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 2. Назначение и функции 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 2022

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.

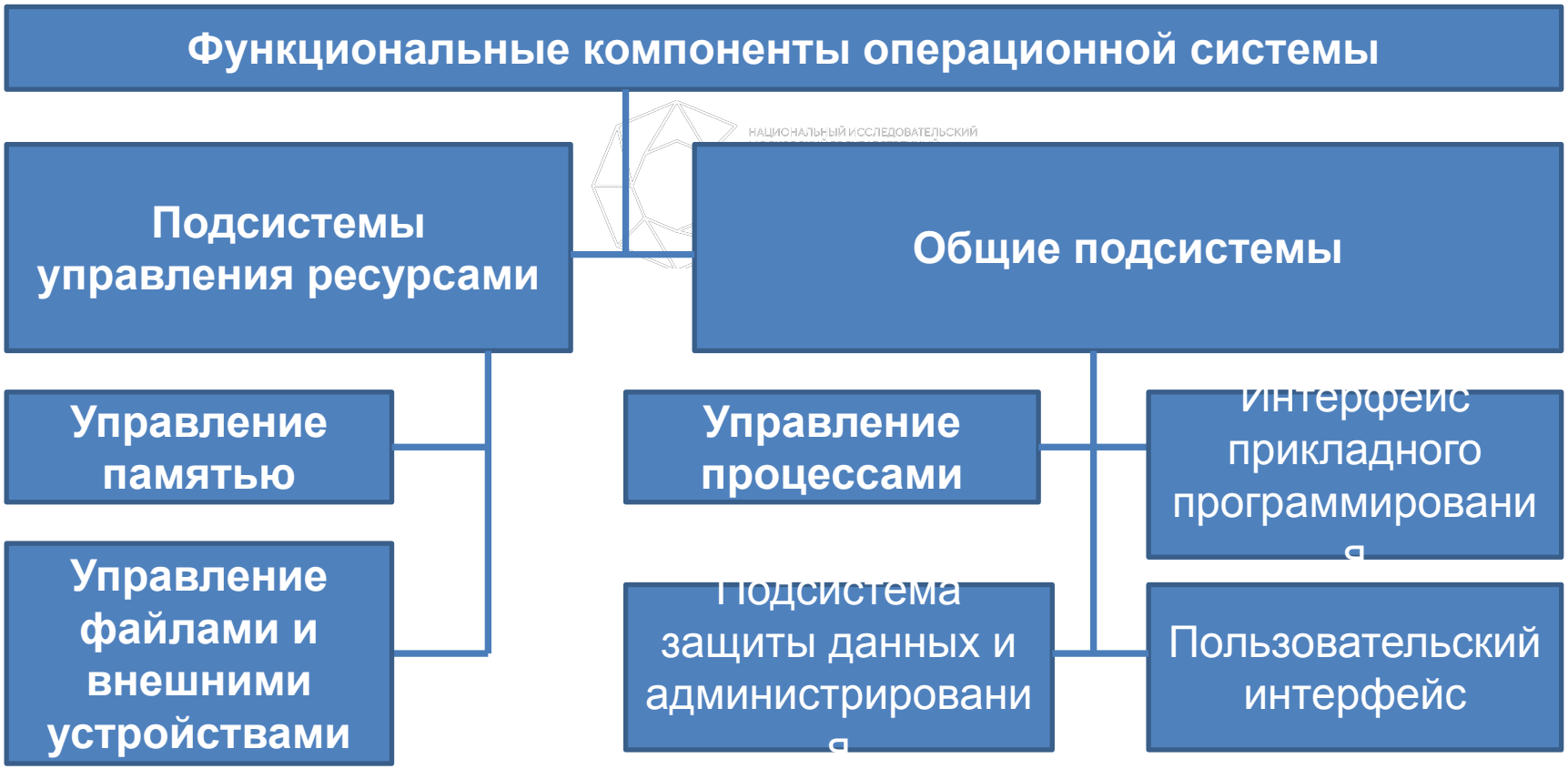
Основные сведения об операционных системах

Операционная система (ОС) – комплекс управляющих и обрабатывающих программ, который,
с одной стороны, выступает как интерфейс между аппаратурой компьютера и пользователем с его задачами,
а с другой – предназначен для наиболее эффективного использования ресурсов вычислительной системы и организации надежных вычислений.

