1. Определение операционной системы. ОС как расширенная машина. ОС как система управления ресурсами. 2

2. Эволюция ОС. 3

3. Классификация ОС. 5

4. Способы построения ОС. Монолитные системы. 6

5. Способы построения ОС. Многоуровневые системы. 7

6. Способы построения ОС. Модель клиент-сервер и микроядра. 8

7. Способы построения ОС. Объектно-ориентированный подход. 9

8. Управление процессами. Состояние процессов. Контекст и дескриптор процесса. 10

9. Управление процессами. Алгоритмы планирования процессов. 12

10. Управление процессами. Способы организации многозадачности. 13

11. Управление памятью. Типы адресов. Классификация методов распределения памяти. 14

12. Управление памятью. Распределение памяти фиксированными разделами. 15

13. Управление памятью. Распределение памяти разделами переменной величины. 16

14. Управление памятью. Перемещаемые разделы. 17

15. Управление памятью. Страничное распределение. 18

16. Управление памятью. Сегментное распределение. 19

17. Управление памятью. Странично-сегментное распределение. 20

18. Управлением памятью. КЭШ. 21

19. Управление вводом-выводом. Физическая организация устройств ввода-вывода. Организация программного обеспечения ввода-вывода. 22

20. Управление вводом-выводом. Обработка прерываний. Драйверы устройств. 24

21. Управление вводом-выводом. Независимый от устройств слой операционной системы. Пользовательский слой программного обеспечения. 25

22. Классификация файловых систем. 27

23. Файловые системы. Файл. Каталоги. Файловая система. Принцип хранения данных на жестком диске. 28

24. Файловая система FAT16 и VFAT. 30

25. Файловая система FAT 32. Разработчик – Microsoft; 31

26. Файловая система NTFS. 32

27. Файловая система EXT2, ETX3 и EXT4 33

# **1. Определение операционной системы. ОС как расширенная машина. ОС как система управления ресурсами.**

Ос выполняет две по существу мало связанные функции:

1. Обеспечение пользователю или программисту вместо реальной аппаратуры компьютера расширенную виртуальную машину, с которой удобней работать и которую легче программировать;
2. Повышение эффективности использования компьютера путем рационального управления ресурсами.

**ОС как расширенная машина**

ОС является предоставление пользователю некоторой расширенной или виртуальной машины, которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину.

**ОС как система управления ресурсами**

ОС должна управлять всеми ресурсами вычислительной машины таким образом, чтобы обеспечить максимальную эффективность её функционирования. Критерием эффективности может быть, например, пропускная способность или реактивность системы. Управление ресурсами включает решение двух общих, не зависищих от типа ресурса задачи:

1. Планирование ресурса – т.е. определение, кому, когда, а для делимых ресурсов и в каком количестве, необходимо выделить данный ресурс;
2. Отслеживание состояния ресурса – т.е. поддержание перативной информации о том, занят или не занят ресурс, а для делимых ресурсов, какое количество ресурса уже распределено, а какое уже свободно.

# **2. Эволюция ОС.**

**Первый период (1945 – 1955)**

После второй мировой войны. В середине 40-х были созданы первые ламповые вычислительные устройства. Программирование осуществлялось исключительно на машинном языке. Об операционных системах не было и речи.

**Второй период (1955 – 1965)**

С середины 50-х годов начался новый период в развитии вычислительной техники, связанный с появлением новой технической базы -полупроводниковых элементов. Компьютеры второго поколения стали более надежными, теперь они смогли непрерывно работать настолько долго, чтобы на них можно было возложить выполнение действительно практически важных задач. Именно в этот период произошло разделение персонала на программистов и операторов, эксплуатационников и разработчиков вычислительных машин.

В эти годы появились первые алгоритмические языки, а, следовательно, и первые системные программы - компиляторы. Появились первые системы пакетной обработки.

Стал доступен резистор, конденсатор. Спулинг – задачи на перфокартах.

**Третий период (1965 – 1980)**

Переход от резисторов к интегральным микросхемам. Первым семейством программно-совместимых машин, построенных на интегральных микросхемах, явилась серия машин IBM/360.

Стандартизация периферийных устройств и интерфейсов.

Программная совместимость – требовала и совместимости ОС. Такие ОС должны были бы работать и на больших, и на малых вычислительных системах, с большим и с малым количеством разнообразной периферии, в коммерческой области и в области с научных исследований.

