Урок 1. Архитектура операционной системы

Наиболее общим подходом к структуризации ОС является разделение всех ее модулей на 2 группы:

1. Ядро, содержащее модули, которые выполняют основные функции ОС.
2. Модули, выполняющие вспомогательные функции ОС.

Модули ядра выполняют такие базовые функции ОС как управление процессами, памятью, устройствами ввода-вывода и т.д. В состав ядра входят функции, решающие внутрисистемные задачи организации вычислительного процесса, такие как переключение контекстов, загрузка-выгрузка страниц, обработка прерываний.

Другой класс функций ядра служит для поддержки приложений, создавая для них так называемую прикладную программную среду.

Функции, выполняемые модулем ядра, являются наиболее часто используемыми функциями ОС. Поэтому скорость их выполнения определяет производительность всей систем в целом.

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

К вспомогательным модулям ОС относятся программы архивирования, программы дефрагментации диска и т.д.

Вспомогательные модули разделяются на группы:

1. Утилиты (архивирование, дефрагментация…)
2. Системные обрабатывающие программы (компиляторы, компоновщики, Отладчики, текстовые и графические редакторы)
3. Программы предоставления пользователю дополнительных услуг (калькулятор, игры)
4. Библиотеки процедур, упрощающие разработку приложений.

Для выполнения своих задач все вспомогательные модули ОС обращаются к функциям ядра посредством системных вызовов, как и обычные приложения.

Для надежного управления ходом выполнения приложений ОС должна иметь по отношению к приложениям определенные привилегии. Кроме того ОС должна обладать исключительными полномочиями для того, чтобы распределить приложениям ресурсы компьютера в мультипрограммном режиме.

Обеспечить привилегии ОС невозможно без специальных средств аппаратной поддержки. Аппаратура компьютера должна поддерживать как минимум 2 режима работы: пользовательский и привилегированный, который еще называют режимом ядра или режимом супервизора.

Поскольку все основные функции ОС выполняет ядро, то именно ядро работает в привилегированном режиме.

Приложение ставится в подчиненное положение за счет запрета выполнения в пользовательском режиме некоторых критичных команд, связанных с переключением процессора с задачи на задачу, управлением устройств ввода/вывода, доступом к механизмам распределения и защиты памяти.

Выполнение некоторых инструкций в пользовательском режиме запрещается безусловно. Другие инструкции запрещается выполнять только при определенных условиях. Важно, что условие разрешения выполнения критичных инструкций находится под полным контролем ОС,

и этот контроль обеспечивается за счет набора инструкций, безусловно запрещенных для пользовательского режима.

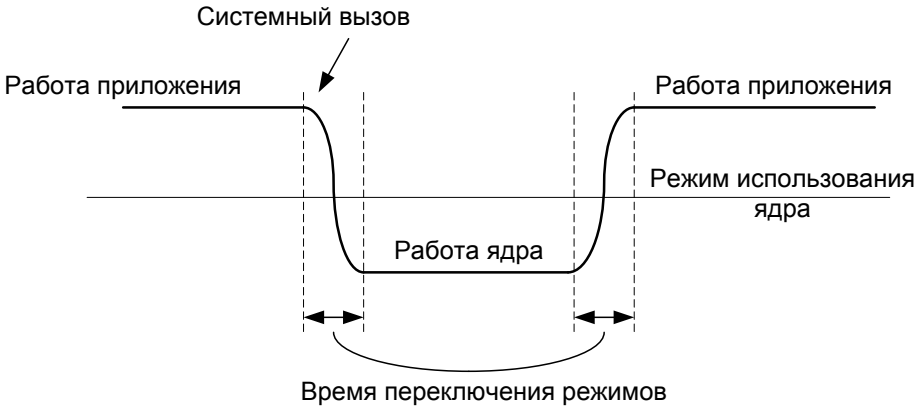
Полный контроль ОС над доступом к памяти достигается именно за счет того, что инструкции конфигурирования механизмов защиты памяти разрешается выполнять только в режиме ядра.

