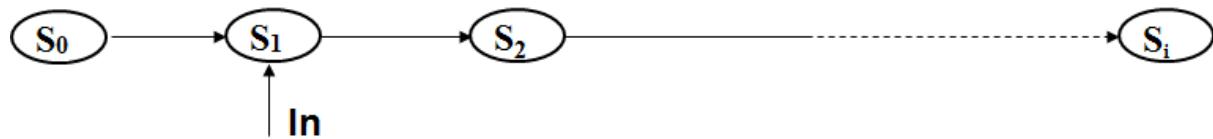


Исходный файл «Еще чьи-то ответы». Дополнен, начиная с 24го вопроса

1. Формализованное описание программного процесса. Понятие вектора состояния процесса, трека процесса, инициатора развития процесса.

ФО: Процесс = Инициатор + трек + процессор



(S_i) - вектор состояния процесса

Вектор состояния в каждый момент времени должен со-держать информацию, достаточную для продолжения выполнения программы или повторного пуска программы с заданной точки

S_i (контекст процесса) – информация для процессора, необходимая для развития процесса:

- выполняемая команда (активная часть вектора, выполнение которой вызывает изменение параметров)
- адрес следующей команды
- другие параметры

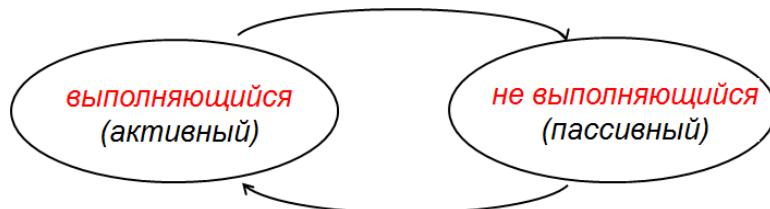
Трек процесса – упорядоченная по времени последовательность векторов состояния

In – инициатор развития процесса. Задает движение процесса, переход от одного вектора к другому.

2. Логическая(абстрактная) модель процесса. Физическая модель процесса. Графы состояний для логической и физической модели.

Логическая модель:

- 1.Каждый процесс имеет собственный процессор (N процессов = N процессоров)
- 2.Обеспечивает решение процессорно-независимых задач (синхронизация и взаимодействие процессов)
- 3.Определяется графиком состояний:

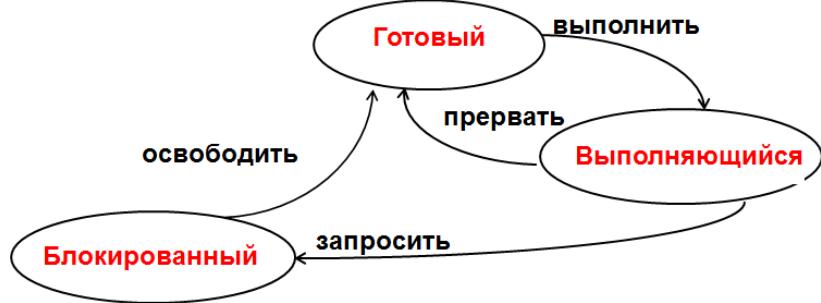


Физическая модель:

- 1.Рассматривает распределение процессам физических процессоров
- 2.N процессов != N процессоров
- 3.Процесс рассматривается как объект управления
- 4.Процессу м.б. выделен процессор, который он может вернуть либо добровольно, либо отобран принудительно (через заданное время, либо при выполнении некоторого условия)

5. В системе д.б. реализован механизм (*т.е. механизм прерывания*), позволяющий:
- сохранять вектор состояния процесса(контекст) с целью будущего его восстановления и прерывать выполнение процесса
 - восстанавливать контекст прерванного процесса и продолжать его выполнение

6. Граф состояний:



3. Прерывания процессов. Назначение прерываний. Внутренние и внешние прерывания

Назначение механизма прерываний

- обеспечение асинхронного режима взаимодействия программных и аппаратных процессов ВС
- организация взаимодействия программных процессов
- поддержка мультипрограммной и мультипроцессорной обработки

Внутренние прерывания (СИНХРОННЫЕ, программные)

- Программируемые прерывания

- обращение к услугам ОС (системные вызовы)
- потребность работы с ресурсами (получить ресурс, отказаться от ресурса, выполнить над ресурсом некоторые действия(в/в и т.п.)
- необходимость какие-либо действия в отношении других процессов (порождение, уничтожение, синхронизация)

- Прерывания, связанные с работой процессора

Прерывания синхронны с операциями процессора:

- арифметическое переполнение,
- исчезновение порядка в операциях с плавающей точкой,
- обращение к защищенной области ОП и т.д.

Внешние прерывания (АСИНХРОННЫЕ, аппаратные)

Вызываются событиями, которые не связаны с выполняющимся процессом (события возникают вне выполняющегося процесса без его ведома, т.е. асинхронно)

- прерывания от внешних устройств ввода-вывода
- прерывания от таймера
- прерывания от другого процессора и т.д.

4. Прерывания процессов. Аппаратная и программная поддержка обработки прерываний.

Аппаратная поддержка:

- Векторный(vectored) способ

- сигнал IRQ(Interrupt request), поступающий от контроллера устройства, контроллер прерываний отображает на определённый элемент вектора прерываний.
- элемент вектора прерывания соответствует определённой программе обработки прерываний.
- процессор формирует начальный адрес программы обработки прерываний – обработчика прерывания.

-Опрашиваемый(polled) способ

- процессор получает информацию об уровне приоритета прерывания.
- С каждым уровнем приоритета связано несколько устройств и процессор опрашивает все обработчики прерываний данного уровня для определения конкретного устройства, которое вызвало прерывание
- процессор формирует начальный адрес программы обработки прерываний – обработчика прерывания.

-Комбинированный способ(Intel Pentium)

- Элемент вектора прерывания указывает на одну из 256 программ обработки прерываний.
- При подключении к линии IRQ одного устройства – векторный способ
- При подключении к линии IRQ нескольких устройств – опрашиваемый способ

Программная поддержка:

Программное управление прерываниями позволяет ОС регулировать обработку сигналов прерывания, заставляя процессор обрабатывать их:

- сразу по приходу;
- откладывать обработку на некоторое время;
- полностью игнорировать прерывания.

Обычно обработка прерывания выполняется только после завершения выполнения текущей команды.

Выбор прерывания для обработки осуществляется на основе приоритетов и очередей прерываний.

Маскирование прерываний – запрет обработки прерываний любого приоритета в некотором промежутке времени

Приоритетное управление обработкой прерываний

- относительные приоритеты
- абсолютные приоритеты

стековая дисциплина

Стековый механизм организации прерываний:

Если запрос на прерывание имеет более высокий приоритет, то инициируется процесс обработки прерывания.

I - Вектор состояния для прерванной программы запоминается в стеке. Из некоторой ячейки (в дальнейшем это будет определено) определяется (подается) адрес первой команды обработки прерываний на счётчик команд, т.е. выбирается вектор прерывания.

II - Из некоторой ячейки выбирается вектор прерывания и помещается на регистры ЦП, который содержит адрес 1-й команды программы обработки прерывания. Далее выполняется программа обработки прерывания (3 этапа):

1. Сохранение РОНов для прерванной программы.
2. Непосредственное выполнение программы обработки прерываний.
3. Восстановление сохраненных РОНов для прерванной программы.

Распределение прерываний по уровням приоритета

Средства контроля процессора (высокий приоритет)

Системный таймер

Внешние устройства

Программные прерывания (низкий приоритет)

5. Синхронизация параллельных процессов. Программная реализация взаимоисключений. Семафорные примитивы Дейкстры. Достоинства и недостатки семафоров.

Программная реализация примитивов взаимоисключения осуществляется с соблюдением следующих четырех ограничений:

- задача должна быть решена чисто программным способом на машине, не имеющей специальных команд взаимоисключения;
- не должно быть никаких предположений об относительных скоростях выполнения асинхронных параллельных процессов;
- процессы, находящиеся вне своих критических участков, не могут препятствовать другим процессам входить в их собственные критические участки;
- не должно быть бесконечного откладывания момента входа процессов в их критические участки.

Семафорные примитивы Дейкстры

Семафор это защищенная переменная, значение которой можно опрашивать и изменять только при помощи операции инициализации и двух специальных атомарных операций P и V.

P(S) (закрытие семафора S)

IF S>0 THEN S:=S-1 (занять единицу семафора)

<продолжить текущий процесс>

IF S=0 THEN <остановить процесс и поместить его в очередь ожидания семафора>

[если s > 0 то s:=s-1 иначе (ожидать на s)]

V(S) (открытие семафора S)

IF S=0 THEN <процесс из очереди ожидания поместить в очередь готовых>

<продолжить текущий процесс>

ELSE S:= S+1(освободить единицу семафора)

[если (имеются процессы, которые ожидают на s)

то (разрешить одному из них продолжить работу)

иначе s:=s+1]

Мьютексы (семафоры взаимного исключения)

Простейшие двоичные семафоры (это те, которые могут принимать значения 0 и 1, есть еще счетчики - они могут принимать неотрицательные значения)

Отмеченное состояние – мьютекс свободен

Неотмеченное состояние – процесс является владельцем мьютекса

Системные вызовы

Создание мьютекса(CreateMutex)

Открытие мьютекса(OpenMutex)

Ожидание (WaitForSingleObject, WaitForMultipleObject)

Освобождение(ReleaseMutex)

Достиоинства семафоров

- Простота
- Независимость от количества процессов
- Отсутствие «активного ожидания»

Недостатки семафоров

- Примитивны (семафор не указывает непосредственно на синхронизирующее условие, с которым он связан или на критический ресурс)
- При построении сложных схем синхронизации алгоритмы получаются сложными и ненаглядными

6. Синхронизация параллельных процессов. Синхронизация и взаимодействие процессов с помощью программных каналов.

Программный канал (pipe) - средство синхронизации и обмена данными между процессами.

- Канал представляет собой поток данных(байтов) между двумя (или более) процессами
- Операции записи и чтения осуществляются потоком байтов
- Конвейер имеет определенный размер, который не может превышать 64 Кбайт. и работает циклически
- Как информационная структура канал имеет
 - идентификатор
 - размер
 - два указателя

Каналы представляют собой системный ресурс: чтобы начать работу с конвейером, процесс сначала должен заказать его у операционной системы и получить в свое распоряжение.

Системные вызовы для работы с каналами

Функция создания канала:

- описатель для чтения из канала,
- описатель для записи в канал,
- размер канала.

Функция чтения из канала:

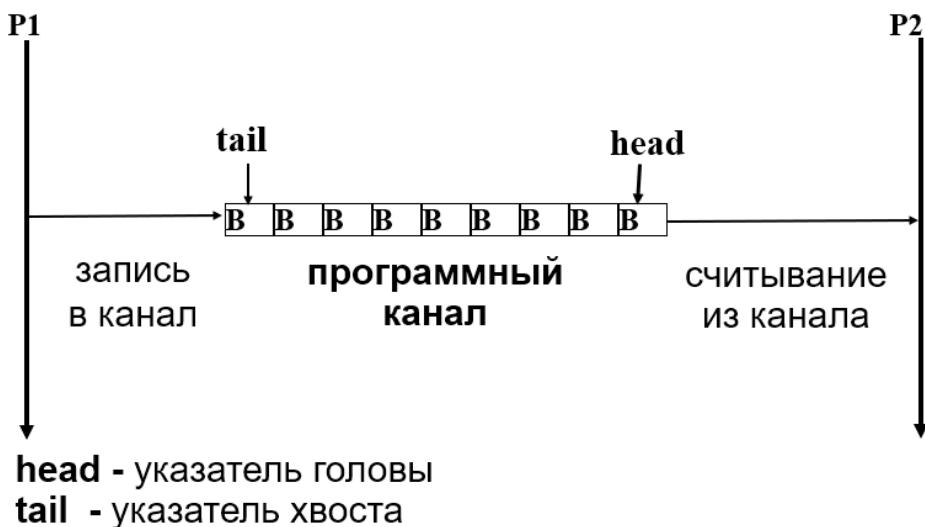
- описатель для чтения из канала,
- переменная любого типа,
- размер переменной,
- количество прочитанных байтов.

Функция записи в канал:

- описатель для записи в конвейер,
- количество записанных байтов.

При обращении к полному каналу для записи процесс будет ждать, пока в канале не освободится место.

При обращении к пустому каналу для чтения процесс будет ждать, пока в канале не появится информация для чтения.



7. Дедлок (тупиковая ситуация). Условия возникновения дедлоков. Решение проблемы дедлоков (стратегии предотвращения и обхода дедлоков)

Дедлок (тупик) – ситуация , при которой множество параллельно выполняющихся процессов находятся в состоянии бесконечного ожидания ресурсов из-за неправильного распределения запросов процессов на ресурсы

Процесс P1 использует ресурс R1 и запрашивает ресурс R2

Процесс P2 использует ресурс R2 и запрашивает ресурс R1

Условия возникновения

1. Условие взаимного исключения

Процессы осуществляют монопольный доступ к ресурсам.

2. Условие ожидания

Процесс, запросивший ресурс, будет ждать, пока запрос не будет удовлетворен, продолжая удерживать все остальные ресурсы, которые он уже получил.

3. Условие отсутствия перераспределения.

Никакие ресурсы нельзя отобрать у процесса, если они ему уже выделены.

4. Условие кругового ожидания.

Существует замкнутая цепь процессов, каждый из которых ждет ресурса, удерживаемый его предшественником в этой цепи.

Стратегии ПРЕДОТВРАЩЕНИЯ

Стратегии предотвращения дедлоков представляет собой такую реализацию системы, которая позволит исключить саму возможность взаимоблокировок

Косвенный метод предотвращения

основан на предотвращении одного из первых трёх условий

1. Условие взаимного исключения

-можно предотвратить неограниченным выделением ресурсов

- неприемлемо к совместно используемым переменным в критических интервалах
- нельзя использовать в случае последовательно используемых ресурсов

2. Условие ожидания

- можно исключить предварительным выделением ресурсов.
- Процесс должен потребовать все ресурсы заранее, и он не сможет начать своё выполнение до тех пор, пока они ему не будут выделены.
- неэффективность использования ресурсов. Каждый процесс должен ждать, пока не получит всех необходимых ресурсов, даже если некоторые из них используются в конце выполнения процесса.
- трудно сформулировать запросы на ресурсы в тех случаях, когда требуемые ресурсы становятся известны только после начала выполнения.

3. Условие отсутствия перераспределения можно исключить, позволяя ОС отнимать у процесса ресурсы

Для перераспределения могут быть использованы различные дисциплины:

- процесс, запросивший дополнительный ресурс, предварительно отказывается от любого количества этого ресурса, которое он уже имеет.
- общее исключение - отнимать у процесса, заблокированного при запросе дополнительного ресурса, все ресурсы.

Прямой метод предотвращения дедлока

основан на предотвращении условия кругового ожидания ресурсов, предотвращая образование цепи используя стратегию иерархического выделения ресурсов:

1. Все ресурсы образуют иерархию
2. Процесс, затребовавший ресурс на одном уровне, может затем потребовать ресурс только на более высоком уровне;
3. Процесс может освободить ресурс на данном уровне только после освобождения всех ресурсов на более высоких уровнях;
4. После того как процесс получил, а потом освободил ресурсы данного уровня, он может снова запросить ресурсы на том же самом уровне.

Предварительное выделение ресурсов – частный случай иерархического выделения, если есть единственный уровень ресурсов

Стратегия ОБХОДА (алгоритм банкира)

Вход в дедлок можно предотвратить, если у системы есть информация о последовательности запросов ресурсов, связанных с каждым из параллельных процессов.

Если система находится в неопасном состоянии, то существует по крайней мере одна последовательность состояний, которая обходит опасные состояния системы.

Решение задачи – контролируемое выделение ресурсов(алгоритм банкира)

В этом случае необходимо для каждого запроса на ресурс, в предположении, что он удовлетворен, определить, существует ли среди последующих запросов от всех других процессов некоторая последовательность запросов, которая может привести к дедлоку.

- Банкир выдаёт денежные кредиты **определенённому** числу клиентов.
- Каждый клиент заранее сообщает банкиру **максимальную сумму**, которая ему нужна.
- Клиент может занимать эту сумму **по частям**, и нет никаких гарантий, что он возвратит часть денег до того, как сделает весь заем.
- Весь капитал банкира обычно меньше, чем суммарные требования клиентов, так как банкир не предполагает, что все клиенты сделают максимальные займы одновременно.
- Банкир предполагает, что он выполнил запрос, и оценивает сложившуюся ситуацию
- Он определяет **клиента**, чей следующий заем наиболее близок к максимальному (минимальный из возможных запросов).
- Если банкир не может ссудить оставшуюся сумму **даже этому клиенту**, то он отклоняет первоначальный запрос.
- Если же он сможет ссудить эту сумму, то он удовлетворяет первоначальный запрос.

Постановка задачи

Если в некоторый момент времени **клиент запросит** некоторую денежную сумму, то банкир должен оценить, сможет ли он ссудить ее **без риска попасть в ситуацию, когда не будет достаточного количества денег**, чтобы обеспечить дальнейшие займы(хотя бы минимальные)

12

8. Долгосрочное и краткосрочное планирование в мультипрограммной операционной системе. Единицы планирования и выполнения процессов.

Мультипрограммная ОС:

- Может выполнять несколько процессов одновременно
- Обеспечивает разделение адресных пространств процессов и взаимную защиту процессов друг от друга.
- Обеспечивает разделение процессоров для выполнения процессов
- Позволяет равномерно загружать ресурсы (ЦП, ОП, ВУ), чтобы каждый процесс был выполнен.

Единица планирования и владениями ресурсами

процесс(process)

задание(task)

Единица выполнения процесса

поток(thread)

облегчённый процесс(lightweight process)

Долгосрочное планирование

Планирование верхнего уровня - Распределение между процессами имеющихся ресурсов ОС (виртуального процессора)

Долгосрочный планировщик решает, какой из процессов, находящихся во входной очереди, должен быть переведен в очередь готовых процессов в случае освобождения виртуальной памяти ОС.

Процесс, которому выделено виртуальная память ОС и другие ресурсы находится в состоянии владения ресурсами (resource ownership).

Процесс помещается в очередь готовых процессов (ready queue). Процессы в этой очереди ожидают выделения процессу ресурса «процессорное время».

Краткосрочное планирование

Планирование нижнего уровня(диспетчеризация) – выделение процессам фактического процессора

Краткосрочный планировщик(диспетчер) решает, какому из процессов, находящихся в очереди готовых процессов, должен быть предоставлен центральный процессор(CPU)
Диспетчер передает управление процессором процессу либо на фиксированный интервал времени, либо процесс может только добровольно вернуть управление процессором

[Долгосрочное планирование (виртуальные процессоры, входная очередь -> очередь готовых)] -> [Краткосрочное планирование (фактические процессоры и выделение процессорного времени, очередь готовых -> на выполнение)]

9. Управление оперативной памятью. Отображение программных модулей на оперативную память. Виртуальное адресное пространство(ВАП) процесса
Исходная программа имеет пространство имен – набор логических имен переменных и входных точек программных модулей (чаще всего являются символьными и для которых отсутствует отношение порядка).

Оперативная память – упорядоченное множество ячеек. Количество ячеек ограничено и фиксировано.

Методы управления памятью:

Ø Свопинг - Метод управления памятью, основанный на том, что адресные пространства процессов, участвующих в мультипрограммной обработке, хранятся во внешней памяти. Временно выгруженные из памяти страницы могут сохраняться на внешних запоминающих устройствах как в файле, так и в специальном разделе на жёстком диске (partition), называемые соответственно swap-файл и swap-раздел.

Ø Смежное размещение адресных пространств процессов

- Однопрограммный режим
- Мультипрограммирование с фиксированными разделами MFT
(Multiprogramming with a fixed number of tasks)

.Фиксируются границы разделов оперативной памяти.

.В каждом разделе может быть размещено адресное пространство одного процесса.

.Фрагментация:

Внутренняя – размер адресного пространства процесса меньше размера раздела ОП, выделенного данному процессу.

Внешняя – размер адресного пространства процесса в очереди больше размера свободного раздела ОП

- Мультипрограммирование с переменными разделами MVT (Multiprogramming with a variable number of tasks)

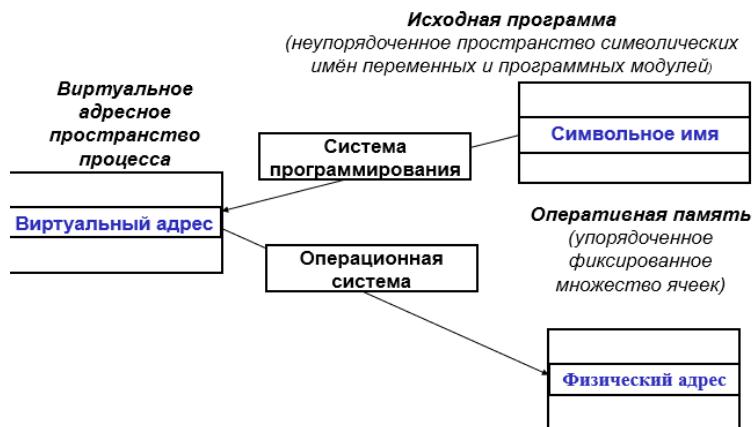
- . Границы разделов не фиксируются.
- . В начальной фазе фрагментации нет. На фазе повторного размещения те же причины фрагментации, что и для MFT.
- . Уплотнение – перемещение занятых разделов по адресному пространству памяти таким образом, чтобы свободный фрагмент занимал одну связную область.

Ø Несмежное размещение адресных пространств процессов

Виртуальное адресное пространство процесса разбивается на множество частей, которые располагаются в различных, не обязательно смежных, участках оперативной памяти.

- Сегментная организация памяти (ВАППроц + Таблица сегментов + ОП, сегменты)
- Страницчная организация памяти (ВАППроц + Таблица страниц + ОП, страницная рамка)
- Сегментно - страницчная организация памяти

Этапы отображения ПМ на ОП



1. Системой программирования на основе логических (символьных) имен формируется **виртуальное адресное пространство (ВАП)** процесса - множество всех допустимых значений виртуального адреса:

§ ВАП зависит от архитектуры процессора и от системы программирования, и не зависит от объема реальной физической памяти, на которую это пространство будет отображаться;

§ В результате работы системы программирования полученные адреса могут иметь как двоичную форму, так и символьно-двоичную (окончательная привязка их к физическим ячейкам осуществляется на этапе загрузки ВАП в память перед ее выполнением).

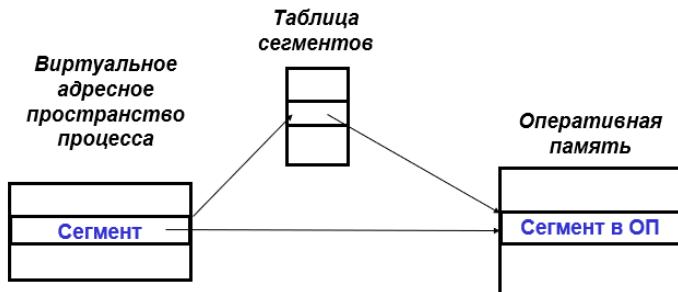
2. Операционной системой с помощью программных модулей управления памятью и использования необходимых аппаратных средств вычислительной системы производится отображение виртуального адресного пространства процесса на физическую память компьютера.