Важным достижением ОС данного поколения явилась реализация мультипрограммирования. Мультипрограммирование – это способ организации вычислительного процесса, при котором на одном процессоре попеременно вычисляется несколько программ.

ОС разделения времени разделяет процессорное время между несколькими одновременно выполняющимися процессами, при этом у пользователя складывается впечатление, что каждое приложение обрабатывается собственным процессором.

Многотерминальные ОС одновременно представляют доступ к ресурсам одного компьютера нескольким пользователем.

**Четвертый период (1980 – н.в.)**

Следующий период в эволюции операционных систем связан с появлением больших интегральных схем (БИС). Компьютер стал доступен отдельному человеку, и наступила эра персональных компьютеров.

На рынке операционных систем доминировали две системы: MS-DOS и UNIX. В середине 80-х стали бурно развиваться сети персональных компьютеров, работающие под управлением сетевых или распределенных ОС.

В сетевых ОС пользователи должны быть осведомлены о наличии других компьютеров и должны делать логический вход в другой компьютер, чтобы воспользоваться его ресурсами, преимущественно файлами. Сетевая ОС не имеет фундаментальных отличий от ОС однопроцессорного компьютера.

Интерактивная ОС – это система разделения времени которая максимально ориентирована на удобство пользователя при работе на компьютере, где активно и адекватно реагирует на действия пользователя.

Система реального времени – это система разделения времени, которая реагирует в предсказуемое время на непредсказуемое появление внешних событий.

Выделяют ОС жесткого и мягкого реального времени. Система жесткого реального времени никогда не опоздает с реакцией на событие, а система мягкого реального времени не должна опаздывать с реакцией на событие.

# **3. Классификация ОС.**

**Особенности алгоритмов управления ресурсами**

Поддержка многозадачности. По числу выполняемых задач ОС могут быть разделены на два класса:

* Однозадачные (MS-DOS, MSX);
* Многозадачные (Windows, UNIX, Android).

Поддержка много пользовательского режима. По числу работающих пользователей ОС:

* Однопользовательские (MS-DOS, Windows 3.x, 95, 98);
* Многопользовательские (UNIX, Windows NT).

Способ организации многозадачности. Важнейшим разделяемым ресурсом является процессорное время. Способ Распределения процессорного времени между несколькими одновременно существующими в системе процессами:

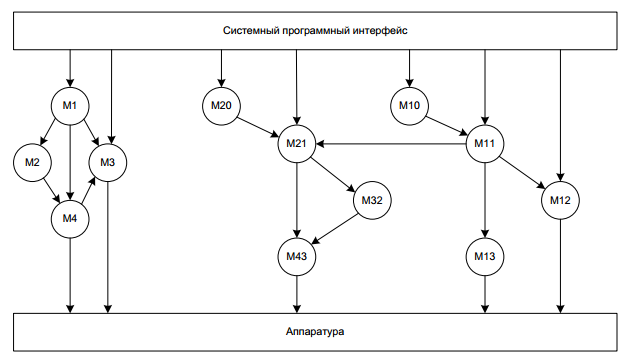
* Невытесняющая многозадачность (NetWare, Windows 3.x);
* Вытесняющая многозадачность (Windows NT, UNIX).

Поддержка многопоточности. Многопоточная (многонитевая) ОС разделяет

процессорное время не между задачами, а между их отдельными ветвями –потоками (нитями).

Многопроцессорная обработка. Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки - мультипроцессирование. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами. Асимметричная ОС целиком выполняется только на одном из процессоров системы, распределяя прикладные задачи по остальным процессорам. Симметричная ОС полностью децентрализована и использует весь пул процессоров, разделяя их между системными и прикладными задачами.

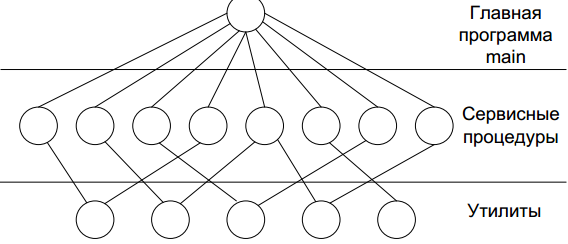
# **4. Способы построения ОС. Монолитные системы.**

Операционная система написана как набор процедур, каждая из которых может вызывать другие, когда ей это нужно. При использовании этой техники каждая процедура системы имеет хорошо определенный интерфейс в терминах параметров и результатов, и каждая вольна вызвать любую другую для выполнения некоторой нужной для нее полезной работы.