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

На базе 4х уровней, обеспечиваемых процессором компании Intel, ОС OS/2 строит 3х уровневую систему привилегий, а ОС Windows NT и UNIX имеют 2х уровневую систему защиты.

Даже на основе 2-х режимов работы привилегий процессора, ОС может построить сколь угодно развитую систему защиты.

Повышение устойчивости ОС обеспечивается переходом ядра в привилегированный режим работы, который достигается за счет некоторого замедления выполнения системных вызовов.



Многослойная структура ядра (монолитное ядро)

Ядро может состоять из следующих слоев:

1. Средства аппаратной поддержки ОС
2. Машинно-зависимые компоненты ОС. Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале этот слой полностью экранирует вышележащие слои ядра от особенностей аппаратуры, что позволяет разрабатывать их на основе машинно-независимых модулей.
3. Базовые механизмы ядра. Этот слой выполняет наиболее примитивные операции ядра, такие как программные переключения контекстов процессов, диспетчеризация прерываний, перемещение страниц. Модули данного слоя являются исполнительными механизмами для модулей верхних слоев.
4. Менеджеры ресурсов. Этот слой состоит из функциональных модулей, реализующих стратегические задачи по управлению основными ресурсами ВС. Здесь работают диспетчеры процессов, оперативной памяти, ввода/вывода, файловой системы. Внутри этого слоя существуют тесные взаимосвязи.
5. Интерфейс системных вызовов. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя

прикладной программный интерфейс ОС. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме, без их физического расположения.

Нет четкой границы между программной и аппаратной реализацией функций ОС. Но практически все современные аппаратные платформы имеют некоторый типичный набор средств аппаратной поддержки ОС в который входят следующие компоненты:

1. Средства поддержки привилегированного режима. Они основаны на системном регистре процессора, который называется **словом состояния**. Этот регистр содержит признаки, определяющие режимы работы процессора.
2. Средства трансляции адресов. Выполняют преобразование виртуальных адресов в физические адреса памяти. Аппаратура процессора содержит указатели на области, где расположены таблицы трансляции адресов.
3. Средства переключения процессов – предназначены для быстрого сохранения контекста приостанавливаемого процесса и восстановления контекста процесса, который становиться активным. Переключение контекста производится по специальным командам процессора.
4. Система прерываний. Позволяет реагировать на внешние события, синхронизировать выполнение процессов и устройств ввода/вывода.
5. Системный таймер, который реализуется в виде быстродействующего регистра-счетчика и используется ОС для определения интервалов времени.
6. Средства защиты областей памяти. Обеспечивают на аппаратном уровне проверку возможности программного кода работать с данными в ОП.

Ядро ОС должно быть спроектировано таким образом, что только часть модулей будут машинно-зависимыми. Они должны находиться в отдельном программном слое, что упрощает перенос ОС на другую аппаратную платформу.

Объем машинно-зависимых компонентов ОС зависит от того, насколько велики отличия в аппаратных платформах, для которых реализуется ОС.

Если код ОС может быть сравнительно легко перенесен с процессора одного типа на процессор другого типа и с аппаратной платформы одного типа – на другой тип, то такую ОС называют переносимой – portable.

Для того, чтобы обеспечить свойство мобильности ОС разработчики должны следовать следующим правилам:

1. Большая часть кода должна быть написана на языке, трансляторы которого имеются на всех машинах, куда предполагается переносить системы.
2. Объем машинно-зависимых частей кода, который непосредственно взаимодействует с аппаратными средствами, должен быть по возможности минимизирован.
3. Аппаратно-зависимый код должен быть изолирован в нескольких модулях, а не быть распределенным по всей системе. Изоляции подлежат все части ОС, которые отражают специфику как процессора, так и аппаратной платформы в целом.