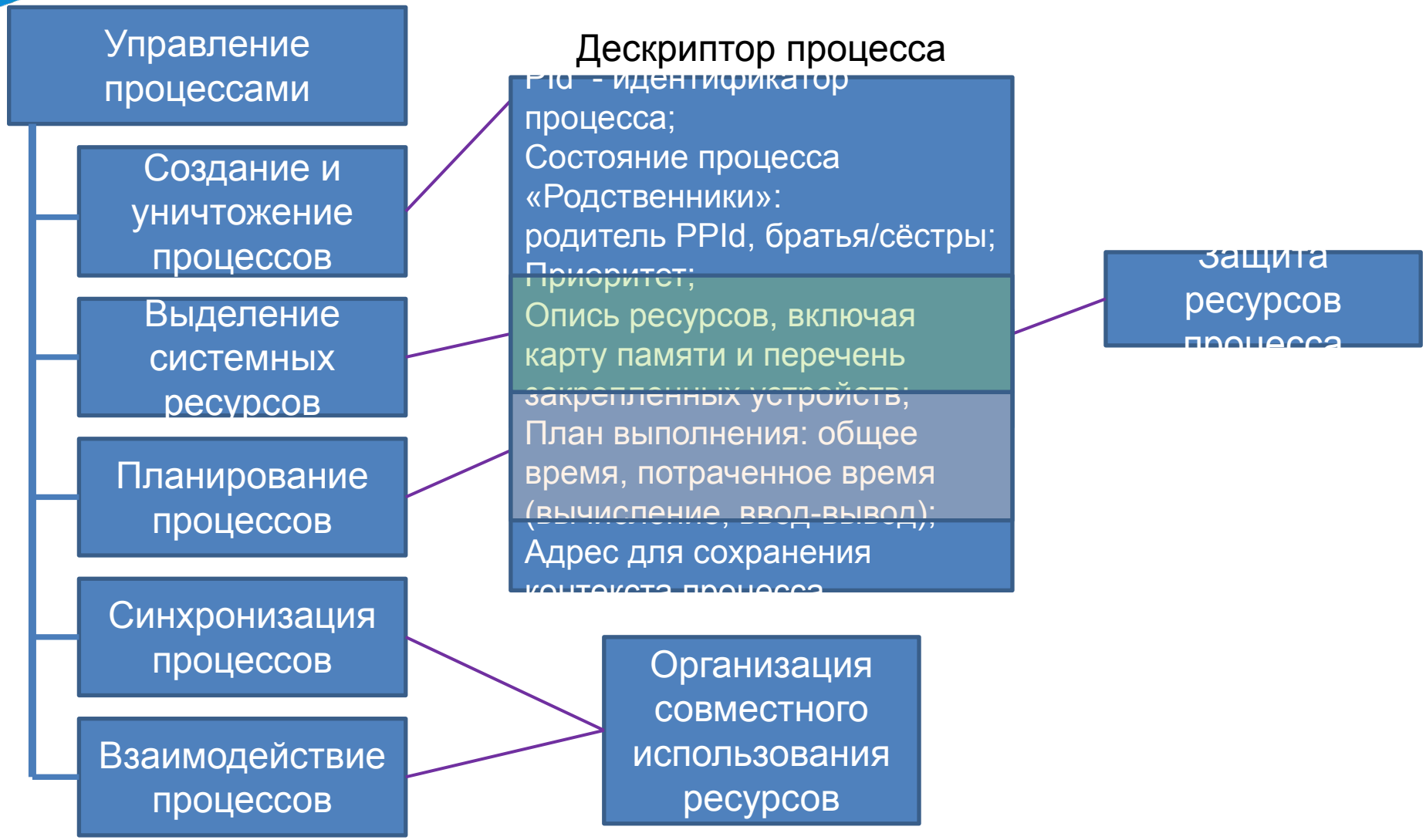
В соответствии с этим определением ОС выполняет две группы функций:
- предоставление пользователю или программисту вместо реальной аппаратуры компьютера абстрактной *виртуальной машины*, с которой удобней работать и которую легче программировать;
- повышение эффективности использования компьютера путем *рационального управления его ресурсами* в соответствии с некоторым критерием.