Однако даже такие монолитные системы могут быть немного структурированными. Такая организация ОС предполагает следующую структуру:

1. Главная программа, которая вызывает требуемые сервисные процедуры.
2. Набор сервисных процедур, реализующих системные вызовы.
3. Набор утилит, обслуживающих сервисные процедуры.

В этой модели для каждого системного вызова имеется одна сервисная процедура. Утилиты выполняют функции, которые нужны нескольким сервисным процедурам.

****

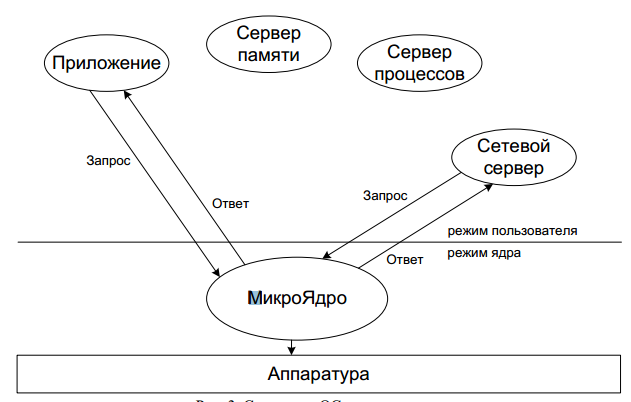
# **5. Способы построения ОС. Многоуровневые системы.**

Первой системой, построенной таким образом была простая пакетная система THE, которую построил Дейкстра и его студенты в 1968 году. Система имела 6 уровней. *Уровень 0* занимался распределением времени процессора. *Уровень 1* управлял памятью - выполнял функции виртуальной памяти. *Уровень 2* управлял связью между консолью оператора и процессами. *Уровень 3* управлял устройствами ввода-вывода и буферизовал потоки информации к ним и от них. На *уровне 4* работали пользовательские программы, которым не надо было заботиться ни о процессах, ни о памяти, ни о консоли, ни об управлении устройствами ввода-вывода. Процесс системного оператора размещался на *уровне 5*.

Дальнейшее обобщение многоуровневой концепции было сделано в ОС MULTICS.В системе MULTICS каждый уровень (называемый кольцом) является более привилегированным, чем вышележащий. Преимущество подхода MULTICS заключается в том, что он может быть расширен и на структуру пользовательских подсистем.

# **6. Способы построения ОС. Модель клиент-сервер и микроядра.**

В широком смысле модель клиент-сервер предполагает наличие программного компонента -потребителя какого-либо сервиса - клиента, и программного компонента -поставщика этого сервиса - сервера.



Применительно к структурированию ОС идея состоит в разбиение ее на несколько процессов - серверов, каждый из которых выполняет отдельный набор сервисных функций - например, управление памятью, создание или планирование процессов. Каждый сервер выполняется в пользовательском режиме. Клиент, которым может быть либо другой компонент ОС, либо прикладная программа, запрашивает сервис, посылая сообщение на сервер. Ядро ОС (называемое здесь микроядром), работая в привилегированном режиме, доставляет сообщение нужному серверу, сервер выполняет операцию, после чего ядро возвращает результаты клиенту с помощью другого сообщения.

Использование модели клиент-сервер повышает надежность. Каждый сервер выполняется в виде отдельного процесса в своей собственной области памяти, и таким образом защищен от других процессов.

# **7. Способы построения ОС. Объектно-ориентированный подход.**

Основным понятием этого подхода является "объект". Объект - это единица

программ и данных, взаимодействующая с другими объектам посредством приема и передачи сообщений. Объект может быть представлением как некоторых конкретных вещей - прикладной программы или документа, так и некоторых абстракций -процесса, события.

Программы (функции) объекта определяют перечень действий, которые могут быть выполнены над данными этого объекта. Объект-клиент может обратиться к другому объекту, послав сообщение с запросом на выполнение какой-либо функции объекта-сервера.

Объекты могут описывать сущности, которые они представляют, с разной степенью детализации. Для обеспечения преемственности при переходе к более детальному описанию разработчикам предлагается механизм наследования свойств уже существующих объектов, то есть механизм, позволяющий порождать более конкретные объекты из более общих. Например, при наличии объекта "текстовый документ" разработчик может легко создать объект "текстовый документ в формате Word 6.0", добавив соответствующее свойство к базовому объекту. Механизм наследования позволяет создать иерархию объектов, в которой каждый объект более низкого уровня приобретает все свойства своего предка.

