Операционные системы Автор В.А.Серков

Архитектура операционных систем

- Архитектурой системы является распределение функций между ее элементами (модулями) и организация взаимодействия этих элементов.
- Наиболее общим подходом к структуризации операционной системы является разделение всех ее модулей на две группы (ядро и вспомогательные модули).

Ядро и вспомогательные модули операционной системы

- При функциональной декомпозиции ОС модули разделяются на две группы:
  - ядро – модули, выполняющие основные функции ОС;
  - модули, выполняющие вспомогательные функции ОС.

Модули ядра ОС

- Модули ядра ОС выполняют следующие базовые функции ОС:
  - управление процессами
  - управление памятью
  - управление устройствами ввода-вывода
- Ядро обеспечивает решение задачи **организации вычислительного процесса**: переключение контекстов, загрузка/выгрузка страниц, обработка прерываний и т.п.
- Другая задача – поддержка приложений, создание для них **прикладной программной среды**. Приложения обращаются к ядру с запросами (**системными вызовами**) для выполнения базовых операций (открытие и чтение файла, вывод информации на дисплей и т.п.)
- Функции выполняемые ядром ОС требуют высокой скорости выполнения и для этого размещаются постоянно в оперативной памяти (**резидентные модули**).

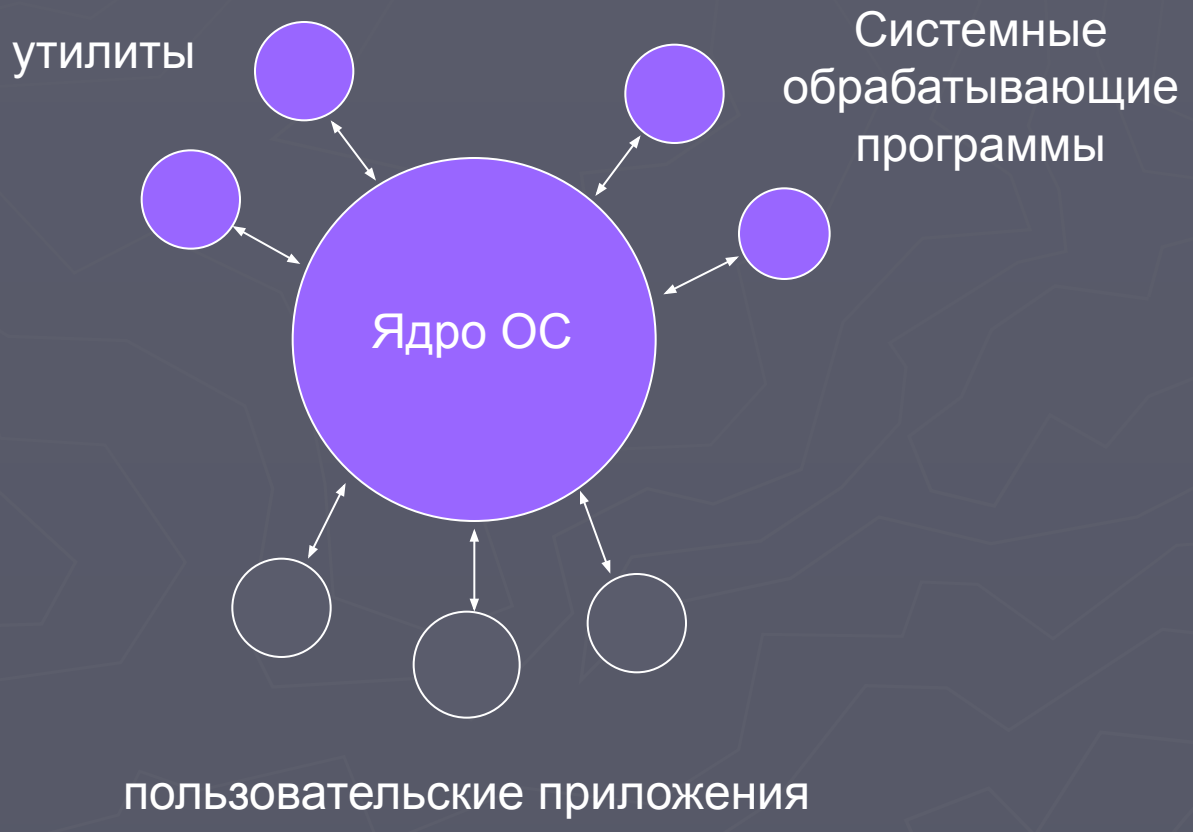
Вспомогательные модули операционной системы