Функциональные компоненты операционной системы

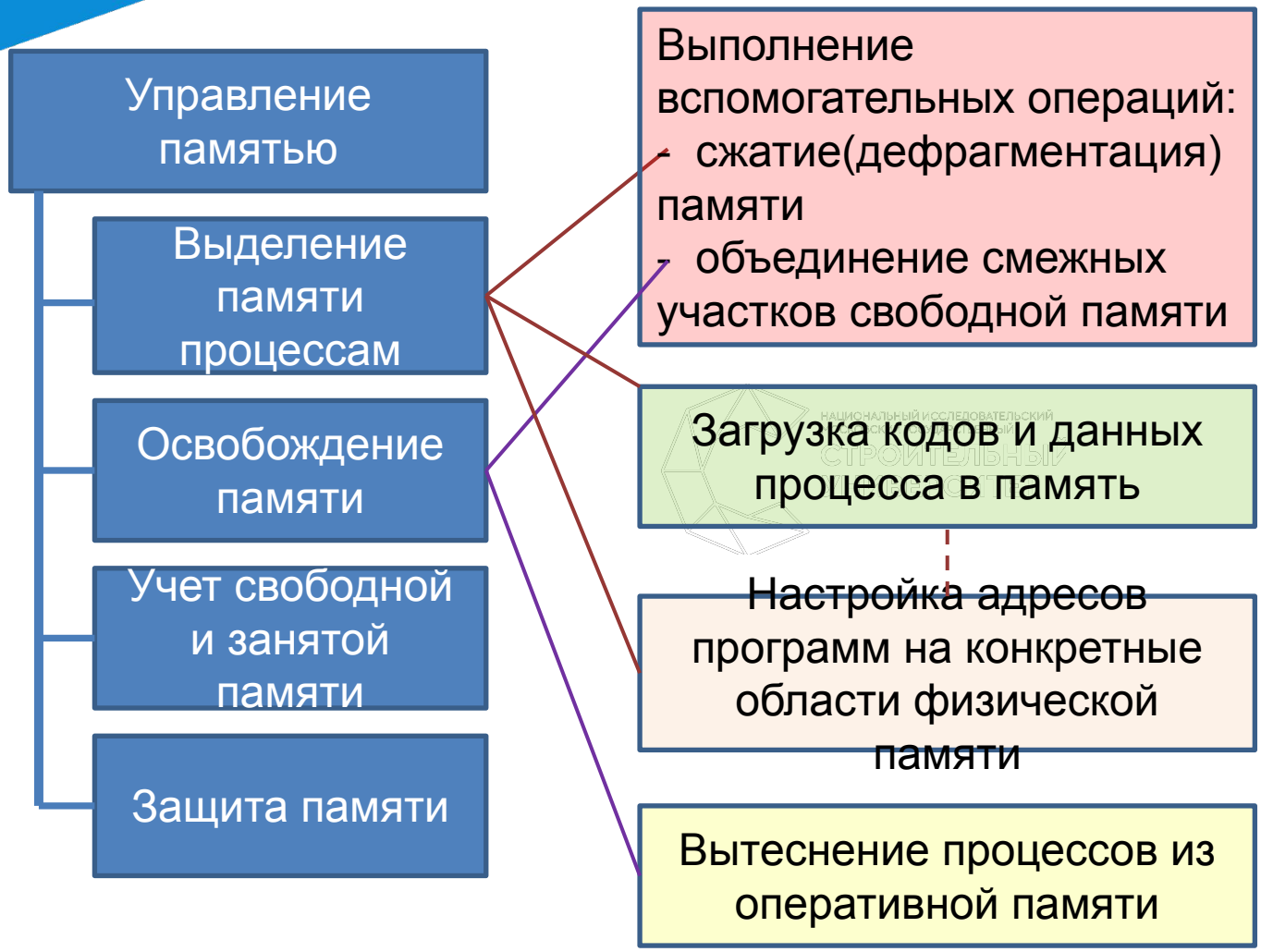
Функции операционной системы автономного компьютера обычно группируются либо в соответствии *с типами локальных ресурсов*, которыми управляет ОС, либо в соответствии *со специфическими задачами, применимыми ко всем ресурсам*.
Такие группы функций называют *функциональными компонентами* или *подсистемами*.