Внутренняя структура данных объекта скрыта от наблюдения. Нельзя произвольно изменять данные объекта. Для того, чтобы получить данные из объекта или поместить данные в объект, необходимо вызывать соответствующие объектные функции. Это изолирует объект от того кода, который использует его. Разработчик может обращаться к функциям других объектов, или строить новые объекты путем наследования свойств других объектов, ничего не зная о том, как они сконструированы. Это свойство называется инкапсуляцией.

Использование объектно-ориентированного подхода особенно эффективно при создании активно развивающегося программного обеспечения, например, при разработке приложений, предназначенных для выполнения на разных аппаратных платформах.

Полностью объектно-ориентированные операционные системы очень привлекательны для системных программистов, так как, используя объекты системного уровня, программисты смогут залезать вглубь операционных систем для приспособления их к своим нуждам, не нарушая целостность системы.

# **8. Управление процессами. Состояние процессов. Контекст и дескриптор процесса.**

Процесс (или по-другому, задача) - абстракция, описывающая выполняющуюся программу. Для операционной системы процесс представляет собой единицу работы, заявку на потребление системных ресурсов. Подсистема управления процессами планирует выполнение процессов, то есть распределяет процессорное время между несколькими одновременно существующими в системе процессами, а также занимается созданием и уничтожением процессов, обеспечивает процессы необходимыми системными ресурсами, поддерживает взаимодействие между процессами.

**Состояние процессов**

В многозадачной (многопроцессной) системе процесс может находиться в одном из трех основных состояний:

1. ВЫПОЛНЕНИЕ - активное состояние процесса, во время которого процесс обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
2. ОЖИДАНИЕ - пассивное состояние процесса, процесс заблокирован, он не может выполняться по своим внутренним причинам, он ждет осуществления некоторого события, например, завершения операции ввода-вывода, получения сообщения от другого процесса, освобождения какого-либо необходимого ему ресурса;
3. ГОТОВНОСТЬ - также пассивное состояние процесса, но в этом случае процесс заблокирован в связи с внешними по отношению к нему обстоятельствами: процесс имеет все требуемые для него ресурсы, он готов выполняться, однако процессор занят выполнением другого процесса.

В ходе жизненного цикла каждый процесс переходит из одного состояния в другое в соответствии с алгоритмом планирования процессов, реализуемым в данной операционной системе.

На протяжении существования процесса его выполнение может быть многократно прервано и продолжено.

Состояние операционной среды отображается состоянием регистров и программного счетчика, режимом работы процессора, указателями на открытые файлы, информацией о незавершенных операциях ввода-вывода, кодами ошибок выполняемых данным процессом системных вызовов и т.д. Эта информация называется контекстом процесса.

Кроме этого, операционной системе для реализации планирования процессов требуется дополнительная информация: идентификатор процесса, состояние процесса, данные о степени привилегированности процесса, место нахождения кодового сегмента и другая информация. В некоторых ОС (например, в ОС UNIX) информацию такого рода, используемую ОС для планирования процессов, называют дескриптором процесса.

# **9. Управление процессами. Алгоритмы планирования процессов.**

Планирование процессов включает в себя решение следующих задач:

1. определение момента времени для смены выполняемого процесса;

2. выбор процесса на выполнение из очереди готовых процессов;

3. переключение контекстов "старого" и "нового" процессов.

В соответствии с алгоритмами, основанными на квантовании, смена активного процесса происходит, если:

• процесс завершился и покинул систему,

• произошла ошибка,

• процесс перешел в состояние ОЖИДАНИЕ,

• исчерпан квант процессорного времени, отведенный данному процессу.

Процесс, который исчерпал свой квант, переводится в состояние ГОТОВНОСТЬ и ожидает, когда ему будет предоставлен новый квант процессорного времени

**СИСТЕМА РАЗДЕЛЕНИЯ ВРЕМЕНИ**

Кванты, выделяемые процессам, могут быть одинаковыми для всех процессов или различными. Кванты, выделяемые одному процессу, могут быть фиксированной величины или изменяться в разные периоды жизни процесса. Процессы, которые не полностью использовали выделенный им квант (например, из-за ухода на выполнение операций ввода-вывода), могут получить или не получить компенсацию в виде привилегий при последующем обслуживании.

