Операционные системы

Лекция 3

Ядро операционной системы

План лекции

- Ресурсы операционных систем
- Структура ядра и его функции
- Принципы построения ядра
- Объекты ядра и основные операции над ними
- Многослойная структура ОС
- Средства аппаратной поддержки ОС
- Машинно-зависимые компоненты ОС
- Утилиты и СОП
- Библиотеки и программы дополнительных услуг

Ресурсы операционных систем

Продолжая наше повествование про операционные системы, хотелось бы поговорить про то как устроена и работает система, НО для начала затронем базовые вещи о таком понятии как «**ресурс**»

Почему? Так ведь как уже говорилось операционная система управляет ими, а значит базовые знания нам не помешают

Что же такое ресурс в нашем случае?

Ресурсом является любой объект, который может распределяться внутри системы. Ресурсы могут быть **разделяемыми**, когда несколько процессов могут их использовать ***одновременно*** (в один и тот же момент времени) или ***параллельно*** (в течение некоторого интервала времени процессы используют ресурс попеременно), а могут быть и **неделимыми**

Ресурсы операционных систем

Термин «**ресурс**» обычно применяется по отношению к неоднократно используемым, относительно стабильным и «дефицитным» объектам, которые запрашиваются, используются и освобождаются процессами в период их активности

В первых системах программирования под понятием «ресурсы» понимали: процессорное время, память, каналы ввода/вывода и периферийные устройства. Впоследствии понятие ресурса стало более универсальным и общим

В настоящее время понятие ресурса превратилось в абстрактную структуру с целым рядом атрибутов, характеризующих способы доступа к этой структуре и ее физическое представление в системе. К ресурсам стали относиться и такие объекты, как сообщения и синхросигналы, которыми обмениваются задачи

Общая схема выделения ресурсов

А как выделяются эти самые ресурсы?

При необходимости использовать какой-либо ресурс задача обращается к **супервизору** ОС (центральному управляющему модулю) и сообщает о своем требовании. При этом указывается вид ресурса и, если необходимо, его объем

Затем управление передаётся ОС, которая:

1. Переключает процессор в привилегированный режим
2. Выполняет запрос
3. Возвращает управление задаче, выдавшей данный запрос

Но что если ресурс недоступен? Или его недостаточно?

Общая схема выделения ресурсов

Всё просто, если ресурс занят, задача блокируется и помещается в очередь ожидания

А вообще, ресурс может быть выделен задаче в следующих случаях:

- Ресурс свободен, и в системе нет запросов от задач более высокого приоритета к запрашиваемому ресурсу
- Текущий запрос и ранее выданные запросы допускают совместное использование ресурсов
- Ресурс используется задачей низшего приоритета и может быть временно отобран (разделяемый ресурс)

Схема освобождения ресурсов

Если же говорить об обратном процессе, то освобождение ресурса происходит следующим образом:

1. Задача сообщает супервизору об освобождении ресурса (либо ОС забирает ресурс после выполнения системной задачи)
2. Супервизор освобождает ресурс и проверяет очередь к освободившемуся ресурсу
3. При наличии очереди в соответствии с принятой дисциплиной обслуживания и в зависимости от приоритета заявки он выводит из состояния ожидания ждущую ресурс задачу и переводит ее в состояние готовности к выполнению

Распределением ресурсов между задачами (процессами) пользователей и системными процессами занимается **ядро операционной системы**

Классификация ресурсов

Ну и куда же без классификации ресурсов, а бывают они такими:

По **реальности их существования** :

- Физический – ресурс, который реально существует и при распределении обладает всеми физическими свойствами и характеристиками (диски, принтеры и сетевые устройства)
- Виртуальный – мнимый ресурс, модель некоторого ресурса. которая реализуется в программно-аппаратной форме (RAM)

По **возможности расширения свойств**:

- Эластичный – допускает виртуализацию
- Жёсткий – не допускает создание виртуального процесса

По **степени активности**:

- Активные – при использовании способны выполнять действия по отношению к другим ресурсам или процессам, которые приводят к изменению последних (ЦП)
- Пассивные – ресурсы, над которыми можно производить дополнительные действия, которые приводят к их изменению (RAM)

Классификация ресурсов

По **степени важности**:

- Главный – без выделения этих ресурсов процесс принципиально существовать не может (ЦП, RAM)
- Второстепенный – допускает некоторые альтернативные различия (хранение данных на HDD)

По **функциональной избыточности**:

- Дорогие
- Дешёвые

По **структуре**:

- Простые – ресурсы, которые не содержат составных частей
- Составные

Классификация ресурсов

По **восстанавливаемости**:

- Воспроизводимые ресурсы – ресурсы, при распределение которых допускается многократное выполнение следующей последовательности: запрос – использование – освобождение
- Потребляемые ресурсы – ресурсы, при распределении которых выполняется следующая последовательность: освобождение – запрос – использование

По **характеру использования**:

- Последовательно используемые – в отношение, которого допустимо строго последовательное действие: запрос – использование – освобождение (печатное устройство)
- Параллельно используемые – ресурсы, которые одновременно используются более чем одним процессом (массив данных находится в некоторой области RAM)
- Критически последовательный ресурс, разделяемый несколькими параллельными процессами (буфер, хранящий принятые данные)

Объекты ядра

Но всё ли так просто у ядра ОС? Как понять какой ресурс запрашивается?

Для этого есть «объекты»**. Объекты ядра** используются системой и приложениями для управления множеством разных ресурсов: процессами, потоками, файлами и т.д. Приложение не может напрямую обращаться к объектам ядра читать и изменять их содержимое

Понятие «**объект ядра**» имеет разный смысл не только в разных операционных системах, но даже у разных авторов, описывающих одну и ту же операционную систему

К тому же, одни объекты существуют во всех операционных системах (процесс, поток), а другие специфичны для конкретной операционной системы (WindowStation). Здесь будет представлена только общая картина и несколько отрывочных иллюстраций. Многие из объектов ядра будут подробно рассматриваться в соответствующих лекциях

Объекты ядра

Как было сказано, определение различно в различной литературе, так например, согласно Microsoft объект это:

Объект ядра – это коллекция данных, являющихся частью режима ядра операционной системы, которыми управляет Диспетчер объектов Windows (Windows Object Manager)

В свою очередь в рамках ядра Linux разработчики явно не используют определение «объект ядра». Вместо этого акцент делается не на формальном определении объектов ядра, а на практической реализации механизмов управления ресурсами и взаимодействия с аппаратурой

При этом структуры данных которые можно было рассматривать в роли «объектов» всё же имеются

Категории объектов ядра в Windows

Типичные объекты режима ядра включают следующие категории объектов:

- Объекты устройств
- Объекты файлов
- Символические ссылки
- Разделы реестра
- Потоки и процессы
- Объекты диспетчера ядра, такие как объекты событий и объекты мьютексов
- Объекты обратного вызова
- Объекты section

Категории объектов ядра в Windows

В качестве примера конкретных объектов можно привести:

- Маркеры доступа (access token objects)
- Файлы (file objects)
- Проекции файлов (file-mapping objects)
- Порты завершения ввода вывода (I/O completion port objects)
- Задания (jobs)
- Почтовые ящики (mailslot objects)
- Мьютексы (mutex objects)
- Каналы (pipe objects)
- Процессы (thread objects)
- Ожидаемые таймеры (waitable timer objects)

Внутренняя структура объекта ядра

Каждый объект ядра на самом деле **просто блок памяти**, выделенный ядром и доступный только ему. Блок представляет собой структуру данных, в элементах которой содержится информация об объекте. Некоторые элементы (дескриптор защиты, счетчик числа пользователей и др.) присутствуют во всех объектах, но большая их часть специфична для объектов конкретного типа

Например, у объекта процесс может быть идентификатор, базовый приоритет и код завершения, а у объекта файл – смещение в байтах, режим разделения и режим открытия

Внутренняя структура объекта ядра

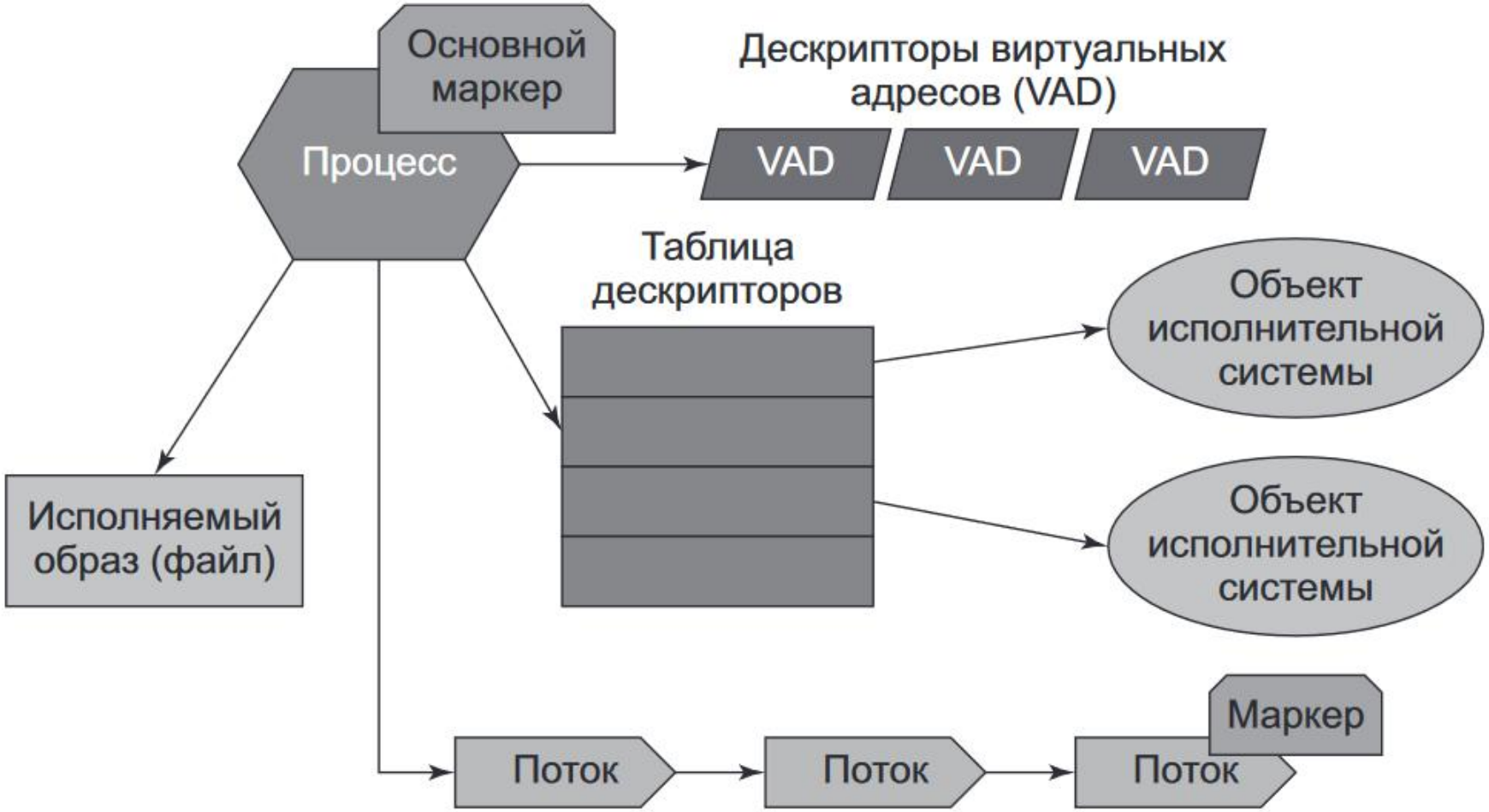
Приложению пользовательского режима предоставляется лишь описатель (**дескриптор**, HANDLE) – индекс в таблице объектов ядра процесса. Но сами объекты ядра принадлежат ядру, а не процессу

