Билет № 1

Определение ОС. История развития ОС.

♦ **ОС как расширенная машина**

♦ **ОС как система управления ресурсами**

• Управление ресурсами включает решение двух общих, не зависящих от типа ресурса задач:
- ***планирование ресурса*** - то есть определение, кому, когда, а для делимых ресурсов и в каком количестве, необходимо выделить данный ресурс;
- ***отслеживание состояния ресурса*** - то есть поддержание оперативной информации о том, занят или не занят ресурс, а для делимых ресурсов - какое количество ресурса уже распределено, а какое свободно.

♦ Эволюция ОС

• Первый период (1945 -1955)

Программирование осуществлялось исключительно на машинном языке. Об операционных системах не было и речи, все задачи организации вычислительного процесса решались вручную каждым программистом с пульта управления. Не было никакого другого системного программного обеспечения, кроме библиотек математических и служебных подпрограмм.

• Второй период (1955 - 1965)

В эти годы появились ***первые алгоритмические языки***, а следовательно и первые системные программы - компиляторы. Стоимость процессорного времени возросла, что потребовало уменьшения непроизводительных затрат времени между запусками программ. Появились ***первые системы пакетной обработки***, которые просто автоматизировали запуск одной программ за другой и тем самым увеличивали коэффициент загрузки процессора. Системы пакетной обработки явились прообразом современных операционных систем, они стали первыми системными программами, предназначенными для управления вычислительным процессом. В ходе реализации систем пакетной обработки был разработан формализованный язык управления заданиями, с помощью которого программист сообщал системе и оператору, какую работу он хочет выполнить на вычислительной машине. Совокупность нескольких заданий, как правило, в виде колоды перфокарт, получила название пакета заданий.

• Третий период (1965 - 1980)

Операционные системы должны были бы работать и на больших, и на малых вычислительных системах.

Важнейшим достижением ОС данного поколения явилась реализация ***мультипрограммирования***. *Мультипрограммирование* - это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняются несколько программ. Пока одна программа выполняет операцию ввода-вывода, процессор не простаивает, как это происходило при последовательном выполнении программ (однопрограммный режим), а выполняет другую программу (многопрограммный режим). При этом каждая программа загружается в свой участок оперативной памяти, называемый разделом.

Другое нововведение - ***спулинг***. *Спулинг* в то время определялся как способ организации вычислительного процесса, в соответствии с которым задания считывались с перфокарт на диск в том темпе, в котором они появлялись в помещении вычислительного центра, а затем, когда очередное задание завершалось, новое задание с диска загружалось в освободившийся раздел.

Наряду с мультипрограммной реализацией систем пакетной обработки появился новый тип ОС - ***системы разделения времени***. Вариант мультипрограммирования, применяемый в системах разделения времени, нацелен на создание для каждого отдельного пользователя иллюзии единоличного использования вычислительной машины.

• Четвертый период (1980 - настоящее время)

На рынке операционных систем доминировали две системы: MS-DOS и UNIX. Однопрограммная однопользовательская ОС MS-DOS широко использовалась для компьютеров, построенных на базе микропроцессоров Intel 8088, а затем 80286, 80386 и 80486. Мультипрограммная многопользовательская ОС UNIX доминировала в среде «не-интеловских» компьютеров, особенно построенных на базе высокопроизводительных RISC-процессоров.

В середине 80-х стали бурно развиваться сети персональных компьютеров, работающие под управлением сетевых или распределенных ОС.

Билет № 2

Классификация ОС

♦ Особенности алгоритмов управления ресурсами

• Поддержка многозадачности.
По числу одновременно выполняемых задач операционные системы могут быть разделены на два класса:

- однозадачные (например, MS-DOS, MSX) и
- многозадачные (ОС ЕС, OS/2, UNIX, Windows 95).

Многозадачные ОС управляют разделением совместно используемых ресурсов, таких как процессор, оперативная память, файлы и внешние устройства.

• Поддержка многопользовательского режима.
По числу одновременно работающих пользователей ОС делятся на:

- однопользовательские (MS-DOS, Windows 3.x, ранние версии OS/2);
- многопользовательские (UNIX, Windows NT).

• Вытесняющая и невытесняющая многозадачность.
Среди множества существующих вариантов реализации многозадачности можно выделить две группы алгоритмов:

- невытесняющая многозадачность (NetWare, Windows 3.x);
- вытесняющая многозадачность (Windows NT, OS/2, UNIX).

• Поддержка многонитевости.
Важным свойством операционных систем является возможность распараллеливания вычислений в рамках одной задачи.

• Многопроцессорная обработка.
Важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки - ***мультипроцессирование***.

В наши дни становится общепринятым введение в ОС функций поддержки многопроцессорной обработки данных. Такие функции имеются в операционных системах Solaris 2.x фирмы Sun, Open Server 3.x компании Santa Crus Operations, OS/2 фирмы IBM, Windows NT фирмы Microsoft и NetWare 4.1 фирмы Novell.

Многопроцессорные ОС могут классифицироваться по способу организации вычислительного процесса в системе с многопроцессорной архитектурой: ***асимметричные ОС*** и ***симметричные ОС***. Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам. Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

♦ Особенности аппаратных платформ

• ***По типу аппаратуры различают операционные системы*** персональных компьютеров, мини-компьютеров, мейнфреймов, кластеров и сетей ЭВМ.
• ***Многопроцессорные системы*** требуют от операционной системы особой организации, с помощью которой сама операционная система, а также поддерживаемые ею приложения могли бы выполняться параллельно отдельными процессорами системы.
• ***Кластер*** - слабо связанная совокупность нескольких вычислительных систем, работающих совместно для выполнения общих приложений, и представляющихся пользователю единой системой. Наряду со специальной аппаратурой для функционирования кластерных систем необходима и программная поддержка со стороны операционной системы, которая сводится в основном к синхронизации доступа к разделяемым ресурсам, обнаружению отказов и динамической реконфигурации системы.
• Наряду с ОС, ориентированными на совершенно определенный тип аппаратной платформы, существуют операционные системы, специально разработанные таким образом, чтобы они могли быть легко перенесены с компьютера одного типа на компьютер другого типа, так называемые ***мобильные ОС***. Наиболее ярким примером такой ОС является популярная система UNIX. В этих системах аппаратно-зависимые места тщательно локализованы, так что при переносе системы на новую платформу переписываются только они.

♦ Особенности областей использования

• Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки (например, ОС ЕС),
- системы разделения времени (UNIX, VMS),
- системы реального времени (QNX, RT/11).

• ***Системы пакетной обработки*** предназначались для решения задач в основном вычислительного характера, не требующих быстрого получения результатов.

Для достижения этой цели в системах пакетной обработки используются следующая ***схема функционирования***: в начале работы формируется пакет заданий, каждое задание содержит требование к системным ресурсам; из этого пакета заданий формируется мультипрограммная смесь, то есть множество одновременно выполняемых задач.

Выбор нового задания из пакета заданий зависит от внутренней ситуации, складывающейся в системе, то есть выбирается «выгодное» задание.

В таких ОС ***невозможно гарантировать*** выполнение того или иного задания в течение определенного периода времени.

• ***Системы разделения времени.***

Каждому пользователю системы разделения времени предоставляется терминал, с которого он может вести диалог со своей программой. Так как в системах разделения времени каждой задаче выделяется только квант процессорного времени, ни одна задача не занимает процессор надолго, и время ответа оказывается приемлемым.

Критерием эффективности систем разделения времени является не максимальная пропускная способность, а удобство и эффективность работы пользователя.

• ***Системы реального времени*** применяются для управления различными техническими объектами, такими, например как спутник.

Критерием эффективности для систем реального времени является их способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата (управляющего воздействия). Это время называется временем реакции системы, а соответствующее свойство системы - реактивностью.

♦ Особенности методов построения

К базовым концепциям относятся:

- Способы построения ядра системы - монолитное ядро или микроядерный подход.
- Построение ОС на базе объектно-ориентированного подхода дает возможность использовать все его достоинства.
- 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 (POSIX), 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 операционной системы позволяет упростить работу пользователей и программистов в сетевых средах.

Билет №3

Сетевые операционные системы и их классификация

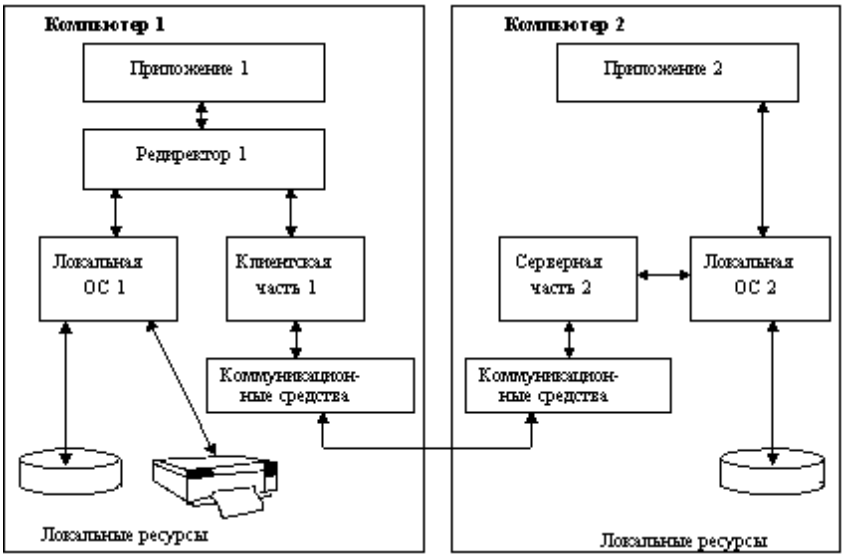
♦ **Структура сетевой операционной системы**

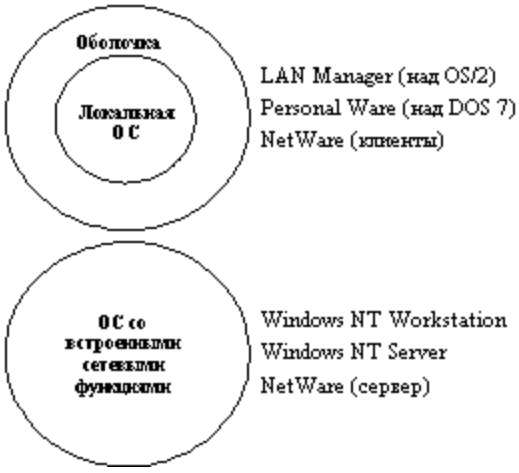
• Сетевая операционная система составляет основу любой вычислительной сети.

• Под ***сетевой операционной системой в широком смысле*** понимается совокупность операционных систем отдельных компьютеров, взаимодействующих с целью обмена сообщениями и разделения ресурсов по единым правилам - ***протоколам***.

• ***В узком смысле сетевая ОС*** - это операционная система отдельного компьютера, обеспечивающая ему возможность работать в сети.

• Компонент клиентской части - ***редиректор***. Именно редиректор перехватывает все запросы, поступающие от приложений, и анализирует их: рисунок ниже слева.



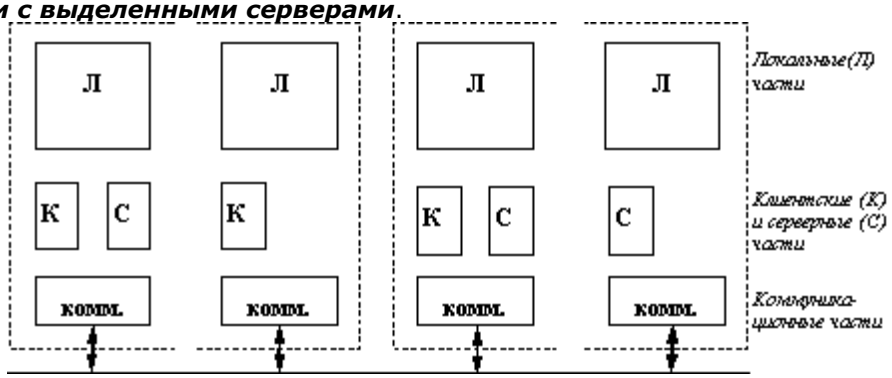


• На практике сложилось несколько подходов к построению сетевых операционных систем: рисунок выше справа

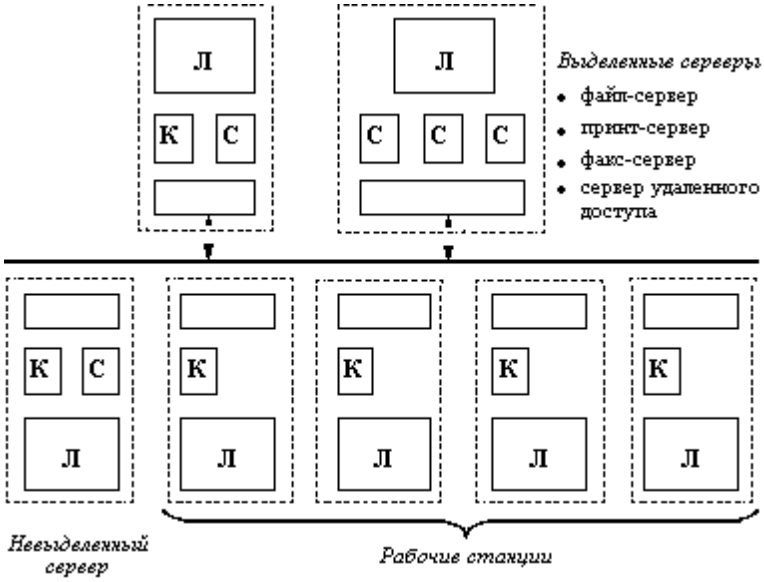
♦ **Классификация сетевых ОС**

• **Одноранговые сетевые ОС и ОС с выделенными серверами**

• В зависимости от того, как распределены функции между компьютерами сети, сетевые операционные системы, а следовательно, и сети делятся на два класса: ***одноранговые*** и ***двухранговые***. Последние чаще называют ***сетями с выделенными серверами***.