- Вспомогательные модули выполняют полезные, но менее обязательные функции. Например:
  - архивирование информации;
  - дефрагментация данных на диске;
  - поиск необходимого файла и т.п.
- Вспомогательные модули часто оформляются как обычные приложения и провести границу между ними и обычными приложениями сложно.
- Деление на основные и вспомогательные модули ОС условно. Некоторые программы переходят из разряда вспомогательных модулей в основные и наоборот.

Вспомогательные модули операционной системы

- Вспомогательные модули ОС условно разделяются на следующие группы:
  - *Утилиты* – приложения, решающие отельные задачи управления и сопровождения ОС
  - *Системные обрабатывающие программы* – текстовые и графические редакторы, компиляторы, компоновщики и т.п.
  - *Программы предоставления пользователю дополнительных услуг* – специальный вариант пользовательского интерфейса, калькулятор, игры и т.п.
  - *Библиотеки процедур* – модули различного назначения, упрощающие разработку приложений.
- Вспомогательные модули обращаются к функциям ядра ОС посредством системных вызовов.

Ядро и вспомогательные модули операционной системы



Привилегированный режим процессора

- Для надежного управления работой приложений ядро ОС должно обладать некоторыми привилегиями по отношению к остальным приложениям.
- Обеспечивается привилегированный режим специальными средствами аппаратной поддержкой. Процессор компьютера поддерживает как минимум два режима работы – **пользовательский** (user mode) и **привилегированный** (kernel mode).
- Приложения в пользовательском режиме не могут выполнять некоторые критичные команды (переключение процессора с задачи на задачу, доступ к механизму выделения и защиты областей памяти и т. п.).

Привилегированный режим работы

- Между числом привилегий, поддерживаемых аппаратурой и операционной системой нет однозначного соответствия:
  - процессор Intel поддерживает 4 режима работы процессора – операционные системы Windows используют два из них.
- Для реализации привилегированного режима достаточно поддержки двух режимов работы
- Повышение устойчивости ОС, обеспечивающееся использованием работы в привилегированном режиме, достигается за счет некоторого замедления, вызванного необходимостью переключения работы ядра.
- Архитектура ОС, основанная на разделении привилегированного режима для ядра и пользовательского режима для приложений – стала классической.