Частные случаи отображения:

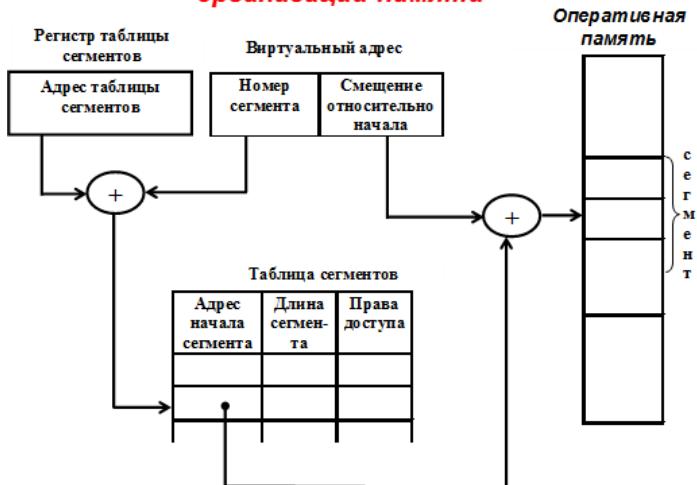
- Виртуальное адресное пространство тождественно физической памяти
 - ∅ Нет необходимости во втором отображении
 - ∅ Виртуальные адреса точно соответствуют физическим адресам ОП
- Виртуальное адресное пространство тождественно исходному пространству имён
 - ∅ Отображение исходной программы на ОП выполняет ОС на основе таблицы символьных имён

10. Отображение виртуального адресного пространства процесса на оперативную память при сегментной и страничной организации памяти. Вычисление адреса ячейки памяти

Сегментная:

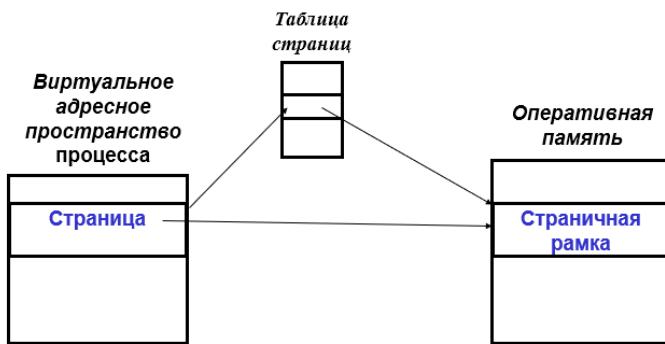


Вычисление адреса при сегментном способе организации памяти

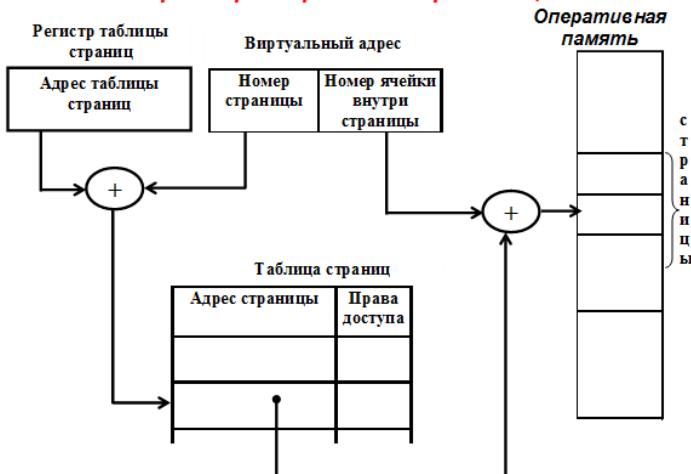


Создается таблица сегментов и сохраняется ее адрес в регистре ->
из ВА берется номер сегмента/селектор + адрес таблицы = адрес начала сегмента ->
адрес начала сегмента + смещение из ВА = нужная ячейка ОП в данном сегменте

Страницчная:



Вычисление адреса при страницной организации памяти



Создается таблица страниц и ее адрес сохраняется в регистре ->
из ВА берется номер страницы + адрес таблицы страниц = адрес страницы ->
адрес страницы + № ячейки из ВА = нужная ячейка ОП внутри данной страницы

11. Технология виртуальной памяти(ВП). Отображение виртуального адресного пространства процесса на ВП ОС при сегментной, страничной и сегментно-страничной организации ВП. Вычисление адреса ячейки памяти.

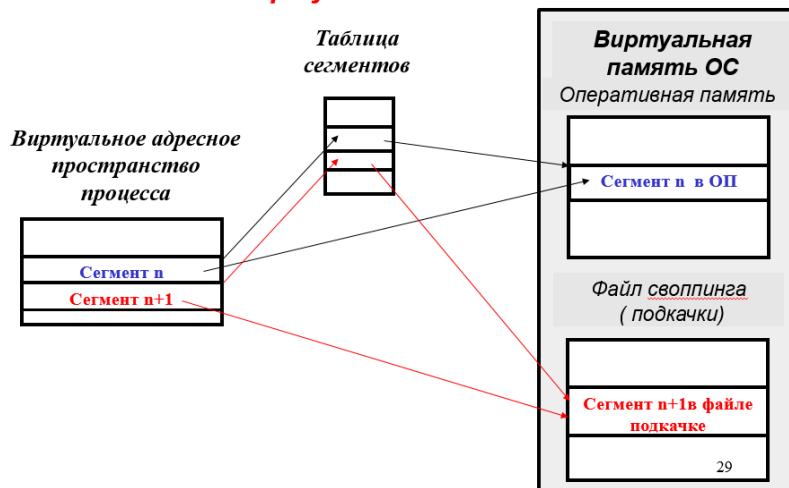
Виртуальная память – это технология, которая позволяет выполнять процесс, виртуальное адресное пространство которого может только частично располагаться в ОП.

Виртуальная память ОС – виртуальный ресурс, построенный на основе оперативной и внешней памяти.

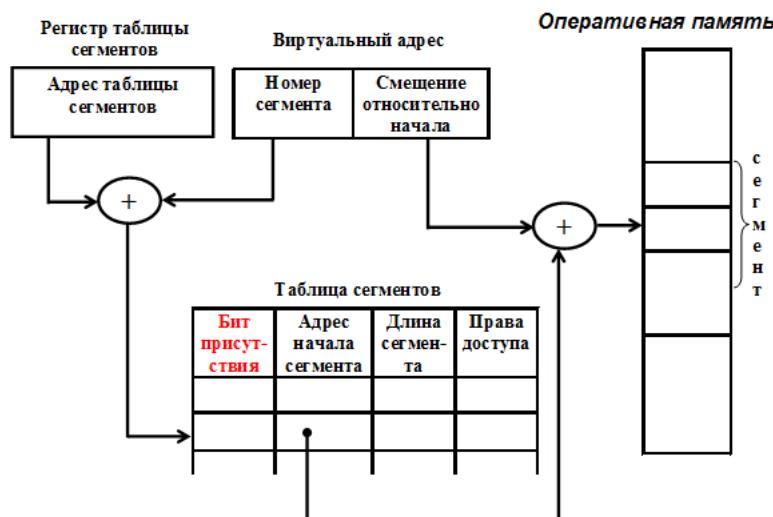
Операционная система отображает ВАП процесса в раздел виртуальной памяти операционной системы, выделенный данному процессу

Каждому процессу предоставляется непрерывное адресное пространство виртуальной памяти операционной системы, которое включает и оперативную память и внешнюю (файл или раздел).

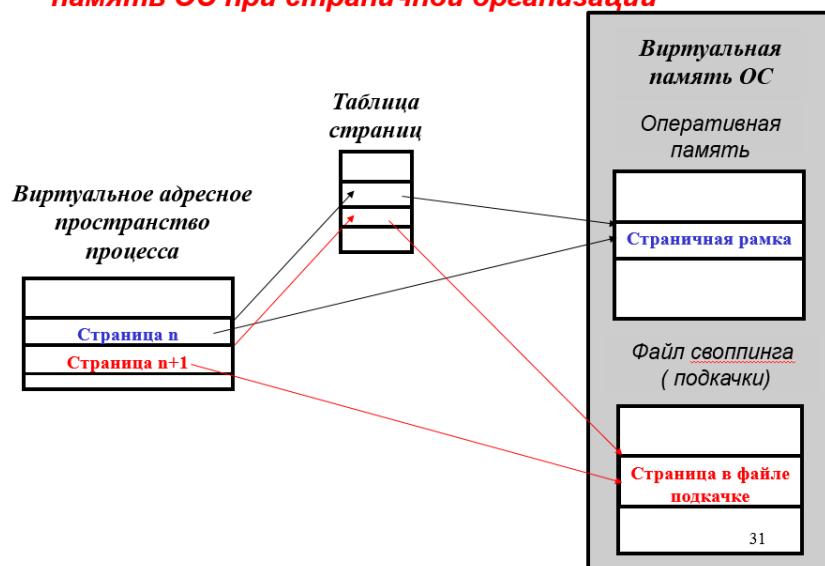
Отображение ВАП процесса на виртуальную память операционной системы при сегментной организации виртуальной памяти



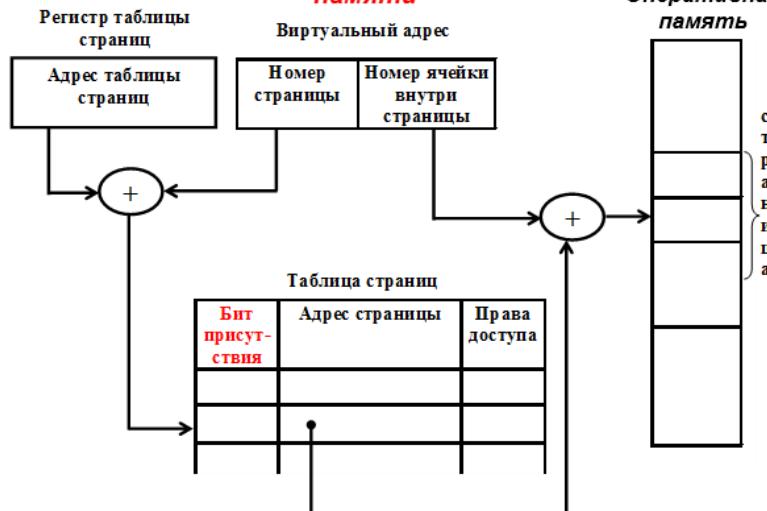
Вычисление адреса при сегментном способе организации виртуальной памяти

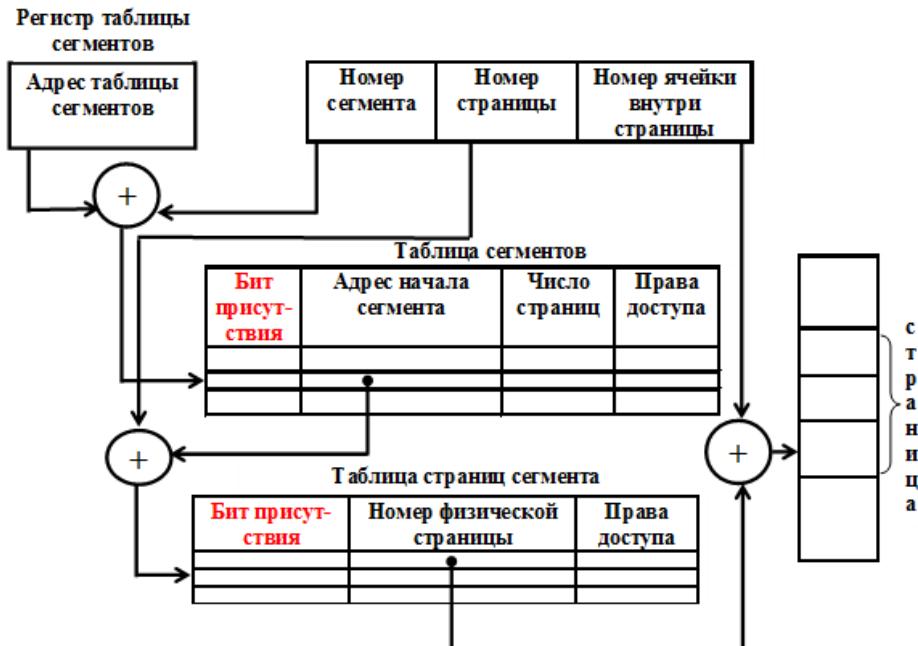


Отображение ВАП процесса на виртуальную память ОС при страничной организации



Вычисление адреса при страничной организации виртуальной памяти





Вычисление адреса при сегментно-страничном способе организации виртуальной памяти

33

Ключ сначала вычисляется нужный сегмент, потом нужная страница внутри сегмента, потом нужная ячейка внутри страницы

12. Диспетчеризация процессов при мультипрограммировании. Дисциплины диспетчеризации в однопроцессорных и многопроцессорных ОС.

- Дисциплины диспетчеризации процессов в **однопроцессорной мультипрограммной ОС**

ØНевытесняющее мультипрограммирование, при котором процесс, владеющий процессором, может отдать его только **добровольно**

1. Первый пришел – первый обслужен (FCFS, first come-first served)

§ Для всех процессов одно и то же среднее время ожидания

§ Короткие процессы вынуждены ждать столько же, сколько и длинные(«эффект конвоя»)

2. Следующий - с кратчайшим заданием (SJN, shortest job-next)

§ Снижается общее среднее время ожидания

§ Среднее время ожидания коротких процессов меньше, чем длинных

§ Растет дисперсия времени ожидания

ØВытесняющее мультипрограммирование, при котором процесс может быть вытеснен из процессора **принудительно**.

1. Круговой циклический алгоритм (RR, round robin)

§ Каждому процессу по очереди из очереди готовых процессов выделяется фиксированный квант времени q, в конце которого, если процесс к этому

времени не закончился или не был заблокирован, он снимается с процессора, а процессор передаётся следующему готовому процессу

§ Процесс, снятый с процессора, ставится в конец очереди

§ Приоритет процесса возрастает с увеличением времени ожидания с момента получения последнего кванта.

§ Если существует n готовых процессов, то каждый получит $1/n$ часть процессорного времени.

§ Такая форма равного обслуживания неявно отдает предпочтение коротким процессам

§ Из-за расходов на переключение процессов общее время ожидания может быть больше, чем при FCFS

§ Если дисперсия времени выполнения процессов велика, то при циклической дисциплине общее среднее время ожидания может быть меньше, чем при FCFS(первый пришёл - первый обслуживается)

Существенную роль играет размер кванта. Он может зависеть от размера очереди и параметров процессов

2. Многоуровневый циклический алгоритм (FB, Feed Back, обратная связь)

§ Процессы поступают в первую очередь готовых процессов $N=1$

§ Из нее процессы поступают на CPU и/или полностью обслуживается или снова поступает в очередь, но с номером на 1 больше

§ Заявка в i -ой очереди обслуживается, если пусты все предыдущие очереди.

§ В очереди N_{max} заявки обслуживаются до завершения (в очереди N_{max} принцип FIFO + RR)

3. Смешанный циклический алгоритм (RR+FB)

§ Каждый процесс получает в i -ой очереди несколько квантов процессорного времени и только потом переходит в очередь $i+1$.

§ Достоинства смешанного алгоритма:

- + сокращаются накладные расходы на перевод процесса из очереди в очередь;
- + возможно подобрать параметры алгоритма под поток процессов (т.е., можно варьировать квантом времени q)

Особенности бесприоритетных дисциплин диспетчеризации

- Линейные дисциплины характеризуются одинаковым средним временем ожидания.
- Циклические дисциплины обслуживают короткие процессы с неявным приоритетом.
- Бесприоритетные дисциплины не требуют предварительной информации о времени выполнения процессов.
- Уменьшение длительности ожидания коротких процессов происходит за счет увеличения $t_{ожид.}$ длинных процессов.

Приоритет – это право на первоочередное обслуживание

Принцип, заложенный для приоритетов: HPF – highest priority first (наивысший приоритет - первым)

Приоритетные дисциплины диспетчеризации

Назначение приоритета процессам происходит на основе

- статистических характеристик
(фиксированный приоритет):

- ожидаемый объем ввода/вывода;
- априорное $t_{выполнения.}$;
- ожидаемый объем виртуальной памяти

- динамических характеристик
(динамический приоритет):

- время нахождения в системе;
- объем вводимых/выводимых операций в предыдущий момент времени.

Дисциплины диспетчеризации с фиксированным приоритетом

Процессы располагаются в очереди готовых процессов в соответствии со своим приоритетом. Если поступило несколько процессов одного приоритета, то в очереди они обслуживаются по принципу FIFO.

Недостатки дисциплин с фиксированным приоритетом

- неточность априорных характеристик приводит к приблизительности приоритета;
- возможность приспособляемости пользователей к системе приоритета. Если пользователю известно о системе приоритетов, т.е. больше «уважают» короткие заявки в системе, то пользователь составит 2 короткие вместо 1-ой длинной, но тогда система становится на уровень FIFO;
- при хороших и средних характеристиках параметра **ожидания**, отдельные заявки в системе могут длительное время ждать. Для устранения этого недостатка надо бы повысить приоритет задачи, но т.к. он фиксирован, это невозможно;
- отсутствует настраиваемость системы на изменение характеристик входного потока заявок.

Дисциплины диспетчеризации с динамическим приоритетом

Приоритет изменяется в период пребывания процесса в системе (либо во время ожидания, либо во время выполнения)

Пример: изменение динамического приоритета процессов от времени

Если низкоприоритетный процесс ждет длительное время, его приоритет повышается

$$P_i(t) = a_i t_{\text{ожидан.}} + b_i t_{\text{выполн.}},$$

где $P_i(t)$ – приоритет i -го процесса

12

Дисциплины диспетчеризации с динамическим приоритетом

1. Следующий - с минимальным оставшимся временем(SRT – shortest-remaining-time)

- Оставшееся время – разность между временем, запрошенным процессом, и временем, которое процесс уже получил (измеряется с помощью системных часов).
- Аналог SJN с перераспределением процессора.
- Короткие процессы еще более выигрывают за счет длинных.
- Так как SJN и SRT основаны на оценке времени, то они не подходят для диспетчеризации интерактивных процессов.
- Недостатком дисциплин, основанных на сокращении общего времени ожидания, является задержка выполнения длинных процессов (процессы могут теряться практически навсегда).

Дисциплины диспетчеризации с динамическим приоритетом

2. Следующий – с наивысшим отношением отклика(HRRN - highest response ratio next)

- Отношение отклика

$$R = (w+s)/s,$$

где W – время затраченное процессом на ожидание,

S – ожидаемое время выполнения.

- Минимальное значение R (равное 1) – при входе процесса в систему.

При завершении, прерывании или блокировании текущего процесса, для выполнения из очереди готовых процессов выбирается тот, который имеет наибольшее значение R (Учитывается возраст процесса).

15

Дисциплины диспетчеризации с несколькими очередями

Процессы делятся на классы и процессы разных классов обслуживаются по разному.

Классы процессов:

- 1. Реального времени** (д.б. обслужен до наступления конкретного момента времени) (**Оперативные процессы с высоким приоритетом**).
- 2. Интерактивные** (д.б. обслужен в течение приемлемого времени реакции). (**Оперативные процессы с высоким приоритетом**).
- 3. Пакетные** (нет жестких ограничений на время обслуживания). (**Фоновые процессы с низким приоритетом**).

Для каждой из очередей можно использовать свою дисциплину и разные значения управляющих параметров (например, длина кванта).

17

Диспетчеризация процессов в мультипроцессорной системе

В мультипроцессорной системе

- Виртуальные адресные пространства **всех** процессов располагаются в **едином виртуальном адресном пространстве ОС**
- Есть множество процессоров, которые могут быть выделены для выполнения процессов

Задача заключается в том, что операционная система должна решать, **какой процесс на каком процессоре** должен выполняться.

19

Диспетчеризация процессов в мультипроцессорной системе

1. Алгоритм разделения времени

- Простейший алгоритм диспетчеризации процессов (**потоков**) состоит в поддержании единой очереди готовых потоков (возможно с различными приоритетами).
- Наличие единой очереди, используемой всеми центральными процессорами, обеспечивает процессорам режим разделения времени подобно тому, как это выполняется на однопроцессорной системе.
- Такая организация позволяет автоматически **балансировать нагрузку**, то есть она исключает ситуацию, при которой один центральный процессор простояивает, в то время как другие процессоры ²⁰ перегружены.

Диспетчеризация процессов в мультипроцессорной системе

2. Алгоритм родственного планирования

- Основная идея данного алгоритма заключается в том, чтобы поток был запущен на том же центральном процессоре, что и в прошлый раз
- Стремления удерживать процессы на одном и том же центральном процессоре максимизируют **родственность кэша**
- Однако если у какого-либо центрального процессора нет работы, у загруженного работой процессора отнимается поток и отдается свободному

21

Диспетчеризация процессов в мультипроцессорной системе

3. Алгоритм бригадного планирования

Бригадное планирование состоит из трех частей:

1. Группы связанных потоков планируются как одно целое, бригада
2. Все члены бригады запускаются одновременно, на разных центральных процессорах с разделением времени
3. Все члены бригады начинают и завершают свои временные интервалы (кванты времени) одновременно

Бригадное планирование работает благодаря синхронности работы всех центральных процессоров:

- Время разделяется на дискретные кванты
- В начале каждого нового кванта все центральные процессоры перепланируются заново, и на каждом процессоре запускается новый поток
- В начале следующего кванта опять принимается решение о планировании. В середине кванта планирование не выполняется. Если какой-либо поток блокируется, его центральный процессор простояивает до конца кванта времени.

22

13. Устройства ввода-вывода. Контроллеры устройств ввода-вывода. Порты ввода-вывода. Способы доступа к контроллерам.

Функции управления вводом-выводом

Операционная система должна:

- § обеспечить простой и удобный интерфейс между пользовательским уровнем и устройствами ввода-вывода
- § предоставлять пользователю возможность использования блокирующих (синхронных) и неблокирующих (асинхронных) операций ввода-вывода
- § управлять внешними устройствами и передавать данные с помощью команд ввода-вывода
- § обрабатывать ошибки ввода-вывода.

Интерфейс ввода-вывода, насколько это возможно, должен быть одинаковым для всех устройств (для достижения независимости от применяемых устройств).

Архитектура внешних устройств ввода-вывода

Внешние устройства ввода-вывода состоят из механических и электронных компонент.

Механический компонент – собственно внешнее устройство.

Электронный компонент – контроллер устройства (адаптер) предназначен для управления устройством.

Каждый контроллер может управлять одним или несколькими устройствами.

Каждый контроллер устройства ввода-вывода имеет:

- несколько регистров, которые используются для взаимодействия с центральным процессором
- буферы данных для приема и передачи информации

Центральный процессор выполняет ввод-вывод, записывая команды в регистры контроллера устройства

Контроллер устройства может генерировать сигнал прерывания, для того, чтобы сообщить контроллеру прерываний и процессору о завершении соответствующей команды

Центральный процессор получает результаты и информацию о состоянии устройства, читая информацию из регистров контроллера

Параметры контроллеров ВВ-ВЫВ:

Адреса управляющих регистров контроллера (портов)

Адреса буферов данных контроллера

Уровень приоритета запроса прерывания от контроллера

/ Необходима настройки каждого контроллера таким образом, чтобы адреса портов, буферов и номера прерываний различных устройств не конфликтовали друг с другом.

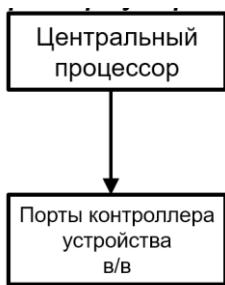
/ Установка адресов портов и номеров прерываний вручную

/ Использование стандарта plug and play, который позволяет операционной системе автоматически собирать информацию об устройствах ввода-вывода, централизованно назначать уровни прерывания и адреса портов ввода-вывода и устанавливать эти параметры в каждом контроллере

- Команды управления. Используются для приведения внешнего устройства в действие и завершения его работы

- Команды состояния. Используются для проверки состояния контроллера ввода-вывода и соответствующих устройств.

- Команды передачи. Используется для чтения и/или записи данных в регистры процессора и внешние устройства и из регистров процессора и внешних устройств.



→ Команды ввода/вывода

СПОСОБЫ ДОСТУПА

1. Раздельные адресные пространства оперативной памяти и контроллера устройства

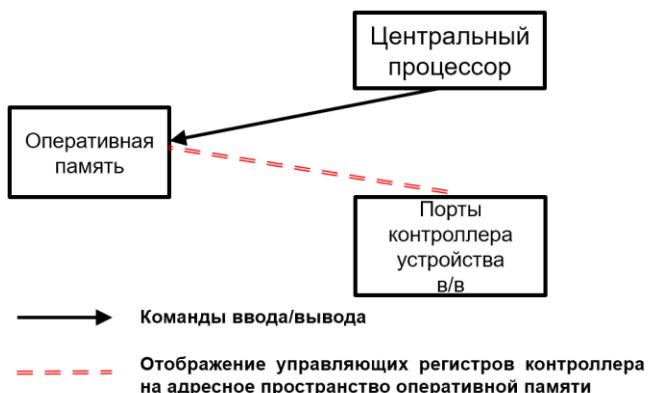
Каждому управляющему регистру назначается номер порта ввода-вывода (8- или 16-разрядное целое число)

С помощью команды процессора IN REG,PORT центральный процессор читает управляющий регистр устройства из порта PORT в регистр процессора REG.

