Архитектура Операционной системы

Состав ОС

Первая часть — ядро, низкоуровневая основа любой ОС, 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 - командный интерпретатор, «переводчик» с программного языка на «железный», язык машинных кодов.

- **Вторая часть** - **Подсистема управления ресурсами (resource allocator)** - управляет вычислительными ресурсами компьютера - оперативной и внешней памятью, процессором
- **Третья часть - Управляющая программа (control program, supervisor) –** управляет исполнением других программ и функционированием устройств ввода-вывода.(используются специализированные программы для управления различными устройствами, входящими в состав компьютера. Драйвера «системные библиотеки»)
- **4 часть** — удобная оболочка, с которой общается пользователь — интерфейс. Своего рода красивая обертка, в которую упаковано скучное и не интересное для пользователя ядро.

Архитектура ядра ОС

Ядро – самый ключевой, основной компонент ОС, именно в нем реализуется большая часть функциональности ОС

Появились различные подходы к проектированию и реализации ядра ОС

- Монолитное ядро
- Поуровневый подход
- Микроядро
- Модули ядра

Монолитное ядро (все вместе)

Появилось исторически первым в ходе эволюции ОС

ОС используют большое монолитное ядро

- **Монолит** – все вместе, все библиотеки, сервисные функции в одном ядре

Монолитное ядро содержит след. базовые элементы:

- Планирование процессов
- Управление файловой системой
- Сетевое взаимодействие
- Драйверы устройств
- Управление памятью

Монолитное ядро

Преимущества:

- **Производительность** - в виду того, что количество переключений из контекста режима пользователя в режим ядра сведено к минимуму;

Недостатки

- **Неустойчивость к сбоям** – так как все базовые элементы и их работа выполняются в режиме ядра, и если хотя бы в одном модуле или блоке ядра произойдет какой-либо сбой, то ему будет подвержена вся ОС(все ядро), вариантов других нет, закончится все – перезапуском ОС.