Объекты в режиме ядра имеют очень определенный жизненный цикл. В каждом объекте, есть счетчик пользователей объектом

Система удаляет объект ядра, когда счетчик обнуляется

Однако стоит отметить, что сама система не сможет определить перестал ли кто-то пользоваться объектом без явного на то указания со стороны приложения

Внутренняя структура объекта ядра



Пространство имен объектов

Среда режима ядра хранит объекты в виртуальной системе каталогов, также называемой пространством имен объектов. Это позволяет получить иерархический доступ к объектам с помощью родительских и дочерних объектов

Это пространство имен похоже на набор каталогов файловой системы, но не соответствует определенной файловой системе на компьютере

Данное понятие – пространство имён объектов относится к семейству операционных систем Windows NT

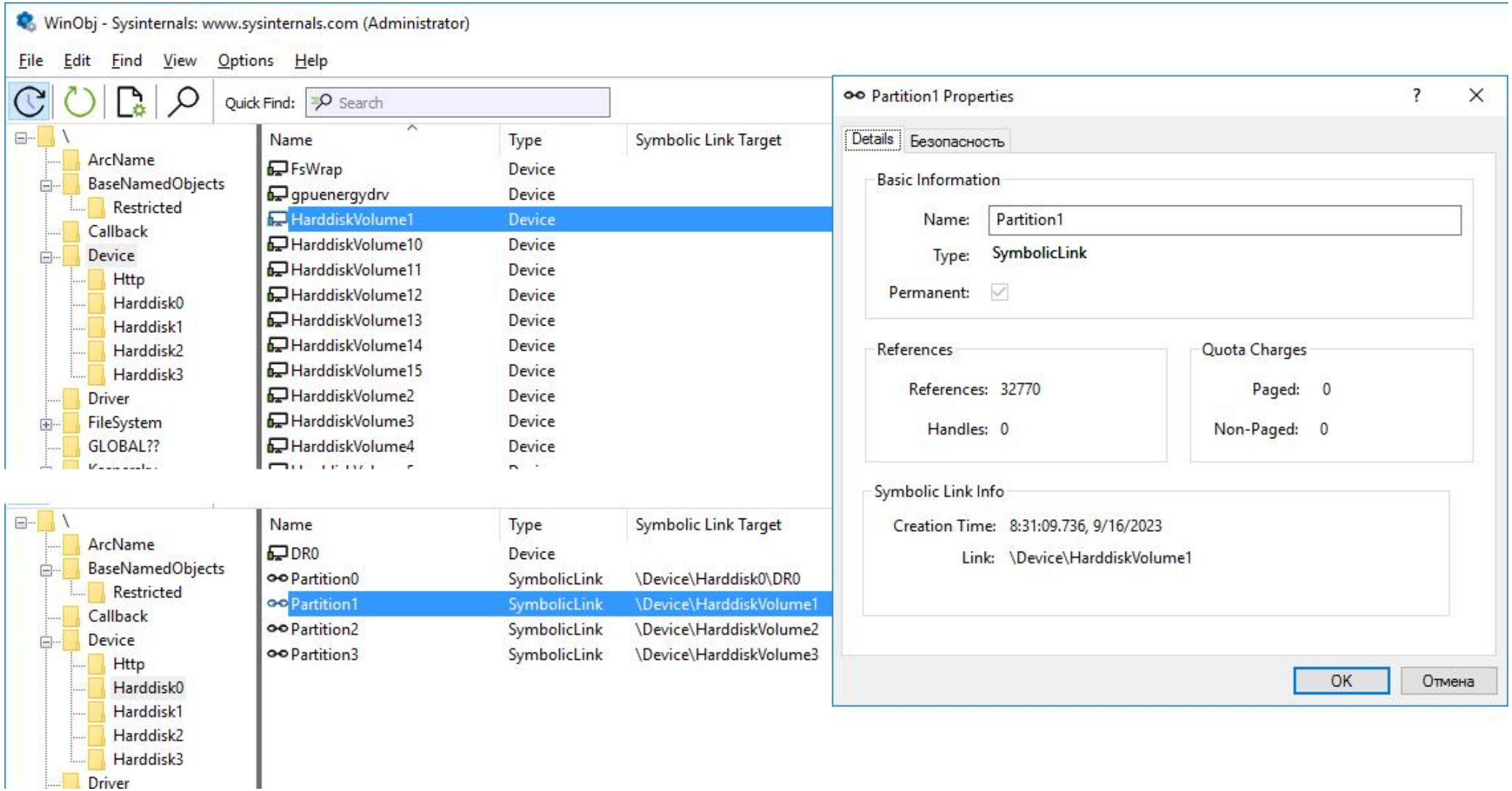
В Linux имеется нечто подобное, однако имеет куда более сложную структуру

Пространство имен объектов

В качестве нетерминальной вершины дерева используется объект «каталог объектов». Каталог включает информацию, необходимую для трансляции имен объектов в указатели на сами объекты. Вследствие необходимости выполнения навигации по каталогам ссылка на объект по имени работает существенно дольше, чем по описателю

«Увидеть» пространство имен можно только при помощи специальных инструментальных средств, например, с помощью утилиты WinObj из SysInternals

Пространство имен объектов



Основные операции над объектами ядра

Говоря об операциях над объектами, стоит отметить следующее:

- В большинстве случаев при создании объекта ему можно присвоить имя, некоторые атрибуты защиты и дополнительные параметры
- Операции над объектами происходят через описатель (какого-либо вида, в некоторых ОС описатель одного объекта может отличаться от описателей других объектов)

В Windows, например, функции для работы с объектами имеют общий вид CreateXxx или OpenXxx, где Xxx – имя объекта над которым выполняется операция. Далее по лекциям познакомимся поближе с такими функциями

В Linux нету подобных шаблонов в наименовании функций

Архитектура современных ОС

Так, а теперь рассмотрим такой вопрос: ну что ж там спрятано за кулисами ОС? Какова их архитектура?

Сначала определимся, что под архитектурой ОС обычно понимают структурную и функциональную организацию ОС на основе некоторой совокупности программных модулей

Современные ОС представляют собой хорошо структурированные модульные системы

Единой архитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС

Общий подход к структуризации ОС

Наиболее общим подходом к структуризации ОС является подразделение модулей две группы:

- Модули, выполняющие основные функции ОС – **ядро ОС**
- Модули, выполняющие вспомогательные функции ОС

Модули ядра выполняют базовые функции ОС:

- Управление процессами
- Управление памятью
- Управление устройствами ввода-вывода

Функции ядра

Функции, входящие в состав ядра, можно разделить на два класса:

1 класс – Функции для решения внутрисистемных задач организации вычислительного процесса (переключение контекстов процессов, загрузка/выгрузка страниц, обработка прерываний). Эти функции **недоступны** для приложений

2 класс – Функции для поддержки приложений (доступны приложениям). Эти функции создают для приложений так называемую **прикладную программную среду** и образуют **интерфейс прикладного программирования –** API. Приложения обращаются к ядру с запросами **– системными вызовами**. Функции API обслуживают системные вызовы – предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения

Функции ядра

Функции модулей ядра – это наиболее часто используемые функции ОС, а следовательно можно выделить следующие их свойства:

- Скорость выполнения этих функций определяет производительность всей системы в целом
- Все (большинство) модули ядра являются **резидентными**

Поясним, модуль считается резидентным, когда такой модуль находится в оперативной памяти **постоянно**