С помощью команды процессора OUT PORT,REG центральный процессор записывает содержимое своего регистра REG в управляющий регистр устройства через порт PORT.

При такой схеме адресные пространства оперативной памяти и устройств ввода-вывода различны

2. Отображаемый на адресное пространство памяти ввод-вывод



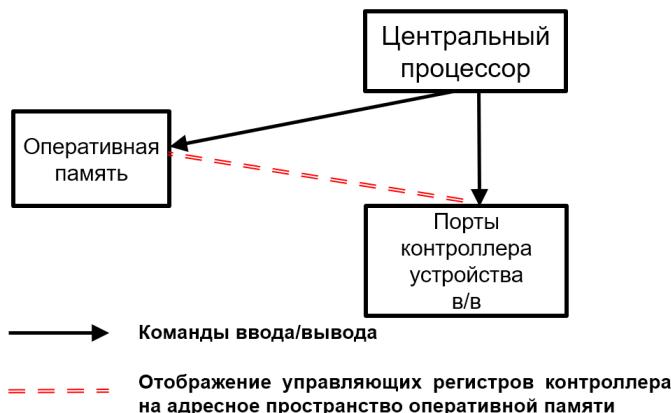
Все управляющие регистры контроллеров внешних устройств отображаются на адресное пространство оперативной памяти. Каждому управляющему регистру назначается уникальный адрес в памяти.

Обычно для регистров устройств отводятся адреса на вершине адресного пространства

Для взаимодействия с

контроллером внешних устройств используются команды чтения-записи регистров памяти MOV REG, PAM

3. Гибридный способ доступа



14. Структура программного обеспечения ввода-вывода. Процессы ввода-вывода.

Основная идея организации программного обеспечения ввода-вывода состоит в разбиении его на уровни:

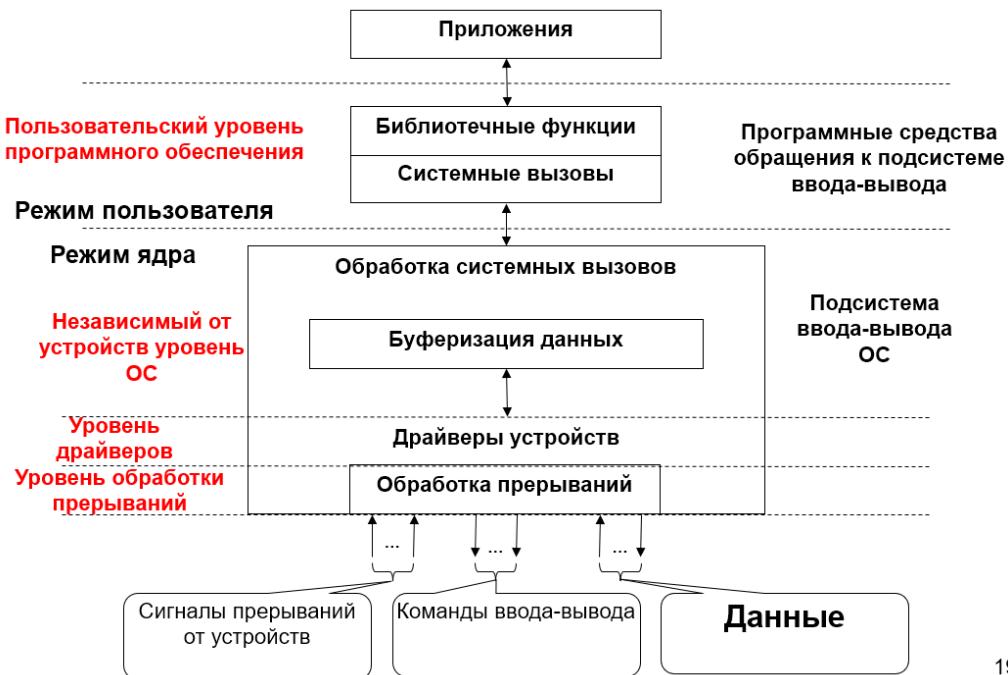
Нижние уровни ОС скрывают особенности аппаратуры от верхних уровней

Верхние уровни ОС в свою очередь, обеспечивают удобный интерфейс для пользователей.

Ошибки должны обрабатываться как можно ближе к аппаратуре. Нижний уровень должен сообщать об ошибке верхнему, только если он не может справиться с ошибкой.

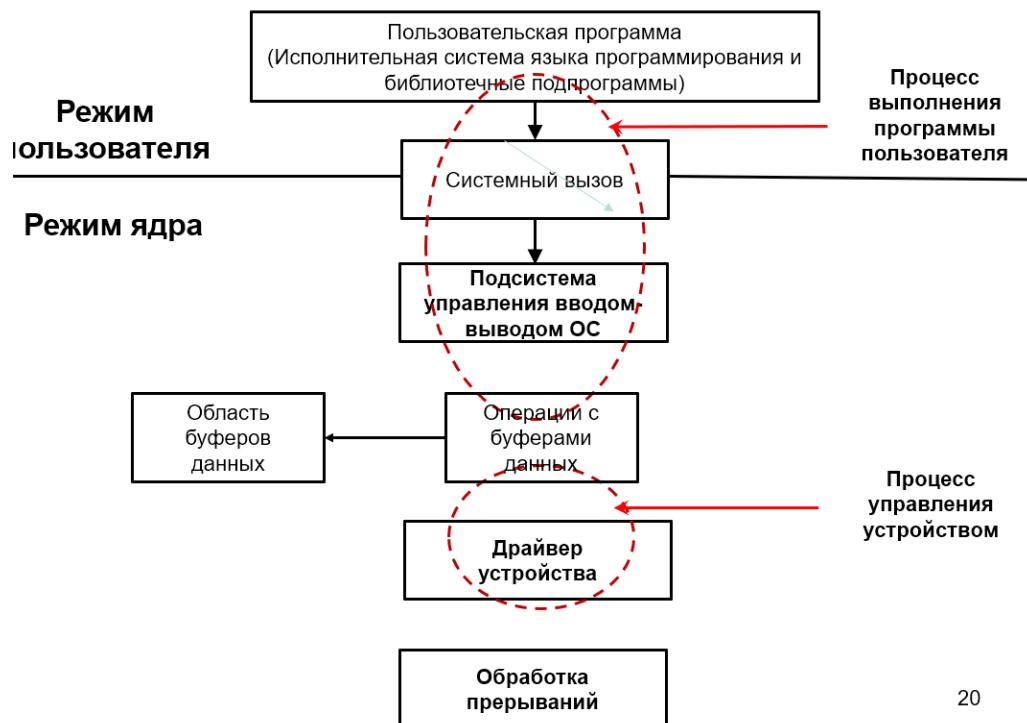
ЗАДАЧИ П.О. ВВ-ВЫВ:

- Создание виртуального интерфейса, позволяющего прикладным программистам работать с внешними устройствами, не обращая внимания на специфику устройств
- Использование блокирующих (синхронных) и неблокирующих(асинхронных) передач данных
- Использование устройств ввода-вывода:
 - разделяемых (параллельно-используемых) и
 - неразделяемых (последовательно-используемых)



19

Процессы В-В:



20

УРОВЕНЬ ОБРАБОТКИ ПРЕРЫВАНИЙ:

Процедура обработки прерывания (обработчик прерывания) вызывается при поступлении прерывания от контроллера устройства, сигнализирующего о завершении команды ввода-вывода

УРОВЕНЬ ДРАЙВЕРОВ УСТРОЙСТВ В-В:

Каждый драйвер управляет устройствами одного типа. Драйвер реализует функции ввода-вывода, исходя из конкретных особенностей соответствующего устройства

Драйвер преобразует операцию ввода-вывода в последовательность команд контроллера устройства ввода-вывода

Драйвер либо блокируется до завершения операции ввода-вывода, либо драйвер ожидает ее завершения без блокирования.

СПОСОБЫ УСТАНОВКИ ДРАЙВЕРОВ УСТРОЙСТВ В ЯДРО ОС:

Заново компонуется ядро ОС вместе с новым драйвером и затем перезагружается система (множество UNIX)

В параметрах ОС отмечается, что требуется драйвер, и затем требуется перезагрузка системы. Во время начальной загрузки операционная система находит нужные драйверы и загружает их (Windows)

Операционная система может принимать новые драйверы, не прерывая работы, и динамически устанавливать их, не нуждаясь при этом в перезагрузке

НЕЗАВИСИМЫЙ ОТ УСТРОЙСТВ УРОВЕНЬ ОС (ПОДСИСТЕМА УПРАВЛЕНИЯ ВВОДОМ-ВЫВОДОМ ОС)

Типичные функции подсистемы:

- ◆ 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 УРОВЕНЬ ПО (программные средства обращения к подсистеме в-в):

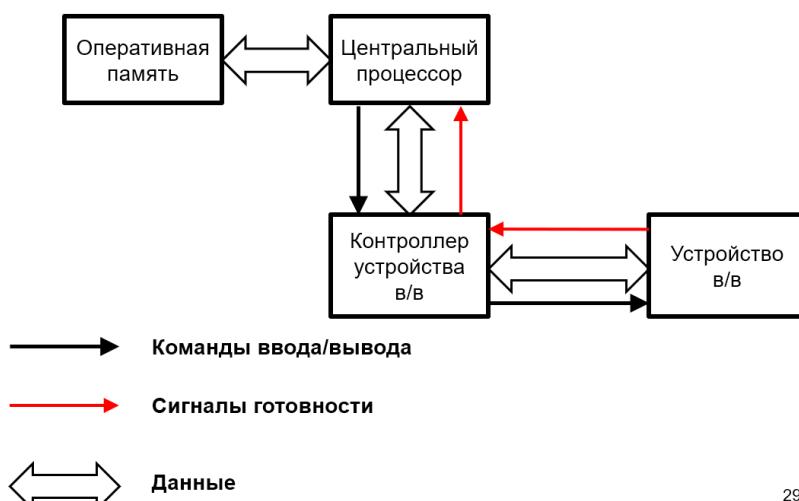
-Системные вызовы ввода-вывода

-Часть программного обеспечения ввода-вывода, содержащаяся в стандартных библиотеках, связываемых с пользовательскими программами.

15. Способы ввода-вывода. Ввод-вывод без использования и с использованием прерываний. Прямой доступ к памяти. Буферизация ввода-вывода.

1. Обмен данных между устройством ввода-вывода и оперативной памятью с использованием центрального процессора

1.1 Программируемый ввод-вывод без использования прерываний (режим опроса готовности).



29

Режим синхронного управления (активное ожидание готовности устройства)

Пользовательская программа выдает системный запрос операции ввода-вывода

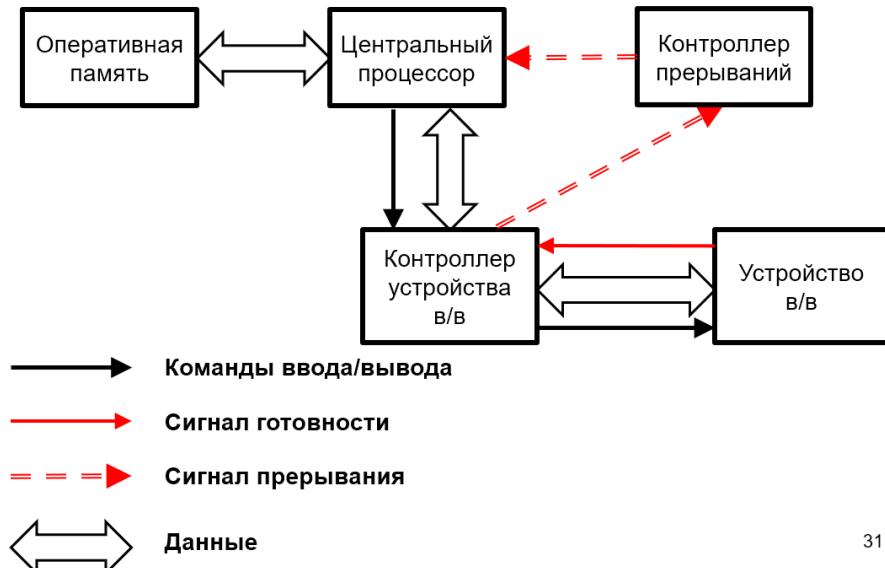
Ядро ОС транслирует запрос в вызов соответствующего драйвера устройства ввода-вывода

Драйвер выполняет ввод-вывод, выполняя команду опроса готовности устройства для выполнения каждой следующей команды ввода вывода

После завершения ввода вывода драйвер передаёт данные и возвращается в исходное состояние

ОС возвращает управление пользовательской программе

1.2 Ввод-вывод, управляемый прерыванием (режим обмена с прерыванием / асинхронного выполнения ввода-вывода)



31

Драйвер передаёт команду контроллеру ввода/вывода устройство

После выполнения команды чтения или записи контроллер устройства посыпает сигнал контроллеру прерываний

Контроллер прерываний подаёт сигнал процессору и выставляет номер устройства

Чтобы не потерять связь с устройством после выдачи команды ввода/вывода и переключения на другие процессы, устанавливается тайм-аут, после которого, если команда ввода-вывода не выполнена, делается вывод о потере связи с устройством

2. Прямая передача данных из устройства ввода-вывода в оперативную память и из оперативной памяти в устройство ввода-вывода с использованием процессора ввода-вывода (прямой доступ к памяти)

Используется контроллер прямого доступа к памяти (DMA, Direct Memory Access), который управляет потоком данных между оперативной памятью и контроллерами устройств ввода-вывода без постоянного участия центрального процессора.

DMA-контроллер может получать доступ к системной шине независимо от центрального процессора

DMA-контроллер содержит регистры, доступные центральному процессору для чтения и записи.

К ним относятся регистр адреса памяти, счетчик байтов и один или более управляющих регистров.

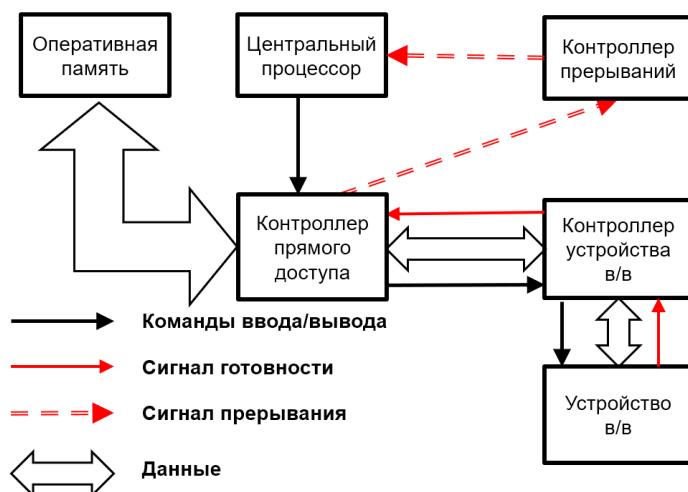
Управляющие регистры задают:

какой порт ввода-вывода должен быть использован,

направление переноса данных (чтение из устройства ввода-вывода или запись в него),

единицу переноса (осуществлять перенос данных побайтно или пословно), число байтов, которые следует перенести за одну операцию.

1. ЦП устанавливает регистры DMA-контроллера
 2. DMA-контроллер посыпает команду ввода-вывода контроллеру устройства ввода-вывода
 3. Контроллер устройства ввода-вывода выполняет команду ввода-вывода
 4. DMA-контроллер производит обмен данными между ОП и устройством ввода-вывода
 5. Контроллер устройства ввода-вывода подтверждает DMA-контроллеру окончание обмена
- Шаги 2 - 5 повторяются до окончания выполнения операции ввода-вывода
6. После выполнения операции ввода-вывода DMA-контроллер с помощью сигнала прерывания на контроллер прерываний ЦП сообщает об окончании операции



БУФЕРИЗАЦИЯ:

Процессы прикладного уровня могут запрашивать ввод и вывод произвольного количества данных. Однако устройства, с которыми они работают, способны передавать данные фиксированными порциями.

Поэтому между модулями ввода-вывода ОС, которые вызывает приложение (сверху вниз), и модулями управления устройством (снизу-вверх), необходимы буфера данных.

Буфера являются критическим ресурсом в отношении процесса выполнения программы пользователя и процесса управления устройством, которые при параллельном своем выполнении информационно взаимодействуют.

Через буфер(буфера) данные либо посыпаются от прикладного процесса к адресуемому внешнему устройству (операция вывода данных на внешнее устройство), либо от внешнего устройства передаются некоторому прикладному процессу (операция считывания данных с внешнего устройства).



ОДИНАРНАЯ БУФЕРИЗАЦИЯ:

Примерная оценка общего времени выполнения, приходящегося на ввод одного блока данных для блочно-ориентированных устройств

Без буферизации:

$$Тобщ = T + C$$

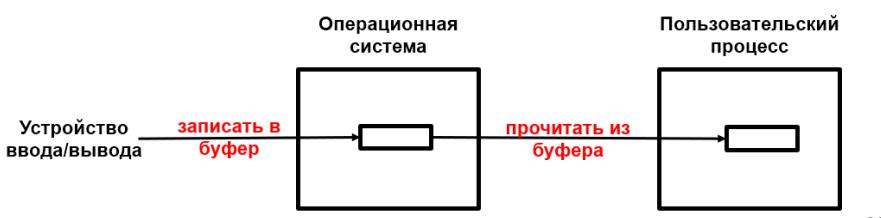
T — это время, необходимое для ввода одного блока,

C — время, необходимое для вычислений, выполняющихся между запросами на ввод данных.

При использовании одинарной буферизации:

$$Тобщ = \max[C, T] + M,$$

где **M** — время, необходимое для перемещения данных из системного буфера в пользовательскую память. **В большинстве случаев это время значительно меньше времени работы без буферизации.**



ДВОЙНАЯ БУФЕРИЗАЦИЯ:

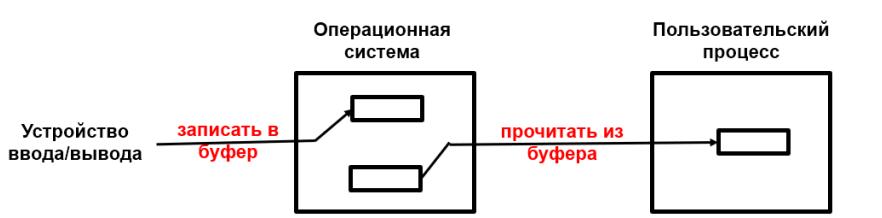
T — это время, необходимое для ввода одного блока,

C — время, необходимое для вычислений, выполняющихся между запросами на ввод данных.

Если **C < T**, то блочно-ориентированное устройство может работать с максимальной скоростью.

Если же **C > T**, то двойная буферизация избавляет процесс от необходимости ожидания завершения ввода-вывода.

В любом случае достигается преимущество перед одинарной буферизацией. Это улучшение буферизации осуществляется за счет увеличения ее сложности



ЦИКЛИЧЕСКАЯ:

Схема двойного буфера призвана выровнять поток данных между устройством ввода-вывода и прикладным процессом.

Двойная буферизация может оказаться недостаточной, если процесс часто выполняет ввод или вывод. В таком случае иногда решить проблему помогает наращивание **количество буферов**.

При использовании множества буферов, состоящего более чем из двух, схема именуется циклической буферизацией. В ней каждый индивидуальный буфер представляет собой модуль циклического буфера.

Такая буферизация описывается моделью **производителя/потребителя с ограниченным буфером**.



16. Логическая организация файла. Атрибуты и данные файла. Логическая организация данных файла. Операции с каталогами и файлами

Под **файлом** понимают некоторый набор данных, связанные с этим набором атрибуты (имя, размер и т.д.), множество допустимых операций над атрибутами и данными

Логическая организация файла:



Атрибуты:

- Список атрибутов файлов зависит от конкретной файловой системы
- При создании файла файлу присваивается основной атрибут – его имя. По имени осуществляют доступ к файлу.
- Точные правила именования файлов варьируются от системы к системе. В именах файлов разрешается использование букв, цифр и специальных символов.
- Файловые системы могут поддерживать короткие или длинные имена файлов.
- В некоторых файловых системах в именах файлов различаются прописные и строчные символы.

Возможные атрибуты:

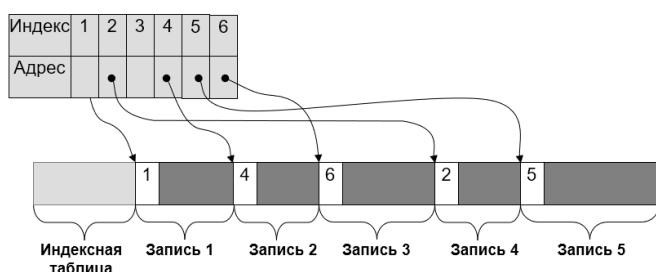
Атрибут	Значение
Имя	Имя файла
Защита	Кто и каким образом может получать доступ к файлу
Пароль	Пароль для получения доступа к файлу
Создатель	Идентификатор пользователя, создавшего файл
Владелец	Текущий владелец
Флаг «только чтение»	0 - для чтения/записи; 1 - только для чтения
Флаг «скрытый»	0 - нормальный; 1 - не показывать в перечне файлов каталога
Флаг «системный»	0 - нормальный; 1 - системный
Флаг «архивный»	0 — заархивирован; 1 - требуется архивация
Флаг ASCII/двоичный	0- ASCII; 1 - двоичный
Флаг произвольного доступа	0 - только последовательный доступ; 1 - произвольный доступ
Флаг «временный»	0 — нормальный; 1 — для удаления файла по окончании работы процесса
Длина записи	Количество байтов в записи
Время создания	Дата и время создания файла
Время последнего доступа	Дата и время последнего доступа файла
Время последнего изменения	Дата и время последнего изменения файла
Текущий размер	Количество байтов в файле
Максимальный размер	Количество байтов, до которого можно увеличивать размер файла

Данные файла:

Поле (field)	основной элемент данных. Содержит единственное значение. Характеризуется длиной и типов данных(например , строка ASCII, десятичное число и т.п.). В зависимости от структуры файла поля <u>м.б.</u> либо фиксированной , либо переменной длины .
Запись (record)	набор связанных между собой полей , которые могут быть обработаны как единое целое некоторой прикладной программой. В зависимости от структуры записи могут быть фиксированной или переменной длины . Запись имеет переменную длину, если некоторые из ее полей – переменной длины или если переменно количество полей в записи.
Файл (file)	набор данных, организованных в виде <ul style="list-style-type: none"> ▪ совокупности записей одинаковой структуры(однородных записей) и ▪ совокупности атрибутов, определяющих характеристики файла

Логическая организация данных файла:

- Неструктурированная последовательность байтов (поток байтов)
- Последовательная организация с логическими записями фиксированной длины
- Последовательная организация с логическими записями переменной длины
- Организация размещения записей файлов с использованием таблицы индексов (индексированный файл):



Каталог - структурированный объект, состоящий из списка элементов

Каждый элемент каталога содержит информацию:

- либо о файле, включённом в данный каталог,
- либо о каталоге, являющимся каталогом нижнего уровня по отношению к данному

Каталоги образуют иерархическую структуру файловой системы

Каталоги и файлы в такой структуре именуются относительно каталога верхнего уровня файловой системы (корневого каталога)

*Некоторые операции с каталогами,
доступные клиентам файловой системы*

- **Создание** каталога
- **Удаление** каталога
- **Установка** прав доступа к каталогу
- **Создание** файла
- **Удаление** файла
- **Создание символической ссылки на**
файл другого каталога
- **Удаление символической ссылки на**
файл другого каталога

*Некоторые операции с файлами,
доступные клиентам файловой системы*

- **Открытие** файла
- **Запись** в файл
- **Чтение** из файла
- **Установка** прав доступа к файлу
- **Закрытие** файла

87

17. Физическая реализация файловой системы. Понятие раздела, логического тома, файловой системы.

Физическая реализация файловых систем решает вопросы

- хранения файловых систем,
- физической организации файлов и каталогов файловой системы,
- управления дисковым пространством,
- обеспечения надежной и эффективной работы файловой системы.