Одноранговая сеть



Двухранговая сеть

• В одноранговых сетях все компьютеры равны в правах доступа к ресурсам друг друга.

• Одноранговые сети могут быть построены, например, на базе ОС LANtastic, Personal Ware, Windows for Workgroup, Windows NT Workstation.

• При повышенных требованиях к характеристикам компьютера более подходящими являются ***двухранговые сети***, где сервер лучше решает задачу обслуживания пользователей своими ресурсами, так как его аппаратура и сетевая операционная система специально спроектированы для этой цели.

- ***<u>ОС для рабочих групп и ОС для сетей масштаба предприятия</u>***

Вид	Топология	Услуги	ОС
Сети отделов	Несколько компьютеров в одном здании	Разделение локальных ресурсов: приложения, данные, лазерные принтеры и модемы	Чаще всего это сеть с выделенным сервером NetWare 3.x или Windows NT, или же одноранговая сеть, например сеть Windows for Workgroups.
Сети кампусов	Соединяют несколько сетей отделов внутри отдельного здания или внутри одной территории предприятия	Сервисы такой сети включают взаимодействие между сетями отделов, доступ к базам данных предприятия, доступ к факс-серверам, высокоскоростным модемам и высокоскоростным принтерам.	Отделы используют для себя разные типы компьютеров, сетевое оборудование и сетевые операционные системы
Сети предприятия	Объединяют все компьютеры всех территорий отдельного предприятия	В таких сетях пользователям предоставляется доступ к информации и приложениям, находящимся в других рабочих группах, других отделах, подразделениях и штаб-квартирах корпорации: почтовая служба, средства коллективной работы, поддержка удаленных пользователей, факс-сервис, обработка голосовых сообщений, организация видеоконференций и др. Особое значение приобрели задачи преодоления ***гетерогенности*** - в сети появились многочисленные шлюзы, обеспечивающие согласованную работу различных ОС и сетевых системных приложений. ***Поддержка приложений***. В корпоративных сетях выполняются сложные приложения, требующие для выполнения большой вычислительной мощности – исполняются на *сервер приложений.* Сервер приложений должен базироваться на мощной аппаратной платформе (мультипроцессорные системы, часто на базе RISC-процессоров, специализированные кластерные архитектуры). ОС сервера приложений должна обеспечивать высокую производительность вычислений, а значит поддерживать многонитевую обработку, вытесняющую многозадачность, мультипроцессирование, виртуальную память и наиболее популярные прикладные среды (UNIX, Windows, MS-DOS, OS/2). ***Справочная служба***. ***Безопасность.***	Unix, Windows NT

Билет № 4

Понятие процесса. Управление процессами.

♦ Управление процессами

• ***Процесс*** - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов.

♦ Состояние процессов

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний: ***ВЫПОЛНЕНИЕ***, ***ОЖИДАНИЕ***, ***ГОТОВНОСТЬ***.

♦ Контекст и дескриптор процесса

• *Контекст процесса* - состоянием регистров и программного счетчика, режим работы процессора, указатели на открытые файлы, информация о незавершенных операциях ввода-вывода, коды ошибок выполняемых данным процессом системных вызовов и т.д.
• *Дескриптор процесса* - идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация.
• ***Очереди процессов*** представляют собой дескрипторы отдельных процессов, объединенные в списки.
• Программный код только тогда начнет выполняться, когда для него операционной системой будет создан процесс*. Создать процесс - это значит*:

1. создать информационные структуры - дескриптор и контекст;
2. включить дескриптор нового процесса в очередь готовых процессов;
3. загрузить кодовый сегмент процесса в оперативную память или в область свопинга.

♦ Алгоритмы планирования процессов

• Алгоритмы, основанные на ***квантовании,*** и алгоритмы, основанные на ***приоритетах***.
• В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

- процесс завершился и покинул систему,
- произошла ошибка,
- процесс перешел в состояние ОЖИДАНИЕ,
- исчерпан квант процессорного времени, отведенный данному процессу.

• ***Приоритет*** - это число, характеризующее степень привилегированности процесса при использовании ресурсов вычислительной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии.
• Алгоритмы, использующие ***относительные приоритеты*** и ***абсолютные приоритеты***.

- В системах с относительными приоритетами активный процесс выполняется до тех пор, пока он сам не покинет процессор, перейдя в состояние ОЖИДАНИЕ.
- В системах с абсолютными приоритетами выполнение активного процесса прерывается еще при одном условии: если в очереди готовых процессов появился процесс, приоритет которого выше приоритета активного процесса.

♦ Вытесняющие и невытесняющие алгоритмы планирования

• ***Невытесняющая многозадачность*** - это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.
• ***Вытесняющая многозадачность*** - это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Билет № 5

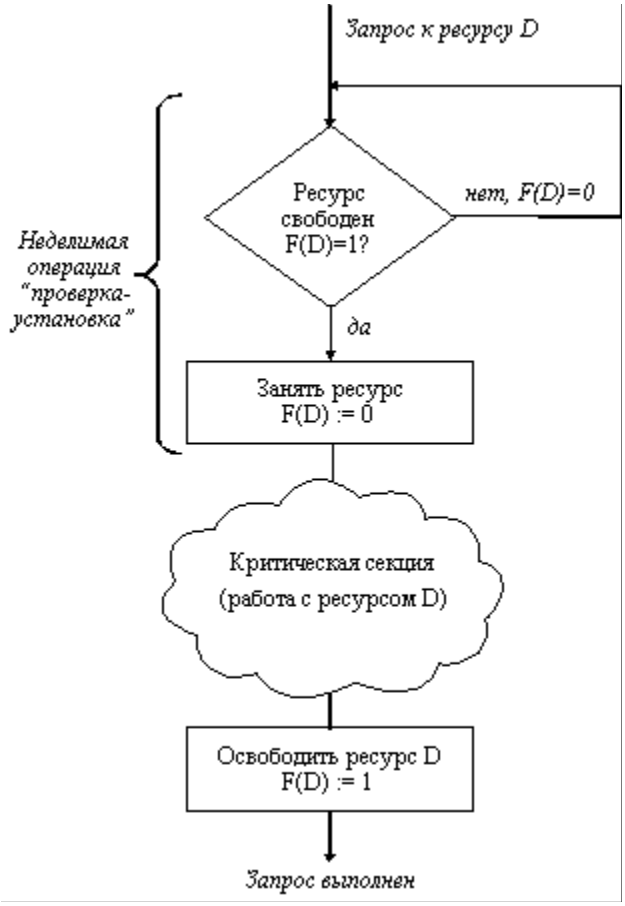
Средства синхронизации процессов.

♦ Проблема синхронизации

• Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме мультипрограммирования, может привести к их неправильной работе или даже к краху системы.
• Ситуация когда два или более процессов обрабатывают разделяемые данные, и конечный результат зависит от соотношения скоростей процессов, называются ***гонками***.

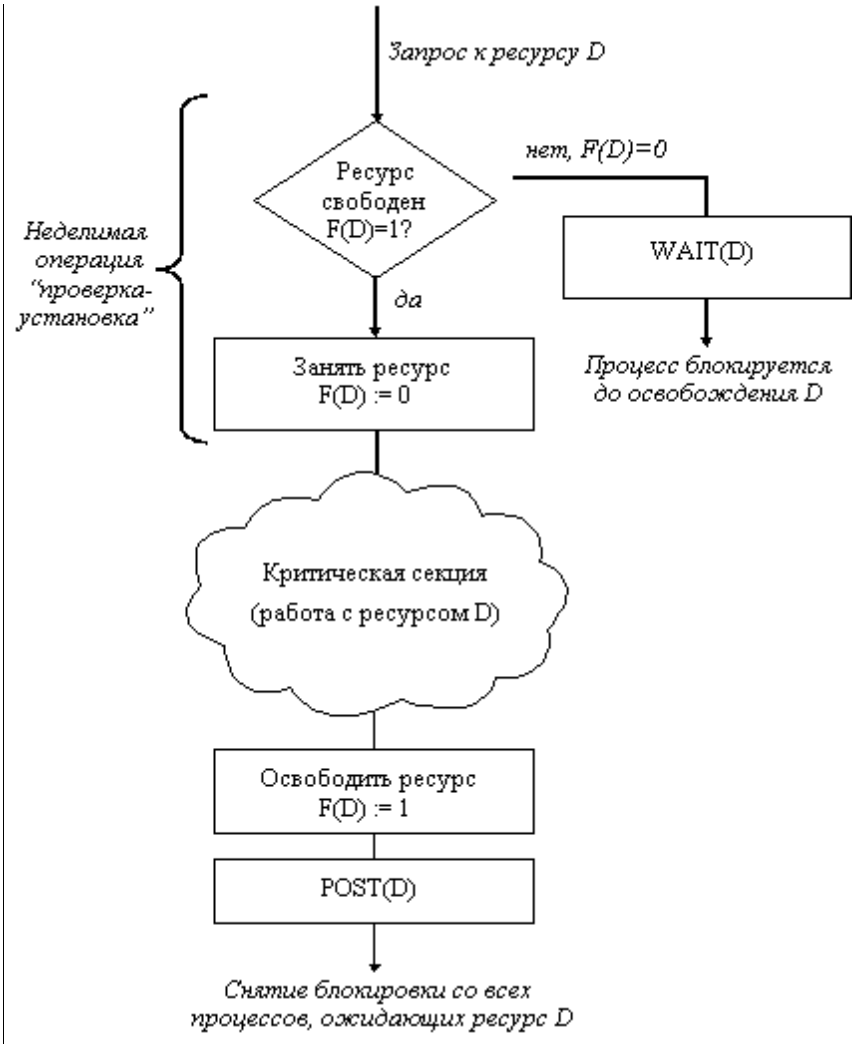
♦ Критическая секция

• ***Критическая секция*** - это часть программы, в которой осуществляется доступ к разделяемым данным.
• ***Взаимное исключение*** – прием, при котором в каждый момент в критической секции, связанной с этим ресурсом, находится максимум один процесс.
• ***Простейший способ обеспечить взаимное исключение*** - позволить процессу, находящемуся в критической секции, запрещать все прерывания.
• Другим способом является использование ***блокирующих переменных***. С каждым разделяемым ресурсом связывается двоичная переменная, которая принимает значение 1, если ресурс свободен, и значение 0, если ресурс занят.



Реализация критических секций с использованием блокирующих переменных

• Реализация критических секций с использованием блокирующих переменных имеет существенный недостаток: в течение времени, когда один процесс находится в критической секции, другой процесс, которому требуется тот же ресурс, будет выполнять рутинные действия по опросу блокирующей переменной, бесполезно тратя процессорное время. Для устранения таких ситуаций может быть использован так называемый аппарат событий. С помощью этого средства могут решаться не только проблемы взаимного исключения, но и более общие задачи синхронизации процессов. В разных операционных системах аппарат событий реализуется по своему, но в любом случае используются системные функции аналогичного назначения, которые условно назовем ***WAIT(x)*** и ***POST(x)***, где x - идентификатор некоторого события. На рисунке ниже показан фрагмент алгоритма процесса, использующего эти функции.



Реализация критической секции с использованием системных функций WAIT(D) и POST(D)

• ***Обобщающее средство синхронизации процессов*** предложил Дейкстра, который ввел два новых примитива. В абстрактной форме эти примитивы, обозначаемые P и V, оперируют над целыми неотрицательными переменными, называемыми ***семафорами***. Пусть S такой семафор. Операции определяются следующим образом:

V(S): переменная S увеличивается на 1 одним неделимым действием; выборка, инкремент и запоминание не могут быть прерваны, и к S нет доступа другим процессам во время выполнения этой операции.

P(S): уменьшение S на 1, если это возможно. Если S=0, то невозможно уменьшить S и остаться в области целых неотрицательных значений, в этом случае процесс, вызывающий P-операцию, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также является неделимой операцией.

♦ Мьютексы

• Мьютекс – упрощенная версия семафора.
• Мьютекс – не способен считать, что он может лишь управлять взаимным исключением доступа к совместно используемым ресурсам.
• Используются в пространстве потоков.
• Мьютекс – переменная, которая может находится в одном из двух состояний: блокированном или неблокированном.
• Для его описания требуется всего один бит. 0 – неблокированное состояние.
• Если Мьютекс не заблокирован, то вход в критическую секцию разрешен и вызывающий поток может попасть в критическую секцию.

♦ Тупики

• Проблема синхронизации - ***взаимные блокировки*** - ***тупики***.
• Проблема тупиков включает в себя следующие задачи:
- предотвращение тупиков,
- распознавание тупиков,
- восстановление системы после тупиков.

• Тупики могут быть предотвращены на стадии написания программ, то есть программы должны быть написаны таким образом, чтобы тупик не мог возникнуть ни при каком соотношении взаимных скоростей процессов.
• Второй подход к предотвращению тупиков называется динамическим и заключается в использовании определенных правил при назначении ресурсов процессам, например, ресурсы могут выделяться в определенной последовательности, общей для всех процессов.
• ***Монитор*** - это набор процедур, переменных и структур данных.
• Процессы могут вызывать процедуры монитора, но не имеют доступа к внутренним данным монитора.
• Только один процесс может быть активным по отношению к монитору.

Билет № 6

Нити.