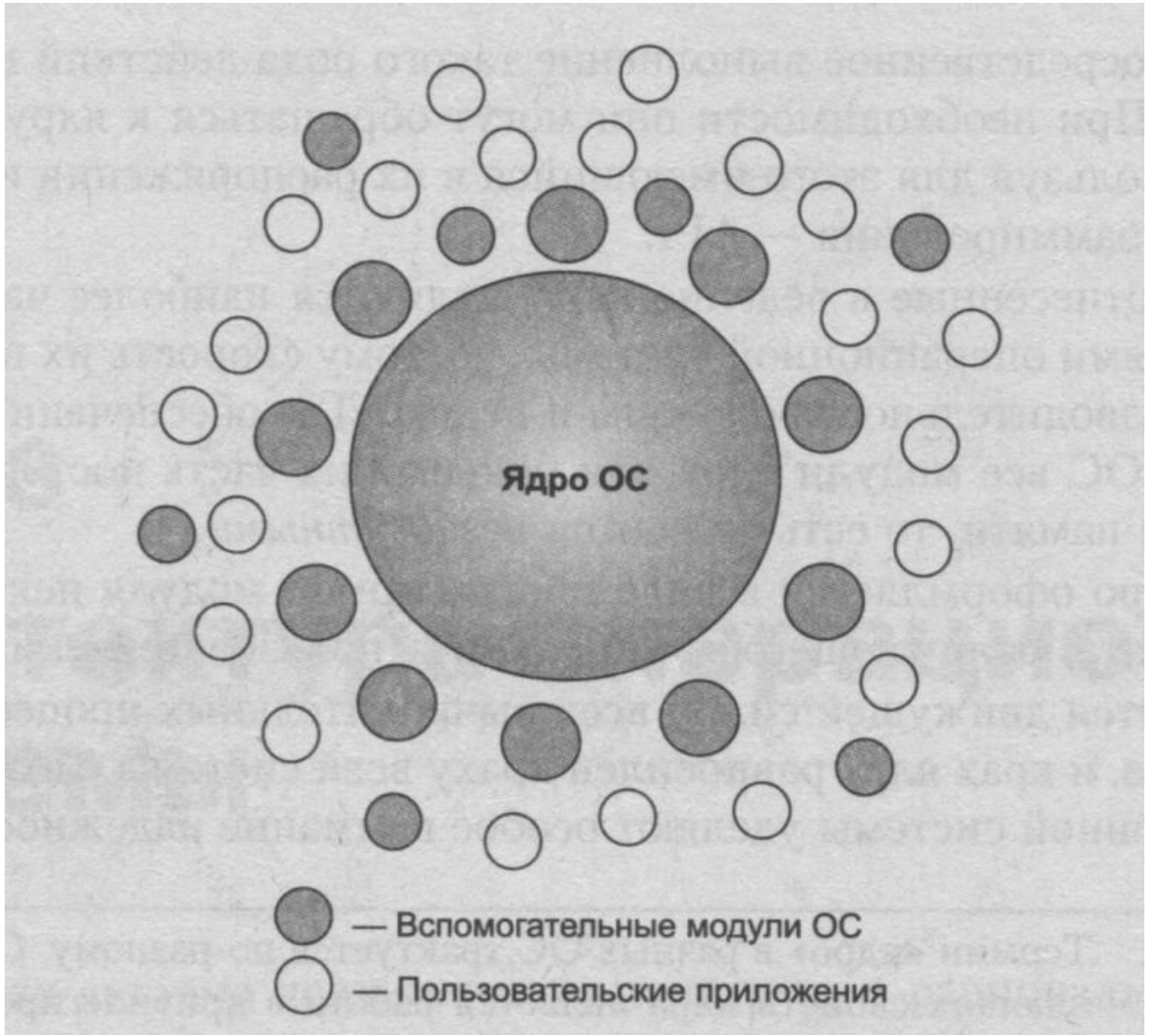
Т. е. модули операционной системы в большинстве случаев никогда не выгружаются из оперативной памяти!

Вспомогательные модули ОС

Остальные модули ОС выполняют полезные, но менее обязательные функции. Например, к таким вспомогательным модулям могут быть отнесены программы архивирования, дефрагментации диска и т.п. Вспомогательные модули ОС оформляются либо в виде приложений, либо в виде библиотек процедур и функций

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

Вспомогательные модули ОС



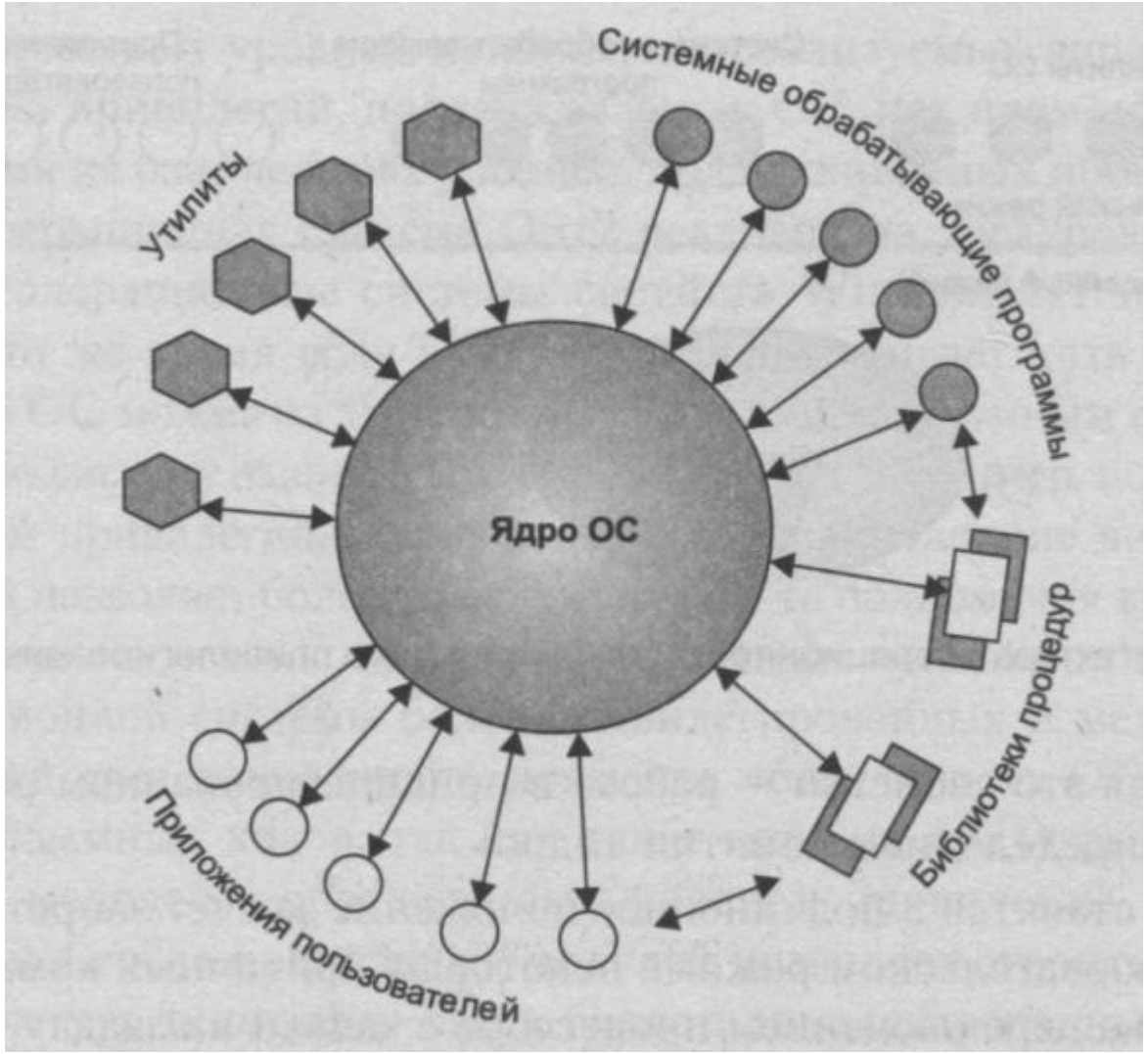
Вспомогательные модули ОС

Обычно вспомогательные модули подразделяются на следующие группы:

- **Утилиты** – программы, которые решают отдельные задачи управления и сопровождения компьютерной системы (сжатие, дефрагментация …)
- **Библиотеки процедур** и функций различного назначения упрощающие разработку приложений (библиотека математических функций, ввода-вывода и т.д.)
- **Программы предоставления пользователю дополнительных услуг** – специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (поставляемые в составе ОС)
- **Системные обрабатывающие программы** – текстовые и графические редакторы, компиляторы, компоновщики, отладчики

Вспомогательные модули ОС

Вспомогательные модули ОС обращаются к функциям ядра, как и обычные приложения, посредством системных вызовов



Вспомогательные модули ОС

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**

Постоянно в оперативной памяти располагаются только самые необходимые коды ОС, составляющие ее ядро

Такая организация ОС экономит оперативную память компьютера

Важным свойством архитектуры ОС, основанной на ядре, является возможность защиты кодов и данных операционной системы за счет выполнения функций ядра в **привилегированном режиме**

Главные задачи, решаемые ядром

Ядро операционной системы включает в свой состав следующие программы, решающие соответствующие задачи:

- Обработка прерываний
- Формирование и ликвидация различных процессов
- Переключение состояний процессов
- Диспетчеризация процессов
- Приостановка и последующая активация процесса

Главные задачи, решаемые ядром

- Синхронизация выполняемых процессов и организация обменов данными между ними
- Выполнение ввода и вывода данных
- Управление функциями распределения ресурсов памяти
- Поддержка файловых систем
- Обеспечение вызова и возврата при работе с процедурами
- Поддержка операций по учёту работ ЭВМ

Подсистемы

Модуль подсистемы управления процессами выполняет следующие функции:

- Создание и удаление процессов
- Распределение системных ресурсов между процессами
- Синхронизацию процессов
- Взаимодействие процессов

Специальная функция ядра, выполняемая **планировщиком процессов** (scheduler), разрешает конфликты между процессами в конкурентной борьбе за системные ресурсы

Модуль подсистемы управления памятью обеспечивает распределение памяти между процессами. Если для всех процессов недостаточно памяти, ядро перемещает части процесса или несколько процессов (чаще пассивных, ожидающих каких-либо событий в системе) в специальную область диска (область «подкачки»), освобождая ресурсы для выполняющихся (активных) процессов

Подсистемы

Файловая подсистема обеспечивает унифицированный интерфейс доступа к данным, расположенным на дисковых накопителях, и к периферийным устройствам. Одни и те же функции open, read, write могут использоваться как при чтении или записи данных на диск, так и при выводе текста на принтер или терминал

Файловая подсистема контролирует права доступа к файлу, выполняет операции размещения и удаления файла, а также выполняет запись/чтение данных файла. Поскольку большинство прикладных функций выполняется через интерфейс файловой системы (в том числе и доступ к периферийным устройствам), права доступа к файлам определяют привилегии пользователя в системе

Подсистемы

Файловая подсистема обеспечивает перенаправление запросов, адресованных периферийным устройствам, соответствующим модулям подсистемы ввода/вывода

Подсистема ввода/вывода выполняет запросы файловой подсистемы и подсистемы управления процессами на доступ к периферийным устройствам. Она взаимодействует с драйверами устройств – специальными программами ядра, обслуживающими внешние устройства

Ядро в сравнении с пользовательскими программами

Некоторые специфические особенности ядра Linux:

- **Ядро не имеет доступа к стандартным библиотекам языка С** (для повышения скорости выполнения и уменьшения объема кода)
- **Отсутствие защиты памяти**
- **Отсутствие замещения страниц**: каждый байт, используемый в ядре – это один байт физической памяти
- **В ядре нельзя использовать вычисления с плавающей точкой.** Активизация режима вычислений с плавающей точкой требует сохранения и проставления регистров устройства поддержки вычислений с плавающей точкой, помимо других рутинных операций
- **Фиксированный стек.** Стек в режиме ядра ни большой, ни изменяющийся. Поэтому в коде ядра не рекомендуется использовать рекурсию
- **Переносимость.** Платформо-независимый код, написанный на языке С, должен компилироваться без ошибок на максимально возможном количестве систем

Место ядра в операционной системе

Подводя некоторые итоги, можно сказать что:

Ядро (kernel) – это центральная и главная часть операционной системы, которая обеспечивает архитектуру связи с приложениями, организует и регулирует доступ к ресурсам компьютера. Дополнительно, но не как правило, предоставляет доступ к сетевым протоколам и к файловой системе

Ядро операционной системы взаимодействует с приложениями пользователя, с утилитами и с системными обрабатывающими программами

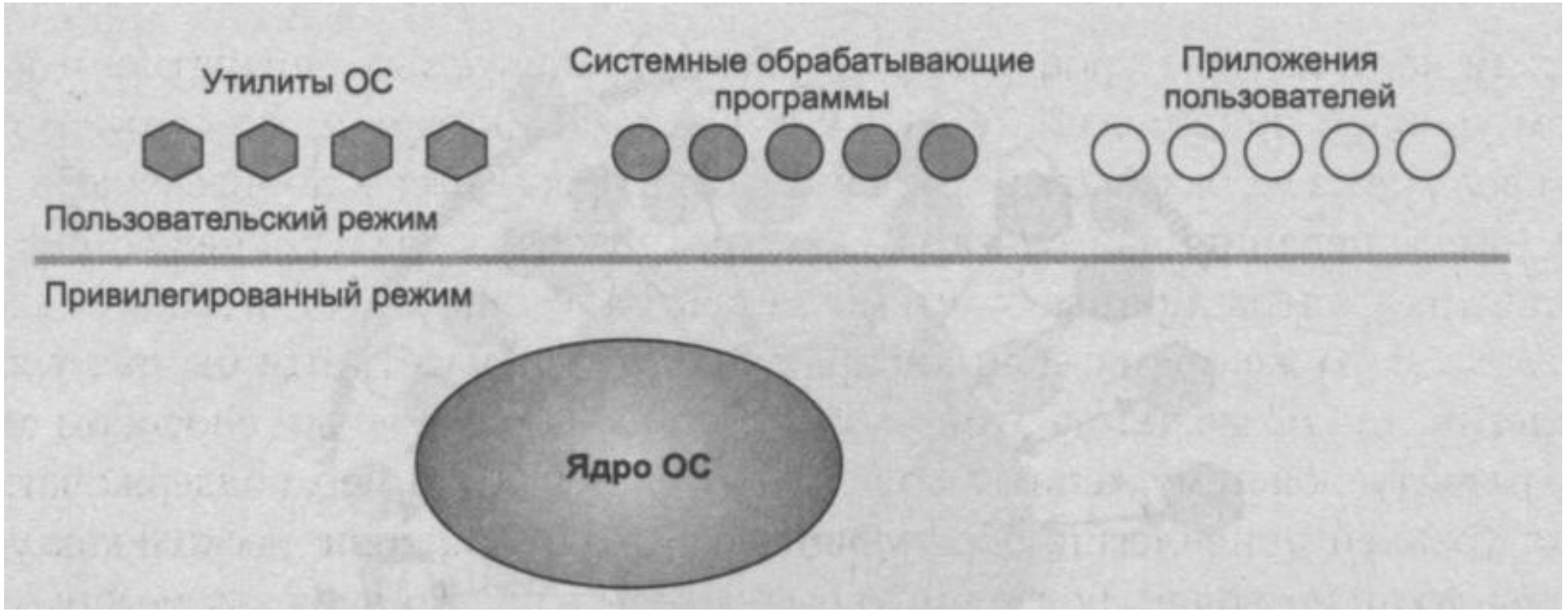
Ядро и привилегированный режим

Для того, чтобы многозадачная ОС могла выполнять функции защиты приложений от влияния друг друга и защиты самой себя от приложений, для нее на аппаратном уровне обеспечиваются определенные привилегии

Подразумевается, что ОС или некоторые ее части работают в **привилегированном** режиме, а приложения – в **пользовательском** режиме

Привилегии ОС обеспечиваются тем, что выполнение некоторых инструкций в пользовательском режиме запрещается. Например, выполнение инструкции доступа к памяти для приложения разрешается, если происходит обращение к области памяти, отведенной данному приложению, и запрещается при обращении к областям памяти, занимаемым ОС или другими приложениями

Ядро и привилегированный режим



Ядро и привилегированный режим

Так как ядро выполняет все основные функции ОС, то чаще всего именно ядро – та часть ОС, которая работает в привилегированном режиме

Ядро – низкоуровневая основа любой операционной системы, выполняемая аппаратурой в особом привилегированном режиме. Ядро загружается в память один раз и находится в памяти резидентно (постоянно), по одним и тем же адресам

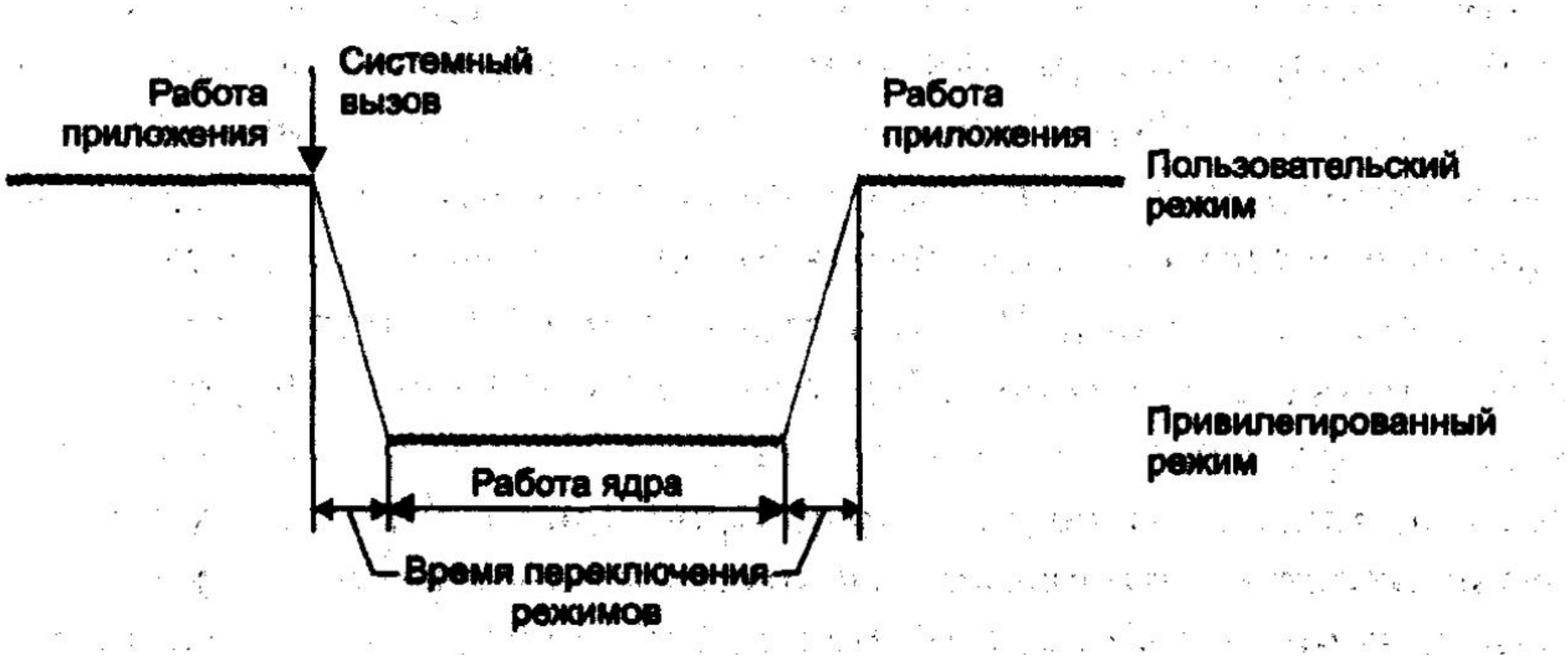
Ядро и приложения пользовательского режима

Каждое приложение пользовательского режима работает в своем адресном пространстве и защищено тем самым от вмешательства других приложений. Код ядра имеет доступ к областям памяти всех приложений, но сам полностью от них защищен. Приложения обращаются к ядру с запросами на выполнение системных функций

Необходимо обратить внимание, на то, что работа системы с привилегированным ядром замедляется за счет замедления выполнения системных вызовов

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

Ядро и приложения пользовательского режима



Виды структуры ядра операционной системы

Как уже говорилось ранее, возможны следующие типы структуры (архитектуры) ядра операционной системы:

- Монолитная структура ядра
- Модульная структура ядра
- Микроструктура ядра
- Экзо структура ядра
- Нано структура ядра
- Гибридная структура ядра
- Комбинированная структура ядра