Объекты физической реализации ФС:

Физический жёсткий диск (или совместимое внешнее устройство)

Раздел (partition) – непрерывная часть адресного пространства физического диска

Логический том – область, занимаемая файловой системой (адресное пространство файловой системы)

Файловая система – физическая организация логического тома для хранения файлов и каталогов



Физическая адресация данных на жёстком диске

Сектор(sector) – минимальный блок хранения и передачи информации

Дорожки(track) – концентрические магнитные дорожки на поверхности диска

Цилиндр(cylinder) – группа дорожек одного радиуса

Физический адрес сектора на диске:

[c-h-s]

с – номер цилиндра(cylinder)

h – номер рабочей поверхности(head)

s – номер сектора на дорожке(sector)

Этапы форматирования жесткого диска

↑Низкоуровневое форматирование, Low-Level formating (физическая разметка диска на цилиндры, дорожки, секторы)

-Разбиение диска на разделы (partition)

-Создание логических томов файловых систем

-Высокоуровневое форматирование логических томов (форматирование файловых систем)

Разбиение жёсткого диска на разделы

- С использованием таблицы разделов PT (partition table) и MBR (Master Boot Record)

Разделы:

первичные (primary)

расширенные (extended)

логические диски (logical)

- С использованием таблицы разделов GPT (GUID Partition Table)

Допускает неограниченное количество разделов. Лимит устанавливает операционная система (Windows допускает не более 128 разделов).

18.Модели размещения логических томов файловых систем. Форматирование логических томов.

1. Модель на основе создания логического тома в разделе физического устройства

Каждый жёсткий диск делится на один или несколько разделов, каждый из которых может содержать один логический том и одну файловую систему

2. Модель на основе создания логического тома в нескольких разделах одного или нескольких физических устройств

Концепция диспетчера томов, обеспечивающая представление нескольких устройств в виде одного устройства (динамические диски и логические тома)

Диспетчер томов поддерживает изменение конфигурации логического тома

3. Модель на основе создания логического тома в пуле устройств хранения данных

Концепция объединения устройств в пул устройств хранения данных. Пул устройств хранения данных описывает физические характеристики хранения (размещение устройств, избыточность данных и т.д.) и выступает в качестве хранилища данных для создания файловых систем. Файловые системы автоматически расширяются в рамках пространства, выделенного для пула

Стратегия форматирования логических томов

Адресное пространства логического тома разбивается на дисковые блоки данных

Файловая система хранит адреса дисковых блоков и информацию об их состоянии

Выбор размера дискового блока

Дисковый блок данных состоит из одного или нескольких смежных секторов

Минимальный размер блока – один сектор

Маленький размер блока – хорошее использование дискового пространства, низкая производительность

Большой размер блока – высокая производительность, неэффективное использование дискового пространства

Учет свободных и занятых блоков

Связные списки свободных и занятых дисковых блоков

Хранение информации о свободных и занятых блоках в виде битового массива

19.Физическая реализация хранения атрибутов файлов. Каталоги и индексные узлы.

Физическая реализация хранения файла

Хранение файла

=

Хранение атрибутов файла

+

Хранение данных файла

1. Хранение атрибутов файла в каталоге

Все атрибуты файла хранятся только в элементе каталога, относящегося к данному файлу.

Каталог состоит из списка элементов фиксированной длины

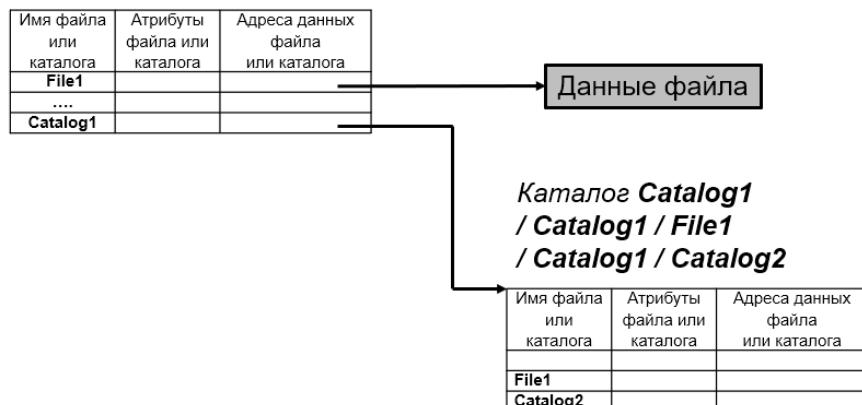
Каждый элемент списка описывает файл (атрибуты файла и дисковые адреса блоков данных файла) или каталог нижнего уровня

Корневой каталог файловой системы создаётся при форматировании логического тома

Корневой каталог

/File1

/Catalog1

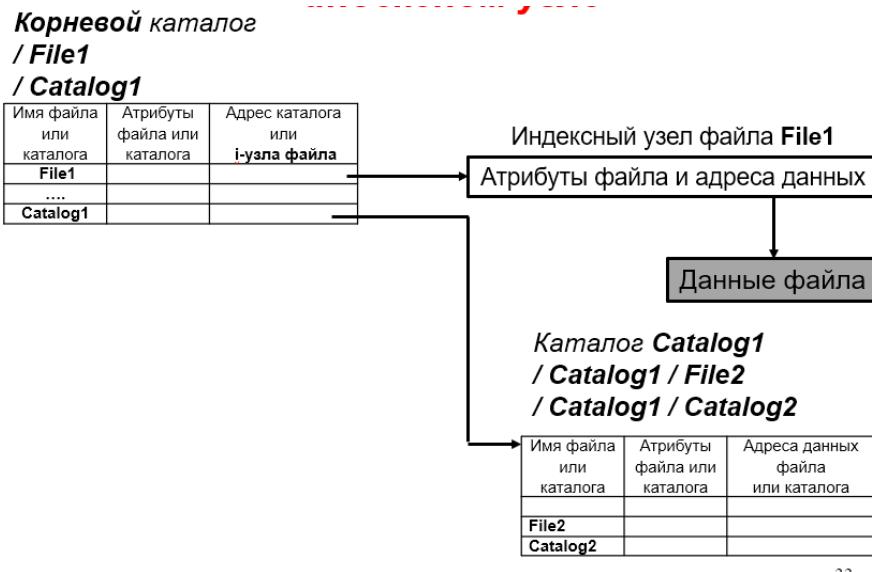


2. Хранение атрибутов файла в каталоге и индексном узле

Атрибуты файла хранятся как в элементе каталога, так и в специальной структуре данных (индексном узле)

Каталог содержит имя файла и адрес индексного узла

Индексный узел содержит атрибуты файла и дисковые адреса блоков данных файла



22

20. Физическая реализация хранения данных файла. Способы адресации блоков данных.

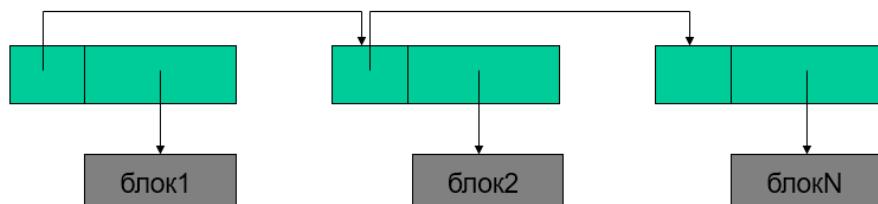
Физическая реализация хранения данных файла

1. Непрерывные файлы

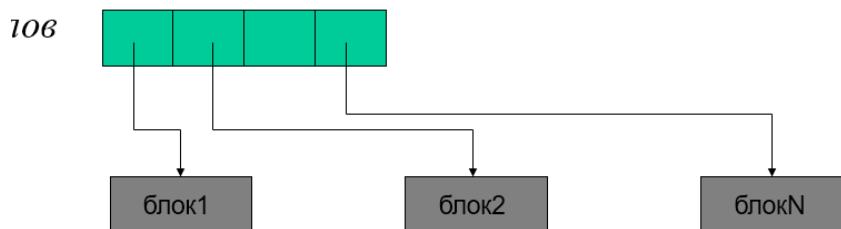
2. Связные списки



3. Связный список с таблицей адресов блоков данных



4. Адресация блоков данных с помощью индексных узлов



1) прямую адресацию;

эффективный адрес содержится в самой команде и для его формирования не используется никаких дополнительных источников или регистров. Эффективный адрес берется непосредственно из поля смещения машинной команды

Относительная прямая адресация

Используется для команд условных переходов, для указания относительного адреса перехода. В поле смещения машинной команды содержится N-битное значение, которое в результате работы команды будет

складываться с содержимым регистра указателя команды ip/eip. В результате такого сложения получается адрес, по которому и осуществляется переход.

Абсолютная прямая адресация

в этом случае эффективный адрес является частью машинной команды, но формируется этот адрес только из значения поля смещения в команде.

2) косвенную базовую (регистровую) адресацию;

При такой адресации эффективный адрес операнда может находиться в любом из регистров общего назначения,

Для того, чтобы небольшая структура индекса позволяла работать с большими файлами, таблица адресов дисковых блоков приводится в соответствие со структурой, представленной на Рисунке 4.6. Версия V системы UNIX работает с 13 точками входа в таблицу адресов индекса, но принципиальные моменты не зависят от количества точек входа. Блок, имеющий пометку "прямая адресация" на рисунке, содержит номера дисковых блоков, в которых хранятся реальные данные. Блок, имеющий пометку "одинарная косвенная адресация", указывает на блок, содержащий список номеров блоков прямой адресации. Чтобы обратиться к данным с помощью блока косвенной адресации, ядро должно считать этот блок, найти соответствующий вход в блок прямой адресации и, считав блок прямой адресации, обнаружить данные. Блок, имеющий пометку "двойная косвенная адресация", содержит список номеров блоков одинарной косвенной адресации, а блок, имеющий пометку "тройная косвенная адресация", содержит список номеров блоков двойной косвенной адресации.

В принципе, этот метод можно было бы распространить и на поддержку блоков четверной косвенной адресации, блоков пятерной косвенной адресации и так далее, но на практике оказывается достаточно имеющейся структуры

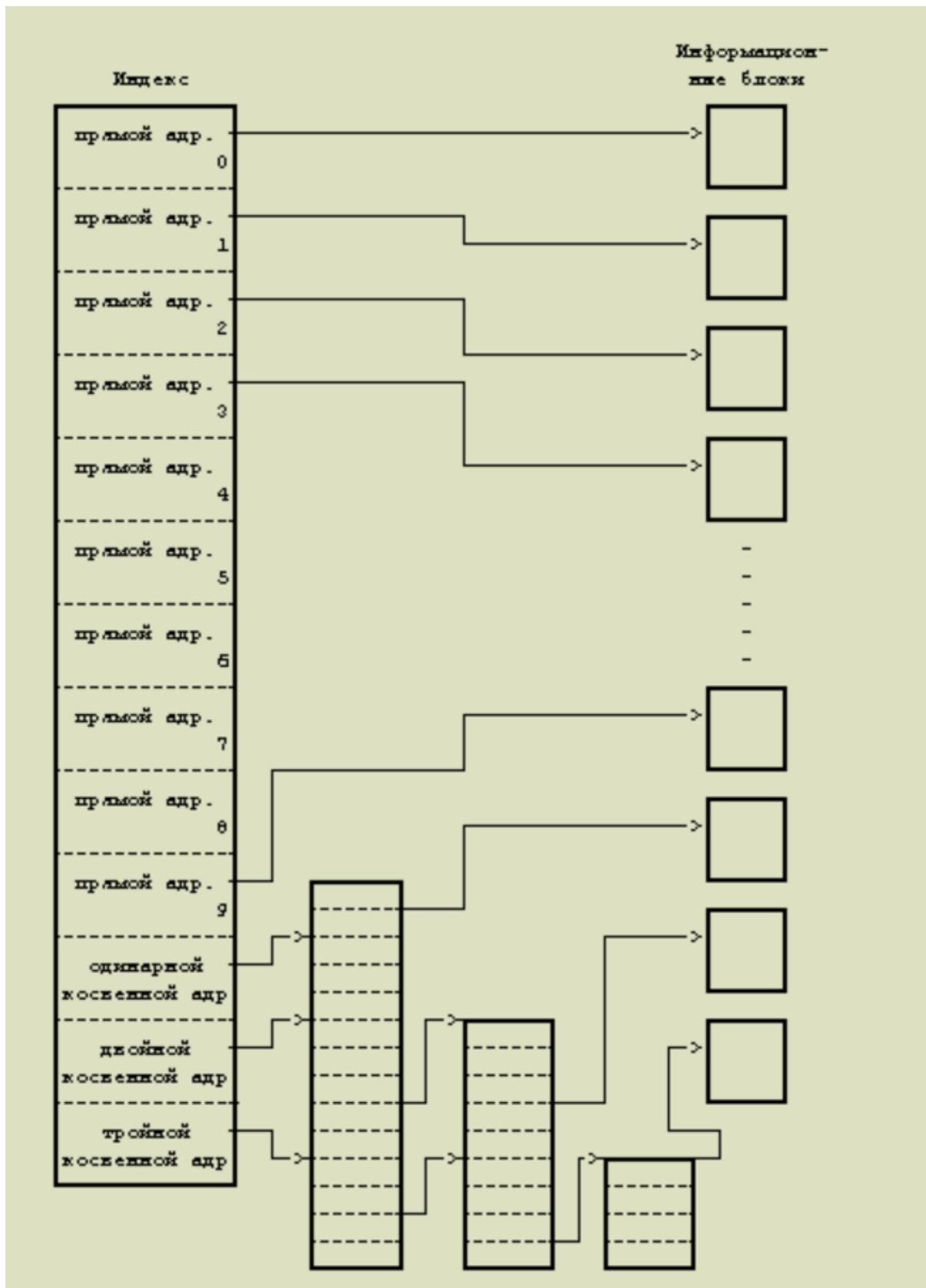


Рисунок 4.6. Блоки прямой и косвенной адресации в индексе

Процессы обращаются к информации в файле, задавая смещение в байтах. Они рассматривают файл как поток байтов и ведут подсчет байтов, начиная с нулевого адреса и заканчивая адресом, равным размеру файла. Ядро переходит от байтов к блокам: файл начинается с нулевого логического блока и заканчивается блоком, номер которого определяется исходя из размера файла. Ядро обращается к индексу и

превращает логический блок, принадлежащий файлу, в соответствующий дисковый блок

```
алгоритм bmap /* отображение адреса смещения в байтах от
                  начала логического файла на адрес блока
                  в файловой системе */
входная информация: (1) индекс
                      (2) смещение в байтах
выходная информация: (1) номер блока в файловой системе
                      (2) смещение в байтах внутри блока
                      (3) число байт ввода-вывода в блок
                      (4) номер блока с продвижением
```

Преобразование адресов с большими смещениями, в частности с использованием блоков тройной косвенной адресации, является сложной процедурой, требующей от ядра обращения уже к трем дисковым блокам в дополнение к индексу и информационному блоку. Даже если ядро обнаружит блоки в буферном кеше, операция останется дорогостоящей, так как ядру придется многократно обращаться к буферному кешу и приостанавливать свою работу в ожидании снятия блокировки с буферов. Насколько эффективен этот алгоритм на практике? Это зависит от того, как используется система, а также от того, кто является пользователем и каков состав задач, вызывающий потребность в более частом обращении к большим или, наоборот, маленьким файлам. Однако, как уже было замечено [Mullender 84], большинство файлов в системе UNIX имеет размер, не превышающий 10 Кбайт и даже 1 Кбайта ! (*) Поскольку 10 Кбайт файла располагаются в блоках прямой адресации, к большей части данных, хранящихся в файлах, доступ может производиться за одно обращение к диску. Поэтому в отличие от обращения к большим файлам, работа с файлами стандартного размера протекает быстро.

На примере 19978 файлов Маллендер и Танненбаум говорят, что приблизительно 85% файлов имеют размер менее 8 Кбайт и 48% - менее 1 Кбайта.

21. Совместное использование файлов. Жёсткие и символические ссылки на атрибуты и данные файла.

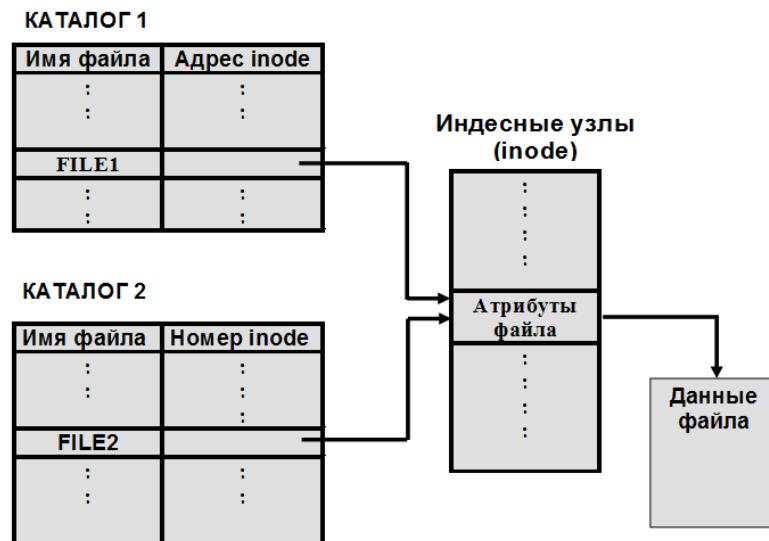
При совместном использовании файла несколькими процессами, изменения, вносимые в атрибуты и данные файла каким-либо процессом, должны быть видны другим процессам, использующим данный файл.

Два способа решения проблемы:

информация об атрибутах и блоках данных, занимаемых файлом, содержится не в каталоге, а в связанном с данным файлом индексном узле. В этом случае записи в каталогах будут просто указывать на эту структуру данных (жёсткая ссылка).

элемент каталога содержит символическую ссылку на элемент другого каталога, имеющего жёсткую ссылку на данные (мягкая ссылка)

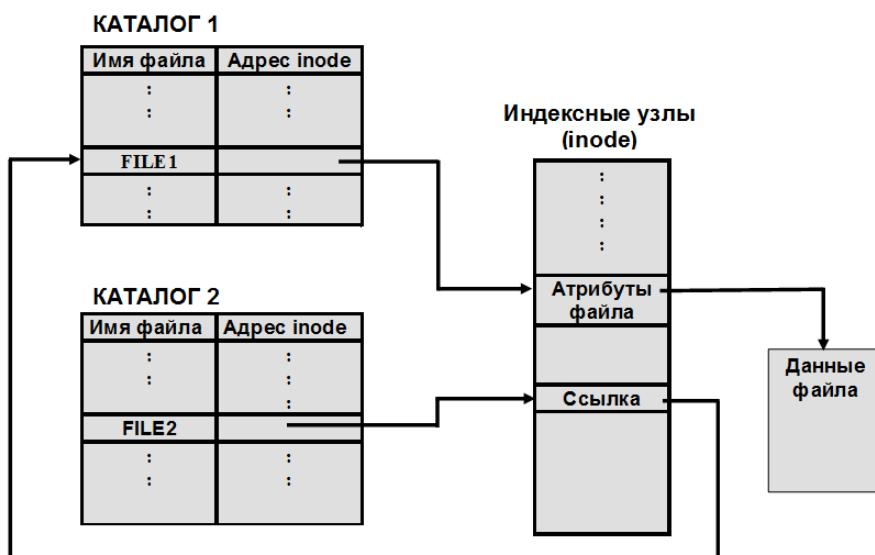
Жесткая связь:



Microsoft использует жесткие ссылки для обеспечения совместимости и безотказного обслуживания операционной системы

Жёсткие ссылки (hard links) - это просто файловые записи, ссылающиеся на одно и то же место в области данных. После создания жёсткой ссылки абсолютно невозможно различить, какой из файлов был "главным", а какой был создан как ссылка; теперь это два абсолютно равноправных файла. Создание жёстких ссылок разрешено только в пределах одного NTFS-раздела. Т.е. создать на одном разделе ссылку на файл, находящийся на другом разделе, невозможно. При удалении файла, счётчик которого больше единицы, удаляются не его данные, а только ссылка на них. Сами данные будут уничтожены только тогда, когда счётчик достигает значения 0, т.е. когда в системе больше нет ни одной записи, ссылающейся на эти данные.

Мягкая (символическая):



Символическая ссылка на Unix (soft link) - это обычный файл, у которого установлен определённый атрибут, а в качестве содержимого прописано то, на что он указывает. По сути это очень сильно напоминает ярлыки в Windows. В случае символьических ссылок есть чётко выраженный "главный" файл или каталог, и одна или несколько ссылок, которые на него указывают. Если этот файл/каталог переименовать, переместить или удалить, то все символьические ссылки, указывавшие на него, станут нерабочими. При удалении "главного" файла данные уничтожаются сразу (если, конечно, у него нет ещё и жёстких ссылок!), никакого учёта символьических ссылок в файловой системе нет.

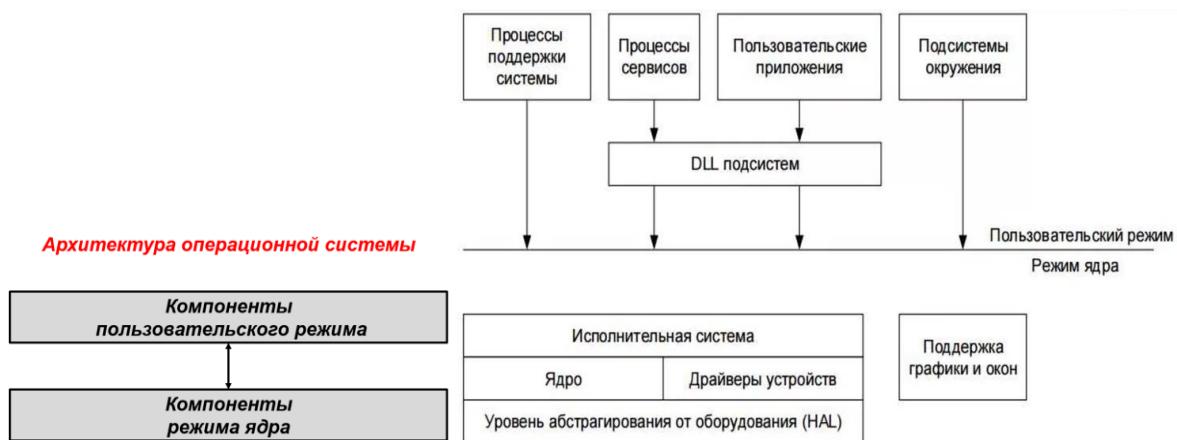
Символьная ссылка занимает ровно столько места на файловой системе, сколько требуется для записи её содержимого (нормальный файл занимает как минимум один блок раздела).

Непротиворечивость файловой системы

Традиционная файловая система	В промежутке времени между выполнением операций файловая система остается в противоречивом состоянии(команды fsck используется для просмотра и проверки состояния файловой системы с попыткой устранения противоречивости).
Файловая система с журналированием	Файловые операции регистрируются в отдельном журнале, и при необходимости могут быть безопасно воспроизведены.
Транзакционная файловая система	Любая последовательность операций либо полностью выполняется, либо полностью игнорируется. Состояние файловой системы всегда является непротиворечивым.

40

22.Архитектура операционной системы Windows. Компоненты пользовательского режима и режима ядра.



§Приложения пользовательского режима взаимодействуют с исполнительной системой опосредованно, через защищенные подсистемы среды, которые реализуют интерфейсы прикладного программирования.

§Защищенные подсистемы среды могут либо взаимодействовать с клиентскими приложениями

-либо по принципу клиент-сервер,

-либо функционировать как совместно используемые библиотеки, связываемые с клиентскими приложениями во время их компоновки.

§На практике часто используется сочетание этих двух механизмов.

§Благодаря такой организации Windows соединяет в себе достоинства микроядерной и расширяемой библиотечной архитектур.

Компоненты пользовательского режима

•**Вспомогательные системные процессы** – процесс входа в систему, авторизация локальных пользователей и т.д.

•**Служебные процессы ОС** – диспетчер задач, спулер печати и т.д.

•**Пользовательские приложения** -32 и 64-разрядные приложения Windows

• **Подсистемы среды окружения** – реализация интерфейсов прикладного программирования для Windows и Posix

•**Подсистема DLL-библиотек** – обеспечение интерфейса приложений с исполнительной системой

Компоненты режима ядра

ØИсполняющая система содержит основные службы ОС:

§управление памятью,

§управление процессами и потоками,

§безопасность,

§ввод-вывод,

§взаимодействие между процессами

§сетевые службы.