По-разному может быть организована очередь готовых процессов: циклически, по правилу "первый пришел - первый обслужился" (FIFO) или по правилу "последний пришел - первый обслужился" (LIFO).

Существует два основных типа процедур планирования процессов - вытесняющие (preemptive) и невытесняющие (non-preemptive).

Non-preemptive multitasking - невытесняющая многозадачность - это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking - вытесняющая многозадачность - это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

# **10. Управление процессами. Способы организации многозадачности.**

Существует 2 типа многозадачности:

1. *Процессная многозадачность* (основанная на процессах — одновременно выполняющихся программах). Здесь программа — наименьший элемент кода, которым может управлять планировщик операционной системы. Более известна большинству пользователей (работа в текстовом редакторе и прослушивание музыки).
2. *Поточная многозадачность* (основанная на потоках). Наименьший элемент управляемого кода — поток (одна программа может выполнять 2 и более задачи одновременно).

Существует два основных типа процедур планирования процессов - вытесняющие (preemptive) и невытесняющие (non-preemptive).

Non-preemptive multitasking - невытесняющая многозадачность - это способ планирования процессов, при котором активный процесс выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление планировщику операционной системы для того, чтобы тот выбрал из очереди другой, готовый к выполнению процесс.

Preemptive multitasking - вытесняющая многозадачность - это такой способ, при котором решение о переключении процессора с выполнения одного процесса на выполнение другого процесса принимается планировщиком операционной системы, а не самой активной задачей.

Мультипрограммирование теперь реализуется на уровне нитей, и задача, оформленная в виде нескольких нитей в рамках одного процесса, может быть выполнена быстрее за счет псевдопараллельного (или параллельного в мультипроцессорной системе) выполнения ее отдельных частей.

Подобно традиционным процессам (то есть процессам, состоящим из одной нити), нити могут находится в одном из следующих состояний: ВЫПОЛНЕНИЕ, ОЖИДАНИЕ и ГОТОВНОСТЬ.

Нити разделяют:

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

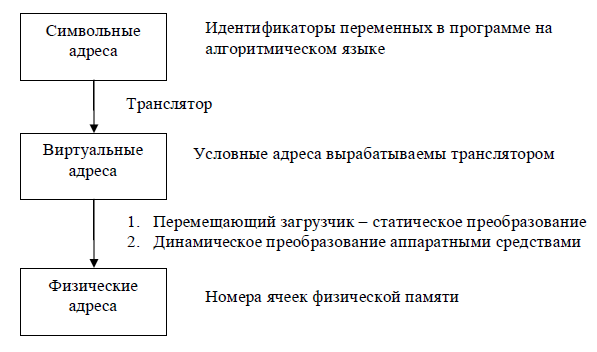
Особенно эффективно можно использовать многонитевость для выполнения распределенных приложений, например, многонитевой сервер может параллельно выполнять запросы сразу нескольких клиентов.

# **11. Управление памятью.** **Типы адресов. Классификация методов распределения памяти.**

Память является важнейшим ресурсом, требующим тщательного управления со стороны мультипрограммной операционной системы. Распределению подлежит вся оперативная память, не занятая операционной системой. Обычно ОС располагается в самых младших адресах, однако может занимать и самые старшие адреса.

Функциями ОС по управлению памятью являются: отслеживание свободной и занятой памяти, выделение памяти процессам и освобождение памяти при завершении процессов, вытеснение процессов из оперативной памяти на диск, когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место, а также настройка адресов программы на конкретную область физической памяти.

Для идентификации переменных и команд используются символьные имена (метки), виртуальные адреса и физические адреса. Символьные имена присваивает пользователь при написании программы на алгоритмическом языке или ассемблере. Виртуальные адреса вырабатывает транслятор, переводящий программу на машинный язык.

Начинаются с 0, виртуальное адресное пространство ограничено архитектурой и разрядностью компьютера. Физические адреса соответствуют номерам ячеек оперативной памяти, где в действительности расположены или будут расположены переменные и команды.

Переход от виртуальных адресов к физическим может осуществляться двумя способами. В первом случае замену виртуальных адресов на физические делает специальная системная программа - перемещающий загрузчик. Второй способ заключается в том, что программа загружается в память в неизмененном виде в виртуальных адресах, при этом операционная система фиксирует смещение действительного расположения программного кода относительно виртуального адресного пространства.

# **12. Управление памятью. Распределение памяти фиксированными разделами.**

Все методы управления памятью могут быть разделены на два класса: методы, которые используют перемещение процессов между оперативной памятью и диском, и методы, которые не делают этого. Начнем с последнего, более простого класса методов.