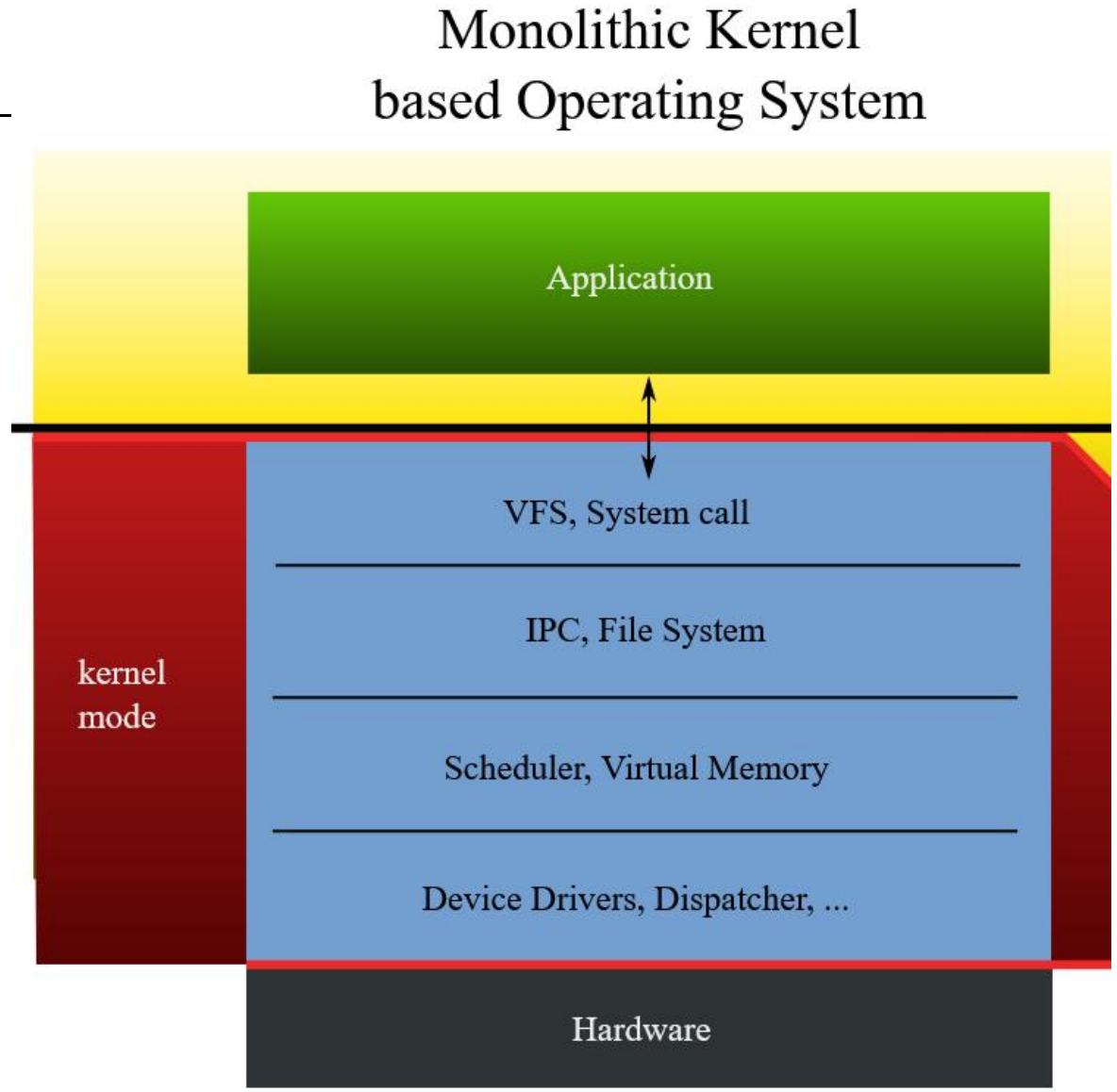
Монолитное ядро

Монолитное ядро формируется из обширного комплекта абстракций оборудования. Все элементы монолитного ядра работают в едином адресном формате. При такой организации операционной системы все составляющие части её ядра выступают как элементы основной программы, применяют одни и те же системы организации данных и работают друг с другом, используя непосредственный вызов процедуры. Определённой структуры данные операционной системы не имеют. По сути, это большой набор сервисных функций. Нет деления на слои и модули

Это самый старый метод формирования операционной системы. В качестве примера можно привести UNIX и классический MS-DOS. Достоинством является большая скорость выполнения операций и простота конструирования модулей. Недостатком можно считать работу ядра в едином адресном пространстве, так как неисправность в любом элементе способна блокировать работу всей системы

Монолитное ядро

Как видно на схеме, весь функционал по управлению вашим компьютером выполняется в режиме ядра!



Модульное ядро

Модульное ядро является современной и модифицированной версией структуры монолитного ядра операционной системы.

Она отличается от классического монолитного ядра тем, что не требует общей реструктуризации ядра при различных вариациях аппаратной оснастки компьютеров

У модульных ядер есть возможность подгружать различные модули (элементы) ядра, которые поддерживают нужное аппаратное оборудование (как пример, загрузка драйвера), причём подзагрузка модуля возможна как в динамическом режиме, то есть без перезагрузки операционной системы, так и в статике, когда выполняется переконфигурирование системы и её перезагрузка

Примеры модулей ядра в Linux: драйверы устройств, файловые системы, сетевые протоколы.

Микроядро

Микроядро решает лишь самые простые задачи по управлению процессами и имеет небольшой комплект абстракций оборудования

Основная часть функций выполняется специальными процессами пользователя, которые называются сервисами. Главным признаком микроядра можно считать распределение практически всех драйверов и элементов в процессах сервиса

Часто нет возможности загрузки расширительных модулей в такое микроядро

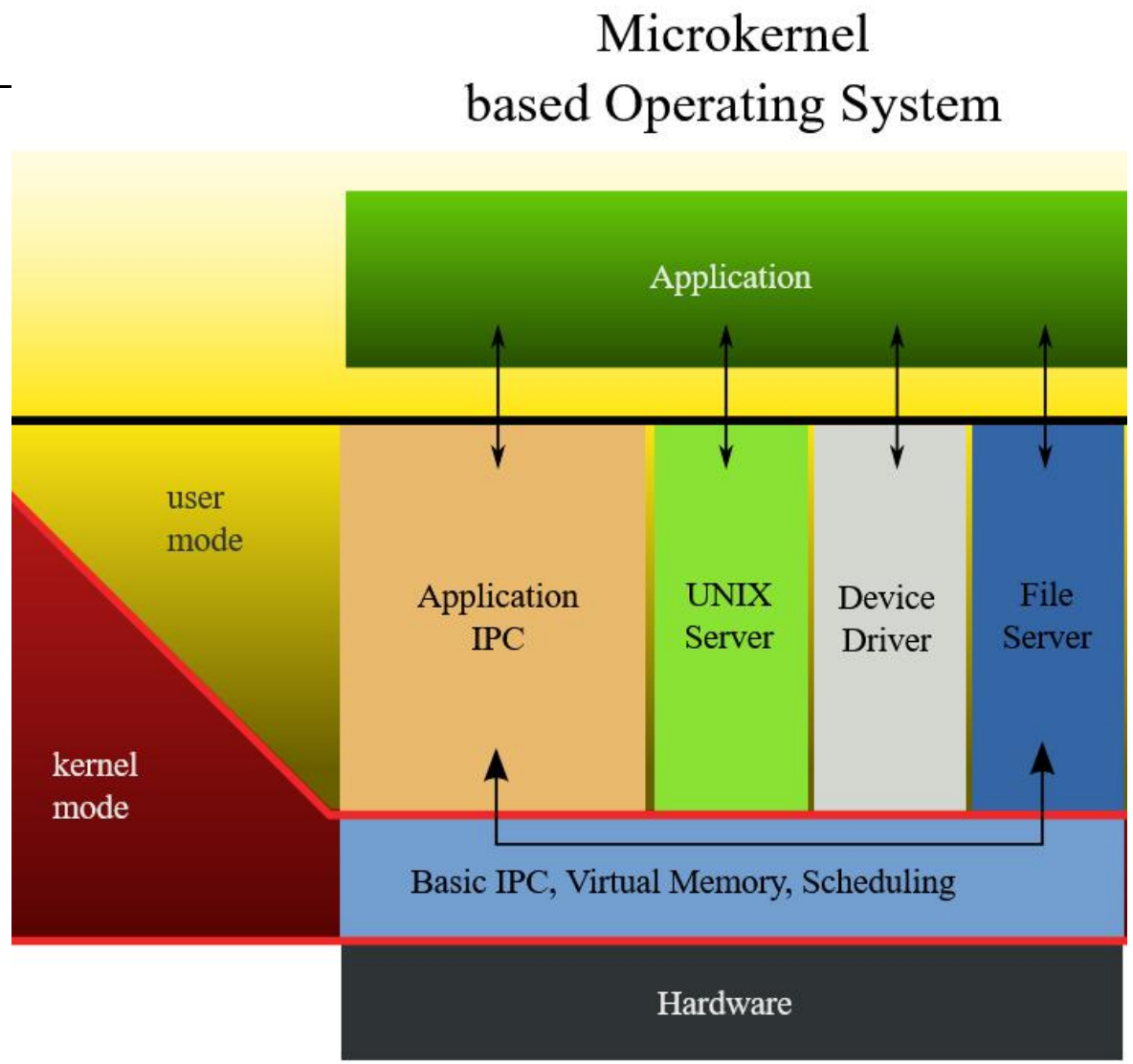
Преимущества: обладают переносимостью, высокая степень расширяемости, достаточная надежность, поддержка распределенных вычислений

Недостатки: снижение производительности из-за накладных расходов

Микроядро

В случае же с микроядром мы можем увидеть, что большая часть функционала по управлению компьютером вынесена в пользовательский режим!

Обычно на уровне ядра остаются только обработка прерываний, механизмы IPC, управление памятью, диспетчеризация процессов

