядерные, так и SMP-системы.

Структура ОС Linux и архитектура ядра Linux

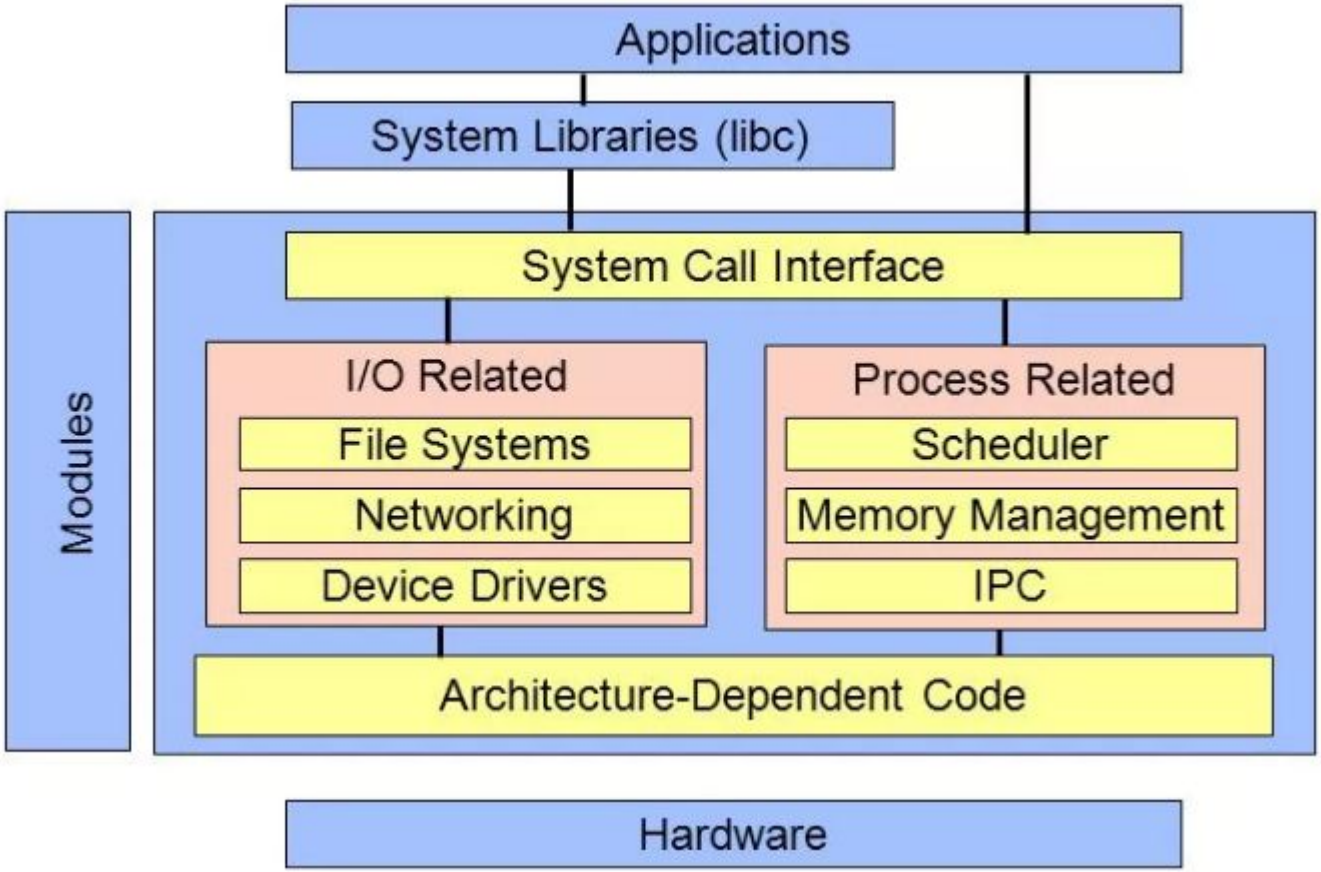


Структура ОС Linux

Структура ядра ОС Linux

Архитектура ядра Linux

I/O Related – Управление вводом выводом:
File Systems - Файловые системы.
Networking - Сетевой стек.
Device Drivers - Драйверы устройств.



Process Related – Управление процессами:
Scheduler - Диспетчер процессов.
Memory Management - Управление памятью.
IPC – механизм межпроцессного взаимодействия.

https://slideplayer.com/slide/7320201/