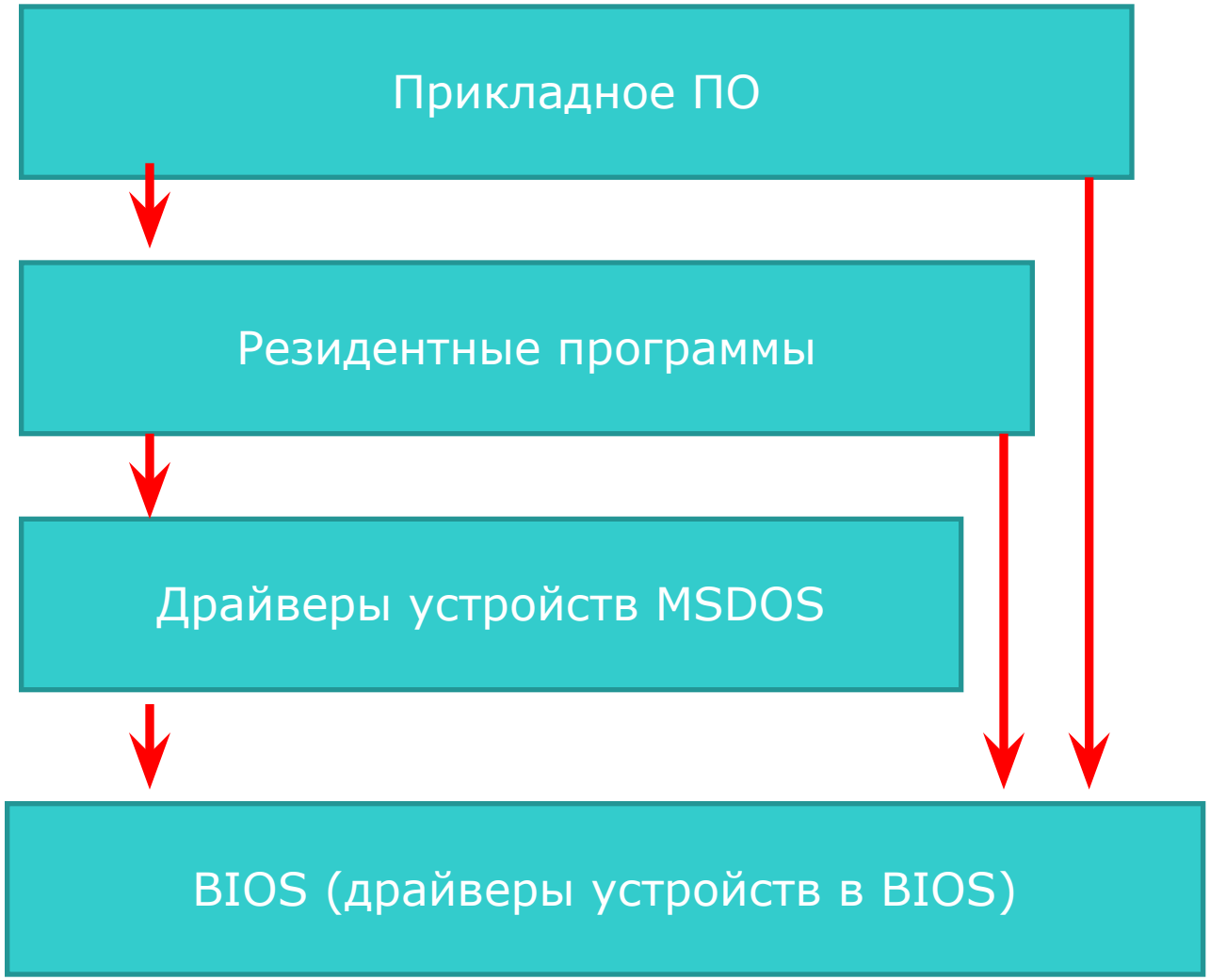
Монолитный подход - простейший

Первые ОС были монолитными, в них
- Нет определенной структуры
- Нет уровней
- Нет разделения на модули

Определенной структуры данные ОС не имеют. Это большой набор сервисных функций. Нет деления на слои и модули.

- ОС были простыми и большими, так как все время дописывались, засчет этого росли.
- Пример – классический MSDOS
  - Больше функциональности, меньше места (в то время было очень маленькое адресное пространство)
  - Разделение на уровни не было, программы могли работать с оборудованием напрямую, могли сами вызывать нужные драйверы

MS DOS


Прикладное ПО
Резидентные программы
Драйверы устройств MSDOS
BIOS (драйверы устройств в BIOS)

Монолитный подход - Уровни

- **Простая неорганизованная структура - плохо;**
- **Улучшение – ОС разбивается на ряд уровней (слоев) – каждый следующий уровень базируется на предыдущем(вводится понятие иерархии уровней)**
  - **Самый нижний – аппаратное обеспечение**
  - **Самый верхний – интерфейс пользователя**
- **Каждый уровень использует только функции (сервисы), предоставляемые нижестоящим уровнем**
- **Все (или почти все уровни) работают в режиме ядра**
- **Примеры таких ОС – MULTICS, VAX\VMC**

Простой поуровневый подход

Разбиение на уровни помогает разбить программирование ОС, становится легче программировать.

Используется оригинальным ядром Unix

- Небольшое число монолитных слоев
- Нет инкапсуляции, функции и сервисы, предоставляемые на разных уровнях доступны всей системе
- По сути ядро – набор процедур, которые могут вызывать любые другие процедуры
- Огромное количество функциональности в ядре (все что можно и нужно

Современные ОС более четко разделены на уровни

Ядро ОС Unix

Пользователи

- Интерфейс системных вызовов

Символьный вв/выв, блочный вв/выв, драйверы терминалов, файловая система, дисковые драйверы, планировщик, виртуальная память

- Интерфейс с аппаратурой

Терминалы | Контроллеры дисков | Физическая память

Поуровневый подход позволяет независимо работать, менять и отлаживать ОС

Поуровневый подход

Если уровни строго разделены:

+ можно над ними независимо работать

+ можно независимо заменять

Например, сетевой стек TCP/IP является примером строгого разделения на уровни (уровни все четко выверены и подробно описаны)

Сложности

- как определить эти слои (непростая задача)

- разделение на уровни возможно только при наличии иерархии вызовов и отсутствии циклических зависимостей.