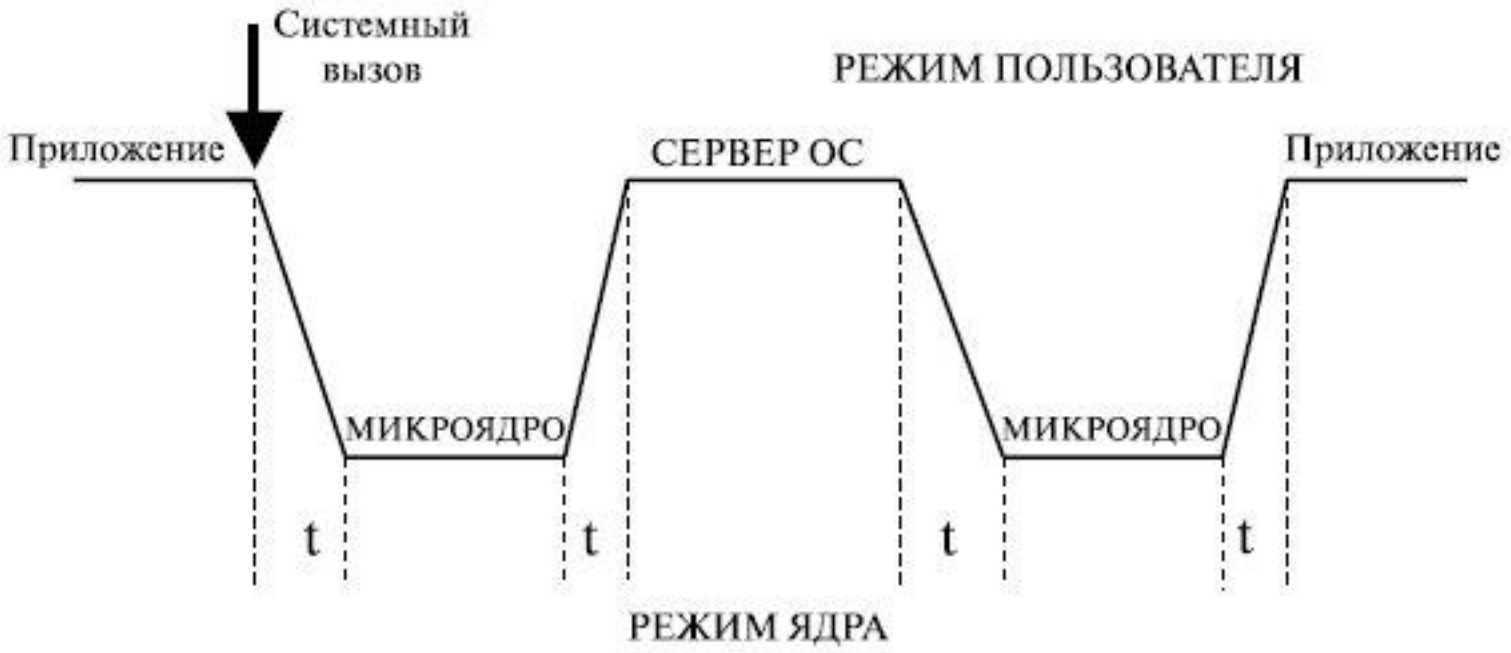


Микроядро

Из минусов упоминалось снижение производительности из-за накладных расходов, но почему? Если внимательно слушали как работает ОС и запомнили что происходит при любых запросах от ПО к ядру, то вы уже скорее всего поняли!

У нас добавляется дополнительные переключения между режимами!

Общение с сервером ОС идёт сугубо через микроядро



Гибридное ядро

Гибридное ядро является попыткой объединить преимущества и разные аспекты монолитного и микроядерного подходов:

- Отсутствие накладных расходов на переключение контекста между режимами
- Сохранение высокой производительности за счет работы основных компонентов в пространстве ядра
- Возможность изолировать определенные подсистемы для повышения безопасности

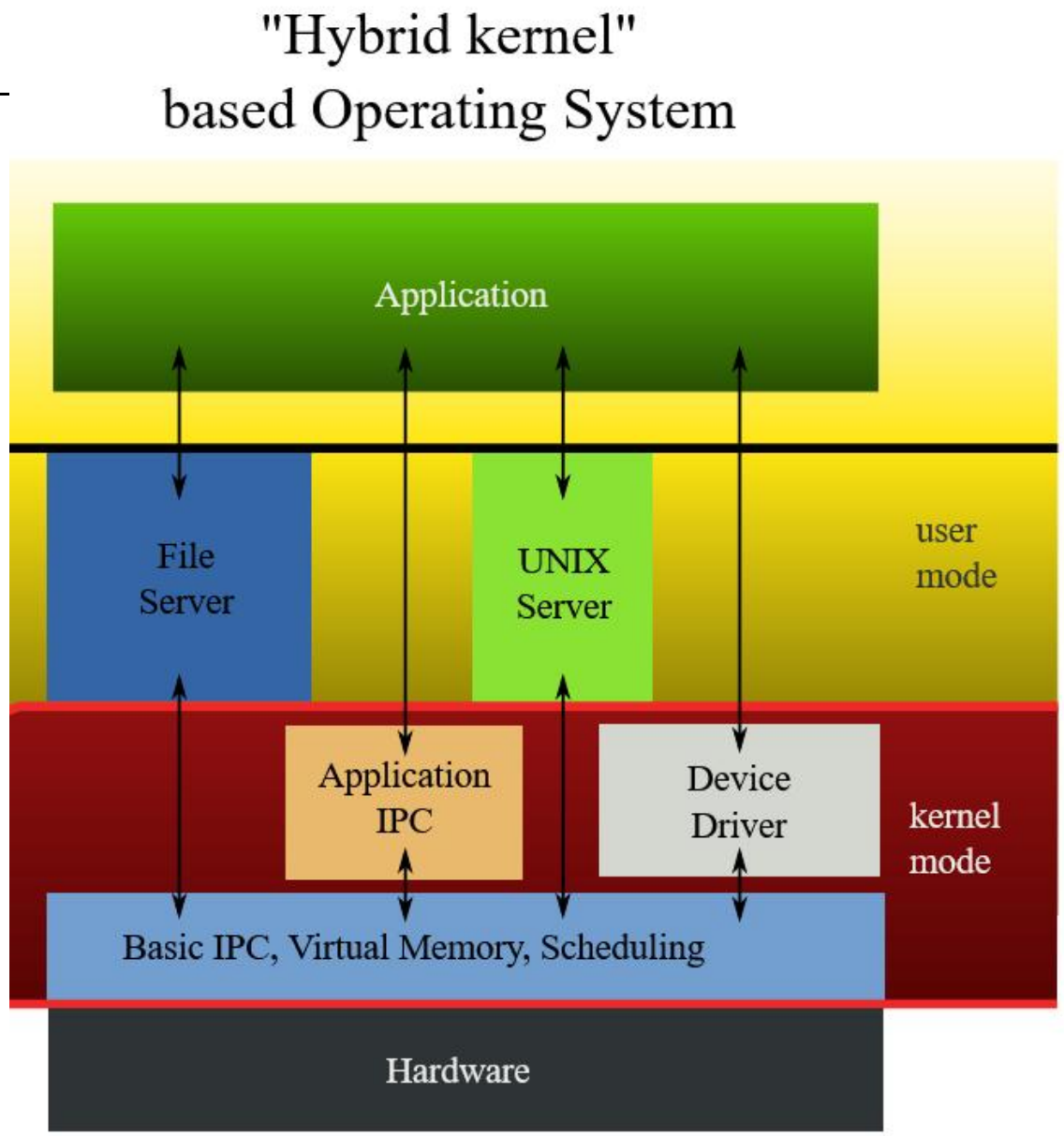
Таким образом, гибридные ядра достигают баланса между производительностью монолитных ядер и безопасностью микроядер за счет продуманного распределения компонентов и использования оптимальных механизмов взаимодействия между ними

Гибридное ядро

Как и говорилось, лишь основные компоненты работают в режиме ядра

Остальное же может быть вынесено для выполнения в пространство пользователя

Однако не каждое обращение к компонентам ОС приводит к системным вызовам



Другие виды ядер

Экзоядро – это ядро операционной системы, которое предоставляет только возможность взаимного обмена между процессами и надёжного распределения и высвобождения ресурсов

Наноядро имеет такую структуру, при которой очень простое ядро решает лишь проблему обработки аппаратного прерывания программы, которое генерируют различные блоки компьютера. Когда обработка прерывания, например, при нажатии символа на клавиатуре, завершается, наноядро пересылает результаты программе, которая выше по рангу. При этом пересылка тоже выполняется посредством прерываний

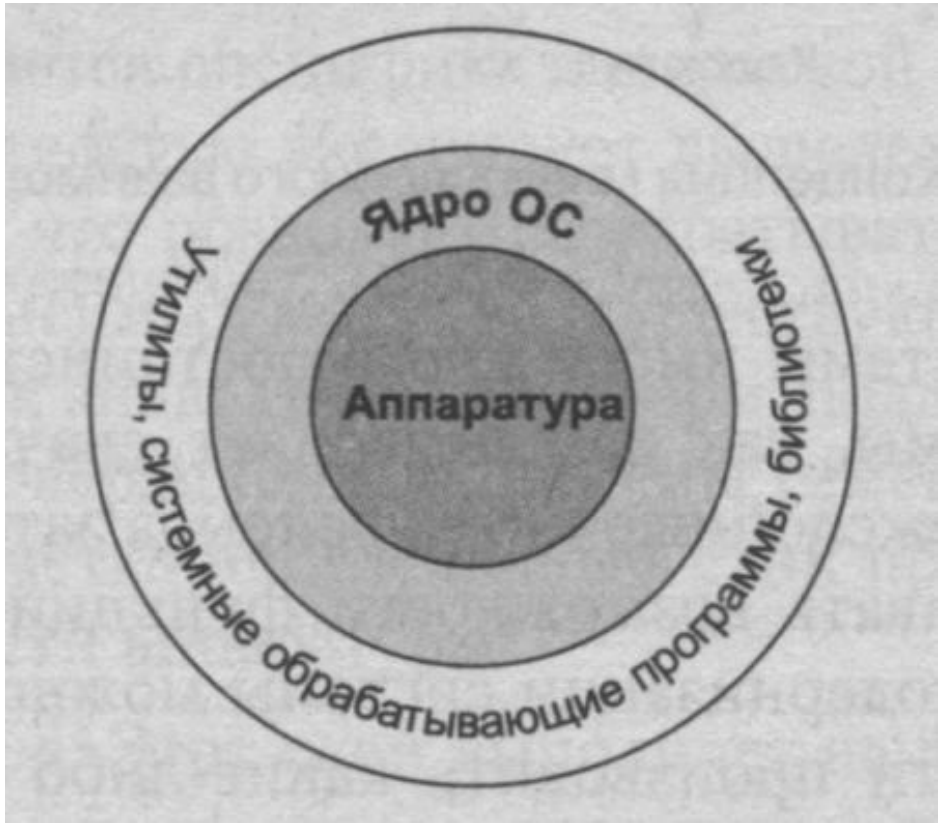
И, наконец, возможен **вариант комбинированного применения** вышеперечисленных вариантов построения структуры ядра операционной системы

Многослойная структура ОС

Вычислительную систему, работающую под управлением ОС на основе ядра, можно рассматривать как состоящую из трех иерархически расположенных слоев:

- Нижний слой образует **аппаратура**
- Промежуточный – **ядро**
- Верхний слой – **утилиты, обрабатывающие программы и приложения**

Каждый слой может взаимодействовать только со смежными слоями



Многослойная структура ОС

Многослойный подход является универсальным и эффективным способом декомпозиции сложных систем любого типа, в том числе программных

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 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

Многослойная структура ОС

Преимущества:

- Многослойные системы **хорошо реализуются.** При использовании операций нижнего слоя не нужно знать, как они реализованы, нужно лишь понимать, что они делают
- Такие системы **хорошо тестируются**. Отладка начинается с нижнего слоя и проводится послойно. При возникновении ошибки мы можем быть уверены, что она находится в тестируемом слое
- Многослойные системы **хорошо модифицируются.** При необходимости можно заменить лишь один слой, не трогая остальные

Недостатки:

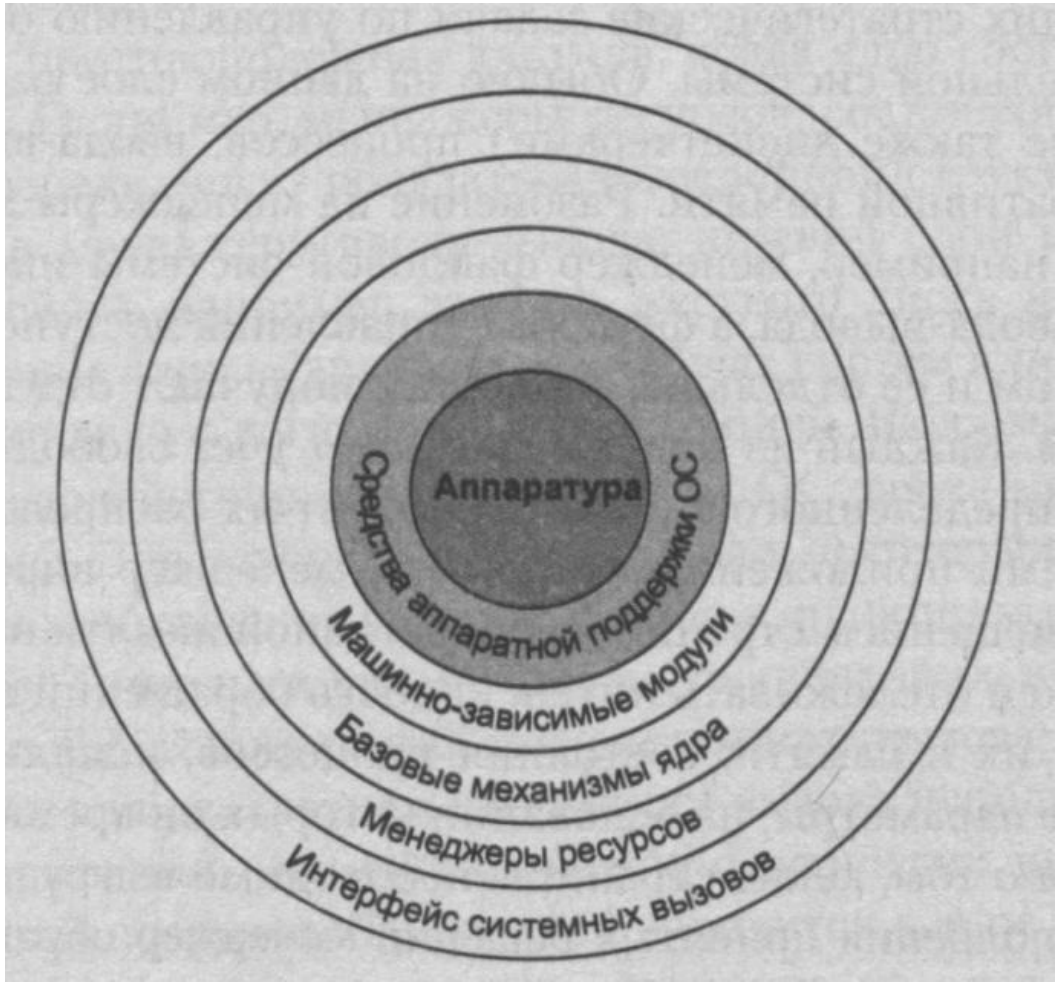
- Многослойные системы **сложны для разработки**: тяжело правильно определить порядок слоев и что к какому слою относится
- Многослойные системы **менее эффективны**, **чем монолитные.** Так, например, для выполнения операций ввода-вывода программе пользователя придется последовательно проходить все слои от верхнего до нижнего

Многослойная структура ядра ОС

Поскольку ядро представляет собой сложный многофункциональный комплекс, то многослойный подход обычно распространяется и на структуру ядра

Ядро может состоять из следующих слоев:

1) Средства аппаратной поддержки ОС
2) Машинно-зависимые модули
3) Базовые механизмы ядра
4) Менеджеры ресурсов
5) Интерфейс системных вызовов



Многослойная структура ядра ОС

Средства аппаратной поддержки операционной системы. До сих пор об операционной системе говорилось как о комплексе программ, хотя часть функций операционной системы может выполняться и аппаратными средствами. В этом случае речь идет не о всей аппаратуре компьютера, а только об аппаратуре, непосредственно участвующей в организации вычислительного процесса. Например, средства поддержки привилегированного режима, система прерываний, средства защиты памяти и т.п.

Машинно-зависимые компоненты операционной системы. Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. За счет этого слоя происходит полное экранирование вышележащих слоев ядра от особенностей аппаратной реализации, тем самым позволяя разрабатывать вышележащие слои на основе машинно-независимых модулей

Многослойная структура ядра ОС

Базовые механизмы ядра. Этот слой выполняет наиболее примитивные операции ядра. 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

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

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

Многослойная структура ядра ОС

Приведенное разбиение ядра ОС на слои является **достаточно условным**. В реальной системе количество слоев и распределение функций между ними может быть иным

В некоторых системах соседние слои могут сливаться в один

Возможна и противоположная картина, когда ядро состоит из большего количества слоев

Способ взаимодействия слоев в реальной ОС также может отклоняться от описанной схемы. Для ускорения работы ядра в некоторых случаях происходит непосредственное обращение с верхнего слоя к функциям нижних слоев, минуя промежуточные

Выбор количества слоев ядра является ответственным и сложным делом: увеличение числа слоев ведет к некоторому замедлению работы ядра за счет дополнительных накладных расходов на межслойное взаимодействие, а уменьшение числа слоев ухудшает расширяемость и логичность системы

Аппаратная зависимость и переносимость ОС

Многие операционные системы успешно работают на различных аппаратных платформах без существенных изменений в своем составе

Во многом это объясняется тем, что, несмотря на различия в деталях, средства аппаратной поддержки ОС большинства компьютеров приобрели сегодня много типовых черт, а именно эти средства в первую очередь влияют на работу компонентов операционной системы

В результате в ОС можно выделить достаточно компактный слой машинно-зависимых компонентов ядра и сделать остальные слои ОС общими для разных аппаратных платформ

Аппаратная зависимость и переносимость ОС

Практически все современные аппаратные платформы имеют некоторый типичный набор средств аппаратной поддержки ОС, в который входят следующие компоненты:

- Средства поддержки привилегированного режима
- Средства трансляции адресов
- Средства переключения процессов
- Система прерываний
- Системный таймер
- Средства защиты областей памяти

Средства поддержки привилегированного режима

Средства поддержки привилегированного режима обычно основаны на системном регистре процессора, часто называемом «словом состояния» машины или процессора

Этот регистр содержит некоторые признаки, определяющие режимы работы процессора, в том числе признак текущего режима привилегий

Смена режима привилегий выполняется за счет изменения слова состояния машины в результате прерывания или выполнения привилегированной команды

Число градаций привилегированности может быть разным у разных типов процессоров, наиболее часто используются два уровня (ядро-пользователь)

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

Средства трансляции адресов

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

Таблицы, предназначенные при 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 применяют данные 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

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 поддерживаются указателями процессора

Переключение контекста выполняется по определенным командам процессора, например, по команде перехода на новую задачу

Такая команда вызывает автоматическую загрузку данных из сохраненного контекста в регистры процессора, после чего процесс продолжается с прерванного ранее места

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

Прерывания играют важнейшую роль в работе любой операционной системы, являясь ее «движущей силой»

Действительно, большая часть действий ОС инициируется прерываниями различного типа. Даже системные вызовы от приложений выполняются на многих аппаратных платформах, с помощью специальной инструкции прерывания (int), вызывающей переход к выполнению соответствующих процедур ядра

Системный таймер

Системный таймер, часто реализуемый в виде быстродействующего регистра-счетчика, необходим операционной системе для выдержки интервалов времени. Для этого 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. Например, в системе разделения времени при обработке очередного прерывания от таймера планировщик процессов может принудительно передать управление другому процессу, если данный процесс исчерпал выделенный ему квант времени

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)

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

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

Машинно-зависимые компоненты ОС

Стоит подумать над следующим фактом: одна и та же операционная система не может без каких-либо изменений устанавливаться на компьютерах, отличающихся типом процессора и/или способом организации всей аппаратуры

В модулях ядра ОС не могут не отразиться такие особенности аппаратной платформы, как количество типов прерываний и формат таблицы ссылок на процедуры обработки прерываний, состав регистров общего назначения и системных регистров, состояние которых нужно сохранять в контексте процесса, особенности подключения внешних устройств и многие другие

Однако опыт разработки операционных систем показывает: ядро можно спроектировать таким образом, чтобы только часть модулей были **машинно-зависимыми**, а остальные не зависели от особенностей аппаратной платформы

Машинно-зависимые компоненты ОС

Собственно, **Машинно-зависимые компоненты операционной системы** – это программные модули, в которых отражается специфика аппаратной платформы компьютера. За счет этого слоя операционной системы происходит полное экранирование вышележащих слоев ядра от особенностей аппаратной реализации, тем самым позволяя разрабатывать вышележащие слои на основе машинно-независимых модулей

Для уменьшения количества машинно-зависимых модулей производители операционных систем ограничивают распространение своей операционной системы только несколькими типами процессоров и созданными на их базе аппаратными платформами