Функциональные компоненты ОС
Управление процессами



Функциональные компоненты ОС
Управление памятью



Функциональные компоненты ОС
Управление файлами и внешними устройствами

Управление файлами и внешними устройствами (ВУ)

Файловая система

Представление пользователю информации о хранимых на носителях данных в удобной для восприятия форме: иерархия *каталогов/папок* и *файлов.*

Преобразование символьных имен файлов, с которыми работает пользователь или прикладной программист, в физические адреса данных на носителях

Действия с файлами:
- поиск
- вывод содержимого
- изменение
- сохранение
- перемещение
- копирование
- переименование
- удаление

Организация совместного доступа к файлам

Защита файлов от несанкционированного доступа

Поддержание высокоуровневого унифицированного интерфейса прикладного программирования к разнородным ВУ является одной из наиболее важных задач ОС. Со времени появления ОС UNIX такой унифицированный интерфейс в большинстве операционных систем строится на основе **концепции файлового доступа**.

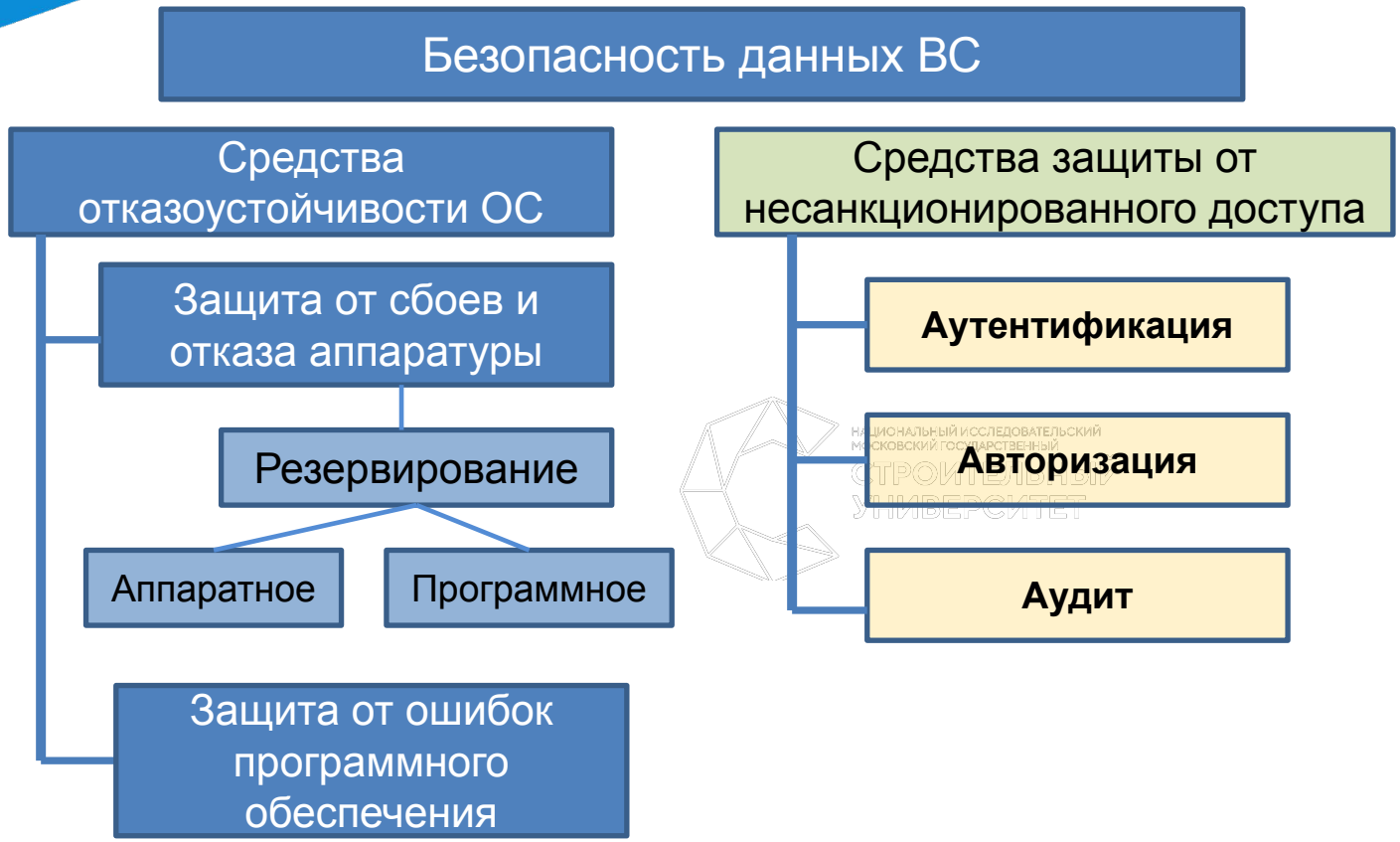
Управление ВУ

Драйвер устройства – программа, управляющая конкретной моделью ВУ и учитывающая все его особенности