Микроядерная архитектура

Суть состоит в том, что в привилегированном режиме остается работать только небольшая часть ОС, называемая **микроядром.**

Микроядро защищено от остальных частей ОС и приложений. В состав микроядра обычно входят машинно-зависимые модули, а также модули, выполняющие базовые функции ядра по

управлению процессам обработки прерываний, управлению виртуальной памятью, пересылке сообщений и управлению устройствами ввода/вывода, связанные с загрузкой или чтением регистров устройств.

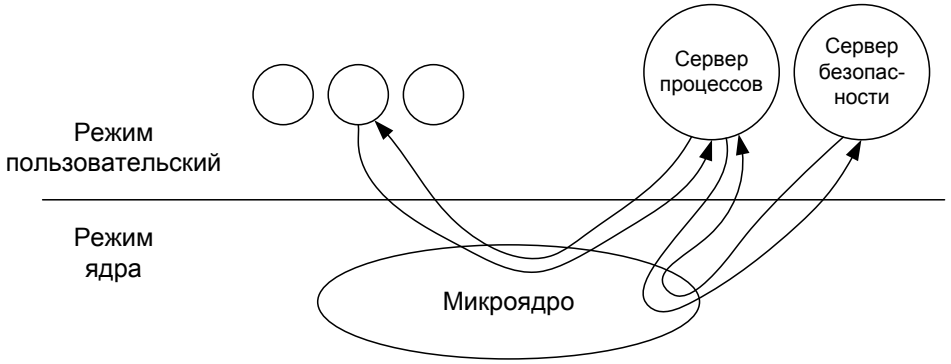
Набор функций микроядра обычно соответствует функциям слоя базовых механизмов обычного ядра. Эти функции чаще всего невозможно выполнить в пользовательском режиме.

Все остальные более высокоуровневые функции ядра формируются в виде приложений, работающих в пространстве пользователя.

Работающие в пользовательском режиме менеджеры ресурсов имеют принципиальное отличие от традиционных утилит и обрабатывающих программ ОС. Утилиты и обрабатывающие программы вызываются в основном пользователями.

Когда в форме приложения формируется часть ОС, основным назначением которого является обслуживание запросов других приложений, то ОС необходимо иметь способ вызова процедур одного процесса из другого.

Менеджеры ресурсов, вынесенные в пользовательский режим, называются серверами ОС, т.е. модулями, основным назначением которых является обеспечение запросов локальных приложений и других модулей ОС.



К достоинствам микроядерной архитектуры относят высокую степень переносимости, т.к. так как весь машинно-независимый код изолирован в ядре + расширяемость ОС. Микроядерная структура при добавлении новой подсистемы требует разработки нового приложения, не затрагивая целостности ядра.

Кроме того можно добавлять не только новые компоненты, но и сократить их число. Использование микроядерной модели повышает надежность ОС. Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти. И если в сервере произойдет сбой, он может быть перезапущен без перезагрузки всей ОС и остальных серверов.

Другим потенциальным источником повышения надежности является уменьшенный объем кода микроядра. Еще одним плюсом можно считать, что модель с микроядром хорошо подходит для поддержки распределенных вычислений, т.к. использует алгоритмы вычислений, аналогичные сетевым.

Самым серьезным минусом микроядерной архитектуры является ее более низкая производительность, т.к. выполнение системного вызова сопровождается 4 переключениями режимов: приложение ⇨ микроядро ⇨ сервер ⇨ микроядро ⇨ приложение.

Совместимость и множественные прикладные среды

Возможность ОС выполнять приложения, написанные для других ОС, называется ***совместимостью***.

Свойство совместимости ОС зависит в первую очередь от архитектуры процессора, на котором работает ОС. Если процессор использует тот же набор команд и тот же диапазон адресов, то двоичная совместимость достигается при соблюдении следующих условий:

1. Вызовы функций API, которые содержат приложения, должны поддерживаться данной ОС.
2. Внутренняя структура исполняемого файла приложения должна соответствовать структуре исполняемых файлов данной ОС.

Если процессоры имеют разную архитектуру, то кроме соблюдения перечисленных условий необходимо организовать эмуляцию двоичного кода.