Поуровневый подход

Циклические зависимости

Пример

Возьмем драйвер диска

- Драйверу устройства можно ожидать завершения выполнения операции вв/выв, это означает обращение к уровню планировщика времени ЦП. (это вызов с верхнего слоя, а драйвер диска находится в самом низу на аппаратном уровне, это циклическая зависимость)
- ЦП может вызвать драйвер устройства для выгрузки и подгрузки процессов

Поэтому в этом случае данная архитектурная модель не работает

Поуровневый подход

- *Чем больше таких уровней, тем больше возникает проблемных ситуаций*.

Выход – отход от строгой поуровневой модели и возврат *к небольшому числу слоев с большой функциональностью*.

Дальнейшим этапом развития придумали **МИКРОЯДРО.**

МИКРОЯДРО

Микроядро – ядро, содержащее только самые необходимые функции

Идея: минимизировать само ядро, вынести как можно функциональности в режим пользователя (т.е. исполнять эту функциональность в виде обычных процессов)

Многие сервисы становятся пользовательскими процессами:

- Драйверы устройств,
- Файловые системы,
- Менеджер виртуальной памяти,
- Оконные системы графического интерфейса пользователя,
- Службы безопасности

Данный подход популяризован ядром MACH («МАК»)

На основе MACH сделаны, среди прочих, Mac OSX (комп.Apple), GNUHurd

Структура микроядра

- Компоненты ОС, являющиеся внешними к микроядру реализуются в виде серверных процессов. Взаимодействие с ними осуществляется через обмен сообщениями, а не путем системных вызовов.
- Микроядро основывается на механизме обмена сообщениями:
  - проверяет корректность сообщений
  - передает их между компонентами
  - Проверяет, разрешен ли обмен сообщениями
- Предоставляет доступ к оборудованию
- Фактически реализует клиент-серверную архитектуру на одном компьютере.

Преимущества микроядра

Унифицированные интерфейсы – процессы могут обмениваться сообщениями, нет разницы между режимом пользователя и режимом ядра, все сервисы предоставляются через обмен сообщениями в клиент-серверной архитектуре (н-р, запросы могут обрабатываться на удаленные машины)

Расширяемость – легче расширить, новые сервисы добавляются как расширяется функциональность программы

Портируемость(переносимость) – на новое оборудование нужно перенести микроядро.

Надежность, безопасность – меньше кода выполняется в режиме ядра, ошибки программ в режиме пользователя не влияют на остальную часть системы.

Недостатки микроядра

Больше расходов на взаимодействие между системными сервисами

- Каждое взаимодействие требует переключения режимов (переходы из режима пользователя в режим ядра и обратно. Механизм передачи сообщений требует очень частого переключения контекста(режим ядра – режим пользователя, они “съедают” значительную часть мощности ЦП)).
- Системные сервисы, работающие в режиме пользователя – это процессы, ОС нужно их планировать(диспетчерезация)

- Решение 1: реинтеграция таких сервисов обратно в ядро (интегрировать некоторые сервисы в ядро и “убрать” переключения
  - Улучшается производительность(меньше переключений, одно адресное пространство)
  - Такое ядро было сделано в ядре Mach
- Решение 2: сделать ядро еще меньше – экспериментальные архитектуры (нано-ядра, пико-ядра)

Минимальная функциональность в микроядре

1) Низкоуровневое управление памятью

- Отображение страниц на физ.память
- Все остальные механизмы предоставляются сервисами, работающими в режиме пользователя

Защита адресного пространства
Механизмы замещения страниц
Управление виртуальной памятью

2) Межпроцессорное взаимодействие

3)Ввод/вывод и обработка прерываний.

Модуль

Модуль – нечто среднее между поуровневой архитектурой и микроядром

- Удобно для разработки (каждый может разрабатывать свой модуль без необходимости трогать основную систему)
- Модули находятся в режиме ядра (уменьшаются затраты на взаимодействие модулей между собой)
- Компромисс ради производительности.