Разнообразие драйверов, поддерживаемых ОС, гарантирует возможность подключения большего числа ВУ различных производителей

Обмен с любым ВУ – это обмен с файлом, имеющим имя и представляющим собой неструктурированную последовательность байтов

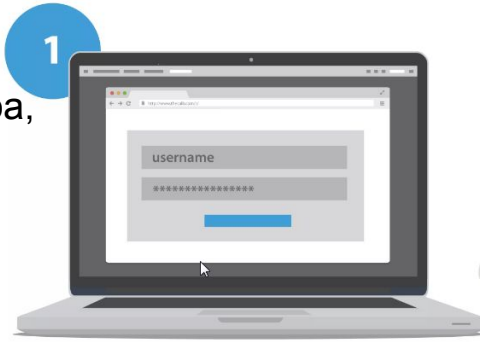
Функциональные компоненты ОС
Защита данных и администрирование



Функциональные компоненты ОС
Защита данных и администрирование

Средства защиты от несанкционированного доступа

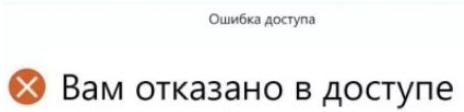
Идентификация — процедура, в результате выполнения которой субъект идентификации указывает системе свой идентификатор



Аутентифика́ция — процедура проверки подлинности,

примеры:
-
проверка подлинности пользователя путём сравнения введённого им пар оля
с паролем в базе данных пользователей;
- подтверждение подлинности электронного письма путём проверки цифровой
подписи письма по ключу проверки подписи отправителя;
-проверка контрольной суммы файла на соответствие сумме, заявленной автором этого файла

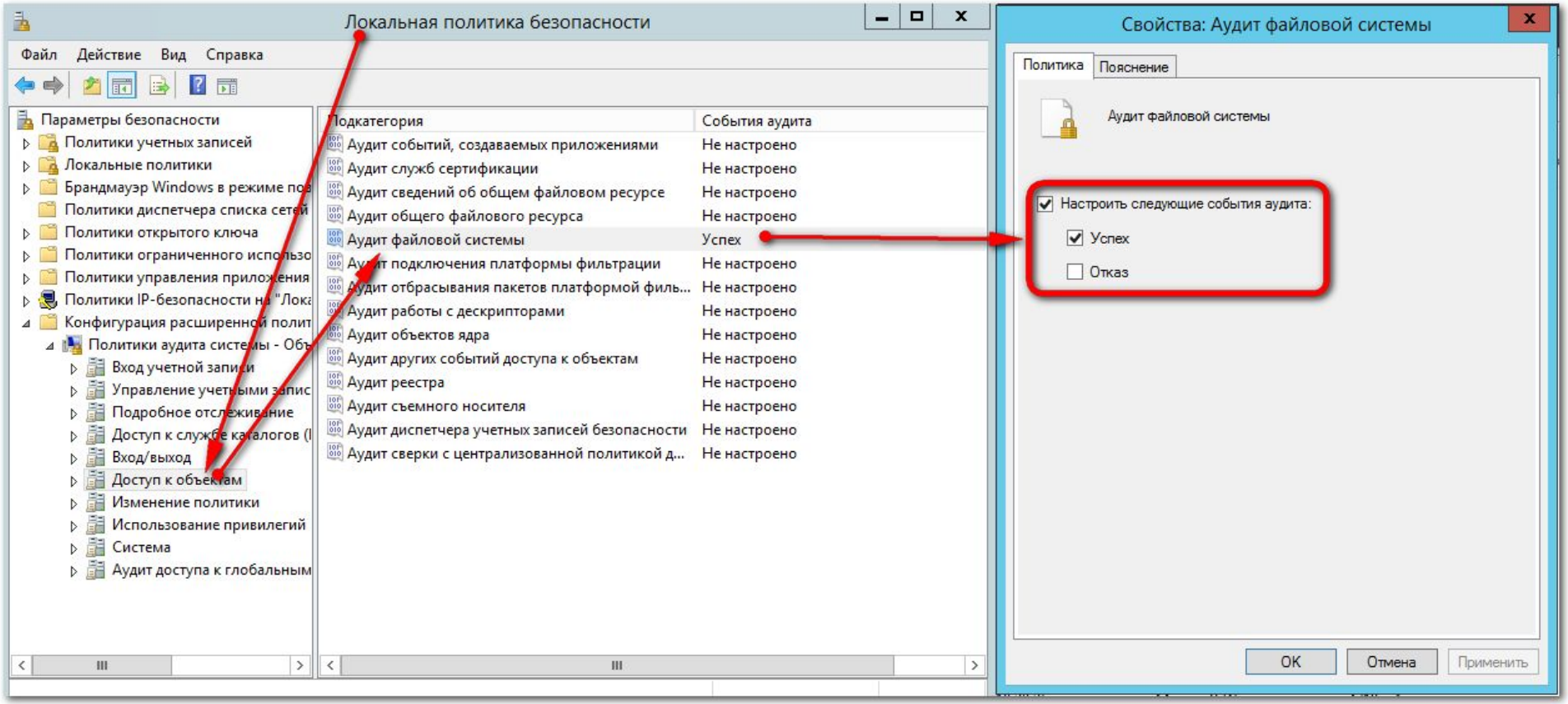
Авторизация — предоставление определенному лицу или группе лиц прав на выполнение определенных действий, а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.