ØДиспетчер объектов.

§создает и удаляет **объекты** исполняющей системы,

§реализует **унифицированный механизм управления объектами** и хранения соответствующих данных и используется всеми компонентами исполнительной подсистемы.

Диспетчер объектов (внутреннее название *Ob*) - это подсистема, реализованная как часть Windows Executive , которая управляет ресурсами Windows. Диспетчер

объектов - это общий ресурс, и все подсистемы, которые имеют дело с ресурсами, должны проходить через диспетчер объектов. Википедия site:star-wiki.ru



Ядро операционной системы

ФУправление процессами

ФДиспетчеризация потоков

ФУправление виртуальной памятью

ФДиспетчеризация прерываний

ФРеализация базовых синхронизационных примитивов

Уровень аппаратных абстракций (Hardware Abstraction Layer, HAL)

Изолирует ядро, драйверы устройств и остальную исполняющую систему Windows от аппаратных различий конкретных платформ (таких, как различия между материнскими платами).

Включает программы, предназначенные для конкретного аппаратного обеспечения, и «изолирует» систему от особенностей последнего (программы взаимодействия с контроллером прерываний, управления взаимодействием между центральными процессорами в мультипроцессорной системе и т.п.).

23. Объектная модель ОС Windows. Понятие объекта. Диспетчер объектов и диспетчер типов объектов.

Объектная модель Windows

Объекты – все системные ресурсы и структуры данных (процессы, потоки, файлы, семафоры и т.д.)

Объекты разделяются на типы.

Объект – экземпляр объектного типа

Каждый тип объекта поддерживается определённой подсистемой

Объекты:

- постоянные(файлы)*
- динамические(потоки)*

Схема именования объектов - иерархическая, с помощью объектов-каталогов пространства имён объектов

*(Device - каталог объектов обнаруженных устройств ввода-вывода
Driver - каталог объектов загруженных драйверов)*

Использование объектов:

- Открытие(Создание) объекта – возвращается дескриптор объекта*
- Работа с объектом – использование методов(интерфейсных операций)*
- Закрытие объекта*

Объектная модель Windows

Создание объектов:

Диспетчер типа объектов + Диспетчер объектов

диспетчер объектов Windows - компонент исполняющей системы, отвечающий за создание, удаление, защиту и отслеживание объектов.

Согласно своему внутреннему устройству у Windows есть три типа объектов: объекты исполняющей системы, объекты ядра и объекты GDI/User. Объекты исполняющей системы представлены объектами, реализованными различными компонентами исполняющей системы (такими как диспетчер процессов, диспетчер памяти, подсистема ввода-вывода и т. д.). Объекты ядра представлены более простым набором, реализованным ядром Windows. Объекты ядра обеспечивают такие основные возможности, как синхронизация, на которых построены объекты исполняющей системы.

(Создать процесс – Диспетчер процессов+ Диспетчер объектов;

Создать файл – Диспетчер ввода-вывода + Диспетчер объектов)

Структура объекта:

- Заголовок объекта(имя, каталог, защита, указатель на типовой объект, методы и т.д.)*
- Тело объекта – данные, специфические для конкретного объекта*

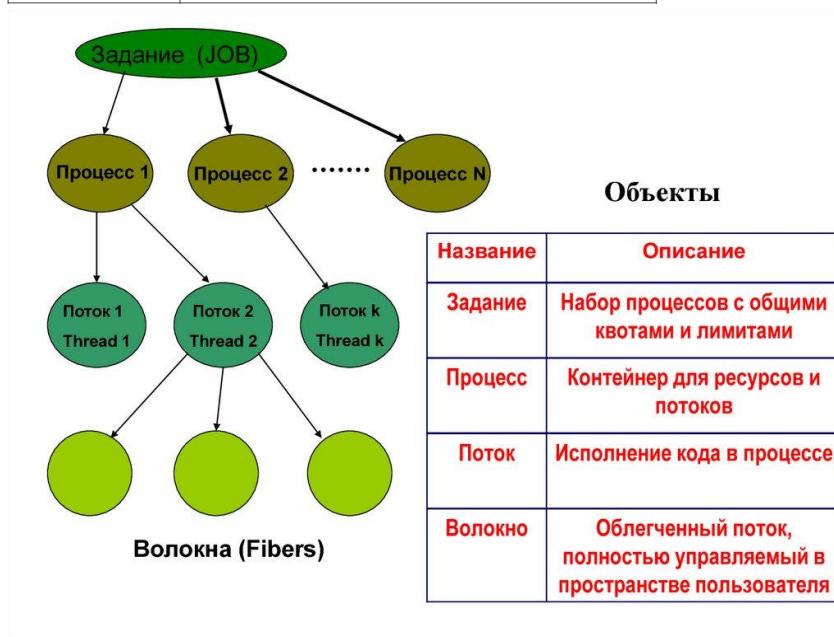
Типовой объект содержит набор атрибутов, общих для всех объектов данного типа

Структура данных типового объекта связывает между собой все объекты соответствующего типа

24.ОС Windows. Управление процессами. Объекты «процесс» и «поток». Состояния потока и график перехода между состояниями.

Основные объекты, используемые для управления процессами в ОС Windows

Название	Описание
Процесс	Контейнер ресурсов(единица планирования ресурсов)
Поток	Объект диспетчеризации(единица выполнения процесса)
Волокно	Объект для организации псевдопараллелизма в составе потока
Задание	Объект для управления группами процессов как единым целым



Объект «Процесс» ОС Windows

Процесс – контейнер ресурсов, используемых при выполнении программы. Процессом обычно называют экземпляр выполняемой программы.

Windows-процесс включает:

- **исполняемую программу**, определяющую исходный код и данные, и отображаемую на виртуальное адресное пространство процесса(ВАП);
- **закрытое виртуальное адресное пространство процесса(ВАП)**, отображаемое на виртуальную память операционной системы;

- **перечень открытых дескрипторов системных ресурсов (семафоров, коммуникационных портов и файлов, доступных всем потокам процесса);**
- **связанную с процессом среду безопасности (маркер доступа)**
- **的独特的 идентификатор процесса (process ID)**
- **как минимум один поток выполнения**
- **каждый процесс также указывает на свой родительский процесс**

Создание объекта «Процесс»

CreateProcess <имя исполняемого модуля, параметры безопасности, ...>
 ∅ Создается объект «раздел» и открывается файл исполняемого модуля

∅ Создается объект «процесс»

∅ Создается объект «поток» в контексте данного процесса

∅ Загружаются DLL и т.п.

∅ Начинается выполнение **первичного потока**

После создания процесса родительский и дочерний процессы обладают своими собственными, отдельными адресными пространствами операционной системы

Объект «поток»

Поток (thread) - некая сущность внутри процесса, получающая процессорное время для выполнения. В каждом процессе есть минимум один поток. Этот первичный поток создается системой автоматически при создании процесса. Далее этот поток может породить другие потоки, те в свою очередь новые и т.д. Таким образом, один процесс может владеть несколькими потоками, и тогда они одновременно исполняют код в адресном пространстве процесса. Каждый поток имеет:

∅ **Идентификатор потока**

∅ **Содержимое регистров ЦП, отображающее состояние потока**

∅ **Два стека:**

- используемый в режиме ядра
- используемый в пользовательском режиме

∅ **Локальная память** потока

∅ **Собственный маркер доступа**

Создание объекта «Поток»

CreateThread <дескриптор защиты, размер стека, начальный адрес потока...>

Локальное хранилище потока-*thread-local storage (TLS)* предназначено для использования подсистемами, библиотеками времени выполнения(RTL) и DLL-библиотеками.

∅Контекст потока - регистры, стеки и локальное хранилище потока

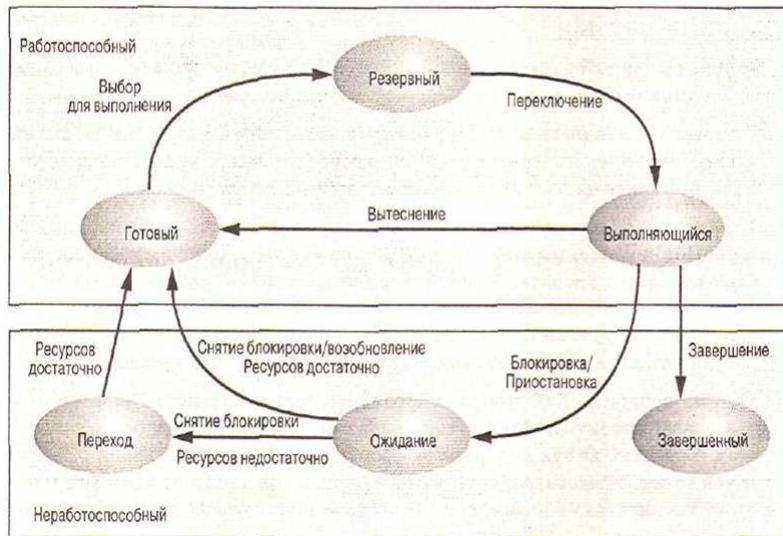
Состояния потока

- Инициализирован (*Initialized*)
- Готов (*Ready*)
- В повышенной готовности (*Standby*)
- Готов, но отложен (*Deferred ready*)
- Выполняется (*Running*)
- Ожидает (*Waiting*)
- В переходном состоянии (*Transition*)
- Завершен (*Terminated*)

Граф переходов между состояниями потока



Граф состояний потока в Windows



Уровни приоритетов потоков



25.ОС Windows. Виртуальные адресные пространства процессов и потоков.

Виртуальное адресное пространство процесса – это совокупность адресов, которыми может манипулировать программный модуль процесса. Операционная система отображает виртуальное адресное пространство процесса на отведенную процессу физическую память.

Виртуальное адресное пространство процесса(ВАП)

ОС Windows выделяет половину ВАП процессам для их уникальных закрытых хранилищ(**пользовательское ВАП**).

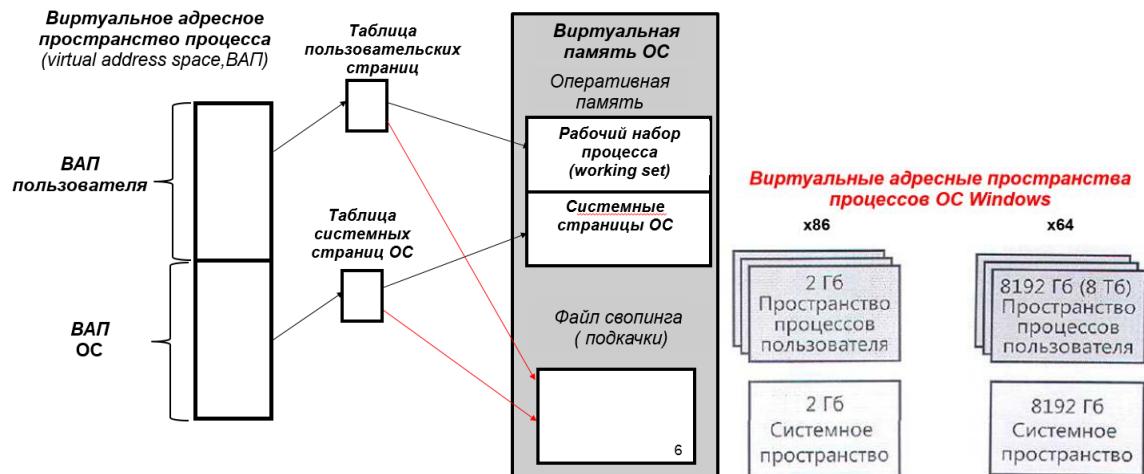
Вторую половину ВАП ОС Windows использует в качестве своей собственной защищенной памяти операционной системы(**ВАП ОС**)

Виртуальное адресное пространство по умолчанию для 32-разрядного Windows

В следующей таблице показан диапазон памяти по умолчанию для каждой секции.

Диапазон памяти	Использование
Низкая 2 ГБ (от 0x00000000 до 0x7FFFFFFF)	Используется процессом.
Высокий 2 ГБ (от 0x80000000 до 0xFFFFFFFF)	Используется системой.

Отображение виртуального адресного пространства процесса(ВАП) на виртуальную память ОС



x86 – 32 битное АП, x64 – 64 битное. Размер адресного пространства x64 – 16 ЕВ (экзобайт). Но на самом деле есть ограничения ОС.

- **32-бита:** размер линейного адресного пространства процесса равен 4Гб, верхние 2Гб (или 1Гб, в зависимости от флагов) из которых защищены на уровне страниц. Поэтому для пользовательского приложения в 32-битной ОС определен лимит в 2Гб (или 3, в зависимости от флагов), за пределы которого процесс выбраться не может (без использования специализированных технологий вида AWE).
- **64-бита:** размер линейного адресного пространства процесса равен 16Тб или 256Тб, из которых (верхняя) часть защищена на уровне страниц. Поэтому 32/64-битному пользовательским приложениям может быть определен лимит в 2Гб, 4Гб, 8Тб и 128Тб (в зависимости от разрядности/версии/флагов).

26.ОС Windows. Управление вводом-выводом. Компоненты подсистемы ввода-вывода. Обработка запросов ввода-вывода. Типы ввода-вывода.

Самым главным является следующий **принцип организации управления вводом/выводом**: любые операции по управлению вводом/выводом объявляются привилегированными и могут выполняться только кодом операционной системы. Для обеспечения этого принципа в большинстве процессоров вводятся режим пользователя и режим супервизора. В режиме супервизора выполнение команд ввода/вывода разрешено, а пользовательском режиме – запрещено.

Компоненты подсистемы ввода-вывода

Диспетчер ввода-вывода подключает приложения и системные компоненты к виртуальным, логическим и физическим устройствам.

Драйвер устройства предоставляет интерфейс ввода-вывода для устройств конкретного типа

Диспетчер PnP отвечают за загрузку соответствующего драйвера при обнаружении нового устройства

диспетчер самонастраивающийся (PnP) обеспечивает поддержку функций pnp в Windows и отвечает следующим задачам, связанным с PnP:

- Обнаружение и перечисление устройств во время загрузки системы
- Обработка добавления или удаления устройств во время работы системы
- Установка новых устройств с совпадающим пакетом драйверов

Диспетчер PnP в режиме ядра поддерживает дерево устройств, которое отслеживает устройства в системе. Дерево устройств содержит сведения об устройствах, имеющихся в системе. При запуске компьютера диспетчер PnP создает это дерево, используя сведения из драйверов и других компонентов, а затем обновляет дерево при добавлении или удалении устройств.

Реестр – база данных, в которой хранится описание основных устройств, подключенных к системе, параметры инициализации драйверов и конфигурационные настройки.

INF-файлы используются для установки драйверов. Содержимое INF-файла состоит из инструкций, описывающих соответствующее устройство, исходное и целевое местонахождение файлов драйвера, изменения, которые нужно внести в реестр при установке драйвера, и информацию о зависимостях драйвера.

CAT-файлы хранят цифровые подписи, которые удостоверяют файлы драйверов, прошедших испытания в лаборатории Microsoft Windows Hardware Quality Lab (WHQL).

Уровень абстрагирования от оборудования (HAL) изолирует драйверы от специфических особенностей конкретных процессоров и контроллеров прерываний, поддерживая API, скрывающие межплатформенные различия.



Диспетчер ввода вывода(I/O manager)

∅ Запросы ввода вывода представляются пакетами запросов ввода-вывода (I/O request packets, IRP)

∅ IRP – структура данных, описывающая запрос ввода вывода

∅ Диспетчер ввода-вывода создаёт IRP и передаёт указатель на IRP соответствующему драйверу

∅ Драйвер, получивший IRP, выполняет операцию и возвращает IRP диспетчеру ввода-вывода

Типы ввода-вывода.

Синхронный ввод-вывод.

∅ Поток, после иницирования операции ввода-вывода, ждет, когда устройство выполнит передачу данных и вернет код статуса по завершении операции ввода-вывода.

∅ После этого приложение продолжает работу и немедленно использует полученные данные. Win32-функции ReadFile и WriteFile выполняются синхронно.

Обработка синхронного запроса ввода-вывода

1.Запрос на ввод-вывод передается через DLL подсистемы.

2.DLL подсистемы вызывает сервис NtWriteFile диспетчера ввода-вывода.

3.Диспетчер ввода-вывода создает IRP, описывающий запрос, и посыпает его драйверу (в данном случае – драйверу устройства), вызывая свою функцию IoCallDriver.

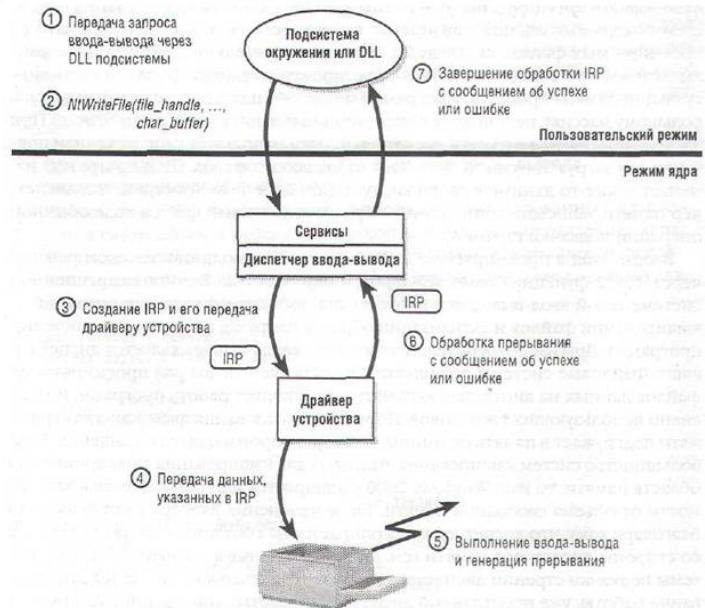
4.Драйвер передает данные из IRP на контроллер устройства и инициирует операцию ввода-вывода.

5.Когда устройство завершает операцию и вызывает прерывание, драйвер устройства обслуживает прерывание.

6.Драйвер уведомляет о завершении ввода-вывода, генерируя прерывание

Драйвер вызывает функцию *IoCompleteRequest* диспетчера ввода-вывода, чтобы уведомить его о завершении обработки IRP, и диспетчер ввода-вывода завершает данный запрос на ввод-вывод

Обработка синхронного запроса ввода-вывод:



8

Типы ввода-вывода.

Асинхронный ввод-вывод

Асинхронный ввод-вывод позволяет потоку выдать запрос на ввод-вывод и продолжить выполнение, не дожидаясь передачи данных устройством.

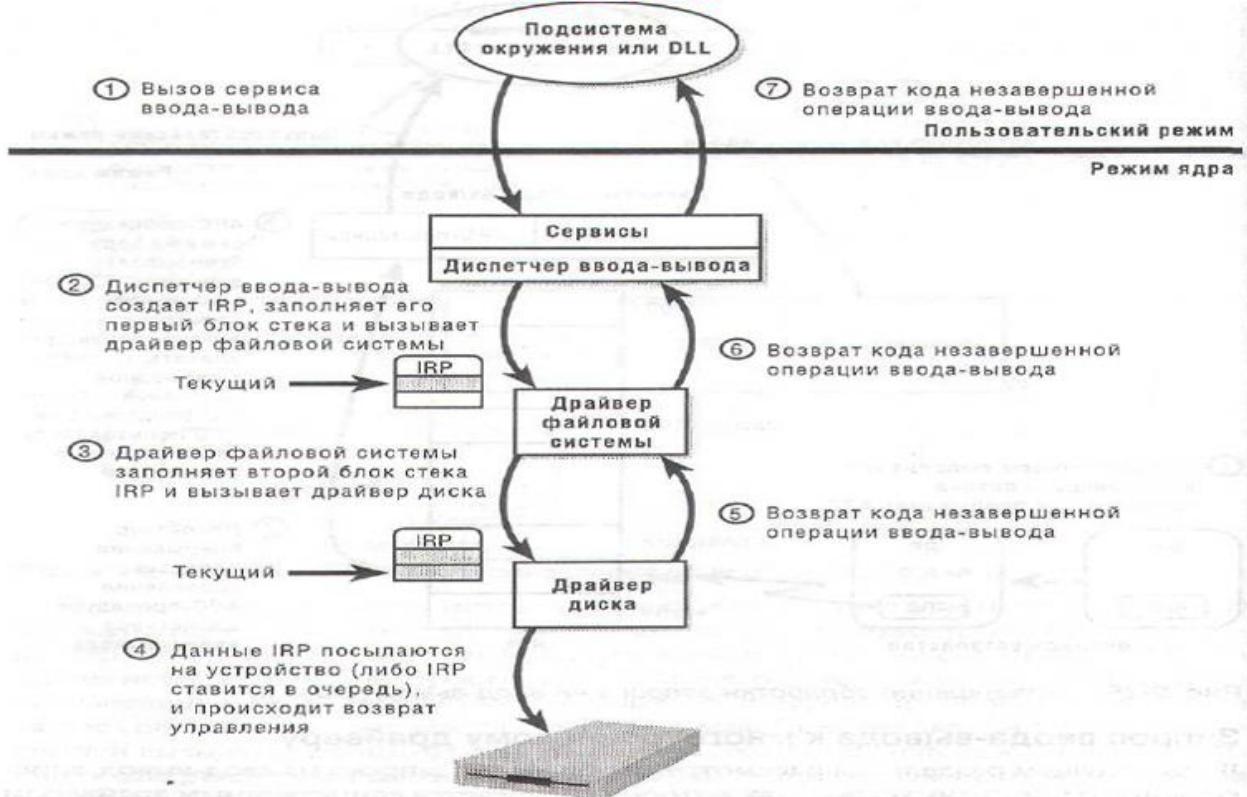
Этот тип ввода-вывода увеличивает эффективность работы.

Поток должен синхронизировать свое выполнение с завершением обработки запроса на ввод-вывод, отслеживая описатель синхронизирующего объекта (которым может быть событие, порт завершения ввода-вывода или сам объект «файл»), который по окончании ввода-вывода перейдет в свободное состояние.

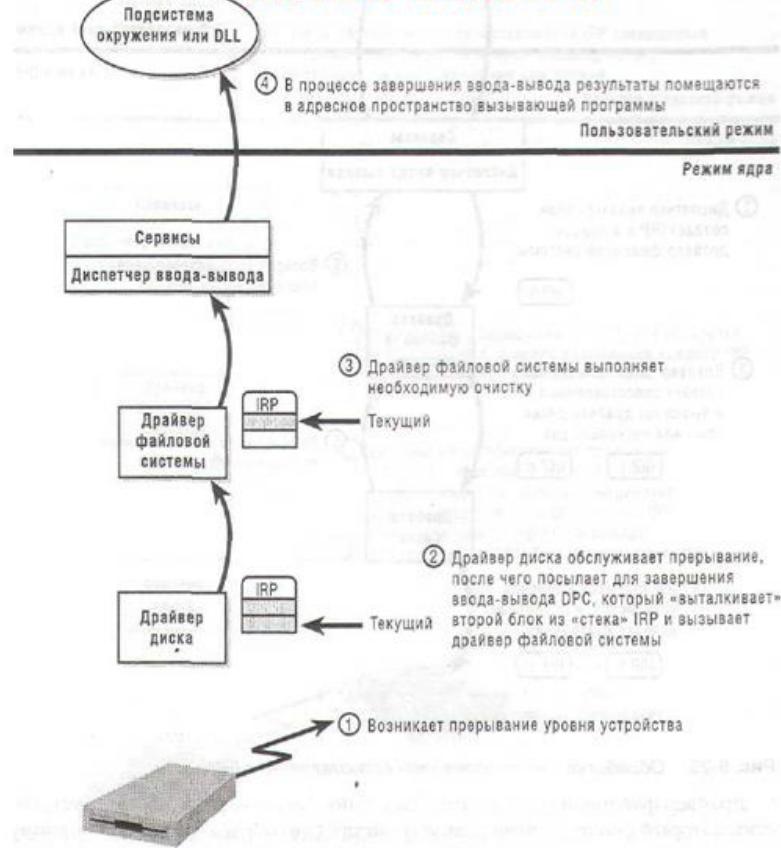
Асинхронный ввод-вывод



Обработка асинхронного запроса ввода-вывода



Завершение обработки асинхронного запроса ввода-вывода



12

Типы ввода-вывода.