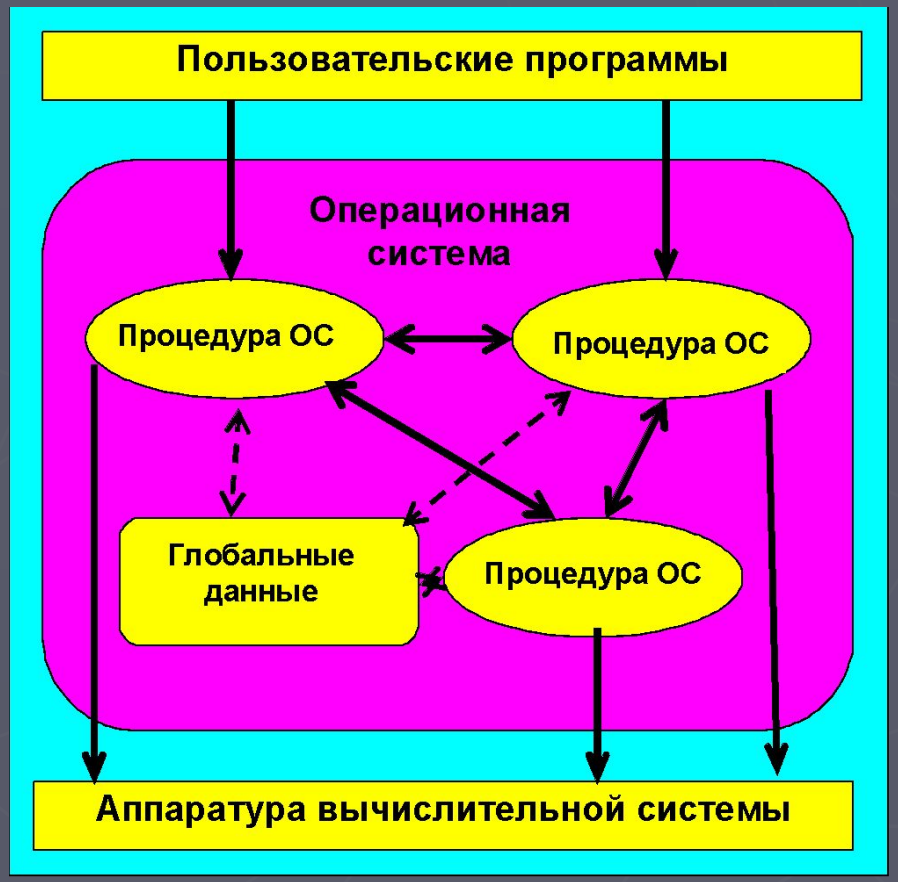
Типы архитектур ОС

Различают всего три базовых типа архитектуры операционных систем:

- монолитная архитектура;
- многоуровневая архитектура;
- архитектура типа клиент-сервер на основе микроядра.

Монолитная архитектура

При монолитной архитектуре операционная система не имеет какой-либо явно выраженной внутренней структуры. Это просто набор процедур, использующих общие глобальные данные, и вызываемые друг другом или пользователем.



Недостатки монолитной архитектуры

1. Затруднено развитие операционной системы или ее перенос на другую аппаратную платформу.
2. Часто возникает необходимость коррекции многих компонентов системы при реальной потребности изменения только одного из них.
3. Серьезные трудности при сопровождении и технической поддержке операционной системы.

Требования к архитектуре операционной системы

Расширяемость операционной системы

Под расширяемостью понимают возможность добавлять в операционную систему новую функциональность без изменения уже существующих компонентов системы.

Необходимость расширения функциональности операционной системы связана с быстрым развитием аппаратных средств и технологий программирования.

Наличие в составе операционной системы монолитного ядра, реализующего базовую функциональность всей операционной системы, существенно затрудняет введение в систему новой функциональности.

Переносимость операционной системы

- Под переносимостью операционной системы понимается способность использования ОС на различных аппаратных платформах с минимальными изменениями в ее структуре. Для уменьшения числа машинно-зависимых модулей разработчики ОС ограничивают универсальность машинно-независимых модулей. Например, Windows разработана для нескольких типов процессоров и для многопроцессорных систем используются собственные модули.
- Для обеспечения переносимости следуют следующим правилам:
  - Большая часть кода написана на языке, трансляторы которого существуют для всех планируемых платформ;
  - Объем машино-зависимых частей кода должен быть минимизирован;
  - Аппаратно-зависимый код должен быть изолирован в нескольких модулях
- В идеале машино-зависимые модули ядра полностью экранируют остальную часть ОС от конкретных деталей аппаратной платформы (кэши, контроллеры прерываний и т.п.).

Аппаратная зависимость ОС

- Операционная система в процессе работы взаимодействует с аппаратными средствами компьютера:
  - Средства поддержки привилегированного режима
  - Средства трансляции адресов
  - Средства переключения процессов
  - Защита областей памяти
  - Система прерываний
  - Системный таймер
- Это делает ОС привязанной к определенной аппаратной платформе

Совместимость операционной системы

Под совместимостью понимается способность операционной системы исполнять прикладные программы, ориентированные на другие операционные системы, или на более ранние версии той же самой операционной системы.

Различают совместимость на уровне исходных кодов, и двоичную совместимость.

Совместимость на уровне исходных кодов – это способность операционной системы исполнять программы, первоначально ориентированные на другую операционную систему, после перекомпиляции этих программ.

Двоичная совместимость – это способность операционной системы загружать исполняемые файлы, первоначально предназначенные для другой операционной системы, и нормально исполнять представленные в них программы.