Под разные задачи необходимо выбирать ОС

- Если ОС должна быть высоко устойчива к сбоям, тогда выбираем соответствующую ОС этим задачам
- Если для нормальных задач, то подойдет и Linux

Модули ядра(развивались параллельно вместе с микроядром)

- Многие ОС реализуют поддержку модульности

Пример – ОС **Linux.** Классифицируют как монолитное ядро.

- Каждый ключевой компонент – отдельный модуль
- Взаимодействие происходит через определенные интерфейсы
- Загружаются модули по требованию

Можно самому компилировать ядро Linux, выбирать нужные модули, решать какие модули войдут в ядро, т.е. компоновать ядро согласно своих требований, что позволяет получать преимущество – гибкость ОС

ОС **Linux**

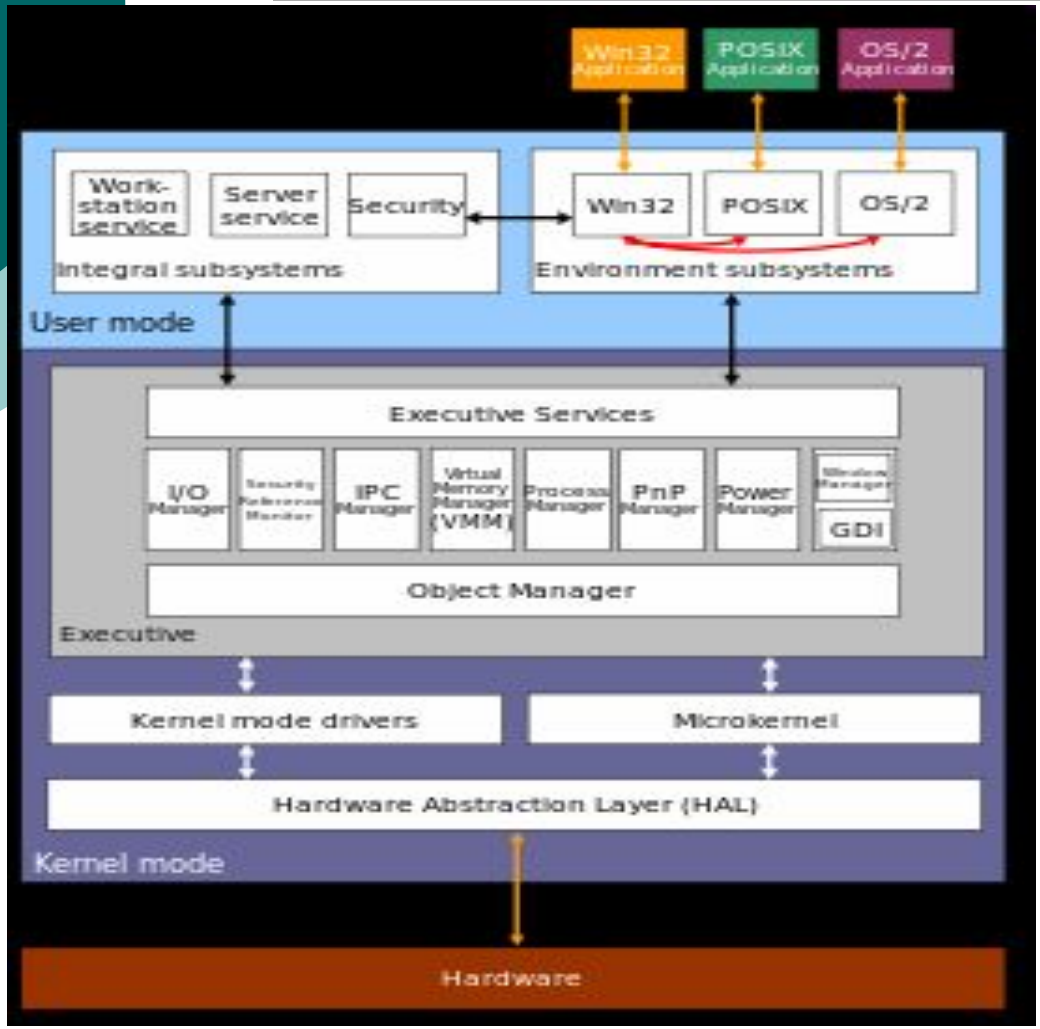
Торговая марка Linux зарегистрирована на Линуса Торвальдса.
Linux, произносится «ли́нукс» . **Создатель Linux** - Линус Бенедикт Торвальдс родился 28 декабря 1969 в Хельсинки — финский программист. Воодушевлённый прочтением книги Эндрю Таненбаума, посвящённой операционной системе Minix, Линус создал Linux — ядро операционной системы GNU/Linux, являющейся на данный момент самой распространённой из свободных операционных систем.