Проектируемый ввод-вывод

Проектируемый ввод-вывод осуществляется совместно системой ввода-вывода и менеджером виртуальной памяти.

Файл рассматривается как часть виртуальной памяти процесса, поэтому, например, для записи данных в файл процессу достаточно будет записать их по соответствующему адресу в памяти. Эти изменения будут записаны диспетчером виртуальной памяти на диск в ходе операции откочки страниц

Производительность приложений, которые выполняют значительный объем ввода-вывода или работают с большим числом файлов, может быть повышена.

Быстрый ввод-вывод

Быстрый ввод-вывод (fast I/O) — специальный механизм, который позволяет подсистеме ввода-вывода напрямую, не генерируя IRP, обращаться к драйверу файловой системы или диспетчеру кэша.

27.OC Windows. Концепция файловых систем семейства FAT. Хранение атрибутов и данных файла. Файловые системы FAT32 и exFAT.

FAT (англ. *File Allocation Table* — «таблица размещения файлов»)- классическая архитектура файловой системы, которая из-за своей простоты всё ещё широко используется для флеш-накопителей. Используется в дискетах, и некоторых других носителях информации. Ранее использовалась и на жестких дисках.

Сейчас существует четыре версии FAT — **FAT8, FAT12, FAT16 и FAT32**. Они отличаются разрядностью записей в дисковой структуре, то есть *количеством бит, отведённых для хранения номера кластера*. FAT12 применяется в основном для дискет, FAT16 — для дисков малого объёма, FAT32- для жестких дисков. На основе FAT была разработана новая файловая система **exFAT** (extended FAT), используемая преимущественно для флеш-накопителей.

Файловая система FAT заполняет свободное место на диске последовательно от начала к концу. При создании нового файла или увеличении уже существующего она ищет первый свободный кластер в таблице размещения файлов. Если одни файлы были удалены, а другие изменились в размере, то появляющиеся в результате пустые кластеры будут рассеяны по диску. Если кластеры, содержащие данные файла, расположены не подряд, то файл оказывается *фрагментированным*. Сильно фрагментированные файлы значительно снижают эффективность работы, так как головки чтения/записи при поиске очередной записи файла должны будут перемещаться от одной области диска к другой. Желательно, чтобы кластеры, выделенные для хранения файла, шли подряд, так, как это позволяет сократить время его поиска. Однако, это можно сделать только с помощью специальной программы, подобная процедура получила название *дефрагментации* файла.

Недостатком FAT так же является то, что ее производительность зависит от количества файлов, находящихся в одном каталоге. При большом количестве файлов (около тысячи), выполнение операции считывания списка файлов в каталоге может занять несколько минут. FAT не предусматривает хранения такой информации, как сведения о владельце или полномочия доступа к файлу.

FAT - простая файловая система, не предотвращающая порчи файлов из-за ненормального завершения работы компьютера, она является одной из самых распространенных файловых систем, и ее поддерживают большинство ОС.

Концепция файловой системы FAT (*File Allocation table*)

Хранение атрибутов файла – каталог

Каталог – файл специального вида для хранения информации о файлах и каталогах

Хранение данных файла

Блок хранения данных – кластер (один или несколько смежных секторов, размер сектора – 512 байт)

Данные файла – один или несколько не обязательно смежных кластеров

Хранение адресов блоков данных файла- связный список адресов кластеров файла

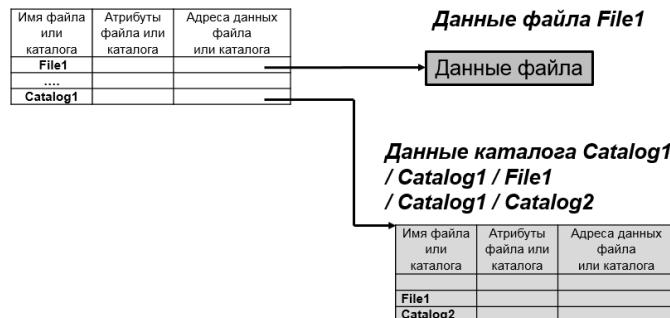


Хранение атрибутов файла в каталоге

Корневой каталог

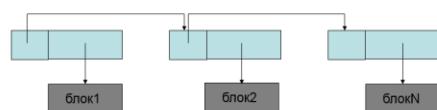
/File1

/Catalog1

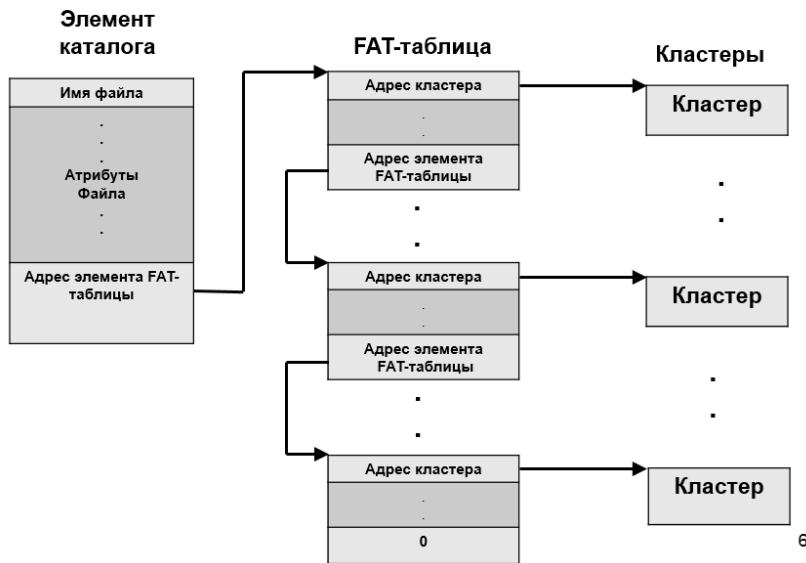


Хранение данных файла

Связный список с таблицей адресов блоков данных



Хранение атрибутов и данных в файловой системе FAT



6

Файловая система FAT32

∅ Размер логического тома с файловой системой до **8 терабайт**

∅ Размер системной области зависит от размера FAT-таблиц

∅ Число строк FAT-таблицы равно числу кластеров области данных

∅ Размер кластера - от **512 байт** до **32 килобайта**

∅ Размер кластера выбирается при **форматировании логического тома**

∅ Размер файла до **4 гигабайт**

∅ Корневой каталог в FAT32 может содержать до **65 535 элементов**.

∅ В загрузочном секторе адрес первого кластера **корневого каталога**

∅ Элемент каталога – **32 байта**

∅ Длинное имя может содержать до **256 символов**.

∅ Для **длинного имени** файла используется **несколько элементов каталога**

Файловая система exFAT (Extended File Allocation Table)

∅ Размер логического тома - до **256 Терабайт**

∅ Размер кластера **от 512 байт** до **32 мегабайт**

ØТеоретический размер файла 16 эксабайт (1 ЭБ=2⁶⁴ =1000000 Терабайт).

Преимущества

Основными преимуществами exFAT перед предыдущими версиями FAT являются:
Уменьшение количества перезаписей одного и того же сектора, что важно для флеш-накопителей, у которых ячейки памяти необратимо изнашиваются после определённого количества операций записи (это сильно смягчается выравниванием износа ([англ.](#) *wear leveling*), встроенным в современные USB-накопители и SD-карты).
Теоретический лимит на размер файла 2⁶⁴ байт (16 ексабайт).

Максимальный размер кластера увеличен до 2²⁵ байт (32 мегабайта).

Улучшение распределения свободного места за счёт введения бит-карты свободного места, что может уменьшать фрагментацию диска.

Введена поддержка списка прав доступа^[3].

Поддержка транзакций (опциональная возможность, должна поддерживаться устройством).

Недостатки и ограничения

Более старые версии Windows NT, вплоть до Windows Vista без Service Pack 1, не поддерживают exFAT. Однако существуют официальные обновления от Microsoft, которые позволяют запускать exFAT на Windows XP SP2 и более поздних версиях.

Windows Vista не способна использовать exFAT-размеченные устройства под ReadyBoost. В Windows 7 это ограничение устранено^[4].

Более сложная структура, в сравнении с FAT, вызывает большее потребление вычислительных ресурсов.

- Количество файлов в подкаталогах ограничено числом 2 796 202 (2²³/3). Для корневого каталога ограничения нет

28.ОС Windows. Концепция файловой системы NTFS. Хранение атрибутов и данных файла. Резидентные и нерезидентные атрибуты. Потоки данных.

В название файловой системы **NTFS** входят слова «новая технология». **NTFS** содержит ряд значительных усовершенствований и изменений, существенно отличающих ее от других файловых систем. С точки зрения пользователей, файлы по-прежнему хранятся в каталогах (часто называемых «папками»). Однако в **NTFS** в отличие от **FAT** работа на дисках большого объема происходит намного эффективнее; имеются средства для ограничения в доступе к файлам и каталогам, введены механизмы, существенно повышающие надежность файловой системы, сняты многие ограничения на максимальное количество дисковых секторов и/или кластеров.

**Основные характеристики NTFS
(New Technology File System)**

ØВосстанавливаемость. После отказа гарантировано восстановление согласованного состояния файловой системы

ØЗащищенность. Аутентифицированный вход в систему и проверка прав доступа к каждому файлу с использованием списка контроля доступа

ØПоддержка множества потоков данных. Каждая единица информации, связанная с файлом, представляет атрибут файла. Каждый атрибут состоит из одного потока. Можно добавлять к файлу новые атрибуты, включая дополнительные именованные потоки данных.

ØХранение имен файлов в кодировке Unicode

Концепция файловой системы NTFS

ØХранение атрибутов файла – каталог + индексный узел

Индексный узел – запись в главной файловой таблице (Master File Table, MFT), относящаяся к данному файлу

ØХранение данных файла

§Блок хранения данных – кластер (один или несколько смежных секторов, размер сектора – 512 байт)

Данные файла – один или несколько не обязательно смежных кластеров

ØХранение адресов блоков данных файла- адреса кластеров хранятся в индексных узлах

Формат логического тома файловой системы NTFS

ØРазмер логического тома файловой системы – до 18,4 эксабайт

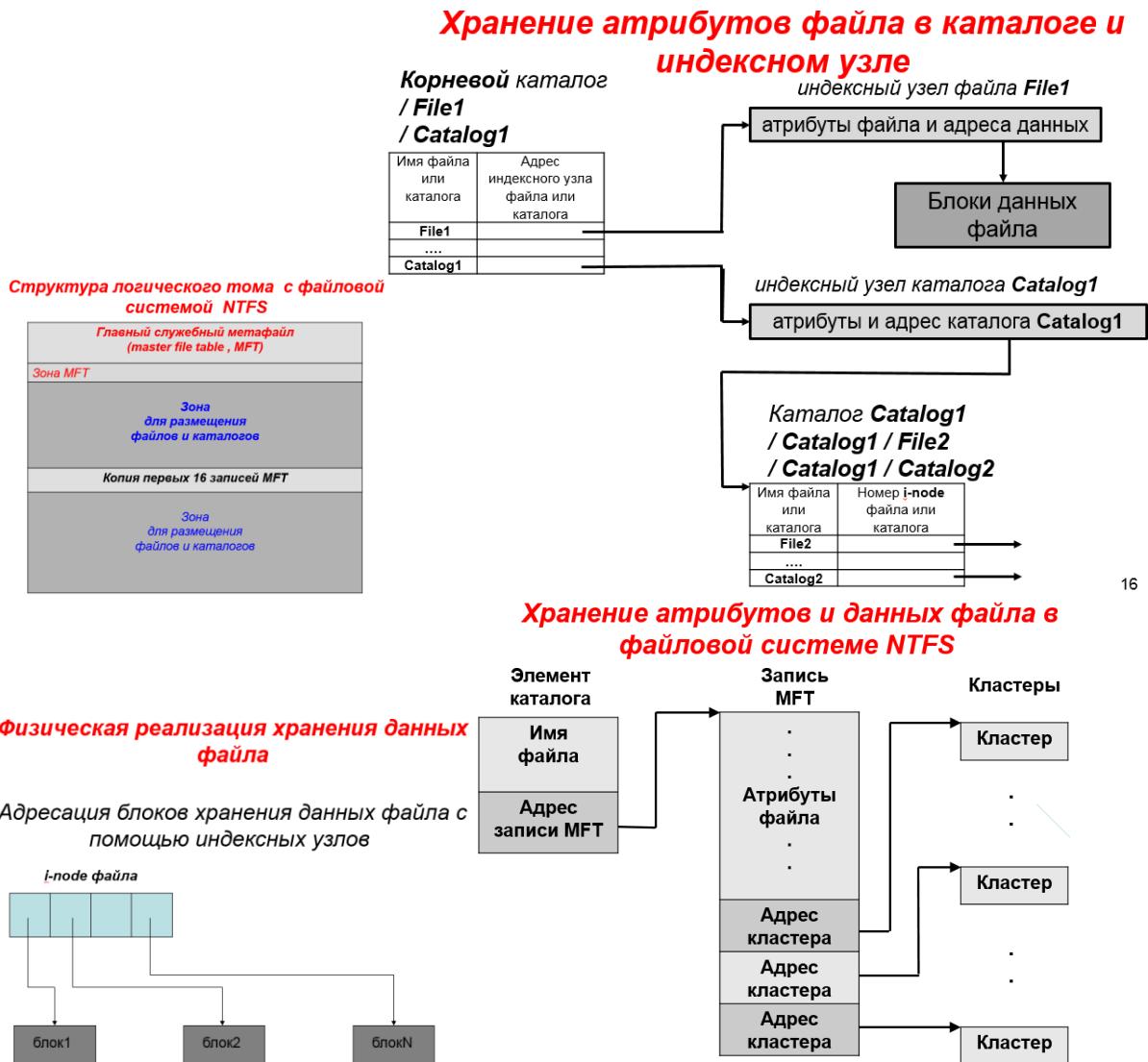
ØКаждый логический том организован как линейная последовательность блоков (кластеров).

ØРазмер блока от 512 байт до 64 Кбайт

ØДля большинства дисков NTFS используются блоки размером в 4 Кбайт, как компромисс между большими блоками (для эффективности операций чтения/записи) и маленькими блоками (для уменьшения потерь дискового пространства на внутреннюю фрагментацию).

ØАдресация блоков осуществляется по их смещению от начала логического тома (используются 64-разрядные числа).

Главной структурой данных в каждом томе является главная файловая таблица MFT (Master File Table), представляющая собой линейную последовательность записей фиксированного (1 Кбайт) размера



Потоки данных

Файл в системе NTFS состоит из множества атрибутов (поток атрибутов)

Файл в системе NTFS имеет имя файла и один длинный (неименованный) поток данных.

У файла NTFS может быть и несколько потоков данных.

При обращении к каждому потоку после имени файла через двоеточие указывается имя потока, например *file1:stream1*.

У каждого потока своя длина.

Каждый поток может блокироваться независимо от остальных потоков.

Максимальная длина потока составляет 2^{64} байт.

Непосредственные файлы

∅ Имя потока данных располагается в заголовке атрибута.

∅ Следом за этим заголовком располагается:

§ либо список дисковых адресов, определяющий положение файла на диске

§ либо, для файлов длиной всего в несколько сотен байтов (а таких файлов довольно много), сами данные файла.

∅ Метод помещения самого содержимого файла в запись MFT называется непосредственным файлом.

∅ В большинстве случаев данные файла не помещаются в запись MFT, поэтому этот атрибут, как правило, является нерезидентным.

Атрибуты файлов в NTFS

Атрибуты бывают двух видов: **резидентные (resident)** и **нерезидентные (nonresident)**. Резидентный атрибут умещается в записи MFT, а нерезидентный нет. У обоих атрибутов есть общий заголовок, куда входят поля типа, длины, и пр. и далее свой собственный заголовок (либо для резидентного, либо для нерезидентного). Атрибуты идентифицируются типом, но также могут быть и именованными.

∅ Заголовок атрибута идентифицирует следующий за ним атрибут, длину и расположение поля значения вместе с флагами и прочей информацией.

∅ Значения атрибутов располагаются непосредственно за заголовками.

∅ Если длина значения слишком велика, чтобы поместиться в запись MFT, она может быть помещена в отдельный блок тома. Такой атрибут называется **нерезидентным атрибутом**. Например, таким атрибутом является **атрибут данных**.

∅ Некоторые атрибуты, такие как атрибуты имени, могут повторяться, но все атрибуты должны располагаться в записи MFT в фиксированном порядке.

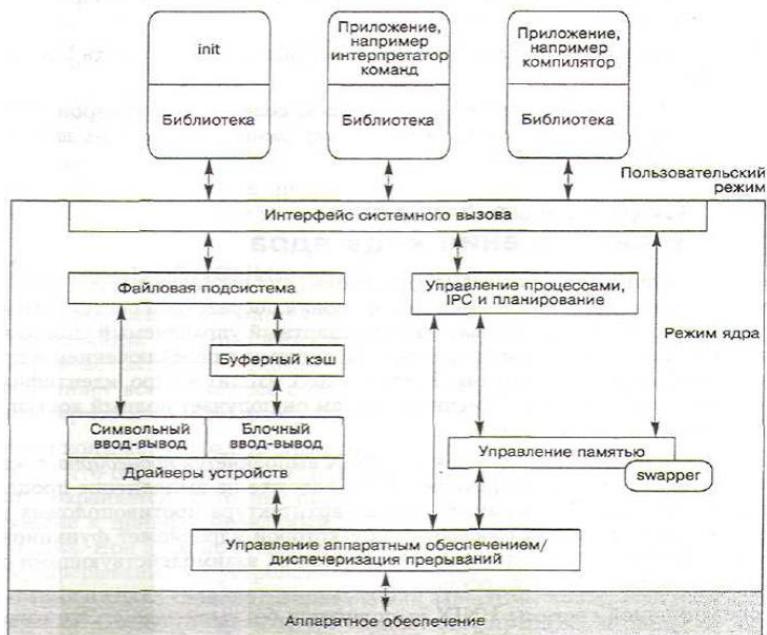
∅ Длина заголовков резидентных атрибутов 24 байт, заголовки для нерезидентных атрибутов длиннее, так как они содержат информацию о месте расположения атрибута.

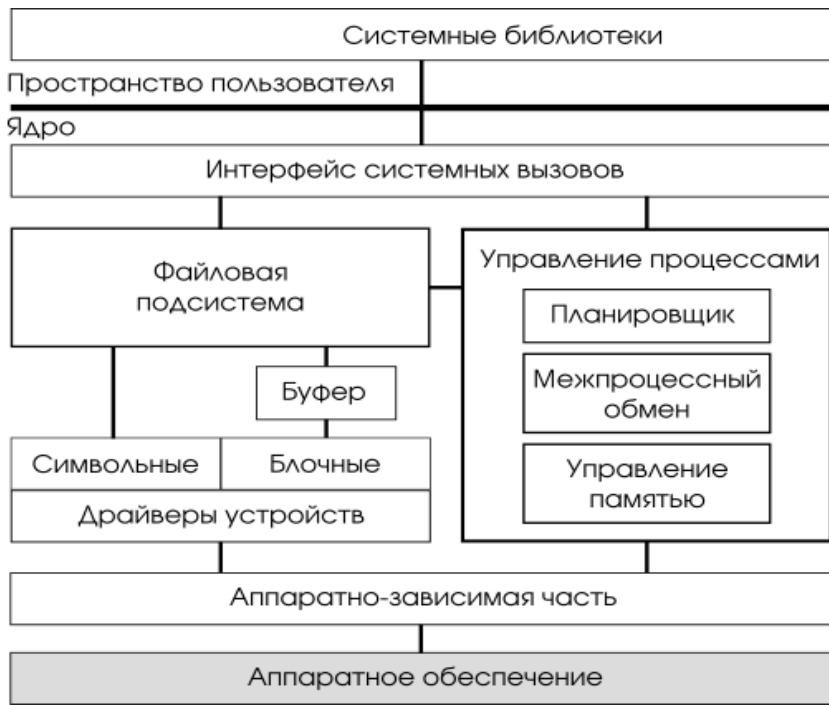
Атрибуты файлов в системе NTFS

Системный атрибут	Описание атрибута
Стандартная информация о файле	Традиционные атрибуты Read Only, Hidden, Archive, System, отметки времени, включая время создания или последней модификации, число каталогов, ссылающихся на файл
Список атрибутов	Список атрибутов, из которых состоит файл, и файловая ссылка на файловую запись и MFT, в которой расположен каждый из атрибутов. Последний используется, если файлу необходимо более одной записи в MFT
Имя файла	Имя файла в символах Unicode. Файл может иметь несколько атрибутов — имен файла
Дескриптор защиты	Структура данных защиты (ACL), предохраняющая файл от несанкционированного доступа. Атрибут «дескриптор защиты» определяет, кто владелец файла и кто имеет доступ к нему
Данные	Собственно данные файла, его содержимое. В NTFS у файла по умолчанию есть один безымянный атрибут данных, и он может иметь дополнительные именованные атрибуты данных.
Корень индекса, битовая карта (только для каталогов)	Атрибуты, используемые для индексов имен файлов в больших каталогах

29.ОС Unix. Архитектура системы и основные модули ядра. Системные вызовы и выполнение процессов ядра ОС.

Архитектура ОС Unix





двууровневую модель системы, состоящую из пользовательской и системной части (ядра). Ядро непосредственно взаимодействует с аппаратной частью компьютера, изолируя прикладные программы (процессы в пользовательской части операционной системы) от особенностей ее архитектуры. Ядро имеет набор услуг, предоставляемых прикладным программам посредством системных вызовов.

Особенности операционной системы Unix

- **Открытость и переносимость.** Код системы написан на языке высокого уровня C. Система простая для понимания, изменений и переноса на другие платформы
- **Стандартизация.** Принципиально одинаковая архитектура и наличие стандартных интерфейсов
- **Единая иерархическая файловая система.** Унифицированный интерфейс файловой системы позволяет также осуществлять доступ к различным устройствам, как к файлам.
- **Наличие большого количества приложений,** в том числе свободно распространяемых, начиная от простейших текстовых редакторов и заканчивая мощными системами управления базами данных и систем мультимедиа

Основные модули ядра

ФИнтерфейс системного вызова - позволяет процессам пользовательского уровня обращаться к сервисам операционной системы;

Файловая подсистема – для реализации символьного и блочного ввода-вывода и доступа к устройствам;

ФБуферный кэш - кэширование данных блочных операций ввода-вывода для повышения производительности системы;

ФМодуль управления процессами — отвечает за создание и уничтожение процессов, планирование и диспетчеризацию их выполнения, поддержку базовых средств взаимодействия процессов.

ФМодуль управления памятью - поддерживает виртуальную память UNIX

ФУправление аппаратным обеспечением и диспетчеризация прерываний

Системные вызовы и выполнение процессов ядра

• Доступ к ядру ОС из процессов прикладного уровня осуществляется посредством **системных вызовов**.

• Когда процесс входит в ядро, идентификатор данного процесса остается **неизменным и ядро UNIX выполняется процедурно**.

• Данная архитектура противоположна архитектуре на основе **передачи сообщений**, при которой ядро может функционировать параллельно с пользовательскими процессами, взаимодействующими с ним путем отправки и получения сообщений.

• **Ядро операционной системы UNIX выполняется без вытеснения.**

Несмотря на то, что система функционирует в одном из двух режимов, ядро действует от имени пользовательского процесса. Ядро не является какой-то особой совокупностью процессов, выполняющихся параллельно с пользовательскими, оно само выступает составной частью любого пользовательского процесса.

При необходимости выполнить привилегированные действия пользовательский процесс обращается с запросом к ядру в форме так называемого **системного вызова**.

30. Управление процессами в ОС Unix. Уровни выполнения процессов.

Под процессом можно понимать совокупность данных ядра системы, необходимых для описания образа программы в память, и управления её выполнением.

С другой стороны процесс это программа в стадии её выполнения, т.к. все программы в ОС UNIX представлены в виде процессов.

Следовательно, процесс - это набор команд или инструкций, выполняемых процессором, совместно с данными и информацией о выполняемой задаче, такой как распределение памяти, открытие файла и статус процесса.

Процесс не отождествляется с программой, т.к. программа может создать (породить) более одного процесса. Выполнение процесса заключается в точном следовании набору инструкций, при котором никогда не передаётся управление другому процессу.