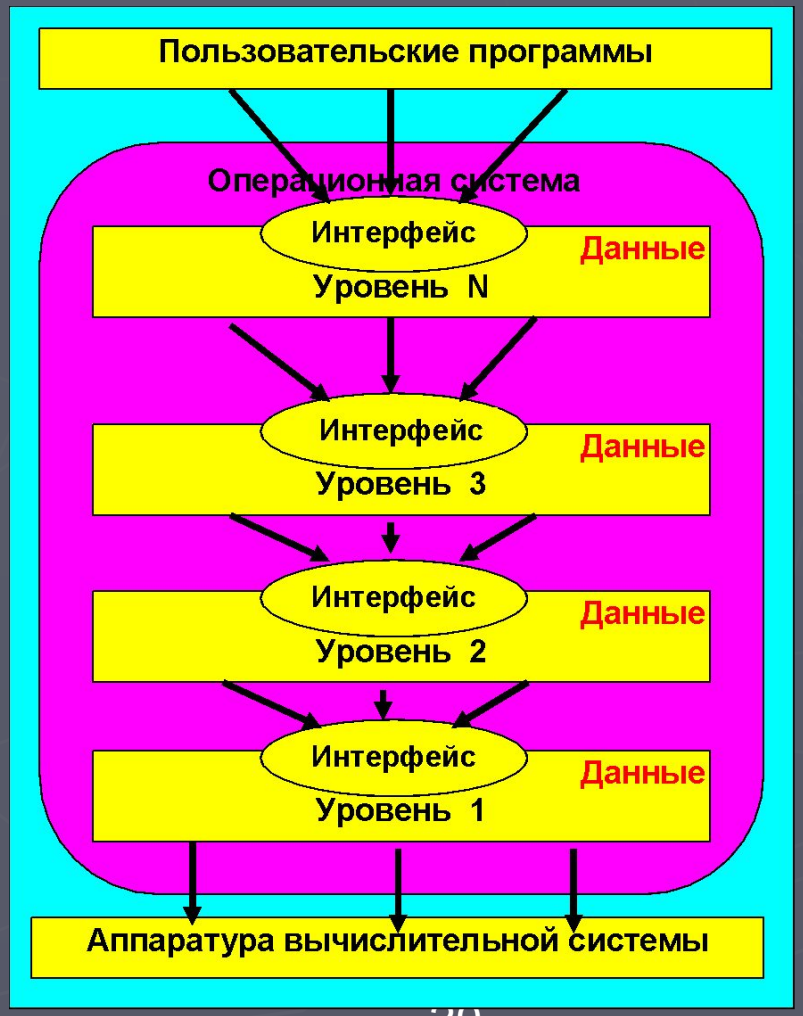
Совместимость операционной системы

Совместимость с какой-нибудь операционной системой может закладываться в новую операционную систему еще на этапе ее проектирования. Однако на практике задача обеспечения совместимости чаще формулируется следующим образом:

Имеются две операционных системы, система А и система В. Обе операционных системы давно и успешно работают на некоторой аппаратной платформе. Необходимо в систему А ввести исполнительную среду системы В, чтобы прикладные программы системы В могли бы исполняться также и в системе А.

Многоуровневая архитектура

1. Полная функциональность операционной системы разделяется на уровни, например, уровень управления аппаратурой, уровень управления памятью, уровень файловой системы, уровень управления процессами и т.п.
2. Для каждого уровня определяются интерфейс взаимодействия, т.е. некоторый набор правил, согласно которым следует обращаться за услугами данного уровня.
3. Взаимодействие уровней строится таким образом, что каждый уровень может обращаться за услугами только к соседнему нижележащему уровню через его интерфейс.
4. Внутренние структуры данных каждого уровня не доступны другим уровням, а реализации процедур уровня скрыты и не зависят от реализаций процедур внутри других уровней.