Эмулятор последовательно выбирает каждую двоичную инструкцию исходного кода, программно дешифрирует ее, чтобы определить, какие действия она задает, имитирует регистры, флаги и выполняет эквивалентную программу в инструкциях текущего процессора.

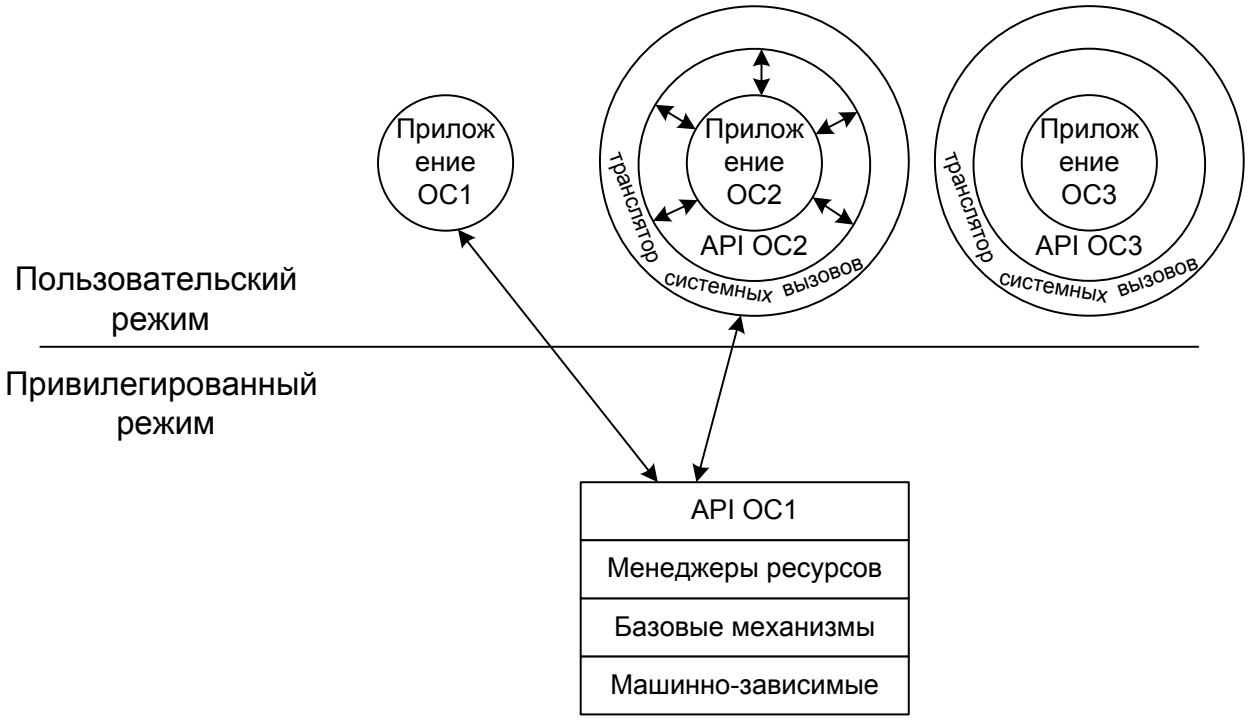
Эмуляция значительно замедляет работу программы.

Для того, чтобы избежать этого недостатка, используются прикладные программные среды. Одной их составляющих, формирующих прикладную программную среду, является набор функций API, который ОС предоставляет своим приложениям.

Для сокращения времени на выполнение чужих программ, прикладные среды имитируют обращение к библиотечным функциям.

Способы реализации прикладных программных сред

Один из более очевидных вариантов реализации множественных прикладных сред основывается на стандартной многоуровневой структуре ОС.