♦ Нити

• Современные ОС предлагают использовать сравнительно новый механизм - механизм ***многонитевой обработки***. При этом вводится новое понятие «нить».
• Мультипрограммирование теперь реализуется на уровне нитей, и задача, оформленная в виде нескольких нитей в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного выполнения ее отдельных частей.
• Нити могут находится в одном из следующих состояний: ***ВЫПОЛНЕНИЕ***, ***ОЖИДАНИЕ*** и ***ГОТОВНОСТЬ***.
• Итак, нити имеют собственные:

- программный счетчик,
- стек,
- регистры,
- нити-потомки,
- состояние.

Нити разделяют:

- адресное пространство,
- глобальные переменные,
- открытые файлы,
- таймеры,
- семафоры,
- статистическую информацию.

• Наконец, в мультипроцессорных системах для нитей из одного адресного пространства имеется возможность выполняться параллельно на разных процессорах.

Билет № 7

Управление памятью ЭВМ с использованием дискового пространства.

♦ Типы адресов

• Для идентификации переменных и команд используются ***символьные имена*** (метки), ***виртуальные адреса*** и ***физические адреса***.
• Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык.
• Каждый процесс имеет собственное виртуальное адресное пространство.

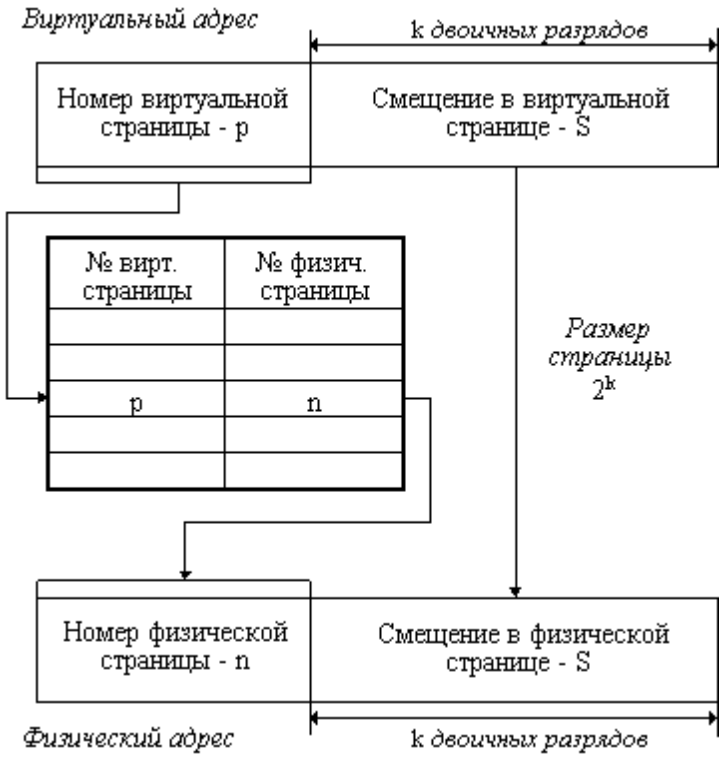
• ***Переход от виртуальных адресов к физическим*** может осуществляться двумя способами:
  • Замену виртуальных адресов на физические делает специальная системная программа - ***перемещающий загрузчик***.
  • Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический.

♦ Понятие виртуальной памяти

• ***Оверлеи***.
• Виртуальным называется ресурс, который предоставляется пользователю или пользовательской программе, которым он в действительности не обладает.
• ***Виртуальная память*** - это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память
• Задачи виртуальной

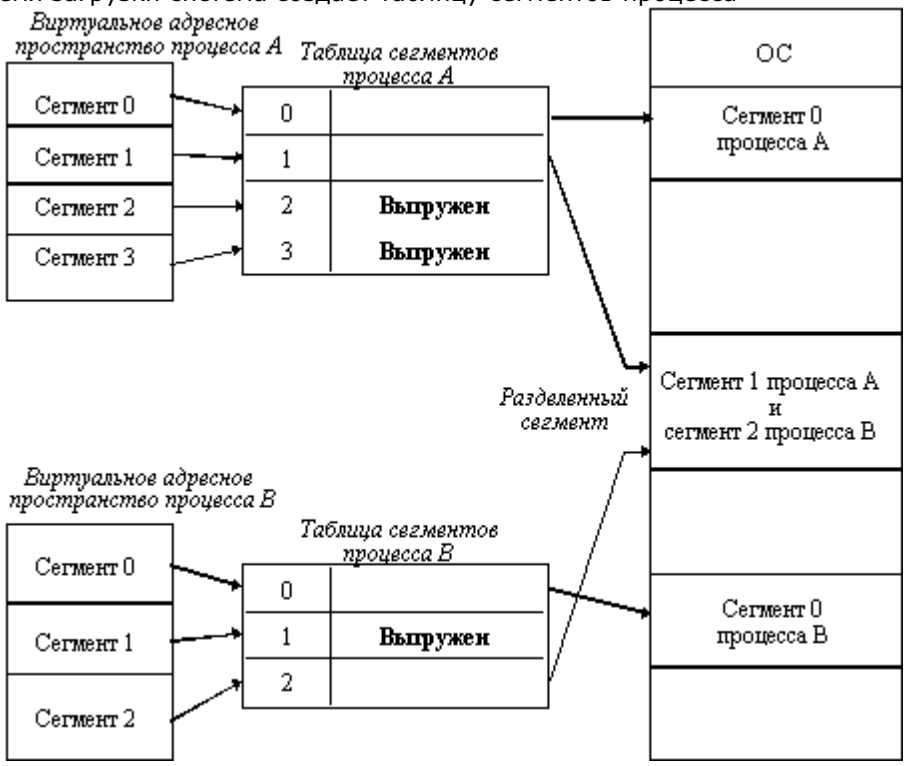
♦ Страничное распределение

• ***Виртуальное адресное пространство каждого процесса*** делится на части одинакового, фиксированного для данной системы размера, называемые ***виртуальными страницами***.
• Вся оперативная память машины также делится на части такого же размера, называемые ***физическими страницами***.
• Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д.
• При загрузке процесса часть его виртуальных страниц помещается в оперативную память, а остальные - на диск.
• Механизм преобразования виртуального адреса в физический при страничной организации памяти:



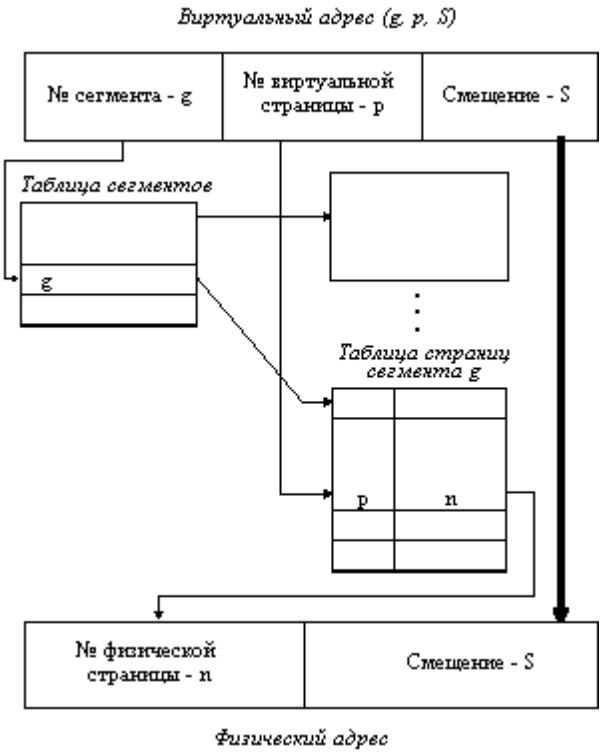
♦ Сегментное распределение

• Виртуальное адресное пространство процесса делится на сегменты, размер которых определяется программистом с учетом смыслового значения содержащейся в них информации.
• При загрузке процесса часть сегментов помещается в оперативную память, а часть сегментов размещается в дисковой памяти.
• Во время загрузки система создает таблицу сегментов процесса



♦ Странично-сегментное распределение

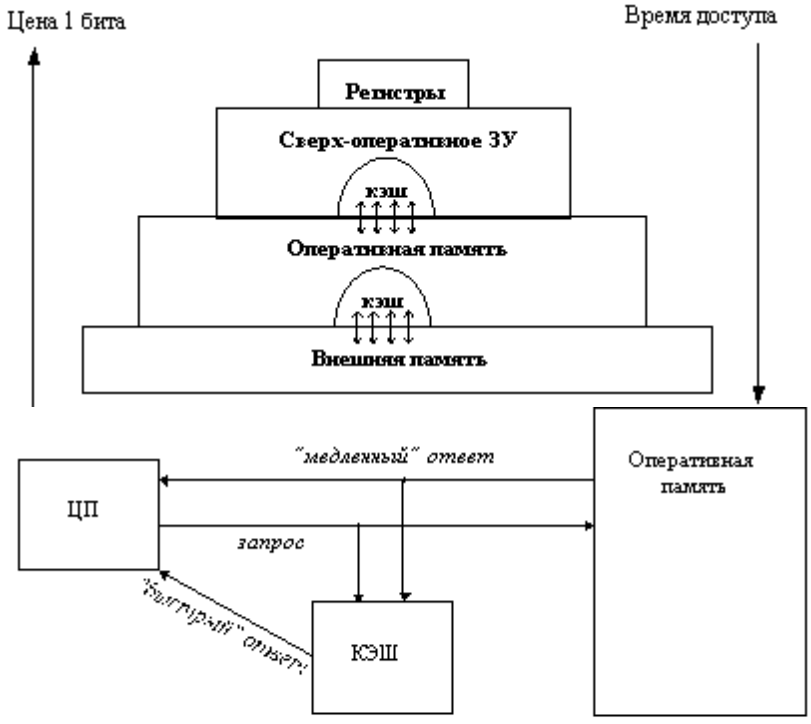
• Виртуальное пространство процесса делится на сегменты, а каждый сегмент в свою очередь делится на виртуальные страницы, которые нумеруются в пределах сегмента.
• Оперативная память делится на физические страницы.
• Загрузка процесса выполняется операционной системой постранично, при этом часть страниц размещается в оперативной памяти, а часть на диске.



♦ Свопинг

При свопинге, в отличие от рассмотренных ранее методов реализации виртуальной памяти, процесс перемещается между памятью и диском целиком.

♦ Иерархия запоминающих устройств. Принцип кэширования данных



Структура кэш-памяти

Адрес данных в ОП	Данные	Управл. информация	
		бит модиф.	бит обращ.

Билет № 8

Управление памятью ЭВМ без использования дискового пространства.

♣ Типы адресов

• Для идентификации переменных и команд используются ***символьные имена*** (метки), ***виртуальные адреса*** и ***физические адреса***.
• Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык.
• Каждый процесс имеет собственное виртуальное адресное пространство.

Типы адресов

• ***Переход от виртуальных адресов к физическим*** может осуществляться двумя способами:
  • Замену виртуальных адресов на физические делает специальная системная программа - ***перемещающий загрузчик***.
  • Во время выполнения программы при каждом обращении к оперативной памяти выполняется преобразование виртуального адреса в физический.

♣ Методы распределения памяти без дискового пространства

♦ Распределение памяти фиксированными разделами

♦ Распределение памяти разделами переменной величины

• В этом случае память машины не делится заранее на разделы. Сначала вся память свободна.
• Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди.
• Программный код не перемещается во время выполнения, то есть может быть проведена единовременная настройка адресов посредством использования перемещающего загрузчика.
• Недостаток - *фрагментация памяти*.

♦ Перемещаемые разделы

• Перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область
• Процедура «сжатия».

Билет № 9

Управление вводом-выводом.

♦ Физическая организация устройств ввода-вывода.

• ***Блок-ориентированные*** устройства ввода-вывода
• ***Байт-ориентированные*** устройства ввода-вывода
• Внешнее устройство обычно состоит из ***механического*** и ***электронного*** компонента. Электронный компонент называется контроллером устройства или адаптером. Механический компонент представляет собственно устройство.
• Операционная система обычно имеет дело не с устройством, а с контроллером.
• ОС выполняет ввод-вывод, записывая команды в регистры контроллера.

♦ Организация программного обеспечения ввода-вывода

• ОС выполняет операции ввода-вывода асинхронно, но представляет их для пользовательских программ в синхронной форме.
• Одни устройства являются ***разделяемыми***, а другие - ***выделенными***.
• Программное обеспечение ввода-вывода делится на четыре слоя:
- ***Обработка прерываний***,
- ***Драйверы устройств***,
- ***Независимый от устройств слой операционной системы***,
- ***Пользовательский слой программного обеспечения***.

♦ Обработка прерываний

• Прерывания должны быть скрыты как можно глубже в недрах операционной системы, чтобы как можно меньшая часть ОС имела с ними дело.
• Эффект от прерывания будет состоять в том, что ранее заблокированный процесс теперь продолжит свое выполнение.

♦ Драйверы устройств

• Весь зависимый от устройства код помещается в драйвер устройства.
• Драйвер должен решить, какие операции контроллера нужно выполнить и в какой последовательности.
• После передачи команды контроллеру драйвер должен решить, блокировать ли себя до окончания заданной операции или нет.

♦ Независимый от устройств слой операционной системы

• Типичными функциями для независимого от устройств слоя являются:
- обеспечение общего интерфейса к драйверам устройств,
- именование устройств,
- защита устройств,
- обеспечение независимого размера блока,
- буферизация,
- распределение памяти на блок-ориентированных устройствах,
- распределение и освобождение выделенных устройств,
- уведомление об ошибках.

♦ Пользовательский слой программного обеспечения

• Большая часть программного обеспечения ввода-вывода находится внутри ОС
• Некоторая его часть содержится в библиотеках, связываемых с пользовательскими программами.
• ***Подсистема спулинг*** - категория программного обеспечения ввода-вывода.
• ***Спулинг*** - это способ работы с выделенными устройствами в мультипрограммной системе.
• ***Каталогом спулинга***.
• ***Процесс-монитор*** по очереди распечатывает все файлы, содержащиеся в каталоге спулинга.