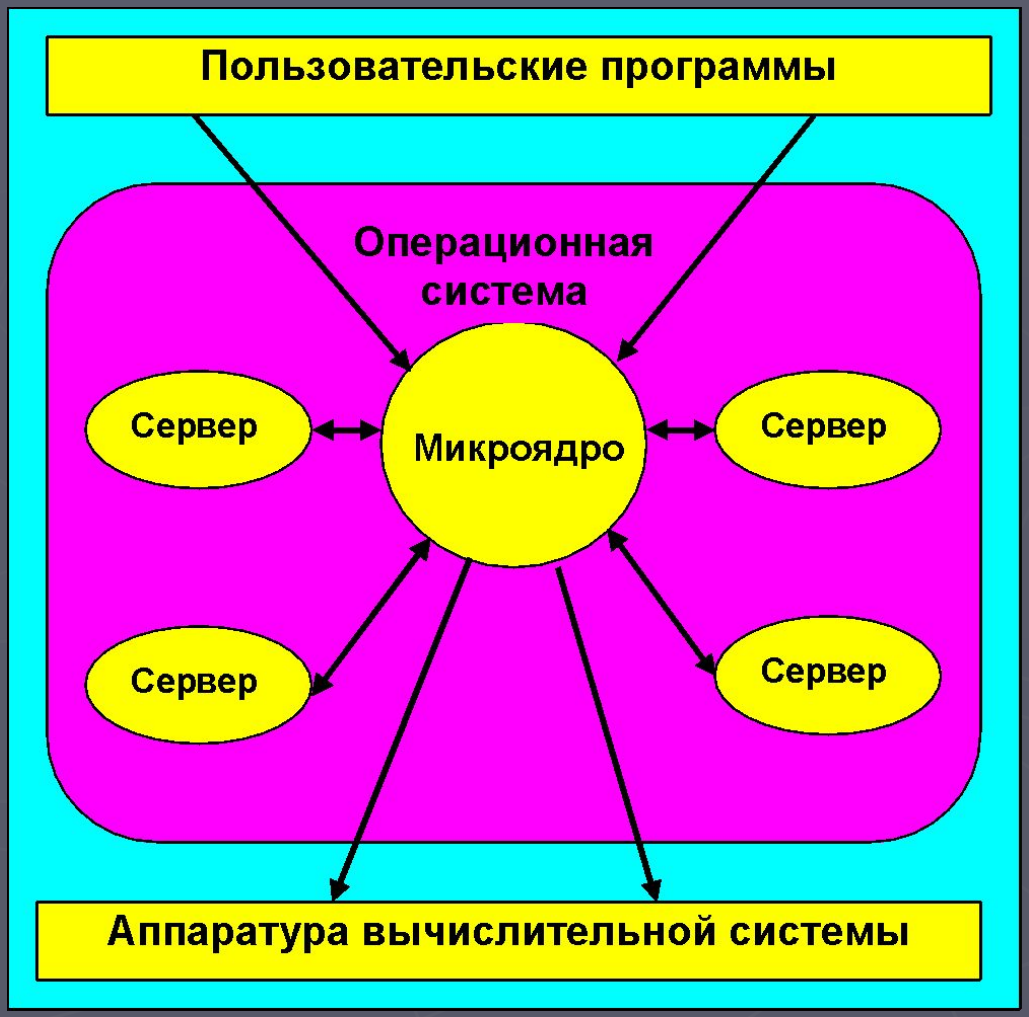
Детализация структуры ядра

- Ядро, являясь структурным элементом ОС, может быть логически разложен на ряд слоев:
  - Средства аппаратной поддержки ОС
  - Машинно-зависимые компоненты ОС (включает модули, отражающие специфику аппаратной платформы компьютера)
  - Базовые механизмы ядра (включает наиболее примитивные операции ядра – переключение контекстов процессов, диспетчеризация прерываний), модули выполняют решения принятые на более высоких уровнях
  - Менеджеры ресурсов (реализует задачи стратегического управления), включает менеджеры – диспетчеры процессов, ввода-вывода и т.п.
  - Интерфейсы системных вызовов (включает модули взаимодействия с приложениями и системными утилитами, функции API.

Микроядерная архитектура

- Концепция микроядерной архитектуры заключается в выделении в качестве работающего в привилегированном режиме части ОС, ответственном за небольшой набор системных функций (управление процессами, обработка прерываний, управление виртуальной памятью, пересылка сообщений). Данная часть ОС называется **микроядром**.
- Все остальные высокоуровневые функции ядра разрабатываются в виде приложений, работающих в пользовательском режиме – **серверы ОС**.
- Взаимодействие между обычными приложениями и серверами ОС осуществляется через механизм обращений. **Клиентское приложение** отправляет запрос к серверу ОС через микроядро ОС. Такой механизм обеспечивает защиту работы приложений.

Микроядерная архитектура



Основные преимущества микроядерной архитектуры

- переносимость операционной системы, т.к. серверы, работающие в пользовательском режиме, аппаратно независимы;
- расширяемость операционной системы, т.к. новая функциональность может быть легко введена за счет введения нового сервера, и это никак не затронет существующую функциональность;
- гибкость операционной системы, т.к. пользователь может запустить только те сервисы, которые ему действительно нужны, и не расходовать ресурсы системы на поддержку невостребованной функциональности, при этом пользователь может изменять набор запущенных серверов без перезапуска системы.