Процессу также недоступны данные и стеки других процессов. Но процессы могут обмениваться друг с другом данными с помощью системы межпроцессного взаимодействия, предоставляемой ОС UNIX.

Процессы ОС UNIX

ØПроцесс ОС UNIX - единица управления и единица потребления ресурсов.

ØКаждый процесс выполняется в собственном виртуальном адресном пространстве.

Ø Совокупность участков виртуальной памяти операционной системы, на которые отображается виртуальное адресное пространство процесса, называется образом процесса.

Типы процессов ОС Unix

Системные процессы

init -процесс инициализации системы(запуск из исполняемого файла ./etc/inin

vhand - диспетчер страничного замещения

bdfflush - диспетчер буферного кэша

shed - диспетчер свопинга

kmadaemon - диспетчер памяти ядра

Прикладные процессы

Интерактивные – монопольно владеют терминалом

Реального времени – требуют гарантированного времени выполнения

Фоновые – выполняются в фоновом режиме

Процессы-демоны

Неинтерактивные процессы, выполняющиеся в фоновом режиме

Атрибуты процесса

Process ID(PID)	Идентификатор процесса. Присваивается при создании процесса и освобождается при завершении
Parent Process ID (PPID)	Идентификатор родительского процесса, породившего данный процесс
Nice Number	Относительный приоритет. Учитывается для определения приоритета выполнения при распределении процессорных ресурсов. Приоритет выполнения зависит от нескольких факторов.
TTY	Терминальная линия(терминал или псевдотерминал, ассоциированный с процессом). Процессы-демоны не имеют ассоциированного терминала
RID	Реальный идентификатор пользователя(идентификатор пользователя, запустившего процесс).
EUID	Эффективный идентификатор пользователя(служит для определения прав доступа процесса к системным ресурсам). <u>М.б.</u> не эквивалентен
RGID	Реальный идентификатор группы(идентификатор первичной или текущей группы пользователя, запустившего процесс).
EGID	Эффективный идентификатор группы(для определения прав доступа к системным ресурсам). ⁴

Структуры данных для управления процессами

Для управления процессами операционная система UNIX использует две информационные структуры:

- Дескриптор процесса(структура proc)
- Контекст процесса(структура user)

Дескриптор процесса (структура proc)

Дескриптор процесса содержит информацию о процессе, необходимую ядру ОС в течение всего жизненного цикла процесса.

Дескрипторы объединены в системную таблицу процессов.

На основании информации из таблицы процессов, операционная система осуществляет планирование и синхронизацию процессов.

Содержимое дескриптора процесса (структура proc)

В дескрипторе прямо или косвенно (через указатели на связанные с ним структуры) содержится информация о

- состоянии процесса,
- расположении образа процесса в оперативной памяти и на диске,
- значении отдельных составляющих приоритета, а также его итоговое значение - глобальный приоритет,
- идентификатор пользователя, создавшего процесс,
- информация о родственных процессах, о событиях, осуществления которых ожидает данный процесс

другая информация.

Контекст процесса (структура user)

Фонкция **Контекст процесса** содержит информацию, необходимую для возобновления выполнения процесса с прерванного места.

∅ Эта информация

- **сохраняется**, когда выполнение процесса приостанавливается,
- и **восстанавливается**, когда планировщик предоставляет процессу вычислительные ресурсы для его продолжения.

Содержимое контекста процесса (структура user)

Содержимое контекста процесса:

- **Адресное пространство процесса в режиме задачи.** Сюда входят код, данные и стек процесса, а также другие области, например, разделяемая память или код и данные динамических библиотек.
- **Управляющая информация.** Ядро использует две основные структуры данных для управления процессом — **proc** и **user**. Сюда же входят данные, необходимые для отображения виртуального адресного пространства процесса в виртуальное адресное пространство ОС

Аппаратный контекст. Сюда входят значения общих и ряда системных регистров процессора.

Переключение контекста

- Текущий процесс переходит в состояние сна, ожидая недоступного ресурса. Ядро вызывает процедуру переключения контекста по функции **sleep ()**.
- Текущий процесс завершает свое выполнение (**exit ()**).
- После пересчета приоритетов в очереди на выполнение находится более высокоприоритетный процесс.
- Происходит пробуждение более высокоприоритетного процесса.

Уровни выполнения процессов

Уровень пользователя

Выполнение пользовательских процессов в системе UNIX осуществляется на двух уровнях: **уровне пользователя (режим задачи, user mode)** и **уровне ядра (режим ядра, kernel mode)**.

§ В режиме задачи процессы имеют доступ только к своим собственным командам и данным

§Когда процесс производит обращение к операционной системе, режим выполнения процесса переключается с **режима задачи на режим ядра**

Уровень ядра

Ø В режиме **ядра** процессу доступны адресные пространства ядра и пользователя. Виртуальное адресное пространство процесса поделено на **адреса, доступные только в режиме ядра, и на адреса, доступные в любом режиме.**

Ø Ядро действует от имени пользовательского процесса. Ядро не является какой-то особой совокупностью процессов, выполняющихся параллельно с пользовательскими, оно само выступает составной частью любого пользовательского процесса.

Ø Некоторые команды процессора являются **привилегированными** и вызывают возникновение ошибок при попытке их использования в режиме задачи: например, команда, управляющая регистром состояния процессора; процессам, выполняющимся в режиме задачи, она недоступна.

31. Управление вводом-выводом в ОС Unix. Компоненты ввода-вывода. Подсистема STREAMS.

Типы ввода-вывода

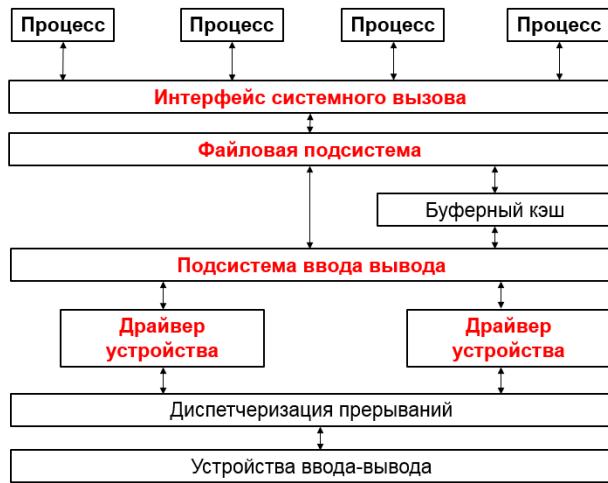
Символьный ввод/вывод - обмен данными с устройством выполняется посимвольно (по байтам), либо строками символов переменного размера (клавиатура, мышь, модем и т.д.).

Блочный ввод/вывод – обмен данными с устройством выполняется блоками фиксированной длины.

- устройства, содержащие файловые системы (жёсткие диски и т.д.).
- устройства, доступ к которым осуществляется через специальные файлы устройств.

§ Потоковый ввод/вывод Похож на символьный ввод/вывод, но имеет возможность включения в поток ввода/вывода промежуточных обрабатывающих модулей и обладает существенно большей гибкостью.

Компоненты ввода-вывода



Типы драйверов ОС UNIX

Символьные драйверы - обслуживание символьного ввода/вывода

Блочные драйверы - обслуживание блочного ввода/вывода. Выполняются с использованием системной буферизации (всегда работают через системный буферный пул).

Драйверы низкого уровня – блочные драйверы, производящие обмен данными с блочными устройствами, минуя буферный пул.

Программные драйверы – драйверы псевдоустройств (доступ к виртуальной памяти ядра, физической памяти, нулевому устройству и т.д.).

Потоковые драйверы

Основное назначение механизма потоков (*streams*) - повышение уровня модульности и гибкости драйверов со сложной внутренней логикой

Специфика таких драйверов - что большая часть программного кода не зависит от особенностей устройства.

Потоковая архитектура драйвера представляет собой двунаправленный конвейер обрабатывающих модулей.

В начале конвейера (ближе всего к пользовательскому процессу) находится заголовок потока, к которому поступают обращения процесса пользователя.

В конце конвейера (ближе всего к устройству) находится драйвер устройства. В промежутке может располагаться произвольное число обрабатывающих модулей, каждый из которых оформляется в соответствии с обязательным потоковым интерфейсом.

Архитектура подсистемы STREAMS

∅ Поток – полнодуплексный канал между прикладным процессом и драйвером устройства

∅ Состав потока

- головной модуль, с которым взаимодействуют прикладные процессы через интерфейс системных вызовов

- модули, выполняющие промежуточную обработку данных

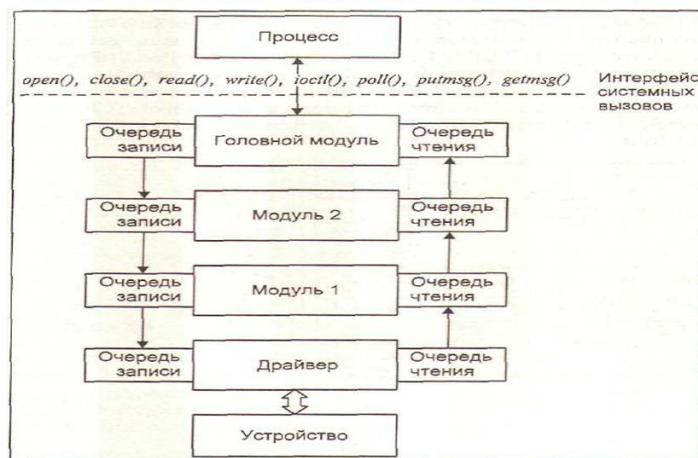
- драйвер, взаимодействующий непосредственно с физическим устройством, псевдоустройством или другим потоком

∅ Каждый модуль(включая драйвер) имеет очередь чтения и очередь записи. Модуль обеспечивает необходимую обработку данных и передаёт их в очередь следующего модуля. Передача в очередь записи осуществляется вниз по потоку (*downstream*), а в очередь чтения — вверх по потоку (*upstream*).

∅ Данные между модулями передаются с помощью сообщений.

∅ Один и тот же модуль может быть включён в несколько потоков

Базовая архитектура потока STREAMS



32. Логическая организация файловых систем ОС Unix. Монтирование файловых систем. Типы файлов. Жёсткие и символические ссылки.

Суть логической организации – все устройства ввода/вывода есть файлы и чтение и запись на них есть чтение и запись в файл.

В ОС UNIX логическая организация ввода/вывода упрощена за счет наличия одинакового интерфейса ввода/вывода на уровне ядра системы.

Все файлы рассматриваются как последовательности байтов, к которым возможно как последовательное, так и прямое обращение. Для того чтобы с файлом можно было работать, он должен быть либо открыт с помощью `open`, либо создан (`creat`). При завершении процесса открытые файлы автоматически закрываются.

Физическая организация ввода/вывода базируется на наличии специальных программных средств, работающих на уровне ядра системы и называемых

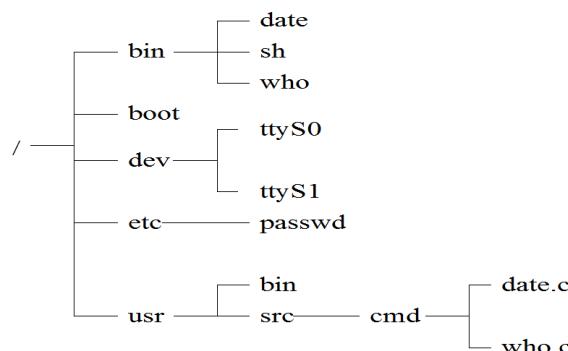
драйверами ввода/вывода. Ядро системы устанавливает однозначную связь между специальными файлами и соответствующими им драйверами. Драйверы ввода/вывода и системы буферизации ядра поддерживают 2 вида интерфейсов:

- Блок-ориентированные.
- Байт-ориентированные.

Особенности файловых систем ОС Unix

- Иерархическая структура с единым корнем
- Трактовка периферийных устройств как файлов
- Существование «жёстких ссылок» (*hard link*) и символьических ссылок (*soft link*)
- Защита доступа к файлам

Логическая организация файловой системы UNIX

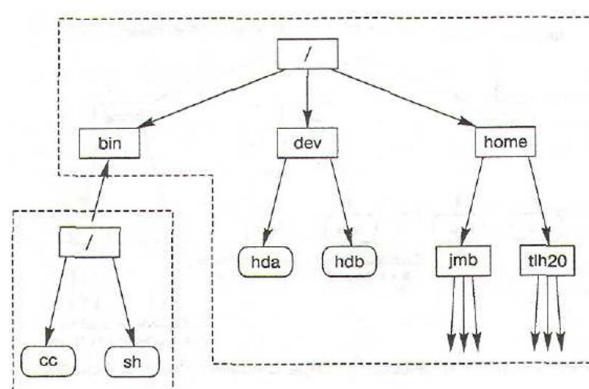


Монтирование каталогов файловой системы UNIX

Когда файловая система монтируется в дерево единой файловой системы UNIX, ее корневой каталог заменяет некоторый заданный каталог (точка монтирования)

Для удаления файловой системы из дерева единой файловой системы используется команда *umount ()*. Для монтирования используется команда *mount ()*.

Монтирование файловых систем UNIX



Типы файлов ОС UNIX

Обычный файл (regular file)	Содержит данные в некотором формате. Интерпретация содержимого производится прикладной программой. Для ОС это просто последовательность байтов.
Каталог (directory)	Содержит имена файлов и указатели на метаданные (номера i-node)
Специальный файл устройства (special device file)	Обеспечивает доступ к физическому устройству путем открытия, чтения и записи в специальный файл устройства: символьные файлы (character) – для небуфризированного обмена данными блочные файлы (block) – обмен данными в виде пакетов фиксированной длины - блоков
Именованный канал (named pipe)	Используется для синхронизации процессов и межпроцессного взаимодействия
Связь (link)	Позволяет косвенно адресовать файл (символическая связь)
Сокет (socket)	Предназначен для организации взаимодействия между процессами

Жесткая связь имен с атрибутами и данными файла

Ø При создании файла создаётся первая жесткая ссылка на атрибуты и данные файла

Ø Для совместного использования данных можно создать множество жестких ссылок на атрибуты и данные существующего файла

Ø Жесткие ссылки можно создавать только для файлов (а не для файлов и каталогов) и только в пределах одной файловой системы.

Ø Жесткие ссылки для операционной системе UNIX идентичны, после создания жесткой ссылки нельзя определить, какое имя файла первоначально являлось оригиналом.

Ø При удалении жесткой ссылки UNIX подсчитывает оставшееся количество ссылок, указывающих на атрибуты и данные файла, и не освобождает блоки данных файла на физическом носителе до тех пор, пока не удалит его последнюю ссылку

Символическая связь имен с каталогами или атрибутами и данными файла

Ø Символическая (косвенная) ссылка обеспечивает возможность вместо имени файла или каталога указывать имя ссылки;

Ø Символическая ссылка представляет собой псевдоним (текстовую подстановку) для имени.

Ø Файл или каталог, на который указывает символическая ссылка, и сама ссылка представляют собой разные объекты файловой системы.

Ø Можно создавать ссылки на несуществующие файлы, удалять оригинальные файлы, не удалив при этом ссылку. Можно создавать ссылки на ссылки и т.п.

Создание жёстких и символьческих связей имен с атрибутами и данными файла

ln — команда, устанавливающая связь между именем файла и атрибутами и данными

Формат команды:

ln file1 file2

создаётся «жёсткая» ссылка (*hard link*)

file1 - существующая жесткая ссылка

file2 - создаваемая жесткая ссылка

ln -s file1 file2

создаётся «символическая» ссылка (*symbolic link*)

file1 - существующая жесткая или символическая ссылка

file2 - создаваемая символическая ссылка

33. Физическая реализация файловых систем ОС Unix. Структура файловой системы.

Каталоги и индексные узлы. Хранение атрибутов и данных файла.

+32 вопрос

Физическая организация файловых систем s5 и usf

Вместо термина «кластер» используется термин «блок», как это принято в файловых системах UNIX.

Блоки хранения данных – размер блока кратен 512 байт.

Раздел ФС делится на 4 области:

- загрузочный блок;
- суперблок(superblock) содержит самую общую информацию о файловой системе

Физическая реализация файловых систем UNIX

Хранение атрибутов файла

§ каталог+индексный узел

§индексный узел(*i-node*) – запись в массиве индексных дескрипторов(*i-list*) , относящаяся к данному файлу

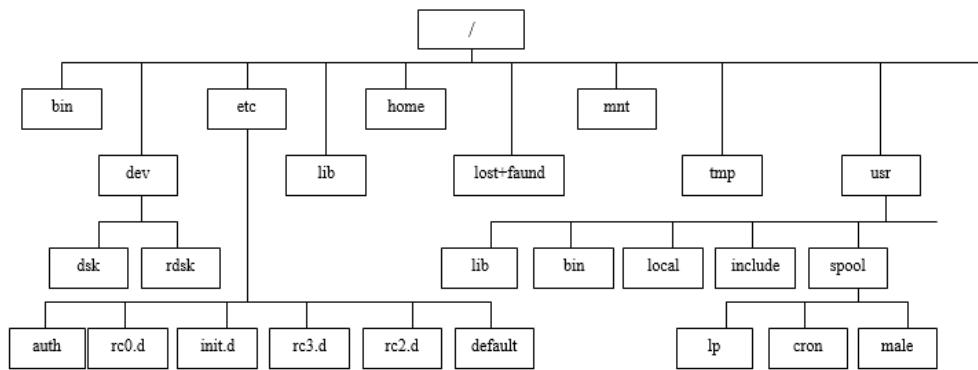
Хранение данных файла

§блок хранения данных – (один или несколько смежных секторов, размер сектора – 512 байт)

§данные файла – один или несколько не обязательно смежных блоков данных

§хранение адресов блоков данных файла- адреса блоков данных хранятся в индексных узлах

Структура файловой системы Unix:



Корневой каталог **(/)** - основа файловой системы *UNIX*. Все остальные файлы и каталоги располагаются в рамках структуры, порождённой корневым каталогом, независимо от их физического местоположения.

В каталоге **/bin** находятся наиболее часто употребимые команды и утилиты системы.

В каталоге **/dev** содержатся специальные файлы устройств, являющиеся интерфейсом доступа к периферийным устройствам. Этот каталог может содержать несколько подкаталогов, группирующих файлы устройств одного типа.

В каталоге **/etc** находятся системные конфигурационные файлы и многие утилиты администрирования. Самые важные из них - скрипты инициализации системы, которые хранятся в каталогах *rcN*, где *N* % номер, определяющий уровень системы.

В каталоге **/lib** находятся библиотечные файлы языка *C* и других языков программирования. Часть их может находиться в каталоге **usr/lib**.

Каталог **/lost+found** % каталог потерянных файлов. При аппаратных сбоях и сбоях операционной системы могут появляться безымянные файлы. Программы проверки и восстановления помещают сюда неповреждённые безымянные файлы под числовыми именами.

Каталог **/mnt** % стандартный каталог для временного связывания (монтирования) физических файловых систем к корневой для получения единого дерева логической файловой системы. Обычно содержимое этого каталога пусто, т.к. при монтировании он перекрывается связанной файловой системой.

Каталог **/home** предназначен для размещения каталогов пользователей.

Каталог **/usr** % каталог различных сервисных подсистем, таких как системы печати, электронная почта, электронные справочники, исполняемые файлы утилит *UNIX*.

В каталоге **/spool** находятся выполняемые файлы утилит *UNIX*.

Каталог **/tmp** предназначен для хранения временных файлов, необходимых для работы различных подсистем *UNIX*.

Структура логического тома с файловой системой *s5fs*

Ø Суперблок(**superblock**) содержит самую общую информацию о файловой системе
 § количество блоков хранения данных в файловой системе,
 § количество свободных блоков в файловой системе,
 § список свободных блоков,
 § размер таблицы *i*-узлов,

§количество свободных i-узлов,

§список свободных i-узлов,

§флаги, используемые для синхронизации доступа к свободным блокам и i-узлам.

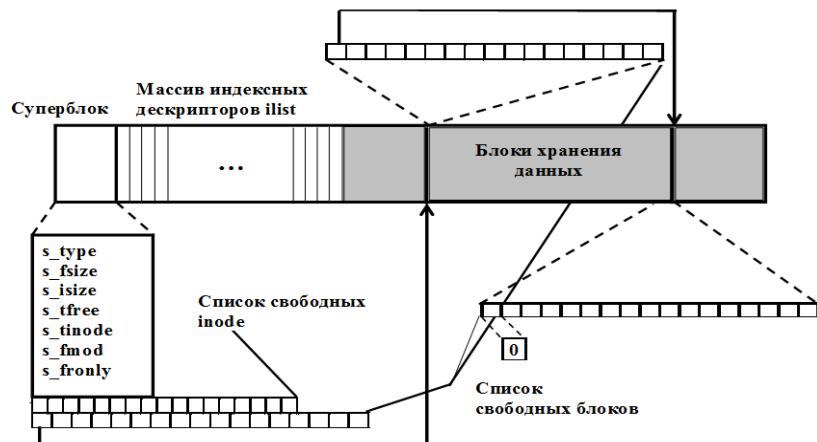
Обычно копии суперблока содержатся в нескольких местах раздела файловой системы, поскольку хранящаяся в нем информация критически важна для файловой системы.

ØМассив индексных дескрипторов(**i-node list**) содержит таблицу индексных узлов(*i-node*)

ØБлоки хранения данных файла, либо выделены для файлов, либо свободны. Они занимают основную часть дискового пространства.

Информация о свободных блоках организована в виде цепочки таблиц свободных блоков. Операция чтения возвращает одну такую таблицу и указатель на следующую таблицу.

Структура логического тома файловой системы *s5fs*



Атрибуты файла, содержащиеся в индексном узле

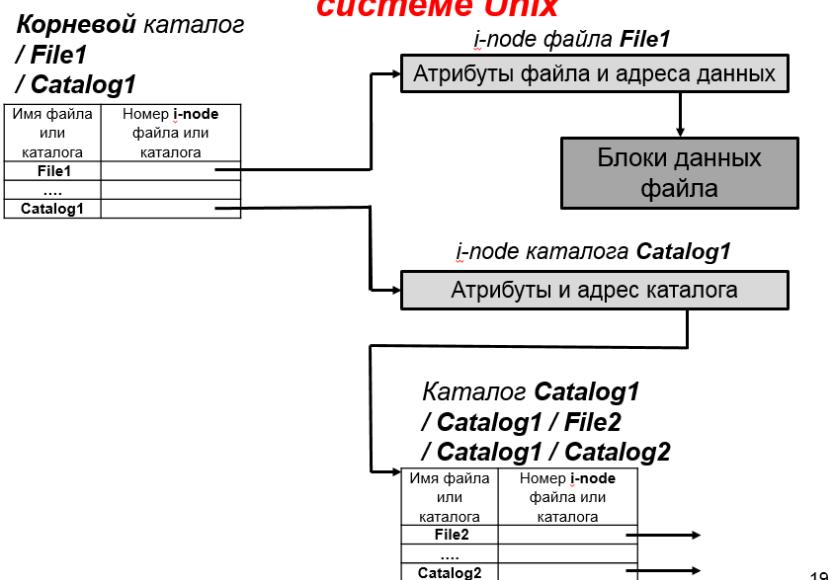
- идентификатор владельца файла;
 - разрешенные способы доступа,
 - время последнего обращения с использованием каждого вида доступа,
 - тип файла (файл, каталог обычного типа, специальный файл, конвейер, символьная связь)
 - число ссылок на данный индексный дескриптор, равный количеству жёстких ссылок;
 - адреса дисковых блоков, содержащих данные файла
- размер файла

Адресация блоков данных файла

Физические адреса блоков данных хранятся в *i-node* в виде массива из 13 элементов.