Билет № 10

Файловые системы.

♦ Файловая система

• ***Файловая система*** - часть операционной системы
• Понятие «файловая система» включает:
- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске,
- комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

♦ Имена файлов

♦ Типы файлов

• Файлы бывают разных типов: ***обычные файлы***, ***специальные файлы***, ***файлы-каталоги***.
• Обычные файлы в свою очередь подразделяются на ***текстовые*** и ***двоичные***.
• ***Специальные файлы*** - это файлы, ассоциированные с устройствами ввода-вывода, которые позволяют пользователю выполнять операции ввода-вывода, используя обычные команды записи в файл или чтения из файла.
• ***Специальные файлы***, так же как и устройства ввода-вывода, делятся на блок-ориентированные и байт-ориентированные.
• Иерархия каталогов может быть ***деревом*** или ***сетью***.

♦ Логическая организация файла

• Программист имеет дело с логической организацией файла, представляя файл в виде определенным образом организованных логических записей.
• Логическая запись - это наименьший элемент данных, которым может оперировать программист при обмене с внешним устройством.

♦ Физическая организация и адрес файла

• Физическая организация файла описывает правила расположения файла на устройстве внешней памяти, в частности на диске.
• Файл состоит из физических записей - блоков. Блок - наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью.

♦ Права доступа к файлу

• ***Матрица прав доступа***
• Различают два основных ***подхода к определению прав доступа***:
- избирательный доступ, когда для каждого файла и каждого пользователя сам владелец может определить допустимые операции;
- мандатный подход, когда система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен.

♦ Кэширование диска

• Программный слой - подсистема буферизации.
• При поступлении запроса на чтение некоторого блока подсистема буферизации просматривает свой буферный пул и, если находит требуемый блок, то копирует его в буфер запрашивающего процесса.
• Подсистема буферизации работает по принципу кэш-памяти.

♦ Общая модель файловой системы

• Функционирование любой файловой системы можно представить многоуровневой моделью:

♦ **Отображаемые в память файлы**

• Отображение файлов в адресное пространство выполняемого процесса.
• MAP (отобразить) и UNMAP (отменить отображение).
• При завершении процесса все отображенные и модифицированные страницы переписываются из памяти в файл.
• ***Проблемы отображения файлов***:
  • Для системы сложно узнать точную длину выходного файла
  • Если один процесс отображает файл, а другой процесс открывает его для обычного файлового доступа.
  • Третья проблема состоит в том, что файл может быть больше, чем сегмент, и даже больше, чем все виртуальное адресное пространство.

♦ Современные архитектуры файловых систем

• Разработчики новых операционных систем стремятся обеспечить пользователя возможностью работать сразу с несколькими файловыми системами.
• Новая файловая система имеет многоуровневую структуру, на верхнем уровне которой располагается так называемый ***переключатель файловых систем*.**
• Каждый компонент уровня файловых систем выполнен в виде ***драйвера соответствующей файловой системы*** и поддерживает определенную организацию файловой системы.
• Драйверы файловых систем обращаются к подсистеме ввода-вывода.
• ***Подсистема ввода вывода*** - это составная часть файловой системы, которая отвечает за загрузку, инициализацию и управление всеми модулями низших уровней файловой системы.
• Драйвер может получить управление на любом этапе выполнения запроса - от вызова приложением функции, которая занимается работой с файлами, до того момента, когда работающий на самом низком уровне драйвер устройства начинает просматривать регистры контроллера.
• Многоуровневый механизм работы файловой системы реализован посредством ***цепочек вызова***.

Билет № 11

Управление распределенными ресурсами

♦ **Базовые примитивы передачи сообщений в распределенных системах**

• Единственным отличием распределенных систем от централизованных является ***межпроцессная взаимосвязь***.
• Основой взаимодействия между машинами может служить только передача по сети ***сообщений*** – примитивы ***ПОСЛАТЬ*** и ***ПОЛУЧИТЬ***.

♦ **Способы адресации**

• Для того чтобы послать сообщение, необходимо указать адрес получателя.
• Одним из вариантов адресации на верхнем уровне является использование физических адресов сетевых адаптеров.
• Если в получающем компьютере выполняется только один процесс, то ядро будет знать, что делать с поступившим сообщением - передать его этому процессу.
• Альтернативная адресная система использует имена назначения, состоящие из двух частей, определяющие ***номер машины*** и ***номер процесса***.
• Другим вариантом могло бы быть назначение каждому процессу ***уникального адреса, который никак не связан с адресом машины***.
• Одним из способов достижения этой цели является использование ***централизованного механизма распределения адресов процессов***, который работает просто, как счетчик.
• Еще один метод назначения процессам уникальных идентификаторов заключается в разрешении каждому процессу выбора своего ***собственного идентификатора из очень большого адресного пространства***, такого как пространство 64-х битных целых чисел. В сети, которая поддерживает широковещательный режим, отправитель может широковещательно передать специальный пакет, который содержит идентификатор процесса назначения. Все ядра получат эти сообщения.
• Специальную машину для отображения высокоуровневых символьных имен – ***DNS сервер***.
• Совершенно иной подход - это использование специальной аппаратуры. Пусть процессы выбирают свои адреса случайно, а конструкция сетевых адаптеров позволяет хранить эти адреса. Теперь адреса процессов не обнаруживаются путем широковещательной передачи, а непосредственно указываются в кадрах, заменяя там адреса сетевых адаптеров.

♦ **Блокирующие и неблокирующие примитивы**

• Примитивы соответственно ***синхронными*** и ***асинхронными***.
• При использовании блокирующего примитива, процесс, выдавший запрос на его выполнение, приостанавливается до полного завершения примитива.
• При использовании неблокирующего примитива управление возвращается вызывающему процессу немедленно, еще до того, как требуемая работа будет выполнена.
• Решение проблемы неблокирующих примитивов:
1. Заставить ядро копировать сообщение в свой внутренний буфер, а затем разрешить процессу продолжить выполнение.
2. Второе решение заключается в прерывании процесса-отправителя после отправки сообщения, чтобы проинформировать его, что буфер снова доступен.

• Вопрос ***тайм-аутов*** – вечная блокировка примитива.

♦ **Буферизуемые и небуферизуемые примитивы**

• ***Небуферизуемые примитивы*** - вызов ПОЛУЧИТЬ сообщает ядру машины, на которой он выполняется, адрес буфера, в который следует поместить пребывающее для него сообщение.
• Что раньше – вызов ПОЛУЧИТЬ или вызов ПОСЛАТЬ. Решение проблемы:
1. Просто отказаться от сообщения, позволить отправителю взять тайм-аут и надеяться, что получатель все-таки выполнит вызов ПОЛУЧИТЬ перед повторной передачей сообщения.
2. Второй подход к этой проблеме заключается в том, чтобы хранить хотя бы некоторое время, поступающие сообщения в ядре получателя на тот случай, что вскоре будет выполнен соответствующий вызов ПОЛУЧИТЬ.

• Простым способом управления буферами является определение новой структуры данных, называемой ***почтовым ящиком -*** буферизуемый примитив.

♦ **Надежные и ненадежные примитивы**

• Реально сообщения могут теряться.
• Для решения этой проблемы существует три подхода:
1. Первый заключается в том, что система не берет на себя никаких обязательств по поводу доставки сообщений.
2. Второй подход заключается в том, что ядро принимающей машины посылает ***квитанцию-подтверждение*** ядру отправляющей машины на каждое сообщение.
3. Третий подход заключается в использовании ответа в качестве подтверждения в тех системах, в которых запрос всегда сопровождается ответом.

Билет № 12

Вызов удаленных процедур (RPC)

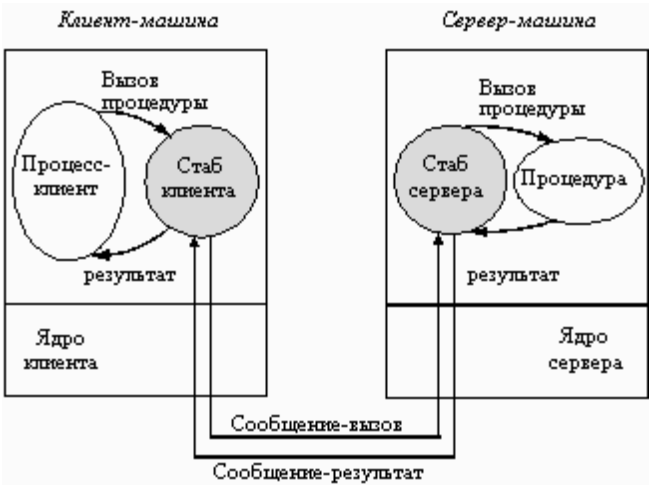
♦ Концепция удаленного вызова процедур

• Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются ***RPC-ориентированными***.
• Параметры RPC не должны содержать указателей на ячейки нестековой памяти.
• RPC обязательно использует нижележащую систему связи.
• В реализации RPC участвуют как минимум два процесса - по одному в каждой машине.
• Неоднородностью языков программирования и операционных сред.

♦ Базовые операции RPC

• Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности также как и вызов локальной процедуры.
• Когда вызываемая процедура действительно является удаленной, в библиотеку помещается вместо локальной процедуры другая версия процедуры, называемая ***клиентским стабом***. Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности, так же происходит прерывание при обращении к ядру. Только в отличие от оригинальной процедуры он не помещает параметры в регистры и не запрашивает у ядра данные, вместо этого он формирует сообщение для отправки ядру удаленной машины.

♦ Этапы выполнения RPC



Remote Procedure Call

Этапы выполнения процедуры RPC

♦ Динамическое связывание

• Начальным моментом для динамического связывания является ***формальное определение сервера***.
• Спецификация содержит имя файл-сервера, номер версии и список процедур-услуг, предоставляемых данным сервером для клиентов.
• Формальная спецификация сервера используется в качестве исходных данных для программы-генератора стабов, которая создает как клиентские, так и серверные стабы. Когда клиентская программа вызывает любую процедуру, определенную в спецификации сервера, соответствующая стаб-процедура связывается с двоичным кодом программы.
• При запуске сервера самым первым его действием является передача своего серверного интерфейса специальной программе, называемой ***binder***'ом. Этот процесс, известный как процесс регистрации сервера.
• Когда клиент вызывает одну из удаленных процедур первый раз клиентский стаб видит, что он еще не подсоединен к серверу, и посылает сообщение binder-программе с просьбой об импорте интерфейса нужной версии нужного сервера. Если такой сервер существует, то binder передает описатель и уникальный идентификатор клиентскому стабу.

♦ Семантика RPC в случае отказов

Классы отказов:

1. Клиент не может определить местонахождения сервера.
2. Потерян запрос от клиента к серверу. Самое простое решение - через определенное время повторить запрос.
3. Потеряно ответное сообщение от сервера клиенту.
4. Сервер потерпел аварию после получения запроса. Существует три подхода к этой проблеме:
   • Ждать до тех пор, пока сервер не перезагрузится и пытаться выполнить операцию снова.
   • Сразу сообщить приложению об ошибке.
   • Когда сервер отказывает, клиенту не оказывается никакой поддержки.
5. Клиент потерпел аварию после отсылки запроса. В этом случае выполняются вычисления результатов, которых никто не ожидает. Такие вычисления называют ***«сиротами»***.
   Как поступать с сиротами? Рассмотрим 4 возможных решения:
   - ***Уничтожение***. До того, как клиентский стаб посылает RPC-сообщение, он делает отметку в журнале, оповещая о том, что он будет сейчас делать. Журнал хранится на диске или в другой памяти, устойчивой к сбоям. После аварии система перезагружается, журнал анализируется и сироты ликвидируются.
   - ***Перевоплощение***. В этом случае все проблемы решаются без использования записи на диск. Метод состоит в делении времени на последовательно пронумерованные периоды. Когда клиент перезагружается, он передает широковещательное сообщение всем машинам о начале нового периода. После приема этого сообщения все удаленные вычисления ликвидируются. Конечно, если сеть сегментированная, то некоторые сироты могут и уцелеть.
   - ***Мягкое перевоплощение*** аналогично предыдущему случаю, за исключением того, что отыскиваются и уничтожаются не все удаленные вычисления, а только вычисления перезагружающегося клиента.
   - ***Истечение срока***. Каждому запросу отводится стандартный отрезок времени T, в течение которого он должен быть выполнен. Если запрос не выполняется за отведенное время, то выделяется дополнительный квант.

Билет № 13

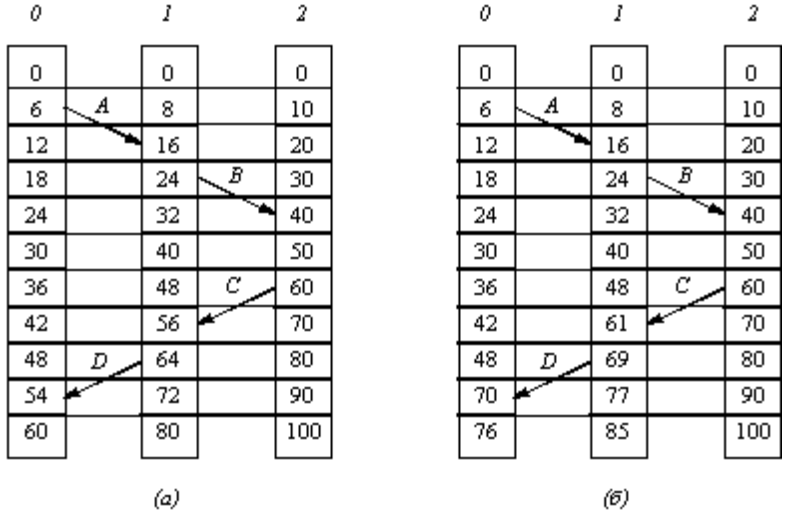
Алгоритмы синхронизации в одиночных и распределенных системах

♦ Синхронизация в распределенных системах

• Синхронизация необходима процессам для организации совместного использования ресурсов, таких как файлы или устройства, а также для обмена данными.