Аудит – фиксация всех событий, от которых зависит безопасность системы.

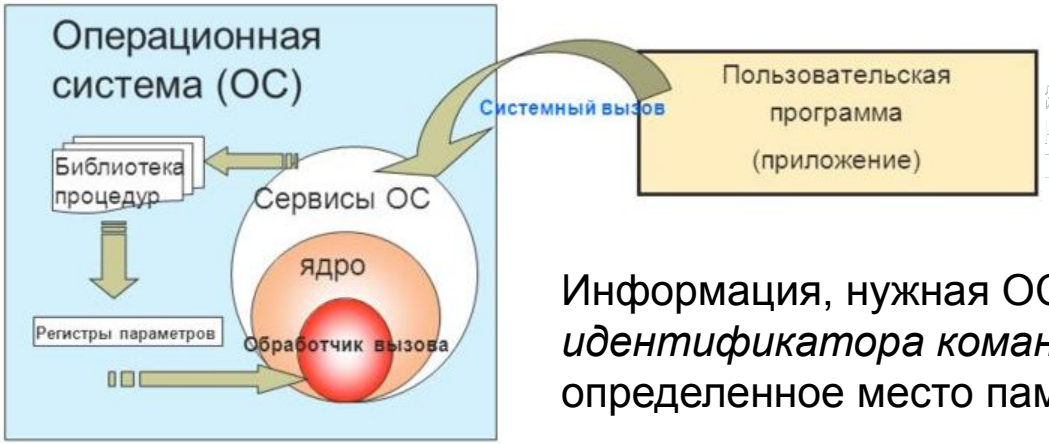
Функциональные компоненты ОС
Защита данных и администрирование

Аудит – фиксация всех событий, от которых зависит безопасность системы.



Функциональные компоненты ОС
Интерфейс прикладного программирования

Когда прикладным программистам для выполнения тех или иных действий требуется особый статус, которым обладает только операционная система, они используют в своих приложениях обращения к ОС. Эти обращения являются обращениями к функциям API с помощью *системных вызовов*.