- Первые **10 элементов** адресуют непосредственно блоки хранения данных файла.
- **11-й элемент** адресует блок, в свою очередь содержащий адреса блоков хранения данных
- **12-й элемент** указывает на дисковый блок, также хранящий адреса блоков, каждый из которых адресует блок хранения данных файла.
- **13-й элемент** используется для тройной косвенной адресации, когда для нахождения адреса блока хранения данных используется три дополнительных блока

Хранение атрибутов и данных в файловой системе Unix



19

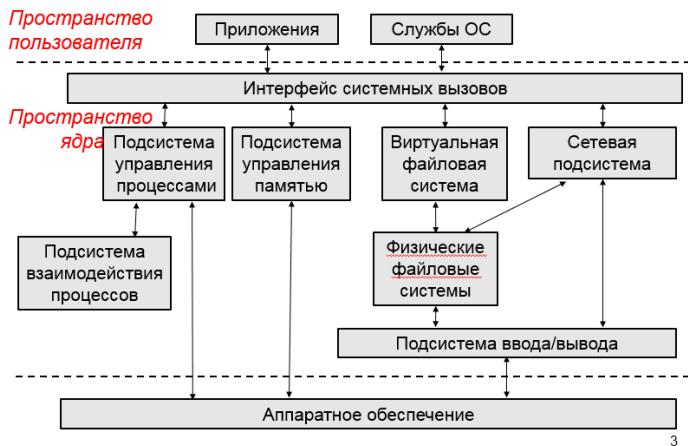
34. Архитектура ОС Linux. Основные подсистемы ядра ОС.

База стандартов Linux

ØLinux Standards Base(LSB, www.linuxbase.org) - проект, ставящий целью стандартизацию операционной системы Linux для того, чтобы приложения, написанные для одного дистрибутива, соответствующего стандарту LSB, точно также компилировались и выполнялись в любом другом LSB-совместимом дистрибутиве.

ØБаза стандартов Linux содержит общие стандарты компонентов операционной системы, включая **библиотеки, форматы пакетов и дистрибутивов, команд и утилит**

Общая архитектура Linux



3

Основные подсистемы ядра Linux

◊ **Подсистема управления процессами (process manager).** Отвечает за создание процессов, обеспечение диспетчеризации доступа к процессору (процессорам) системы, удалению процессов из системы по завершению их работы

◊ **Подсистема взаимодействия процессов (interprocess communication, IPC).** Обеспечивает взаимодействие процессов. Подсистема работает совместно с подсистемой управления процессами.

◊ **Подсистема управления памятью.** Обеспечивает процессам доступ к памяти. Linux выделяет каждому процессу виртуальное адресное пространство, которое делится на **пользовательское адресное пространство и пространство ядра**.

◊ **Виртуальная файловая система (virtual file system, VFS).** Интерфейс VFS обеспечивает единый способ доступа к файлах и каталогам, размещенным в различных файловых системах.

◊ **Физические файловые системы.** Реализуют размещения и хранение данных на внешних устройствах.

◊ **Подсистема ввода-вывода.** Транслирует запросы ввода вывода драйверам устройств для выполнения операций ввода-вывода.

◊ **Сетевая подсистема.** Позволяет взаимодействовать процессам, выполняющимся на разных узлах сети.

35.ОС Linux. Управление процессами и потоками. Граф состояния потока.

Планирование и диспетчеризация потоков.

Понятие процесса и потока ОС Linux

◊ **Процесс(process)** ОС Linux - единица управления и единица потребления ресурсов.

◊ Процесс имеет один или несколько **потоков выполнения (thread)**

◊ В системе процессы и потоки представляют собой единую структуру данных и носят название **задачи(task)**

Ø Каждый процесс имеет собственное виртуальное адресное пространство, отображаемое в образ процесса в виртуальной памяти операционной системы.

Создание процессов ОС Linux

Ø Системный вызов **fork**, выполняемый родительским процессом (*parent process*) создаёт новый процесс, называемый дочерним процессом (*child process*), образ которого создаётся по принципу копирования образа родителя (*copy-on-write*). Образ потомка перезаписывается системным вызовом **execve**.

§ Системный вызов **vfork** позволяет создавать процесс без копирования страниц родительского процесса

vfork() это специальный вариант [clone\(2\)](#). Он используется для создания новых процессов без копирования таблиц страниц родительского процесса. Это может использоваться в приложениях, критичных к производительности, для создания дочерних процессов, сразу же запускающих **execve()**.

vfork() отличается от **fork** тем, что родительский процесс блокируется до тех пор, пока дочерний процесс не вызовет **execve(2)** или [_exit\(2\)](#). Дочерний процесс разделяет всю память с родительским, включая стек, до тех пор, пока не вызовет **execve()**. Дочерний процесс не должен выходить из текущей функции или вызывать **exit()**, но может вызвать [_exit\(\)](#)

§ Системный вызов **clone** создает новый поток либо в *текущем процессе*, либо в *новом процессе*.

*Если новый поток находится в **текущем процессе**, он совместно использует с остальными потоками адресное пространство процесса*

*Если поток создаётся в **новом процессе**, то используется та же семантика, что и у системного вызова **fork***

Виртуальное адресное пространство процесса

Ø Часть виртуального адресного пространства процесса выделяется собственно процессу, часть памяти отводится для данных ядра ОС.

Ø Область ядра не видна в пользовательском режиме, но становится доступной, когда процесс переключается в режим ядра.

Ø Образ потомка перезаписывается системным вызовом **exec**.

Ø Каждый процесс Linux имеет собственное виртуальное адресное пространство памяти.

Ø Пользовательское виртуальное адресное пространство процесса состоит из трёх логических сегментов

§ Сегмент текста (*text segment*). Содержит исполняемый код.

§ Сегмент данных (*data segment*). Содержит данные (переменные, массивы и т.д.)

§ Сегмент стека (*stack segment*). Содержит временно хранимые данные процесса по правилу LIFO

Управление виртуальной памятью операционной системы

§ Используется страничное распределение памяти.

§ Размер страницы фиксирован и зависит от архитектуры процессора.

§ Страницы процессов выгружается либо в раздел подкачки, если он присутствует, либо в файлы подкачки фиксированной длины, которых может быть от одного до восьми.

Файлы и разделы подкачки

Файлы подкачки могут динамически добавляться и удаляться, и у каждого есть свой приоритет.

Выгрузка страниц в раздел подкачки, доступ к которому осуществляется как к отдельному устройству, не содержащему файловой системы, более эффективна, чем выгрузка в файл, по некоторым причинам:

- не требуется преобразование блоков файла в блоки диска.
- физическая запись может быть любого размера, а не только размера блока файла.
- страница всегда пишется прямо на устройство в виде единого непрерывного участка, а при записи в файл подкачки это может быть и не всегда так.

Таблица управления процессами

Каждая задача представляется в таблице процессов **дескриптором процесса**

В дескрипторе хранятся переменные и вложенные структуры, описывающие процесс:

§ сведения о текущем состоянии задачи.

§ переменные, позволяющие планировщику вычислять время выполнения на процессоре.

§ переменные для вычисления приоритета задачи,

§ необходимость выполнения в режиме реального времени, и то, какой алгоритм планирования реального времени должен использоваться в этом случае.

Вложенные структуры могут содержать дополнительные сведения о задаче:

§ описывать выделяемую для задачи память. (например, местоположение таблицы страниц в памяти и число задач, совместно использующих адресное пространство).

значения регистров, хранящие контекст выполнения задачи, обработчики сигналов и права доступа для задачи.



Планирование потоков

В состав подсистемы управления процессами входит **планировщик процессов** (*process scheduler*), обеспечивающий доступ процессов к процессору в режиме квантования времени.

Планировщик процессов хранит список всех задач в виде двух структур данных.

§ **Первая структура** представляет собой кольцевой список, каждая запись которого содержит указатели на предыдущую и последующую задачу (**очередь выполнения** (*run queue*)). Обращение к этой структуре происходит в том случае, когда ядру необходимо проанализировать все задачи, которые должны быть выполнены в системе. Очереди выполнения напоминают **многоуровневые очереди с обратной связью** (*FB+RR*) с различными приоритетами потоков.

§ **Второй структурой** является хэш-таблица. При создании задачи ей присваивается уникальный **идентификатор процесса** (*process identifier, PID*). Идентификаторы процессов передаются хэш-функции для определения местоположения процесса в таблице процессов. **Хэш-метод обеспечивает быстрый доступ к специфическим структурам данных задачи, если ядру известен ее PID.**

Состояния потока

Состояния **active** (активный) и **expired** (неактивный) используются при планировании выполнения потока (диспетчериизации)

Ø Планировщик осуществляет диспетчеризацию потоков, которые находятся в активном состоянии (**active**) (обладающие правом на процессорное время).

Ø Потоки, ожидающие наступления следующего периода дискретизации, находятся в неактивном состоянии (**expired**)

Ø Поток переходит в состояние **running** (выполнения) после передачи ему процессора

∅ При блокировке задача переходит в состояние **sleeping** (спячки), а при приостановке работы в состояние остановов (**stopped**).

∅ Состояние **zombie** (зомби) показывает, что выполнение задачи прекратилось, однако она еще не была удалена из системы (например, если процесс состоит из нескольких потоков, он будет пребывать в состоянии зомби, пока все потоки не получат уведомление о завершении работы основного процесса).

Приоритеты диспетчеризации

∅ Различают **40 уровней** приоритетов от **-20 до 19**

(-20 – наивысший приоритет)

∅ **Статический приоритет (state priority)** (правильное значение, nice value) - присваивается при создании задачи

∅ Для обеспечения высокой степени интерактивности планировщик может динамически увеличивать значение приоритета (понижая уровень) задачи, занимающей процессорное время, до истечения кванта.

∅ **Эффективный приоритет (effective priority)** – изменённый уровень приоритета, вычисляемый во время сна или выполнения.

∅ Эффективный приоритет определяет уровень, на который будет помещена задача

Диспетчеризация заключается в реализации найденного в результате планирования (динамического или статистического) решения, то есть в переключении процессора с одного потока на другой. Прежде чем прервать выполнение потока, ОС запоминает его контекст, с тем, чтобы впоследствии использовать эту информацию для последующего возобновления выполнения данного потока.

Диспетчеризация сводится к следующему:

- сохранение контекста текущего потока, который требуется сменить;
- загрузка контекста нового потока, выбранного в результате планирования;
- запуск нового потока на выполнение.

36. ОС Linux. Управление процессами и потоками.

+35 вопрос

37. Файловые системы ОС Linux. Особенности реализации файловых систем ext2, ext3, ext4.

Файловая система ext2

После своего появления в 1993 году файловая система ext2 (second extended file system, ext2fs) стала самой распространенной файловых систем для Linux. Целью создания файловой системы ext2 являлось обеспечение высокой производительности и устойчивости файловой системы, а также поддержка дополнительных возможностей.

Как и в любой файловой системе UNIX, в ext2 можно выделить следующие элементы:

Блоки;

Группы блоков;

Индексные дескрипторы;

Суперблок;

Блоки

Всё пространство раздела диска разбивается на блоки фиксированного размера, кратные размеру сектора: 1024, 2048, 4096 или 8192 байт. Размер блока указывается при создании файловой системы в разделе диска. Все блоки имеют порядковые номера. По умолчанию, во время форматирования, 5% блоков резервируются исключительно для нужд привилегированных пользователей root.

С целью уменьшения фрагментации и количества перемещений головок жёсткого диска при чтении больших массивов данных блоки объединяются в **блочные группы**.

Блочные группы

Все блоки раздела ext2 разбиваются на группы блоков, называемые блочными группами (block group). Для каждой группы создаётся отдельная запись в глобальной дескрипторной таблице, в которой хранятся основные параметры:

номер блока в битовой карте блоков,

номер блока в битовой карте inode,

номер блока в таблице inode,

число свободных блоков в группе,

число индексных дескрипторов, содержащих каталоги.

Битовая карта блоков - это структура, каждый бит которой показывает, отведен ли соответствующий ему блок какому-либо файлу. Если бит равен 1, то блок занят.

Суперблок содержит важную информацию не только об отдельной блочной группе, но и обо всей файловой системе. Сюда относятся сведения об общем количестве блоков и индексных узлов inode файловой системы, размере данной блочной группы, времени монтирования файловой системы и другая дополнительная информация.

Блочная группа содержит несколько структур данных, обеспечивающих файловые операции над этой группой. В качестве одной из таких структур выступает **таблица**

индексных узлов (inode table), которая содержит записи для каждого inode в блочной группе.

Индексные дескрипторы

Индексный дескриптор, или **inode** (information node) - это специальная структура, которая содержит информацию об атрибутах и физическом расположении данных файла. Индексные дескрипторы объединены в таблицу, которая содержится в начале каждой группы блоков.

Каждый индексный дескриптор в ext2 (ext2 inode) хранит информацию об одном файле либо каталоге.

Каждый индексный дескриптор содержит 15 указателей на блоки данных (каждый величиной в 32 бита). Первые двенадцать указателей ссылаются непосредственно на первые 12 блоков данных. Указатели с 13-го по 15-тый предназначены для косвенной адресации блоков данных.

13-й косвенный указатель (indirect pointer) определяет местоположение блока, содержащего указатели на блоки данных.

14-й указатель представляет собой указатель двойной косвенной адресации (doubly indirect pointer). Указатель двойной косвенной адресации ссылается на блок простых косвенных указателей.

15-й указатель представляет собой указатель тройной косвенной адресации (triply indirect pointer) - он указывает местоположение блока указателей двойной косвенной адресации.

Суперблок

Суперблок - основной элемент файловой системы ext2. Он содержит общую

информацию о файловой системе:

общее число блоков и индексных дескрипторов в файловой системе,

число свободных блоков и индексных дескрипторов в файловой системе,

размер блока файловой системы,

количество блоков и индексных дескрипторов в группе блоков,

размер индексного дескриптора,

идентификатор файловой системы.

Суперблок находится в 1024 байтах от начала раздела. От целостности суперблока напрямую зависит работоспособность файловой системы. Операционная система создаёт несколько резервных копий суперблока на случай повреждения раздела.

В следующем блоке после суперблока располагается глобальная дескрипторная таблица - описание групп блоков, представляющее собой массив, содержащий общую информацию обо всех группах блоков раздела.

2.2 Файловая система ext3

Является файловой системой по умолчанию во многих дистрибутивах. Основана на ФС ext2.

Основное отличие от ext2 состоит в том, что ext3 – **журналируемая** файловая система

Стандартом предусмотрено три режима журналирования:

- **writeback**: в журнал записываются только **метаданные файловой системы**, то есть информация о её изменении. Не может гарантировать целостности данных, но уже заметно сокращает время проверки по сравнению с ext2;
- **ordered**: то же, что и writeback, но **запись данных в файл производится гарантированно до записи информации о изменении этого файла**. Немного снижает производительность, также не может гарантировать целостности данных (хотя и увеличивает вероятность их сохранности при дописывании в конец существующего файла);
- **journal**: полное журналирование как метаданных ФС, так и пользовательских данных. Самый медленный, но и самый безопасный режим; может гарантировать целостность данных при хранении журнала на отдельном разделе (а лучше – на отдельном жёстком диске).

Режим журналирования указывается в строке параметров для программы **mount**, например:

```
mount /dev/hda6 /mnt/disc -t ext3 -o data=<режим>  
либо в файле /etc/fstab.
```

Файловая система ext3 может поддерживать файлы размером до 1 терабайта.

С Linux-ядром 2.4 объём файловой системы ограничен максимальным размером блочного устройства, что составляет 2 терабайта.

В Linux 2.6 (для 32-разрядных процессоров) максимальный размер блочных устройств составляет 16 терабайт, однако ext3 поддерживает только до 4 терабайт.

Табл. Ограничения размеров файлов и каталогов ext3

Размер блока	Макс. размер файла	Макс. размер файловой системы
1 Кб	16 GB	до 2 TB
2 Кб	256 GB	до 4 TB
4 Кб	2 TB	до 8 TB
8 Кб	2 TB	до 16 TB

2.3 Файловая система ext4

В ext4 появилось несколько новых улучшений производительности и надежности. ext4 поддерживает файловые системы до **одного экзабайта** (1000 петабайт).

Хотя это и большая цифра, потребление места на устройствах хранения увеличивается, так что ext4 была разработана с расчетом на будущее.

Файлы в ext4 могут достигать размера 16 ТБ (при блоках размером 4 КБ), что в восемь раз больше, чем в ext3.

Глубина поддиректорий в ext4 была увеличена с 32 КБ до фактически бесконечной. Это может показаться избыточным, но тут надо принимать во внимание возможную иерархию файловой системы размером в экзабайт.

Было оптимизировано индексирование директорий, которое теперь использует хеширующую структуру, подобную В-дереву. Поэтому, несмотря на гораздо больший размер, поиск в ext4 работает очень быстро.

2.3.1 Экстенты

Одним из главных недостатков системы ext3 был ее метод выделения места на дисках. Дисковые ресурсы для файлов выделялись с помощью битовых карт свободного места - способа, не выделяющегося ни скоростью, ни масштабируемостью. Формат, применяемый в ext3, очень эффективен для маленьких файлов, но очень неэффективен для больших.

Поэтому для улучшения выделения ресурсов и поддержки более эффективной структуры хранения данных в ext4 вместо битовых карт применяются экстенты.

Экстент - это способ представления непрерывной последовательности блоков памяти. При использовании экстентов сокращается количество метаданных, так как вместо информации о том, где находится каждый блок памяти, экстенты содержат информацию о том, где находится большой список непрерывных блоков памяти.

В ext4 для эффективного представления маленьких файлов в экстентах применяется уровневый подход, а для эффективного представления больших файлов применяются деревья экстентов.

Например, один индексный дескриптор в ext4 имеет достаточно места, чтобы ссылаться на четыре экстента (каждый из которых представляет множество последовательных блоков). Для больших (в том числе фрагментированных) файлов дескриптор может содержать ссылки на другие индексные дескрипторы, каждый из которых может указывать на концевой узел (указывающий на экстенты). Такое дерево экстентов постоянной глубины предоставляет мощный механизм представления больших, потенциально фрагментированных файлов. Также узлы имеют механизмы самопроверки для защиты от повреждений файловой системы.

2.3.2 Производительность

ext4, вместе с повышением масштабируемости и надежности, имеет ряд улучшений, связанных с производительностью.

- Предварительное выделение на файловом уровне

Некоторые приложения, например, базы данных, рассчитывают, что их файлы будут храниться в непрерывных блоках (чтобы использовать оптимизацию при последовательном чтении данных с дисков, а также минимизировать количество команд Read в расчете на блок данных). Хотя сегменты непрерывных блоков можно получить с помощью экстентов, есть и другой, более грубый метод: предварительно выделять очень большие сегменты непрерывных блоков желаемого размера. ext4 делает это с помощью нового системного вызова, который осуществляет предварительное выделение и инициализацию файла заданного размера. Далее можно записывать необходимые данные и читать их посредством операций Read .

- Отложенное выделение блоков памяти

Суть оптимизации в том, что откладывание выделения физических блоков памяти до того, пока они действительно будут записаны, позволяет выделять больше последовательных блоков. Это похоже на предварительное выделение, за исключением того, что эту задачу система выполняет автоматически. Но если размер файла известен заранее, лучше применять предварительное выделение.

- Выделение блоков памяти группами

И последняя оптимизация, также связанная с последовательными блоками, - это групповое выделение блоков в ext4. В ext3 каждый блок выделяется по отдельности. Поэтому иногда получалось, что для последовательных данных выделенные блоки располагались не последовательно. В ext4 эта проблема решена за счет того,

что выделение группы блоков происходит за один раз, поэтому фрагментирование маловероятно. Связанные данные хранятся на диске вместе, что в свою очередь позволяет оптимизировать их чтение.

Другим аспектом группового выделения блоков является объем работы, необходимой для выделения блоков. В ext3 выделение осуществляется по одному блоку за раз. Выделение блоков группами требует гораздо меньшего количества вызовов, что ускоряет выделение блоков.

- **Надежность**

При увеличении размеров файловых систем до уровней, поддерживаемых ext4, неизбежно встает проблема повышения надежности. Для ее решения в ext4 предусмотрено множество механизмов самозащиты и самовосстановления.

- **Контрольная сумма журнала файловой системы**

Как и ext3, ext4 является журналируемой файловой системой. Журналирование позволяет производить изменения более надежным образом и гарантировать целостность данных даже в случае краха системы или сбоя питания во время выполнения операции. В результате снижается вероятность повреждения файловой системы.

Но даже с применением журналирования повреждения системы возможны, если в журнал попадут ошибочные записи. Для борьбы с этим в ext4 реализована проверка контрольных сумм записей журнала, чтобы гарантировать, что в нажелевшую файловую систему будут внесены правильные изменения.

В зависимости от нужд пользователя ext4 может работать в разных режимах журналирования. Например, ext4 поддерживает режим обратной записи (в журнал заносятся только метаданные), режим упорядочивания (метаданные заносятся в журнал, но только после записи самих данных), а также самый надежный - журнальный режим (в журнал заносятся как данные, так и метаданные). Заметьте, что хотя журнальный режим - это лучший способ гарантировать целостность файловой системы, в то же время это и самый медленный режим, так как в нем через журнал проходят все данные.

- **Дефрагментация "на лету"**

Хотя ext4 включает в себя возможности уменьшения фрагментации внутри файловой системы (экстенты для выделения последовательных блоков), все же при длительной жизни файловой системы некоторая фрагментация неизбежна. Поэтому для улучшения производительности существует инструмент, который на лету дефрагментирует как файловую систему, так и отдельные файлы. Дефрагментатор - это простой инструмент, который копирует (фрагментированные) файлы в новый дескриптор ext4, указывающий на непрерывные экстенты.

Другим результатом дефрагментации на лету является уменьшение времени проверки файловой системы. (fsck). Ext4 помечает неиспользуемые группы блоков в таблице индексных дескрипторов, что позволяет процессу (fsck) полностью их пропускать и ускоряет тем самым процедуру проверки. Поэтому, когда операционная система решит проверить файловую систему после внутреннего повреждения (которые неизбежно будут происходить по мере увеличения размера файловой системы и ее распределенности), благодаря архитектуре ext4 это можно будет сделать быстро и надежно.

- В ext4 представлен механизм "пространственной" (extent) записи файлов. Новая информация добавляется на диск по определенному алгоритму, который специально уменьшает фрагментацию и повышает производительность. А также предполагается механизм дефрагментации, чего не было в предыдущих ext2 и ext3.
 - Для более точной работы с временными атрибутами файлов - время создания, модификации - используются наносекундные временные отметки (timestamps). Максимально возможное время увеличено до 25 апреля 2514 года, против 18 января 2038 года у Ext3.

все тоже, что написано выше, но покороче:

Типы файловых систем Linux и их особенности

Отличительными особенностями является скорость работы с файлами, безопасность и параметры (такие как размер блока), существующие по умолчанию и задаваемые при создании FS. Возможно, самой важной характеристикой является наличие журнала. В системный журнал записываются данные или **метаданные** (только заголовки) по которым информацию можно восстановить в случае сбоя.

Файловая система может создаваться на любом устройстве: на диске или системной партиции.

Файловая система EXT2

EXT2 является в настоящее время устаревшей файловой системой, которая практически не используется в современных инсталляциях. основной недостаток — отсутствие журналирования что, соответственно, делает невозможным восстановление данных в случае сбоя. По прежнему применяется на портативных носителях информации, таких как USB. Журнал для них не требуется, поскольку занимает определенное пространство.

Также гарантирует максимальную скорость работы.

- для EXT2 максимальный размер файла -2 ТВ
- максимальный размер всех файлов — 32 ТВ
-

Файловая система EXT3

Вытеснила EXT2, главной особенностью является появление журнала, является полностью обратно совместимой с EXT2 (EXT2 можно свободно конвертировать в EXT3). Сейчас встречается также редко, практически всегда используется EXT4.

Журнал — специальная область в памяти, в которую записывается информация обо всех изменениях

- для EXT3 максимальный размер файла -2 ТВ
- максимальный размер всех файлов — 32 ТВ
- в каждом каталоге может быть до 32 000 подкаталогов

При журналировании может быть три опции (указываются при создании файловой системы):

- journal – в журнал метаданные, а также сама информация
- ordered – опция по умолчанию, сохраняются только метаданные и после записи на диск
- writeback – также сохраняются только метаданные, можно выбрать сохранять их до записи на диск или после

Файловая система EXT4

Современная версия extended file system, чаще всего применяется именно она

- максимальный размер файла -2 ТВ 16 ТВ
- максимальный размер всех файлов — 1 EB (exabyte). 1 EB = 1024 PB (petabyte). 1 PB = 1024 TB (terabyte).
- в каждом каталоге может быть до 64 000 подкаталогов

В EXT4 ведение журнала можно выключить установив опцию **data** при монтировании в **off**