♦ Алгоритм синхронизации логических часов

• Процессам не нужно, чтобы во всех машинах было правильное время, для них важно, чтобы оно было везде одинаковое, более того, для некоторых процессов важен только правильный порядок событий. В этом случае мы имеем дело с ***логическими часами***.
• Отношение «случилось до» для двух произвольных событий: a ® b читается «a случилось до b».



Синхронизация логических часов
а - три процесса, каждый со своими собственными часами;
б - алгоритм синхронизации логических часов

• Следовательно, каждое сообщение должно нести с собой время своего отправления по часам машины-отправителя. Если в машине, получившей сообщение, часы показывают время, которое меньше времени отправления, то эти часы переводятся вперед, так, чтобы они показали время, большее времени отправления сообщения.

♦ Алгоритмы взаимного исключения

Если процесс уверен, что никакой процесс не будет иметь доступа к этому ресурсу одновременно с ним, то это называется взаимным исключением.

♦ Централизованный алгоритм

• Один из процессов выбирается в качестве ***координатора*** (например, процесс, выполняющийся на машине, имеющей наибольшее значение сетевого адреса).
• Когда какой-либо процесс хочет войти в критическую секцию, он посылает сообщение с запросом к координатору, оповещая его о том, в какую критическую секцию он хочет войти, и ждет от координатора разрешение.

♦ Распределенный алгоритм

• Когда процесс хочет войти в критическую секцию, он формирует сообщение, содержащее имя нужной ему критической секции, номер процесса и текущее значение времени.
• Затем он посылает это сообщение всем другим процессам.
• Когда процесс получает сообщение такого рода, его действия зависят от того, в каком состоянии по отношению к указанной в сообщении критической секции он находится. Имеют место три ситуации:

1. Если получатель не находится и не собирается входить в критическую секцию в данный момент, то он отсылает назад процессу-отправителю сообщение с разрешением.
2. Если получатель уже находится в критической секции, то он не отправляет никакого ответа, а ставит запрос в очередь.
3. Если получатель хочет войти в критическую секцию, но еще не сделал этого, то он сравнивает временную отметку поступившего сообщения со значением времени, которое содержится в его собственном сообщении, разосланном всем другим процессам. Если время в поступившем к нему

сообщении меньше, то есть его собственный запрос возник позже, то он посылает сообщение-разрешение, в обратном случае он не посылает ничего и ставит поступившее сообщение-запрос в очередь.

• Процесс может войти в критическую секцию только в том случае, если он получил ответные сообщения-разрешения от всех остальных процессов.

♦ Алгоритм Token Ring

• Все процессы системы образуют логическое кольцо.
• Каждый процесс знает номер своей позиции в кольце, а также номер ближайшего к нему следующего процесса.
• Когда кольцо инициализируется, процессу 0 передается ***токен***. Токен циркулирует по кольцу. Он переходит от процесса n к процессу n+1 путем передачи сообщения по типу «точка-точка».
• Когда процесс получает токен от своего соседа, он анализирует, не требуется ли ему самому войти в критическую секцию. Если да, то процесс входит в критическую секцию. После того, как процесс выйдет из критической секции, он передает токен дальше по кольцу.

♦ Неделимые транзакции

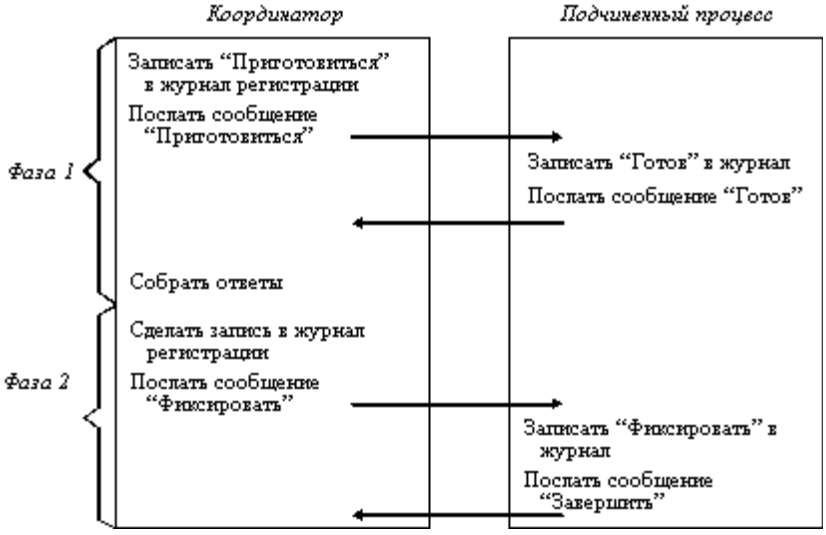
• Один процесс объявляет, что он хочет начать транзакцию с одним или более процессами. Они могут некоторое время создавать и уничтожать разные объекты, выполнять какие-либо операции. Затем инициатор объявляет, что он хочет завершить транзакцию. Если все с ним соглашаются, то результат фиксируется. Если один или более процессов отказываются, тогда измененные объекты возвращается точно к тому состоянию, в котором они находились до начала выполнения транзакции. Это свойство ***«все-или-ничего».***
• Примитивы транзакции:

BEGIN_TRANSACTION	команды, которые следуют за этим примитивом, формируют транзакцию.
END_TRANSACTION	завершает транзакцию и пытается зафиксировать ее.
ABORT_TRANSACTION	прерывает транзакцию, восстанавливает предыдущие значения.
READ	читает данные из файла (или другого объекта)
WRITE	пишет данные в файл (или другой объект).

• Транзакции обладают следующими свойствами: ***упорядочиваемостью***, ***неделимостью***, ***постоянством***.
• ***Упорядочиваемость*** гарантирует, что если две или более транзакции выполняются в одно и то же время, то конечный результат выглядит так, как если бы все транзакции выполнялись последовательно в некотором порядке.
• ***Неделимость*** означает, что когда транзакция находится в процессе выполнения, то никакой другой процесс не видит ее промежуточные результаты.
• ***Постоянство*** означает, что после фиксации транзакции никакой сбой не может отменить результатов ее выполнения.
• Вопросы реализации транзакций:

1. В соответствии с первым подходом, когда процесс начинает транзакцию, то он работает в ***индивидуальном рабочем пространстве***, содержащем все файлы и другие объекты, к которым он имеет доступ.
2. Второй общий подход к реализации механизма транзакций называется ***списком намерений***. Этот метод заключается в том, что модифицируются сами файлы, а не их копии, но перед изменением любого блока производится запись в специальный файл - журнал регистрации.

• ***Двухфазный протокол фиксации транзакции***



Билет № 14

Нити в распределенных системах

♦ Понятие «нить»

• На многопроцессорной системе нити действительно выполняются параллельно. Нити могут, например, порождать нити-потомки, могут переходить в состояние ожидания до завершения системного вызова, как обычные процессы, пока одна нить заблокирована, другая нить того же процесса может выполняться.
• Нити делают возможным сохранение идеи последовательных процессов, которые выполняют блокирующие системные вызовы и в то же время позволяют достичь параллелизма вычислений.

♦ Различные способы организации вычислительного процесса с использованием нитей

• ***Менеджер-планировщик***.
• В модели ***«команда»*** все нити эквивалентны, каждая получает и обрабатывает свои собственные запросы. Иногда работы приходят, а нужная нить занята. В этом случае может создаваться ***очередь незавершенных работ***.
• Нити могут быть также организованы в виде ***конвейера***. В этом случае первая нить порождает некоторые данные и передает их для обработки следующей нити и т.д.
• Нити часто полезны и для клиентов.
• Использование нитей может сократить необходимое количество прерываний пользовательского уровня.
• В многопроцессорных системах нити из одного адресного пространства могут выполняться параллельно на разных процессорах.

♦ Вопросы реализации нитей

• Существует два подхода к управлению нитями: ***статический*** и ***динамический***.
• При ***статическом подходе*** вопрос, сколько будет нитей, решается уже на стадии написания программы или на стадии компиляции. Каждой нити назначается фиксированный стек.
• Более общим является ***динамический подход***, который позволяет создавать и удалять нити оперативно по ходу выполнения. Системный вызов для создания нити обычно содержится в нити главной программы в виде указателя на процедуру с указанием размера стека, а также других параметров.
• Завершаться нити могут одним из двух способов: по своей инициативе, когда завершается работа, и извне.
• Нити могут быть реализованы как в ***пользовательском пространстве*** - для каждого процесса можно организовать свою схему планирования, так и в ***пространстве ядра*** - ядро может при диспетчеризации выбирать нить из другого процесса; при переключение нитей тратится время на переключение из режима пользователя в режим ядра.

♦ Нити и RPC

• Обычно в распределенных системах используются как ***RPC***, так и ***нити***.
• ***Локальный вызов RPC***. Когда стартует серверная нить S, то она экспортирует свой интерфейс, сообщая о нем ядру. Интерфейс определяет, какие процедуры могут быть вызваны, каковы их параметры и т.п. Когда стартует клиентская нить C, то она импортирует интерфейс из ядра в том случае, если собирается вызвать S, и ей дается специальный идентификатор для выполнения определенного вызова. Ядро теперь знает, что C собирается позже вызвать S и создает специальные структуры данных для подготовки к вызову.
• Другой прием широко используется для ускорения ***удаленных RPC***. Идея основана на следующем наблюдении: когда нить-сервер блокируется, ожидая нового запроса, ее контекст почти всегда не содержит важной информации. Следовательно, когда нить завершает обработку запроса, то ее просто удаляют. Эту схему иногда называют ***неявным вызовом***.
• Этот метод имеет несколько преимуществ по сравнению с обычным RPC:

1. Нити не должны блокироваться, ожидая новую работу, следовательно контекст не нужно сохранять,
2. Создание новой нити проще, чем активизация существующей приостановленной, так как не нужно восстанавливать контекст.

Билет № 15

Распределенные файловые системы. Основные алгоритмы.

♦ Распределенные файловые системы

• Ключевым компонентом любой распределенной системы является файловая система.
• Файловая система поддерживается одной или более машинами, называемыми ***файл-серверами***.
• Рабочая станция может подсоединять и монтировать эти файловые системы к своим локальным файловым системам. При этом монтируемые файловые системы остаются на серверах.
• ***Файловый сервис*** - это описание функций, которые файловая система предлагает своим пользователям. Файловый сервис определяет интерфейс файловой системы с клиентами.
• ***Файловый сервер*** - это процесс, который выполняется на отдельной машине и помогает реализовывать файловый сервис.
• Так как обычно файловый сервер - это просто пользовательский процесс, выполняющийся на некоторой машине, в системе может быть несколько файловых серверов, каждый из которых предлагает различный файловый сервис.
• Файловый сервис в распределенных файловых системах имеет две функционально различные части: ***собственно файловый сервис -*** имеет дело с операциями над отдельными файлами и ***сервис каталогов -*** с созданием каталогов и управлением ими.

♦ Интерфейс файлового сервиса.

• Большинство современных распределенных файловых систем поддерживают определение файла как последовательности байтов.
• Файл характеризуется атрибутами.
• Важным аспектом файловой модели является возможность модификации файла после его создания. Обычно файлы могут модифицироваться, но в некоторых распределенных системах единственными операциями с файлами являются СОЗДАТЬ и ПРОЧИТАТЬ.
• Файловый сервис может быть разделен на два типа в зависимости от того, поддерживает ли он ***модель загрузки-выгрузки*** или ***модель удаленного доступа***.
• В *модели загрузки-выгрузки* пользователю предлагаются средства чтения или записи файла целиком.
• Другой тип файлового сервиса соответствует ***модели удаленного доступа***, которая предполагает поддержку большого количества операций над файлами: открытие и закрытие файлов, чтение и запись частей файла, позиционирование в файле, проверка и изменение атрибутов файла и так далее.
• в случае ***модели удаленного доступа*** вся файловая система выполняется на серверах, а не на клиентских машинах.

♦ Интерфейс сервиса каталогов

• Природа сервиса каталогов не зависит от типа используемой модели файлового сервиса.
• В распределенных системах используются многоуровневая организация каталогов.
• ***Прозрачность расположения*** - означает, что имена не дают возможности определить месторасположение файла.
• Система, в которой файлы могут перемещаться без изменения имен, обладает свойством ***независимости от расположения***. Распределенная система, которая включает имена серверов или машин непосредственно в имена файлов, не является независимой от расположения.
• Большинство распределенных систем используют какую-либо форму двухуровневого именования: на одном уровне файлы имеют ***символические имена***, предназначенные для использования людьми, а на другом - внутренние, ***двоичные имена***, для использования самой системой.
• Отличием распределенных систем от централизованных является возможность соответствия одному символьному имени нескольких двоичных имен. Имея несколько двоичных имен, можно при недоступности одной из копий файла получить доступ к другой. Этот метод обеспечивает ***отказоустойчивость за счет избыточности.***

♦ Семантика разделения файлов

• Система придерживается абсолютного временного упорядочивания всех операций, и всегда возвращает самое последнее значение - модель ***семантики UNIX'а***.
• Решение проблемы ***семантики UNIX'а*** - немедленный возврат всех изменений в кэшированном файле клиента на сервер.
• Другим решением является введение так называемой ***сессионной семантики***, в соответствии с которой изменения в открытом файле сначала видны только процессу, который модифицирует файл, и только после закрытия файла эти изменения могут видеть другие процессы.
• Правило ***сессионной семантики***, в соответствии с которым окончательным является тот вариант, который был закрыт последним.
• Решение - сделать ***все файлы неизменяемыми***. Тогда файл нельзя открыть для записи, а можно выполнять только операции СОЗДАТЬ и ЧИТАТЬ.
• Решение - ***механизм неделимых транзакций -*** все изменения делаются по принципу «все или ничего».