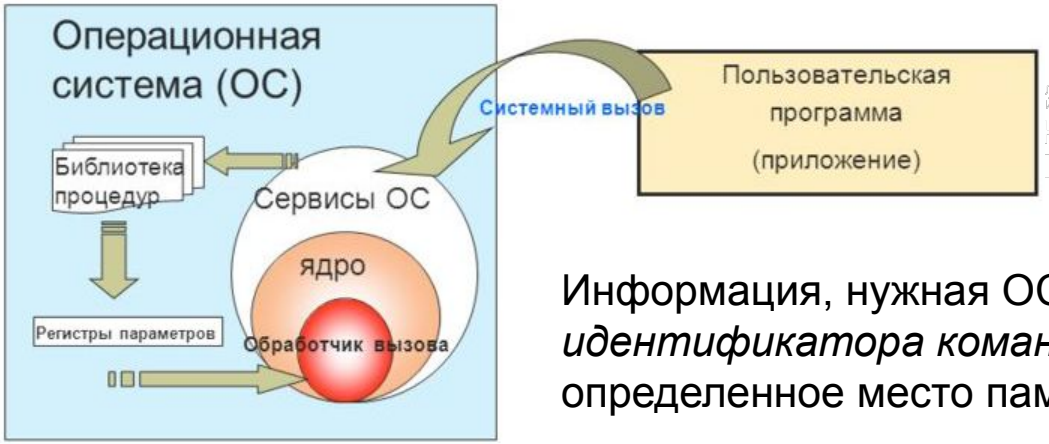
Способ, которым приложение получает услуги операционной системы, очень похож на вызов подпрограмм.

Информация, нужная ОС и состоящая обычно из *идентификатора команды и данных*, помещается в определенное место памяти, в регистры и/или стек.

Затем управление передается операционной системе, которая выполняет требуемую функцию и возвращает результаты через память, регистры или стеки.
Если операция проведена неуспешно, то результат включает индикацию ошибки.

Функциональные компоненты ОС
Интерфейс прикладного программирования

Когда прикладным программистам для выполнения тех или иных действий требуется особый статус, которым обладает только операционная система, они используют в своих приложениях обращения к ОС. Эти обращения являются обращениями к функциям API с помощью *системных вызовов*.



Способ, которым приложение получает услуги операционной системы, очень похож на вызов подпрограмм.

Информация, нужная ОС и состоящая обычно из *идентификатора команды и данных*, помещается в определенное место памяти, в регистры и/или стек.

Для разработчиков приложений все особенности конкретной ОС представлены особенностями её API.
Поэтому ОС с различной внутренней организацией, но с одинаковым набором функций API кажутся разработчику одной и той же ОС, что упрощает стандартизацию операционных систем и обеспечивает *переносимость приложений* между внутренне различными ОС, соответствующими определенному стандарту на API.

Функциональные компоненты ОС
Интерфейс прикладного программирования (API)

Способы реализации API

- **На уровне модулей ОС**
- На уровне системы программирования
- С помощью внешних библиотек

При этом способе ОС ответственна за выполнение функций API. Объектный код, выполняющий функции, либо непосредственно входит в состав ОС, либо поставляется в составе динамически загружаемых библиотек (Dynamic Link Library, DLL), разработанных для данной ОС.
Система программирования ответственна только за то, чтобы организовать интерфейс для вызова этого кода.
Динамически загружаемые модули .dll исполняются при вызове их исполняющейся программой, но **НЕ ПОДКЛЮЧАЮТСЯ** к коду программы!
Достоинства: достигается ***наибольшая эффективность выполнения функций API*** по сравнению со всеми другими вариантами реализации API, так как результирующая программа обращается непосредственно к ОС.
Недостатки: практически ***полное отсутствие переносимости*** не только кода результирующей программы, но и кода исходной программы.

Программа, созданная для одной архитектуры вычислительной системы, не сможет исполняться на вычислительной системе другой архитектуры даже после того, как ее объектный код будет полностью перестроен.
Чаще всего система программирования не сможет выполнить перестроение исходного кода для новой архитектуры вычислительной системы, поскольку многие функции API, ориентированные на определенную ОС, могут просто *отсутствовать* в новой архитектуре. Таким образом, в данной схеме для переноса прикладной программы с одной целевой вычислительной системы на другую будет требоваться *изменение исходного кода* программы.