Самым простым способом управления оперативной памятью является разделение ее на несколько разделов фиксированной величины. Это может быть выполнено вручную оператором во время старта системы или во время ее инсталляции. Очередная задача, поступившая на выполнение, помещается либо в общую очередь, либо в очередь к некоторому разделу.

При очевидном преимуществе - простоте реализации - данный метод имеет существенный недостаток - жесткость. Даже если программа имеет небольшой объем, она будет занимать весь раздел, что приводит к неэффективному использованию памяти.

# **13. Управление памятью. Распределение памяти разделами переменной величины.**

В этом случае память машины не делится заранее на разделы. Сначала вся память свободна. Каждой вновь поступающей задаче выделяется необходимая ей память. Если достаточный объем памяти отсутствует, то задача не принимается на выполнение и стоит в очереди. После завершения задачи память освобождается, и на это место может быть загружена другая задача. Таким образом, в произвольный момент времени оперативная память представляет собой случайную последовательность занятых и свободных участков (разделов) произвольного размера.

Задачами операционной системы при реализации данного метода управления памятью является:

* ведение таблиц свободных и занятых областей, в которых указываются начальные адреса и размеры участков памяти,
* при поступлении новой задачи - анализ запроса, просмотр таблицы свободных областей и выбор раздела, размер которого достаточен для размещения поступившей задачи,
* загрузка задачи в выделенный ей раздел и корректировка таблиц свободных и занятых областей,
* после завершения задачи корректировка таблиц свободных и занятых областей.

Но ему присущ очень серьезный недостаток - фрагментация памяти. Фрагментация - это наличие большого числа несмежных участков свободной памяти очень маленького размера (фрагментов). Настолько маленького, что ни одна из вновь поступающих программ не может поместиться ни в одном из участков, хотя суммарный объем фрагментов может составить значительную величину, намного превышающую требуемый объем памяти.

# **14. Управление памятью. Перемещаемые разделы.**

Одним из методов борьбы с фрагментацией является перемещение всех занятых участков в сторону старших либо в сторону младших адресов, так, чтобы вся свободная память образовывала единую свободную область.

В дополнение к функциям, которые выполняет ОС при распределении памяти переменными разделами, в данном случае она должна еще время от времени копировать содержимое разделов из одного места памяти в другое, корректируя таблицы свободных и занятых областей. Эта процедура называется "сжатием".

Сжатие может выполняться либо при каждом завершении задачи, либо только тогда, когда для вновь поступившей задачи нет свободного раздела достаточного размера.

В первом случае требуется меньше вычислительной работы при корректировке таблиц, а во втором - реже выполняется процедура сжатия. Так как программы перемещаются по оперативной памяти в ходе своего выполнения, то преобразование адресов из виртуальной формы в физическую должно выполняться динамическим способом.

Хотя процедура сжатия и приводит к более эффективному использованию памяти, она может потребовать значительного времени, что часто перевешивает преимущества данного метода.

# **15. Управление памятью.** **Страничное распределение.**

Виртуальная память - это совокупность программно-аппаратных средств, позволяющих пользователям писать программы, размер которых превосходит имеющуюся оперативную память; для этого виртуальная память решает следующие задачи:

* размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
* перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
* преобразует виртуальные адреса в физические.

Современная ОС делит адресное пространство процесса на две части – пользовательскую и системную. По умолчанию обе части равны. Старшая часть для всех процессов одинакова.

Наиболее распространенными реализациями виртуальной памяти является страничное, сегментное и странично-сегментное распределение памяти, а также свопинг.

Страница – непрерывная последовательность адресов, лежащих в зад диапазоне фиксированного размера. 2^N

Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами.

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками).

Размер страницы обычно выбирается равным степени двойки: 512, 1024 и т.д., это позволяет упростить механизм преобразования адресов.

При каждом обращении к памяти происходит чтение из таблицы страниц информации о виртуальной странице, к которой произошло обращение. Если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический. Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди готовых. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то решается вопрос, какую страницу следует выгрузить из оперативной памяти.

В данной ситуации может быть использовано много разных критериев выбора, наиболее популярные из них следующие:

* дольше всего не использовавшаяся страница,
* первая попавшаяся страница,
* страница, к которой в последнее время было меньше всего обращений.