Уникальность данной ОС – открытый исходный код, он разрабатывается сообществом, поэтому данная ОС работает на очень многих существующих платформах (архитектуры процессора таких как Intel x86, x86-64, PowerPC, ARM, Alpha AXP, Sun SPARC, Motorola 68000, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, AXIS CRIS, Renesas M32R, Atmel AVR32, Renesas H8/300, NEC V850, Tensilica Xtensa и многих других).

В отличие от большинства других операционных систем, Linux не имеет единой «официальной» комплектации. Вместо этого Linux поставляется в большом количестве так называемых дистрибутивов, в которых ядро Linux соединяется с утилитами GNU и другими прикладными программами (например, X.org), делающими её полноценной многофункциональной операционной средой.

Сам разработчик сегодня заявляет, что ядро ОС стало настолько большим, что его уже трудно поддерживать, оно недостаточно структурировано.

ОС Windows NT



- Есть все: послойность, модульность и микроядро (используется гибридное ядро)
- Уникальная система с точки зрения теории – гениально структурированное ядро ОС. В структуре спокойно разберется один человек
- В ядре сосредоточены ключевые компоненты, а то, что можно было вынести без потери производительности – вынесено в режим пользователя.
- На основе этого ядра сделаны Windows 2000/XP/7/8/
- Windows NT первоначально работала на 5 платформах.

Критерии выбора ОС

- ***Монолит*** –если есть разбиение на уровни, модули, но большей частью все компоненты выполняются в режиме ядра, этот монолит обычно раструктурированный.
- ***Микроядро*** – если есть механизм обмена сообщениями и он является ключевым для организации взаимодействия компонентов ядра, сделано по клиент-серверной архитектуре. Его части выполняются в режиме пользователя. Такая ОС устойчива к сбоям

Виртуализация (виртуальная машина)

Сегодня применяется практически везде:

- Планшет Android
- Телефон с Java
- Веб-сервер (на хостинге работает вир. сервер)

Впервые коммерческая виртуализация появилась в 1972г. В компьютерах фирмы IBM в минифреймах

- Вирт.машина сама по себе это дальнейшее развитие поуровневого подхода

Виртуализация

- Создает виртуально аппаратное окружение (виртуальный процессор, виртуальную память, виртуальные устройства вв/выв), которые **реализуются программно.**
- Сама виртуальная машина работает как приложение в ОС
- Виртуальная машина позволяет одному ПК (или серверу) выполнять одновременно несколько ОС, создавать ощущения того, что есть несколько полностью изолированных ОС. Или как частный случай, позволяет выполнять несколько сессий одной ОС на одном ПК (на одной платформе).
  Т.О. мы можем на одном ПК выполнять множество различных приложений, программ, работающих на различных ОС.

Архитектура, поддерживающая виртуальные машины

Классическая архитектура

- В классической архитектуре используется иерархия – процессы, ядро, аппаратное обеспечение. Процессы связываются с аппаратурой только через ядро
- Когда появляется виртуализация, появляется слой поддержки вируальных машин, который располагается над аппаратным обеспечением. Он предоставляет уже виртуальное аппаратное обеспечение для каждой из виртуальных машин. Каждая виртуальная машина служит прослойкой к своему ядру ОС, которое в свою очередь выполняет уже пользовательские процессы. Поэтому получается несколько полностью изолированных сред выполнения, которые фактически могут объединяться между собой только на уровне аппаратного обеспечения и уровне поддержки виртуальных машин. Естественно в такой архитектуре поддерживается полный контроль безопасности.