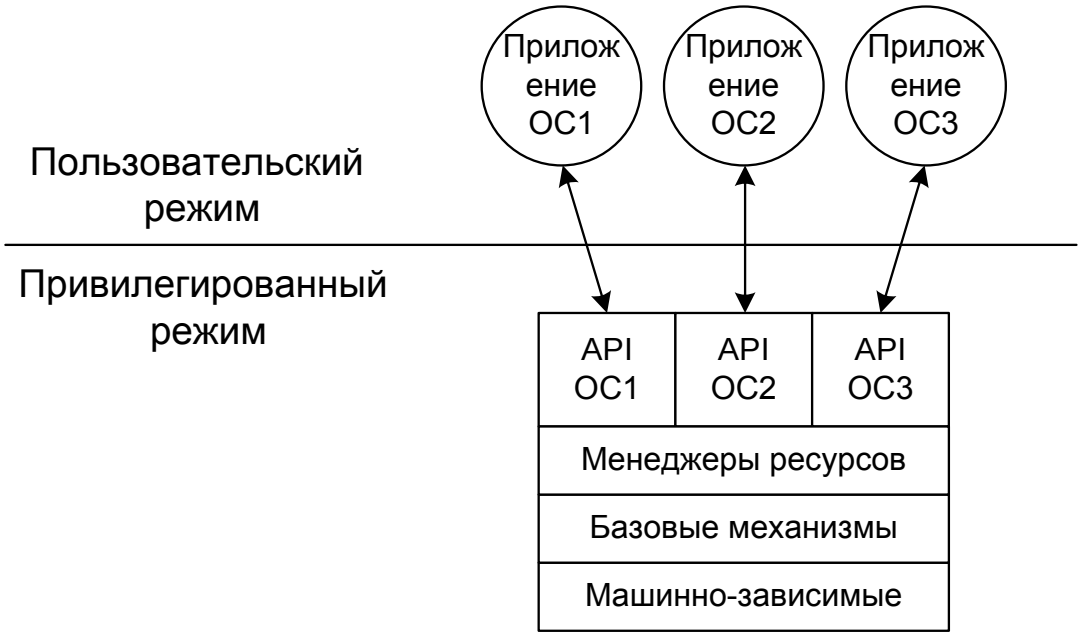


ОС ОС1 кроме своих приложений поддерживает приложения ОС2 и ОС3. Для этого в ее составе имеются специальные приложения, прикладные программные среды, которые

транслируют интерфейсы чужих ОС API ОС2 и API ОС3 в интерфейс своей родной ОС API ОС1.

Другая реализация множественных прикладных сред предполагает наличие в ОС нескольких равноправных прикладных программных интерфейсов.

В пространстве ядра системы размещаются прикладные программные интерфейсы всех ОС.



Функции уровня API обращаются к функциям нижележащего уровня ОС, который должен поддерживать 3 (в данном случае) несовместимые среды.

Функции каждого API реализуются ядром с учетом специфики соответствующей ОС, даже если они имеют аналогичное назначение.