**16. Управление памятью. Сегментное распределение.**

Виртуальное адресное пространство процесса делится на части – сегменты, размер которых определяется с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями. Максимальный размер сегмента определяется разрядностью виртуального адреса, например, при 32-разрядной организации процессора он равен 4 Гбайт. При этом максимально возможное виртуальное адресное пространство процесса представляет собой набор из N виртуальных сегментов, каждый размером по 4 Гбайт. В каждом сегменте виртуальные адреса находятся в диапазоне от 0000000016 до FFFFFFFF16. Сегменты не упорядочиваются друг относительно друга, так что общего для сегментов линейного виртуального адреса не существует, виртуальный адрес задается парой чисел: номером сегмента или не линейным виртуальным адресом внутри сегмента.

На этапе создания процесса во время загрузки его образа в оперативную память система создает таблицу сегментов процесса (аналогичную таблице страниц), в которой для каждого сегмента указывается:

* базовый физический адрес сегмента в оперативной памяти;
* размер сегмента;
* правила доступа к сегменту;
* признаки модификации, присутствия и обращения к данному сегменту, а также некоторая другая информация.

Другим недостатком сегментного распределения является избыточность. При сегментной организации единицей перемещения между памятью и диском является сегмент, имеющий в общем случае объем больший, чем страница. Но главный недостаток сегментного распределения – это фрагментация, которая возникает из-за непредсказуемости размеров сегментов. В процессе работы системы в памяти образуются небольшие участки свободной памяти, в которые не может быть загружен ни один сегмент. Суммарный объем, занимаемый фрагментами, может составить существенную часть общей памяти системы, приводя к ее неэффективному использованию.

**17. Управление памятью. Странично-сегментное распределение.**

При сегментно-страничной организации виртуальной памяти происходит двухуровневая трансляция виртуального адреса в физический. В этом случае виртуальный адрес состоит из трех полей: номера сегмента виртуальной памяти, номера страницы внутри сегмента и смещения внутри страницы. Соответственно, используются две таблицы отображения -таблица сегментов, связывающая номер сегмента с таблицей страниц, и отдельная таблица страниц для каждого сегмента.

Достоинством сегментно-страничного способа организации памяти является

следующее:

* разбиение программы на сегменты позволяет размещать сегменты в памяти целиком;
* наличие сегментов облегчает реализацию разделения программных модулей между параллельными процессами;
* возможность динамической компоновки задачи;
* выделение памяти страницами позволяет минимизировать фрагментацию.

Недостатком сегментно-страничного способа организации памяти является то, что из-за значительных затрат вычислительных ресурсов и непростой реализации, он используется редко, причём в дорогих, мощных вычислительных системах.

# **18. Управлением памятью. КЭШ.**

Память вычислительной машины представляет собой иерархию запоминающих устройств (внутренние регистры процессора, различные типы сверхоперативной и оперативной памяти, диски, ленты), отличающихся средним временем доступа и стоимостью хранения данных в расчете на один бит. Пользователю хотелось бы иметь и недорогую и быструю память. Кэш-память представляет некоторое компромиссное решение этой проблемы.

Цена высокая, время доступа высокое.

1. Регистры процессора (регистры)
2. Кэш процессора (сверхоперативное ЗУ)
3. Оперативная память
4. Флэш\USB накопители
5. Жесткие диски (внешняя память)
6. Лента резервного копирования

Цена низкая, время доступа низкое.

Кэш-память - это способ организации совместного функционирования двух типов запоминающих устройств, отличающихся временем доступа и стоимостью хранения данных, который позволяет уменьшить среднее время доступа к данным за счет динамического копирования в "быстрое" ЗУ наиболее часто используемой информации из "медленного" ЗУ.

Структура кэш памяти:

1. Адрес данных в ОП
2. Данные
3. Информация – признак обращения и признак модификации.

В системах, оснащенных кэш-памятью, каждый запрос к оперативной памяти выполняется в соответствии со следующим алгоритмом:

1. Просматривается содержимое кэш-памяти с целью определения, не находятся ли нужные данные в кэш-памяти;
2. Если данные обнаруживаются в кэш-памяти, то они считываются из нее, и результат передается в процессор;
3. Если нужных данных нет, то они вместе со своим адресом копируются из оперативной памяти в кэш-память, и результат выполнения запроса передается в процессор.