♦ Вопросы разработки структуры файловой системы

• В некоторых системах (например, NFS) нет разницы между клиентом и сервером, на всех машинах работает одно и то же базовое программное обеспечение.
• В других системах файловый сервер - это только пользовательская программа, так что система может быть сконфигурирована как клиент, как сервер или как клиент и сервер одновременно.
• Система, в которой клиенты и серверы - это принципиально различные машины, как в терминах аппаратуры, так и в терминах программного обеспечения.
• ***Структуризация*** сервиса файлов и каталогов:

1. Один подход заключается в комбинировании этих двух сервисов на одном сервере. При другом подходе эти
2. Сервисы разделяются - при открытии файла требуется обращение к серверу каталогов, который отображает символьное имя в двоичное, а затем обращение к файловому серверу с двоичным именем для действительного чтения или записи файла.

• Последний рассматриваемый здесь структурный вопрос связан с хранением на серверах информации о состоянии клиентов. Существует две конкурирующие точки зрения:

1. Сервер не должен хранить такую информацию (сервер stateless).
   - отказоустойчивы;
   - не нужны вызовы OPEN/CLOSE;
   - меньше памяти сервера расходуется на таблицы;
   - нет ограничений на число открытых файлов;
   - отказ клиента не создает проблем для сервера.

2. Сервер должен хранить такую информацию (сервер statefull).
   - более короткие сообщения при запросах;
   - лучше производительность;
   - возможно опережающее чтение;
   - легче достичь идемпотентности;
   - возможна блокировка файлов.

♦ Кэширование

• В системах, состоящих из клиентов и серверов, потенциально имеется четыре различных места для хранения файлов и их частей: диск сервера, память сервера, диск клиента (если имеется) и память клиента.
• Значительное увеличение производительности может быть достигнуто за счет ***кэширования файлов в памяти сервера***. Требуются алгоритмы для определения, какие файлы или их части следует хранить в кэш-памяти.
• Какими ***единицами*** оперирует кэш - дисковые блоки, или целые файлы.
• Необходимо определить ***правило замены*** данных при заполнении кэш-памяти. Здесь можно использовать любой стандартный алгоритм кэширования, например, алгоритм LRU (least recently used), соответствии с которым вытесняется блок, к которому дольше всего не было обращения.
• Кэш-память на сервере легко реализуется и совершенно прозрачна для клиента.
• Кэширование на стороне клиента, которое, однако, порождает много сложностей.
• ***Кэширование файлов непосредственно внутри адресного пространства каждого пользовательского процесса***. Обычно кэш управляется с помощью библиотеки системных вызов. По мере того, как файлы открываются, закрываются, читаются и пишутся, библиотека просто сохраняет наиболее часто используемые файлы. Когда процесс завершается, все модифицированные файлы записываются назад на сервер.
• Другим местом кэширования является ***ядро***. Недостатком этого варианта является то, что во всех случаях требуется выполнять системные вызовы, даже в случае успешного обращения к кэш-памяти (файл оказался в кэше). Но преимуществом является то, что файлы остаются в кэше и после завершения процессов.
• Третьим вариантом организации кэша является создание отдельного процесса пользовательского уровня - ***кэш-менеджера***. Преимущество этого подхода заключается в том, что ядро освобождается от кода файловой системы и тем самым реализуются все достоинства микроядер.
• Четыре алгоритма управления кэшированием обобщаются следующим образом:

1. ***Сквозная запись***. Этот метод эффективен частично, так как уменьшает интенсивность только операций чтения, а интенсивность операций записи остается неизменной. Когда кэшируемый элемент (файл или блок) модифицируется, новое значение записывается в кэш и одновременно посылается на сервер.
2. ***Отложенная запись***. Производительность лучше, но результат чтения кэшированного файла не всегда однозначен. Вместо того, чтобы выполнять запись на сервер, клиент просто помечает, что файл изменен. Примерно каждые 30 секунд все изменения в файлах собираются вместе и отсылаются на сервер за один прием
3. ***«Запись-по-закрытию»***. Удовлетворяет сессионной семантике.
4. ***Централизованное управление***. Ненадежен вследствие своей централизованной природы. Когда файл открыт, машина, открывшая его, посылает сообщение файловому серверу, чтобы оповестить его об этом факте.

♦ Репликация

• ***Репликация*** - это асинхронный перенос изменений данных исходной файловой системы в файловые системы, принадлежащие различным узлам распределенной файловой системы.
• Система оперирует несколькими копиями файлов, причем каждая копия находится на отдельном файловом сервере.
• ***Причины для предоставления этого сервиса***:

1. Увеличение надежности за счет наличия независимых копий каждого файла на разных файл-серверах.
2. Распределение нагрузки между несколькими серверами.

• ***Точная репликация файла*** - программист сам управляет всем процессом репликации.
• При ***точной репликации файла*** сетевые адреса всех копий могут быть ассоциированы с именем файла - когда имя найдено, это означает, что найдены все копии.
• ***Ленивая репликация*** - создается только одна копия каждого файла на некотором сервере. Позже сервер сам автоматически выполнит репликации на другие серверы без участия программиста.
• При ***ленивой репликации*** адресуется один сервер, а не группа.
• ***Ленивая репликация*** происходит в фоновом режиме, когда сервер имеет промежуток свободного времени.
• ***Репликация файла, использующая группу -*** все системные вызовы ЗАПИСАТЬ передаются одновременно на все серверы, таким образом копии создаются одновременно с созданием оригинала.
• При групповой репликации все копии создаются в одно и то же время.
• Два хорошо известных ***алгоритма решения проблемы***:

1. Алгоритм ***«репликация первой копии»*** - требует, чтобы один сервер был выделен как первичный. Остальные серверы являются вторичными.
2. Изменения должны быть сохранены в постоянном запоминающем устройстве еще до изменения первичной копии.

• Метод, предложенный Гиффордом и известный как ***«голосование»***. Пусть имеется n копий, тогда изменения должны быть внесены в любые W копий. При этом серверы, на которых хранятся копии, должны отслеживать порядковые номера их версий. В случае, когда какой-либо сервер выполняет операцию чтения, он обращается с запросом к любым R серверам. Если R+W > n, то, хотя бы один сервер содержит последнюю версию, которую можно определить по максимальному номеру.

Билет № 16

Современные концепции построения ОС

• Современная ОС должна реализовывать мультипрограммную обработку, виртуальную память, свопинг, поддерживать многооконный интерфейс, а также выполнять многие другие, совершенно необходимые функции.

• **Рыночные требования**:

Расширяемость. Код должен быть написан таким образом, чтобы можно было легко внести дополнения и изменения, если это потребуется, и не нарушить целостность системы.
Переносимость. Код должен легко переноситься с процессора одного типа на процессор другого типа и с аппаратной платформы (которая включает наряду с типом процессора и способ организации всей аппаратуры компьютера) одного типа на аппаратную платформу другого типа.
Надежность и отказоустойчивость. Система должна быть защищена как от внутренних, так и от внешних ошибок, сбоев и отказов. Ее действия должны быть всегда предсказуемыми, а приложения не должны быть в состоянии наносить вред ОС.
Совместимость. ОС должна иметь средства для выполнения прикладных программ, написанных для других операционных систем. Кроме того, пользовательский интерфейс должен быть совместим с существующими системами и стандартами.
Безопасность. ОС должна обладать средствами защиты ресурсов одних пользователей от других.
Производительность. Система должна обладать настолько хорошим быстродействием и временем реакции, насколько это позволяет аппаратная платформа.

• **Переносимость**

1. Большая часть кода должна быть написана на языке, который имеется на всех машинах, куда вы хотите переносить систему.
2. В какое физическое окружение программа должна быть перенесена. Различная аппаратура требует различных решений при создании ОС (32 и 16 адреса).
3. Минимизировать или, если возможно, исключить те части кода, которые непосредственно взаимодействуют с аппаратными средствами.
4. Если аппаратно зависимый код не может быть полностью исключен, то он должен быть изолирован в нескольких хорошо локализуемых модулях.

• **Совместимость**

• Способность ОС выполнять программы, написанные для других ОС или для более ранних версий данной операционной системы, а также для другой аппаратной платформы.
• Двоичная совместимость (совместимость на уровне команд процессора, совместимость на уровне системных вызовов и даже на уровне библиотечных вызовов) и совместимость на уровне исходных текстов приложений (наличия соответствующего компилятора в составе программного обеспечения, а также совместимости на уровне библиотек и системных вызовов).

• **Безопасность**

• Иерархия уровней безопасности, приведенная в Оранжевой Книге, помечает низший уровень безопасности как D, а высший - как A.
• ***C-системы***: наличие подсистемы учета событий, связанных с безопасностью, и избирательный контроль доступа. ***Уровень C делится на 2 подуровня***: уровень C1, обеспечивающий защиту данных от ошибок пользователей, но не от действий злоумышленников, и более строгий уровень C2:

- Средства секретного входа
- Избирательный контроль доступа
- Средства учета и наблюдения (auditing)
- Защита памяти

На этом уровне система не защищена от ошибок пользователя.
• ***Системы уровня B*** основаны на помеченных данных и распределении пользователей по категориям, то есть реализуют ***мандатный контроль доступа***. Каждому пользователю присваивается рейтинг защиты, и он может получать доступ к данным только в соответствии с этим рейтингом. Этот уровень в отличие от уровня C защищает систему от ошибочного поведения пользователя.
• ***Уровень A*** является самым высоким уровнем безопасности, он требует в дополнение ко всем требованиям уровня B выполнения формального, математически обоснованного доказательства соответствия системы требованиям безопасности.

Билет № 17

Структурное построение ОС

♦ Монолитные системы

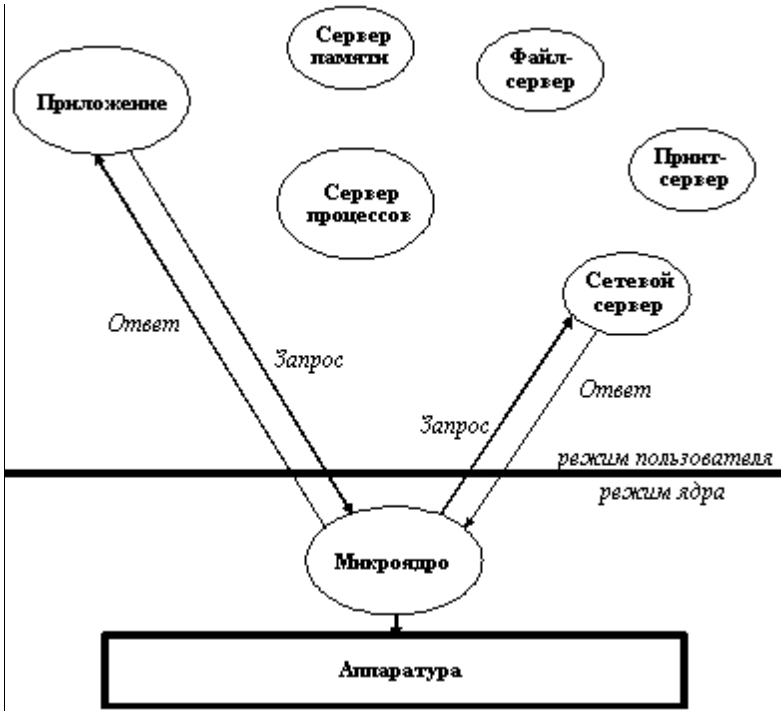
• В общем случае «структура» монолитной системы представляет собой отсутствие структуры.
• ОС написана как набор процедур, каждая из которых может вызывать другие, когда ей это нужно.
• Для построения монолитной системы необходимо скомпилировать все отдельные процедуры, а затем связать их вместе в единый объектный файл с помощью компоновщика.
• При обращении к системным вызовам, поддерживаемым ОС, параметры помещаются в строго определенные места, такие, как регистры или стек, а затем выполняется специальная команда прерывания, известная как вызов ядра или вызов супервизора.

♦ Многоуровневые системы

• Обобщением предыдущего подхода является организация ОС как иерархии уровней.
• Уровни образуются группами функций операционной системы - файловая система, управление процессами и устройствами и т.п.
• Каждый уровень может взаимодействовать только со своим непосредственным соседом - выше- или нижележащим уровнем.
• Прикладные программы или модули самой операционной системы передают запросы вверх и вниз по этим уровням.
• Первой системой, построенной таким образом была простая пакетная система THE, которую построил Дейкстра и его студенты в 1968 году. Система имела 6 уровней.
• Дальнейшее обобщение многоуровневой концепции было сделано в ОС MULTICS. В системе MULTICS каждый уровень (называемый ***кольцом***) является более привилегированным, чем вышележащий. Когда процедура верхнего уровня хочет вызвать процедуру нижележащего, она должна выполнить соответствующий системный вызов, то есть команду TRAP (прерывание).

♦ Модель клиент-сервер и микроядра

• Модель клиент-сервер предполагает наличие программного компонента - потребителя какого-либо сервиса - ***клиента***, и программного компонента - поставщика этого сервиса - ***сервера***.
• ***Взаимодействие между клиентом и сервером стандартизуется***, так что сервер может обслуживать клиентов, реализованных различными способами и, может быть, разными производителями.
• Эта модель успешно применяется не только при построении ОС, но и на всех уровнях программного обеспечения.