Функциональные компоненты ОС
Интерфейс прикладного программирования (API)

- Способы реализации API
  - На уровне модулей ОС
  - На уровне системы программирования
  - С помощью внешних библиотек

При этом способе ОС ответственна за выполнение функций API. Объектный код, выполняющий функции, либо непосредственно входит в состав ОС, либо поставляется в составе динамически загружаемых библиотек (Dynamic Link Library, DLL), разработанных для данной ОС.
Система программирования ответственна только за то, чтобы организовать интерфейс для вызова этого кода.
Динамически загружаемые модули .dll исполняются при вызове их исполняющейся программой, но **НЕ ПОДКЛЮЧАЮТСЯ** к коду программы!
Достоинства: достигается ***наибольшая эффективность выполнения функций API*** по сравнению со всеми другими вариантами реализации API, так как результирующая программа обращается непосредственно к ОС.
Недостатки: практически ***полное отсутствие переносимости*** не только кода результирующей программы, но и кода исходной программы.

Примером API такого рода может служить набор функций, предоставляемых пользователю со стороны ОС типа Microsoft Windows – WinAPI (*Windows API*). Даже внутри этого корпоративного API существует определенная несогласованность, которая несколько ограничивает переносимость программ между различными версиями ОС Windows.
Еще одним примером такого API можно считать набор сервисных функций ОС типа MS-DOS, реализованный в виде *набора подпрограмм обслуживания программных прерываний*.

Функциональные компоненты ОС
Интерфейс прикладного программирования (API)

- Способы реализации API
  - На уровне модулей ОС
  - **На уровне системы программирования**
  - С помощью внешних библиотек

В этом случае функции API предоставляются пользователю в виде *библиотеки функций соответствующего языка программирования*. Обычно речь идет о *библиотеке времени исполнения – RTL (run time library)*.
Система программирования предоставляет пользователю библиотеку соответствующего языка программирования и обеспечивает подключение к результирующей программе объектного кода, ответственного за выполнение этих функций.
Достоинства***: переносимость исходного кода программы в таком варианте будет самой высокой***, поскольку синтаксис и семантика всех функций будут строго регламентированы в стандарте соответствующего языка программирования. Они зависят от языка и не зависят от архитектуры целевой вычислительной системы.
Недостатки: ***эффективность*** функций API в таком варианте будет ***несколько ниже, чем при непосредственном обращении к функциям ОС***, поскольку для выполнения многих функций API библиотека RTL языка программирования должна все равно выполнять обращения к функциям ОС.

Примеры: функции динамического выделения памяти в языках C (malloc, realloc и free), в C++ (new и delete), и Pascal (new и dispose).
Проблема, главным образом, заключается в том, что большинство языков программирования предоставляют пользователю не очень широкий набор стандартизованных функций. Поэтому разработчик исходного кода существенно ограничен в выборе доступных функций API.

Функциональные компоненты ОС
Интерфейс прикладного программирования (API)

- Способы реализации API
  - На уровне модулей ОС
  - На уровне системы программирования
  - С помощью внешних библиотек

Функций API в этом случае предоставляются пользователю в виде библиотеки процедур и функций, созданной сторонним разработчиком. Система программирования ответственна только за то, чтобы подключить объектный код библиотеки к результирующей программе. Причем внешняя библиотека может быть и динамически загружаемой (библиотека RTL).

Достоинства*:* с точки зрения переносимости исходного кода **требование только одно – используемая внешняя библиотека должна быть доступна в любой из архитектур вычислительных систем**, на которые ориентирована прикладная программа. Тогда удастся достигнуть переносимости. Это возможно, если используемая библиотека удовлетворяет какому-то принятому стандарту, а система программирования поддерживает этот стандарт.

Недостатки: ***с точки зрения эффективности выполнения имеет самые низкие результаты***, поскольку внешняя библиотека обращается как к функциям ОС, так и к функциям RTL языка программирования. Только при очень высоком качестве внешней библиотеки ее эффективность становится сравнимой с библиотекой RTL.

Примеры: библиотеки, удовлетворяющие стандарту POSIX, доступны в большинстве систем программирования для языка С. И если прикладная программа использует только библиотеки этого стандарта, то ее исходный код будет переносимым.

Функциональные компоненты ОС
Пользовательский интерфейс

Операционная система должна обеспечивать удобный интерфейс не только для прикладных программ, но и для человека, работающего за терминалом. Этот человек может быть конечным пользователем, администратором ОС или программистом.

В ранних операционных системах пакетного режима функции пользовательского интерфейса были практически отсутствовали и были сведены к минимуму.