Примеры реализации для x86-компьютеров

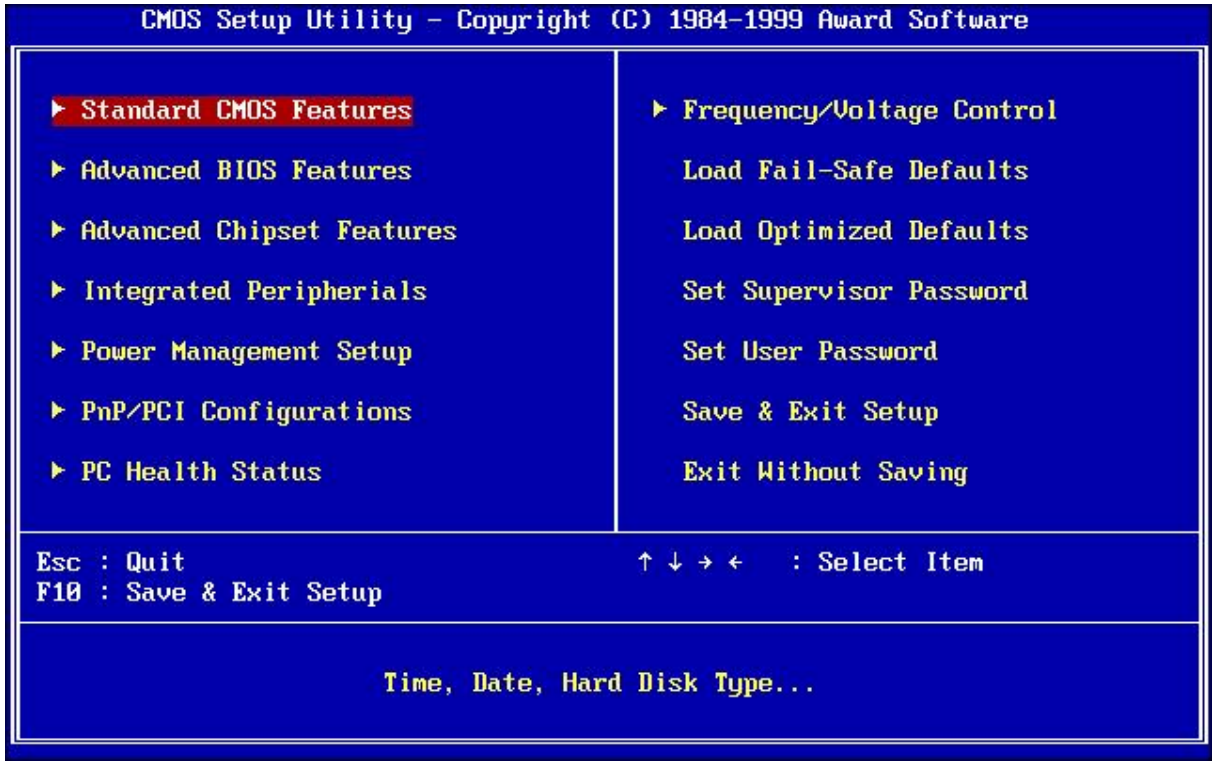
Для компьютеров на основе x86 процессоров разработка экранирующего машинно-зависимого слоя ОС несколько упрощается за счет встроенной в постоянную память компьютера базовой системы ввода-вывода – **BIOS или его аналога UEFI**

BIOS/UEFI содержит драйверы для всех устройств, входящих в базовую конфигурацию компьютера: жестких и гибких дисков, клавиатуры, дисплея и т. д.

Современные версии Windows используют **HAL** (**Hardware Abstraction Layer**) – слой абстрагирования, реализованный в программном обеспечении, находящийся между физическим уровнем аппаратного обеспечения и программным обеспечением, запускаемом на этом компьютере

BIOS (Basic Input-Output system)

BIOS (базовая система ввода-вывода) – это программа низкого уровня, хранящаяся на чипе EEPROM материнской платы вашего компьютера. BIOS загружается при включении компьютера и отвечает за пробуждение его аппаратных компонентов, убеждается в том, что они правильно работают, а потом запускает программу-загрузчик, запускающую операционную систему



Недостатки BIOS

Он может загружаться только с жёстких дисков объёмом не более 2,1 Тб. Это ограничение BIOS MBR

BIOS должен работать в 16-битном режиме процессора и ему доступен всего 1 Мб памяти

У него проблемы с одновременной инициализацией нескольких устройств, что ведёт к замедлению процесса загрузки, во время которого инициализируются все аппаратные интерфейсы и устройства

Именно поэтому в 2007 году произошло то, что произошло, а именно была согласована новая спецификация **Unified Extensible Firmware Interface** (UEFI), унифицированный интерфейс расширяемой прошивки

UEFI (Unified Extensible Firmware Interface)

UEFI – это небольшая операционная система, работающая над прошивкой PC, поэтому она способна на гораздо большее, чем BIOS. Её можно хранить в флэш-памяти на материнской плате или загружать с жёсткого диска или с сети

Он хранит все данные об инициализации и запуске в файле .efi, а не в прошивке

Этот файл .efi хранится в специальном системном разделе EFI (ESP - EFI System Partition) на жестком диске

Этот раздел также содержит загрузчик

Преимущества UEFI

UEFI поддерживает размеры дисков до 9 зеттабайт, тогда как BIOS поддерживает только 2,2 терабайта

UEFI обеспечивает более быструю загрузку

UEFI поддерживает дискретные драйверы, в то время как BIOS поддерживает диски, хранящиеся в его ПЗУ, поэтому обновление прошивки BIOS немного затруднено

UEFI обеспечивает безопасную загрузку, которая предотвращает загрузку компьютера из неавторизированных/неподписанных приложений. Это помогает предотвратить внедрения руткитов, но при этом затрудняет двойную загрузку, так как рассматривает другие ОС как неподписанные приложения

UEFI работает в 32-битном или 64-битном режиме, тогда как BIOS работает в 16-битном режиме. Таким образом, UEFI может предоставить графический интерфейс (то есть управление с помощью мыши), в отличие от BIOS, который поддерживает управление только с помощью клавиатуры

Как происходит загрузка в UEFI?

С GPT-раздела с идентификатором EF00 и файловой системой FAT32 по умолчанию грузится и запускается файл **\efi\boot\boot[название архитектуры].efi**, например **\efi\boot\bootx64.efi**

Т.е. чтобы, например, создать загрузочную флешку с Windows, достаточно просто разметить флешку в GPT, создать на ней FAT32-раздел и просто-напросто скопировать все файлы с ISO-образа. Boot-секторов больше нет, забудьте про них

Загрузка в UEFI происходит гораздо быстрее.

Hardware Abstraction Layer

HAL предназначен для скрытия различий в аппаратном обеспечении от основной части ядра операционной системы, таким образом, чтобы большая часть кода, работающая в режиме ядра, не нуждалась в изменении при её запуске на системах с различным аппаратным обеспечением

В операционных системах семейства Windows NT HAL является неотъемлемой частью кода, исполняемого в режиме ядра, находится в отдельном загрузочном модуле, загружаемом совместно с ядром. Это обеспечивает возможность использования одного и того же загрузочного модуля собственно ядра ОС Windows NT на ряде систем с различными архитектурами шин ввода-вывода, управления прерываниями и таймерами

Hardware Abstraction Layer

Подобные решения можно найти и для Linux, например, **Adeos**. **Adeos** (Адаптивная доменная среда для операционных систем) – это наноядерный уровень аппаратной абстракции (HAL) или гипервизор, который работает между компьютерным оборудованием и операционной системой (ОС), которая на нем работает

Он отличается от других наноядер тем, что представляет собой не только низкоуровневый слой внешнего ядра

Вместо этого предполагается совместная работа нескольких ядер, что делает его похожим на технологии полной виртуализации

Утилиты

Утилиты – это системное программное обеспечение, которое помогает поддерживать правильное и бесперебойное функционирование компьютерной системы. Они помогают операционной системе управлять, организовывать, поддерживать и оптимизировать функционирование компьютерной системы

Утилиты выполняют **определенные** задачи, такие как обнаружение, установка и удаление вирусов, резервное копирование данных, удаление ненужных файлов и т. д.

Утилиты предоставляют доступ к возможностям (параметрам, настройкам, установкам), недоступным без их применения, либо делают процесс изменения некоторых параметров проще (автоматизируют его)

Утилиты

Утилиты могут входить в состав операционных систем, идти в комплекте со специализированным оборудованием или распространяться отдельно

Интегрированные пакеты утилит – набор нескольких программных продуктов, объединенных в единый удобный инструмент

Типы утилит: Антивирусы, Системы управления файлами, Инструменты сжатия, Инструменты управления дисками, Инструменты очистки диска, Дефрагментация диска, Утилита резервного копирования

Системные обрабатывающие программы

Системные обрабатывающие программы – текстовые и графические редакторы, компиляторы, компоновщики, отладчики

Обрабатывающие системные программы отличаются от управляющих программ как по своим функциям, так и по способу их инициирования (запуска). Основные функции обрабатывающих программ:

1. Перенос информации
2. Преобразование информации

В зависимости от того, какая из этих двух функций является основной, обрабатывающие системные программы делятся на **утилиты** и **лингвистические процессоры**

Системные обрабатывающие программы

Лингвистические процессоры делятся на **трансляторы** и **интерпретаторы**

В результате работы транслятора алгоритм, записанный на языке программирования, преобразуется в алгоритм, записанный на машинном языке

Трансляторы делятся на компиляторы и ассемблеры

Интерпретатор в отличие от транслятора не выдаёт машинную программу целиком. Выполнив перевод очередного оператора исходной программы в соответствующую совокупность машинных команд, интерпретатор обеспечивает их выполнение

Системные обрабатывающие программы

Современная реализация командного процессора CMD кроме классического применения, позволяет также использовать возможности:

- Инструментария управления Windows (WMI –Windows Management Instrumentation)
- Сценарии Windows Script Host (WSH)
- Оболочки (shell) операционных систем Linux

Командный файл – это обычный текстовый файл с набором команд, которые последовательно выполняются командным процессором CMD Windows (cmd.exe). Командный процессор последовательно считывает и выполняет команды, которые представляют собой своеобразную программу, реализующую определенный алгоритм

Естественно программирование в командной строке подчиняется определенным правилам и командные файлы должны им полностью соответствовать

Библиотеки

Библиотеки процедур и функций различного назначения включены в категорию вспомогательных модулей операционной системы. К ним можно отнести библиотеки математических функций, библиотеки функций ввода-вывода и т.д.

Библиотеки – это набор функций, которые могут использоваться в различных программах. Библиотеки могут быть **статические** (библиотека привязывается к определенной программе или софт содержит данную библиотеку в своем теле) и **динамическими** (библиотеки грузятся в оперативную память и используются)

Программы дополнительных услуг

В эту категорию входит большое количество разнообразных программ: специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (какие, например, поставляются в составе ОС)

Операционные системы

Лекция 3

Ядро операционной системы