• Компоненты, лежащие выше микроядра, хотя и используют сообщения, пересылаемые через микроядро, взаимодействуют друг с другом непосредственно.
• Микроядро играет роль регулировщика. Оно проверяет сообщения, пересылает их между серверами и клиентами, и предоставляет доступ к аппаратуре.
• Главный принцип разделения работы между микроядром и окружающими его модулями - включать в микроядро только те функции, которым абсолютно необходимо исполняться в режиме супервизора и в привилегированном пространстве.
• В этой модели ***весь машинно-зависимый код изолирован в микроядре***, поэтому для переноса системы на новый процессор требуется меньше изменений и все они логически сгруппированы вместе.
• Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти, и таким образом защищен от других процессов.
• Серверы не имеют непосредственного доступа к аппаратуре и не могут модифицировать память.

♦ Объектно-ориентированный подход

• Объектно-ориентированный подход - каждый программный компонент является функционально изолированным от других.
• ***Объект*** - это единица программ и данных, взаимодействующая с другими объектам посредством приема и передачи сообщений.
• Программы объекта определяют перечень действий, которые могут быть выполнены над данными этого объекта.
• Механизм наследования позволяет создать иерархию объектов, в которой каждый объект более низкого уровня приобретает все свойства своего предка.
• Для того, чтобы получить данные из объекта или поместить данные в объект, необходимо вызывать соответствующие объектные функции.
• Разработчик может обращаться к функциям других объектов, или строить новые объекты путем наследования свойств других объектов - ***инкапсуляцией***.
• Но особенно большие перспективы имеет этот подход в реализации распределенных вычислительных сред. Приложения, созданные для такой сетевой среды, основанной на объектах, могут выполняться, динамически обращаясь к множеству объектов, независимо от их местонахождения в сети и независимо от их операционной среды.

Билет № 18

Средства OLE. Стандарт OpenDoc.CORBA.

• Средства OLE

• Для пользователей Windows объектно-ориентированный подход проявляется при работе с программами, использующими технологию OLE фирмы Microsoft.
• OLE 2.0, доступная в настоящее время в качестве расширения Windows 3.1, переопределяет документ-клиент как контейнер. Когда пользователь щелкает дважды над объектом OLE 2.0, вставленным в документ-контейнер, он активизируется в том же самом месте.
Наиболее заметный недостаток OLE - отсутствие сетевой поддержки.

• Стандарт OpenDoc

• Ключевыми технологиями OpenDoc являются механизм хранения ***Бенто*** и ***технология сценариев*** (scripting).
• В Бенто-документе каждый объект имеет постоянный идентификатор, перемещающийся вместе с ним от системы к системе.
• ***SOM*** нейтральна к языкам программирования и поддерживает ***наследование***
• ***COM*** ориентирована на C ++ и вместо механизма наследования использует альтернативный механизм, который Microsoft называет ***агрегацией***.

• Семейство CORBA

• ***CORBA*** (Common Object Request Broker Architecture - Общая архитектура посредника обработки объектных запросов) закладывает фундамент распределенных вычислений с переносимыми объектами.
• CORBA задает способ поиска объектами других объектов и вызова их методов.
• CORBA может гарантировать только механизм нижнего уровня, посредством которого одни объекты вызывают другие. Для успешного взаимодействия требуется также понимание сообщений друг друга.

Билет № 19

Множественно прикладные среды. Распределенная служба безопасности.

♦ Множественно прикладные среды.

• Концепция множественных прикладных сред приносит пользователю долгожданную возможность выполнять на своей ОС программы, написанные для других операционных систем и других процессоров.
• Множественные прикладные среды обеспечивают совместимость данной ОС с приложениями, написанными для других ОС и процессоров, на двоичном уровне, а не на уровне исходных текстов.
• Самой большой потенциальной проблемой является ***производительность*** - прикладная среда должна выполнять программы с приемлемой скоростью.
• Тщательно сделанная прикладная среда имеет в своем составе библиотеки, имитирующие внутренние библиотеки GUI, но написанные на родном коде, то есть она совместима с программным интерфейсом другой ОС.
• Библиотекам GUI не нужно дешифрировать и имитировать каждую команду, значит в частях программы, относящихся к вызовам ***GUI ABI*** (Application Binary Interface - двоичный интерфейс прикладного программирования), производительность может резко вырасти.
• Модульность операционных систем нового поколения позволяет намного легче реализовать поддержку множественных прикладных сред. Это делает создание дополнительных модулей, объединяющих эмуляцию процессора и трансляцию библиотек, значительно более простым.
• К усовершенствованным операционным системам, явно содержащим средства множественных прикладных сред, относятся: IBM OS/2 2.x и Workplace OS, Microsoft Windows NT, PowerOpen компании PowerOpen Association и версии UNIX от Sun Microsystems.
• В UNIX транслятор прикладных сред обычно делается, как и другие прикладные программы, плавающим на поверхности операционной системы. В более современных операционных системах типа Windows NT или Workplace OS модули прикладной среды выполняются более тесно связанными с операционной системой, хотя и обладают по-прежнему высокой независимостью.

♦ Распределенная служба безопасности.

• Имеется две больших группы функций службы безопасности: ***идентификация*** и ***авторизация***.
• ***Идентификация*** проверяет идентичность объекта (например, пользователя или сервиса).
• ***Авторизация*** (или управление доступом) назначает привилегии объекту, такие как доступ к файлу.
• ***Служба идентификации*** представляет собой механизм передачи третьей стороне функций проверки идентичности пользователя.
• ***Служба безопасности OSF DCE*** базируется на системе идентификации Kerberos, разработанной в 80-е годы и расширенной за счет добавления элементов безопасности. Kerberos использует шифрование, основанное на ***личных ключах***, для обеспечения ***трех уровней защиты***:

- ***Самый нижний уровень*** требует, чтобы пользователь идентифицировался только при установлении начального соединения, предполагая, что дальнейшая последовательность сетевых сообщений исходит от идентифицированного пользователя.
- ***Следующий уровень*** требует идентификацию для каждого сетевого сообщения.
- ***На последнем уровне*** все сообщения не только идентифицируются, но и шифруются.

• Система безопасности не должна сильно усложнять жизнь конечного пользователя в сети, то есть он не должен запоминать десятки паролей и кодов.
• Весьма полезным сетевым средством для целей безопасности является служба прав доступа или, другими словами, авторизация. ***Служба авторизации OSF*** базируется на POSIX-совместимых списках прав доступа - ***ACL***.
• В то время как система Kerberos основана на личных ключах, в настоящее время широкое распространение получили методы, основанные на ***публичных ключах*** (например, ***метод RSA***).

Билет № 20

Распределенная служба каталогов. Файловая система AFS.

♦ Распределенная служба каталогов.

• ***Задачей службы каталогов*** в распределенной сети является поиск сетевых объектов, то есть пользователей, ресурсов, данных или приложений.
• В гетерогенных глобальных сетях служба каталогов становится намного более сложной, из-за необходимости синхронизации различных баз данных каталогов.
• Хорошая служба каталогов делает использование распределенного окружения прозрачным для пользователя. Пользователям не нужно знать расположение удаленного принтера, файла или приложения.
• ***OSF определяет двухъярусную архитектуру для службы каталогов*** для целей адресации межъячеечного и глобального взаимодействия. ***Ячейка (cell)*** - это фундаментальная организационная единица для систем в DCE OSF. Ячейки могут иметь социальные, политические или организационные границы. Ячейки состоят из компьютеров, которые должны часто взаимодействовать друг с другом - это могут быть, например, рабочие группы, отделы, или отделения компаний.
• **Служба каталогов DCE состоит из 4-х элементов:**

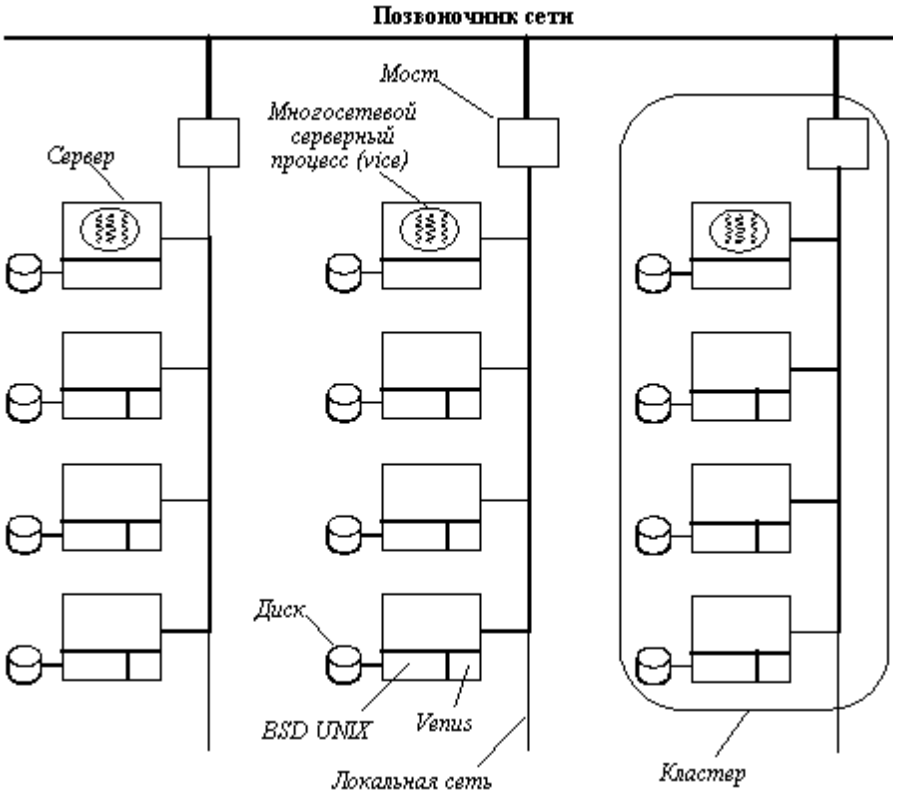
- CDS (Cell Directory Service) - служба каталогов ячейки. Ячейка сети - это группа систем, администрируемых как единое целое. CDS оптимизируется для локального доступа. Большинство запросов к службе каталогов относятся к ресурсам той же ячейки. Каждая ячейка сети нуждается по крайней мере в одной CDS.
- GDA (Global Directory Agent) - агент глобального каталога. GDA - это шлюз имен, который соединяет домен DCE с другими административными доменами посредством глобальной службы каталогов X.500 и DNS (domain name service - сервис имен домена). GDA передает запрос на имя, которое он не смог найти в локальной ячейке, службе каталогов другой ячейки, или глобальной службе каталогов (в зависимости от места хранения имени). Для того, чтобы отыскать имя, клиент делает запрос к локальному агенту GDA. Затем GDA передает запрос на междоменное имя службе X.500. Эта служба возвращает ответ GDA, который в свою очередь передает его клиенту. OSF GDA может быть совместим с любой схемой глобального именования.
- GDS (Global Directory Service) - глобальная служба каталогов. Основанная на стандарте X.500, GDS работает на самом верхнем уровне иерархии и обеспечивает связь множества ячеек в множестве организаций.
- XDS (X/Open Directory Service) - обеспечивает поддержку X/Open API функций службы каталогов и позволяет разработчикам писать приложения, независимые от нижележащих уровней архитектуры службы каталогов. XDS-совместимые приложения будут работать одинаковым образом со службами каталогов DCE и X.500.

♦ Распределенная файловая система DFS OSF

• Распределенная файловая система DFS OSF предназначена для обеспечения прозрачного доступа к любому файлу, расположенному в любом узле сети. Главная концепция такой распределенной файловой системы - это простота ее использования.
• Распределенная файловая система DFS OSF базируется на известной файловой системе AFS (The Andrew File System).

♦ Файловая система AFS

• Файловая система была спроектирована так, чтобы обеспечить прозрачность доступа каждому пользователю, независимо от того, какой рабочей станцией он пользуется.
• Особенностью этой файловой системы является возможность работы с большим (до 10 000) числом рабочих станций.



• Каждый клиент имеет также часть кода - ***venus***, которая управляет интерфейсом между клиентом и серверной частью системы, называемой ***vice***. Вначале venus выполнялся в пользовательском режиме, но позже он был перемещен в ядро для повышения производительности. ***Venus*** работает также в качестве ***менеджера кэша***. В дальнейшем мы будем называть venus просто клиентом, а ***vice - сервером***.

Пространство имен, видимое пользовательскими программами, выглядит как традиционное дерево в ОС UNIX с добавленным к нему каталогом /cmu (рисунок 2). Содержимое каталога /cmu поддерживается AFS посредством vice-серверов и идентично на всех рабочих станциях. Другие каталоги и файлы исключительно локальны и не разделяются. Возможности разделяемой файловой системы предоставляются путем монтирования к каталогу /cmu. Файл, который UNIX ожидает найти в верхней части файловой системы, может быть перемещен символьной связью из разделяемой файловой системы (например, /bin/sh может быть символьно связан с /cmu/bin/sh).

• При открытии удаленного файла весь файл (или его значительная часть, если он очень большой) загружается на диск рабочей станции и кэшируется там, причем процесс, который сделал вызов OPEN, даже не знает об этом. По этой причине каждая рабочая станция имеет диск.

• После загрузки файла на локальный диск он помещается в локальный каталог /cash, так что он выглядит для ОС как нормальный файл.

• ***Безопасность*** - это главный вопрос в системе с 10 000 пользователей. Все сообщения между рабочими станциями шифруются на уровне аппаратуры.