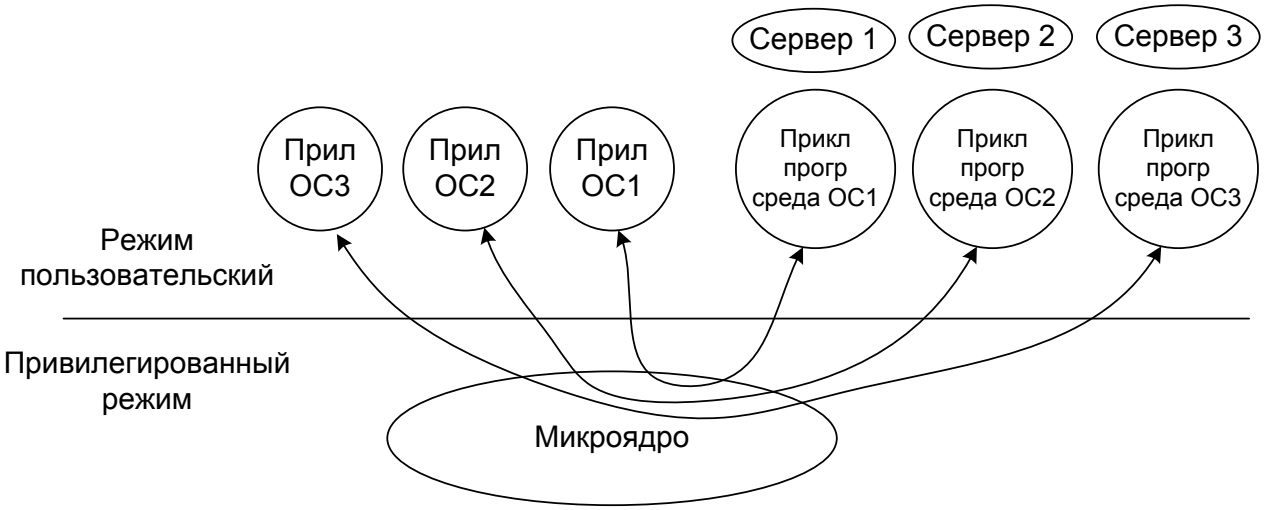
Еще один способ построения множественных прикладных сред основан на микроядерном подходе. При этом важно отделить базовые, общие для всех прикладных сред механизмы ОС от специфических.

В соответствии с микроядерной архитектурой все функции ОС реализуются микроядром и серверами пользовательского режима.

Важно, что каждая прикладная среда оформляется в виде отдельного сервера пользовательского режима и не включает базовых механизмов.

Приложение, используя API, обращается к системным вызовам к соответствующей прикладной среде через микроядро.

Прикладная среда образует запрос, выполняет его, и отсылает приложению результат. В ходе выполнения запроса прикладной среде приходится обращаться к базовым механизмам ОС, реализуемым микроядром и другими серверами ОС.



Такому подходу конструирования множественных прикладных сред присущи все достоинства и недостатки микроядерной архитектуры.

Проблемы проектирования

Чтобы ОС была успешно спроектирована, разработчики должны четкое представление о том, что они хотят.

Для универсальных ОС основными являются следующие 4 положения:

1. Определение абстракций
2. Представление примитивных операций
3. Обеспечение изоляций (одну программу от другой)
4. Управление аппаратурой

Сложность проектирования ОС заключается в следующем:

1. Современные ОС являются громоздкими программами, и при этом все подсистемы должны соответствовать и взаимодействовать между собой.
2. ОС должны управлять параллельно выполняющимися процессами и при этом решать проблемы гонок и тупиков.
3. ОС должна предпринимать меры против вмешательства в ее работу.
4. ОС должны предоставлять пользователям ресурсы для совместного использования, но таким образом, чтобы пользователь, не имеющий прав доступа, не мог пользоваться этой возможностью.
5. ОС должна быть рассчитана на долгое время использования, и проектировщики должны предусмотреть, как изменится аппаратура и приложения с течением времени.
6. У разработчиков ОС на самом деле нет четкого представления о том, как будет использоваться их ОС.
7. От ОС требуется переносимость и поддержка множества устройств ввода/вывода, которые проектируются независимо друг от друга.
8. При разработке ОС необходимо учитывать совместимость с предыдущей версией.

Контрольные вопросы:

1. Что такое операционная система?
2. Функции операционной системы.
3. Что такое API?
4. Структура операционной системы.
5. Микроядерные операционные системы.
6. Монолитные операционные системы.
7. Сервисы, поддерживаемые микроядром.
8. Суть микроядерной архитектуры.
9. Привилегированный и пользовательский режимы.
10. Реализация системных вызовов в микроядерной архитектуре.
11. Достоинства и недостатки микроядерной архитектуры.
12. Модель монолитной структуры ОС.

Список литературы

Операционные системы: Учебник/ Под ред. Э.С. Спиридонова, М.С. Клыкова. Изд. стереотип.- М.: Книжный дом "ЛИБРОКОМ", 2017.- 350 с.

Задание: ответить на контрольные вопросы в письменном виде и оформить отчет.