Многопроцессорность

- Есть несколько вычислительных ядер
- Улучшает производительность за счет введения истинного параллелизма выполнения программ

Преимущества:

- Производительность – на каждом процессоре может работать один или больше процессов
- Доступность – отказ одного процессора не приведет к отказу всей системы
- Расширяемость – увеличение производительности за счет добавление ЦП

Многопроцессорность (изменения в архитектуре)

Многопроцессорная ОС должна предоставлять все тоже, что и однопроцессорная.

Учитываются следующие сложности:

- **Реентабельность ядра** – код ядра д.б. реентабельным – т.е. один и тот же код может быть выполнен одновременно несколькими процессорами. Реента – англ.слово - «входить еще раз» - вход несколько одновременно. Это накладывает определенные требования синхронизации
- **Синхронизация** – истинный параллелизм выполнения процессов и доступ к общим ресурсам (ОП, вв/выв) требуют эффективной синхранизацииотказ одного процессора не приведет к отказу всей системы
  Синхронизация процессов — приведение двух или нескольких процессов к такому их протеканию, когда определённые стадии разных процессов совершаются в определённом порядке, либо одновременно.
- **Планирование** – загрузки ЦП, процессы планируются для загрузки на разные ЦП
- **Надежность и устойчивость к сбоям** – если один процессор отказывает, задачи надо распределять на другие ЦП

Многоядерные процессоры

На одном кристалле процессора есть несколько вычислительных ядер

- Параллелизм в рамках одного процессора
- Каждое ядро по сути – отдельный процессор

Преимущества

- Несколько процессоров в одном чипе существенно увеличивают производительность
- Введение различных уровней КЭШ-памяти

Особенности методов построения ОС

- В руководстве по работе с операционной системой часто указываются особенности ее структурной организации и основные концепции, положенные в ее основу.
- К таким базовым концепциям относится способ построения ядра системы: *монолитное ядро* или *микроядро*.

- Большинство ОС использует монолитное ядро, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский режим и наоборот.

- Альтернативой является построение ОС на базе **микроядра,** работающего также в привилегированном режиме и выполняющего только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС - *серверы*, работающие в пользовательском режиме.
- При такой реализации ОС работает более медленно, так как часто выполняются переходы между привилегированным режимом и пользовательским, зато система получается более гибкой - ее функции можно наращивать, модифицировать или сужать, добавляя, модифицируя или исключая серверы пользовательского режима.
- Кроме того, серверы хорошо защищены друг от друга, как и любые пользовательские процессы.

Построение ОС на базе *объектно-ориентированного подхода* дает возможность использовать все достоинства этого метода (хорошо зарекомендовавшие себя на уровне приложений) внутри операционной системы, а именно:

- аккумуляцию удачных решений в форме стандартных объектов;
- возможность создания новых объектов на базе имеющихся с помощью механизма наследования;
- хорошую защиту данных за счет их инкапсуляции во внутренние структуры объекта, что делает данные недоступными для несанкционированного использования извне;
- структурированность системы, состоящей из набора хорошо определенных объектов.

- Наличие нескольких *прикладных сред* дает возможность в рамках одной ОС одновременно выполнять приложения, разработанные для нескольких ОС.
- Многие современные операционные системы поддерживают одновременно прикладные среды MS-DOS, Windows, Unix, OS/2 или хотя бы некоторого подмножества из этого популярного набора.
- Концепция множественных прикладных сред наиболее просто реализуется в ОС на базе микроядра, над которым работают различные серверы, часть которых реализуют прикладную среду той или иной операционной системы.

Распределенная организация ОС позволяет упростить работу пользователей и программистов в сетевых средах.

В распределенной ОС реализованы механизмы, которые дают возможность пользователю представлять и воспринимать сеть в виде традиционного однопроцессорного компьютера.

Характерными признаками распределенной организации ОС являются:

- наличие единой справочной службы разделяемых ресурсов;
- наличие единой службы времени;
- использование механизма вызова удаленных процедур (RPC) для прозрачного распределения программных процедур по компьютерам;
- применение многонитевой обработки, позволяющей распараллеливать вычисления в рамках одной задачи и выполнять эту задачу сразу на нескольких компьютерах сети;
- наличие других распределенных служб.