• Каталоги защищаются списками прав доступа (ACL), но файлы имеют обычные биты RWX UNIX'а. Разработчики системы предпочитают механизм ACL, но так как многие UNIX-программы работают с битами RWX, то они оставлены для совместимости.

• Диски рабочих станций используются только для временных файлов, кэширования удаленных файлов и хранения страниц виртуальной памяти, но не для постоянной информации.

• Каталога и ячейки AFS поддерживает еще одно важное понятие - ***том***. Том - это собрание каталогов, которые управляются вместе.

• ***Семантика***, предлагаемая AFS, близка к сессионной семантике. Когда файл открывается, он берется у подходящего сервера и помещается в каталог /cash на локальном диске на рабочей станции. Все операции чтения-записи работают с кэшированной копией. При закрытии файла он выгружается назад на сервер.

• Тома, предназначенные только для чтения, такие как системные двоичные файлы, реплицируются, а пользовательские файлы - нет.

• Прикладные программы используют двухуровневую схему именования, при которой каталог содержит структуры, называемые ***fids*** (file identifiers), вместо традиционных номеров i-узлов.

• ***Общий механизм доступа к файлам в AFS***. Когда приложение выполняет системный вызов OPEN, то он перехватывается оболочкой клиента, которая первым делом проверяет, не начинается ли имя файла с /cmu. Если нет, то файл локальный, и обрабатывается обычным способом. Если да, то файл разделяемый. Производится грамматический разбор имени, покомпонентно находится fid. По fid проверяется кэш, и здесь имеется три возможности:

1. Файл находится в кэше, и он достоверен.
2. Файл находится в кэше, и он не достоверен.
3. Файл не находится в кэше.

В первом случае используется кэшированный файл. Во втором случае клиент запрашивает сервер, изменялся ли файл после его загрузки. Файл может быть недостоверным, если рабочая станция недавно

перезагружалась или же некоторый другой процесс открыл файл для записи, но это не означает, что файл уже модифицирован, и его новая копия записана на сервер. Если файл не изменялся, то используется кэшированный файл. Если он изменялся, то используется новая копия. В третьем случае файл также просто загружается с сервера. Во всех трех случаях конечным результатом будет то, что копия файла будет на локальном диске в каталоге /cash, отмеченная как достоверная.

Вызовы приложения READ и WRITE не перехватываются оболочкой клиента, они обрабатываются обычным способом. Вызовы CLOSE перехватываются оболочкой клиента, которая проверяет, был ли модифицирован файл, и, если да, то переписывает его на сервер, который управляет данным томом.

Помимо кэширования файлов, оболочка клиента также управляет кэшем, который отображает имена файлов в идентификаторы файлов fid. Это ускоряет проверку, находится ли имя в кэше. Проблема возникает, когда файл был удален и заменен другим файлом. Однако этот новый файл будет иметь другое значение поля «уникальный номер», так что fid будет выявлен как недостоверный. При этом клиент удалит вход (pass, fid) и начнет грамматический разбор имени с самого начала. Если дисковый кэш переполняется, то клиент удаляет файлы в соответствии с алгоритмом LRU.

Vice работает на каждом сервере как отдельная многонитевая программа. Каждая нить обрабатывает один запрос. Протокол между сервером и клиентом использует RPC и построен непосредственно на API. В нем есть команды для перемещения файлов в обоих направлениях, блокирования файлов, управления каталогами и некоторые другие. Vice хранит свои таблицы в виртуальной памяти, так что они могут быть произвольной величины.

Так как клиент идентифицирует файлы по их идентификаторам fid, то у сервера возникает следующая проблема: как обеспечить доступ к UNIX-файлу, зная его vnode, но не зная его полное имя. Для решения этой проблемы в AFS в UNIX добавлен новый системный вызов, позволяющий обеспечить доступ к файлам по их индексам vnode.

• Реализация DFS на базе AFS дает прекрасный пример того, как работают вместе различные компоненты DCE. DFS работает на каждом узле сети совместно со службой каталогов DCE, обеспечивая единое пространство имен для всех файлов, хранящихся в DFS. DFS использует списки ACL системы безопасности DCE для управления доступом к отдельным файлам. Потоковые функции RPC позволяют DFS передавать через глобальные сети большие объемы данных за одну операцию.

♦ Распределенная служба времени

• В распределенных сетевых системах необходимо иметь ***службу согласования времени***. Многие распределенные службы, такие как распределенная файловая система и служба идентификации, используют сравнение дат, сгенерированных на различных компьютерах. Чтобы сравнение имело смысл, пакет DCE должен обеспечивать согласованные временные отметки.

• ***Сервер времени OSF DCE*** - это система, которая поставляет время другим системам в целях синхронизации. Любая система, не содержащая сервера времени, называется ***клерком*** (clerk). Распределенная служба времени использует ***три типа серверов*** для координации сетевого времени. ***Локальный сервер*** синхронизируется с другими локальными серверами той же локальной сети. ***Глобальный сервер*** доступен через расширенную локальную или глобальную сети. ***Курьер (courier)*** - это специальный локальный сервер, который периодически сверяет время с глобальными серверами. Через периодические интервалы времени серверы синхронизируются друг с другом с помощью протокола DTS OSF. Этот протокол может взаимодействовать с протоколом синхронизации времени NTP сетей Internet.

• Одной из главных особенностей и достоинств пакета DCE OSF является тесная взаимосвязь всех его компонентов. Это свойство пакета иногда становится его недостатком. Так, очень трудно работать в комбинированном окружении, когда одни приложения используют базис DCE, а другие - нет.

• Для того, чтобы стать действительно распространенным базисом для создания гетерогенных распределенных вычислительных сред, ***пакет DCE должен обеспечить поддержку двух ключевых технологий*** - обработку транзакций и объектно-ориентированный подход.

• Поддержка транзакций совершенно необходима для многих деловых приложений, когда недопустима любая потеря данных или их несогласованность. Две фирмы - IBM и Transarc - предлагают дополнительные средства, работающие над DCE и обеспечивающие обработку транзакций. Что же касается объектно-ориентированных свойств DCE, то OSF собирается снабдить этот пакет средствами, совместимыми с объектно-ориентированной архитектурой CORBA, и работающими над инфраструктурой DCE. После достаточно тщательной работы на уровне исходных кодов, разработчики могут создать DCE-сервер, который сможет обслуживать Windows NT-клиентов, и Windows NT-сервер, который работает с DCE-клиентами.

Запуск: bash lab2.bs

Скрипт:

```
#! /bin/bash
###################################
# 1.
ps –A |less
###################################
# 2.
i=1
j=100
while [ “$i” –le “$j” ]
 do
   echo “PID$i=” >> /home/lab2.txt
   ps –o pid >> /home/lab2.txt
   echo “PPID$i=” >> /home/lab2.txt
   ps –o ppid >> /home/lab2.txt
   echo “TTY$i=” >> /home/lab2.txt
   ps –o tty >> /home/lab2.txt
   echo “RUID$i=” >> /home/lab2.txt
   ps –o ruid >> /home/lab2.txt
   echo “COMMAND$i=” >> /home/lab2.txt
   ps –o command >> /home/lab2.txt
   echo >> /home/lab2.txt
   let “i=$i+1”
 done
exit 0
###################################
# 3.
ps –U root >> /home/lab2.txt
###################################
# 4.
ps –c >> /home/lab2.txt
ps –v >> /home/lab2.txt
ps –j >> /home/lab2.txt
ps –u >> /home/lab2.txt
###################################
# 5.
/home/labs/back
ps –A  #Узнали PID back
kill –SIGTERM PID_BACK
ps –A  #PID back уже не должен появиться в списке
###################################
# 6. bash /home/all.bs
#! /bin/bash
i=1
j=2
while [ “$i” -le “$j” ]
 do
  echo “NO_EXIT”
 done
exit 0
###################################
# 7.
#На втором терминале запускаем
ps –x
#В списке нашли процесс /home/all.bs => PID
kill –SIGTERM all.bs_PID
###################################
# 8.
1 терминал:
mkfifo /home/lab2_fifo.txt
cat /home/lab2_fifo.txt > fifo
2 терминал:
cat fifo  #Вывели содерж. именованного канала на экран
###################################
# 9.
tty >> /home/lab2.txt
w  >> /home/lab2.txt
uname –a >> /home/lab2.txt
uptime >> /home/lab2.txt
###################################
# 10.
trap “echo SIGINT” INT
CTRL-C
###################################
# 11.
echo “Poterenko A.G” > /home/lab2Poter1.txt
cp /home/lab2OTCHET.txt /home/lab2Poter2.txt
cat /home/lab2Poter1.txt /home/lab2Poter2.txt >
/home/lab2OTCHET.txt
rm –f /home/lab2Poter1.txt
rm –f /home/lab2Poter2.txt
```

Предмет: Операционные системы
Лабораторная работа: №2
Тема: Команды управления процессами

Выполнил: Потеренко А.Г.
Проверил: Коноплянов А.В.
Дата: 19.06.2005г.

Запуск: bash lab5.bs

Скрипт:

```
#! /bin/bash
#########################################################################################################
#########################################################################################################
# .bashrc
…
# Source global definitions
if [ -f /etc/bashrc ]; then
   . /etc/bashrc
####HELLO#####
echo “Time:”
date
echo “User:”$USER
echo “Pwd:”$PWD
echo “$”
###############
…
#########################################################################################################
#########################################################################################################
echo “Number of parental process=”$PPID
echo “Number of parental process=”$PPID >> /home/lab5POTER.txt
echo “The current working catalogue=”$PWD
echo “The current working catalogue=”$PWD >> /home/lab5POTER.txt
echo “The catalogue which was the worker up to current=”$OLDPWD
echo “The catalogue which was the worker up to current=”$OLDPWD >> /home/lab5POTER.txt
echo “User=”$UID
echo “User=”$UID >> /home/lab5POTER.txt
echo “RANDOM=”$RANDOM
echo “RANDOM=”$RANDOM >> /home/lab5POTER.txt
echo “LINENO=”$LINENO
echo “LINENO=”$LINENO >> /home/lab5POTER.txt
echo “HOSTNAME=”$HOSTNAME
echo “HOSTNAME=”$ HOSTNAME >> /home/lab5POTER.txt
#########################################################################################################
#########################################################################################################
vi /root/.bashrc
# Открыли файл .bashrc
# .bashrc
# User specific aliases and functions
alias ls=’ls -G’
alias ll=’ls -l’
alias la=’ls -a’
alias pa=’ps ax’
alias x=’startx’
# Сохранили файл .bashrc
alias  #Проверили
#########################################################################################################
#########################################################################################################
vi /root/.bashrc
# Открыли файл .bashrc
# .bashrc
# User specific aliases and functions
lab() {
        echo “Laboratory work #”$1; echo “Poterenko A.G.”
       }
# Сохранили файл .bashrc
lab 5  >> /home/lab5POTER.txt
```

Предмет: Операционные системы ***Лабораторная работа***: №5 ***Тема***: Настройка оболочки	***Выполнил***: Потеренко А.Г. ***Проверил***: Коноплянов А.В. ***Дата:*** 19.06.2005г.

Запуск: bash lab6.bs

Скрипт:

```
#! /bin/bash
#########################################################################################################
#########################################################################################################
mkdir /home/lab6POTER
touch /home/lab6POTER/lab6.txt
cd /home/lab6POTER/
echo “The mounted all file systems:” > lab6.txt
df –a >> lab6.txt
echo “The mounted all file systems:”
df –a
#########################################################################################################
mkdir /home/labs
touch /home/labs/text.txt
mkdir /home/floppy
cp /home/labs/text.txt  /home/floppy
echo “Contents of the catalogue floppy:” >> /home/lab6POTER/lab6.txt
ls /home/floppy >> /home/lab6POTER/lab6.txt
echo “Contents of the catalogue floppy:”
ls /home/floppy
#########################################################################################################
mount –t vfat /dev/fd0 /home/floppy/
echo “After assembling a diskette:” >> /home/lab6POTER/lab6.txt
ls /home/floppy >> /home/lab6POTER/lab6.txt
echo “After assembling a diskette:”
ls /home/floppy
echo “The mounted all file systems:” >> /home/lab6POTER/lab6.txt
df –a >> /home/lab6POTER/lab6.txt
echo “The mounted all file systems:”
df –a
#########################################################################################################
#########################################################################################################
umount /dev/fd0  /home/floppy/
echo “The mounted all file systems:”
df –a
echo “The mounted all file systems:” >> /home/lab6POTER/lab6.txt
df –a >> /home/lab6POTER/lab6.txt
#########################################################################################################
#########################################################################################################
fdformat –n /dev/fd0
mkfs –t msdos /dev/fd0
mount –t msdos /dev/fd0 /home/floppy/
cp /home/lab6POTER/lab6.txt  /home/floppy/
echo “FLOPPY+file:” >> /home/lab6POTER/lab6.txt
ls /home/floppy >> /home/lab6POTER/lab6.txt
echo “FLOPPY+file:”
ls /home/floppy
umount /dev/fd0  /home/floppy/
#########################################################################################################
#########################################################################################################
df –m >> /home/lab6POTER/lab6.txt
du –scm /root  >> /home/lab6POTER/lab6.txt    #”Домашний каталог” в данном случае /root
du –scm --exclude=boot  /  >> /home/lab6POTER/lab6.txt
# Корневой каталог монтируется на “/dev/sda2”
# “/boot” монтируется на “/dev/sda1” – поэтому мы исключаем “--exclude=boot”
```

Предмет: Операционные системы ***Лабораторная работа***: №6 ***Тема***: Монтирование и демонтирование ФС	***Выполнил***: Потеренко А.Г. ***Проверил***: Коноплянов А.В. ***Дата:*** 19.06.2005г.