**19. Управление вводом-выводом. Физическая организация устройств ввода-вывода. Организация программного обеспечения ввода-вывода.**

**Физическая организация устройств ввода-вывода**

Устройства ввода-вывода можно грубо разделить на две категории: блочные и символьные. Блочными называются устройства, хранящие информацию в виде адресуемых блоков фиксированного размера. Обычно размеры блоков варьируются от 512 до 32 768 байт. Важное свойство блочного устройства состоит в том, что каждый блок может быть прочитан независимо от остальных блоков. Наиболее распространенными блочными устройствами являются диски.

Другой тип устройств ввода-вывода — символьные устройства. Символьное устройство принимает или предоставляет поток символов без какой-либо блочной структуры. Символьное устройство не является адресуемым и не выполняет операцию поиска. Принтеры, сетевые интерфейсные адаптеры, мыши (для указания позиции на экране) и большинство других устройств, не похожих на диски, можно рассматривать как символьные устройства.

Электронный компонент называется контроллером устройства, или адаптером. В персональных компьютерах он обычно имеет вид печатной платы, вставляемой в слот расширения. Операционная система всегда имеет дело с контроллером, а не с самим устройством. У больших машин, мэйнфреймов, применяется другая модель с несколькими шинами, которые обслуживаются специализированными компьютерами ввода-вывода, называемыми каналами ввода-вывода. Такая организация позволяет снизить нагрузку на основной процессор.

У каждого контроллера есть несколько регистров, с помощью которых с ним может общаться центральный процессор. В дополнение к регистрам управления, многие устройства имеют буфер данных, доступный для чтения и записи со стороны операционной системы. Например, отображение пикселов на экране в большинстве компьютеров осуществляется с помощью видеопамяти. В сущности, видеопамять представляет собой буфер, в который программы операционной системы записывают отображаемые данные.

**Организация программного обеспечения ввода-вывода**

Ключевая концепция разработки программного обеспечения ввода-вывода известна как независимость от устройств. Эта концепция означает возможность написания программ, способных получать доступ к любому устройству ввода-вывода без предварительного указания конкретного устройства. Соответственно, программа, читающая данные из входного файла, должна с одинаковым успехом работать с файлом на дискете, жестком диске или компакт-диске. Причем без каких-либо изменений в программе.

Другим важным аспектом программного обеспечения ввода-вывода является обработка ошибок. Ошибки должны обрабатываться как можно ближе к аппаратуре. Если контроллер обнаружил ошибку чтения, он должен попытаться по возможности исправить эту ошибку сам. Если он не в силах это сделать, тогда ошибку обязан обработать драйвер устройства, допустим, попытавшись прочитать этот блок еще раз.

Еще один ключевой вопрос – способ передачи данных: синхронный (блокирующий) против асинхронного (управляемого прерываниями). Большинство операций ввода-вывода на физическом уровне являются асинхронными – центральный процессор начинает передачу данных и забывает о ней, пока не появится прерывание. Пользовательские программы значительно легче писать, применяя блокирующие операции ввода-вывода, после обращения к системному вызову receive программа автоматически приостанавливается до тех пор, пока данные не появятся в буфере.

Говоря о программном обеспечении ввода-вывода, нельзя обойти вниманием буферизацию. Часто данные, поступающие с устройства, не могут быть сохранены сразу там, куда они в конечном итоге направляются.

И последнее — это понятие выделенных устройств и разделяемых устройств. Если два или более пользователей одновременно станут писать вперемешку блоки на одну ленту, ничего хорошего не получится. Введение понятия выделенных (монопольно используемых) устройств также привносит целый спектр проблем, таких как взаимные блокировки. Тем не менее операционная система обязана управлять как разделяемыми, так и выделенными устройствами и преодолевать различные потенциальные проблемы самостоятельно.

**20. Управление вводом-выводом. Обработка прерываний. Драйверы устройств.**

**Обработчики прерываний**

Хотя программный ввод-вывод иногда бывает полезен, для большинства операций ввода-вывода приходится прибегать к прерываниям. Прерывания должны быть упрятаны как можно глубже во внутренностях операционной системы, чтобы о них знала, как можно меньшая ее часть. Лучший способ завуалировать их заключается в блокировке драйвера, начавшего операцию ввода-вывода, вплоть до окончания этой операции и получения прерывания. Драйвер может заблокировать себя сам, выполнив на семафоре процедуру down, процедуру wait на переменной состояния, процедуру receive на сообщении или что-либо подобное.

**Драйверы устройств**

Для управления каждым устройством ввода-вывода, подключенным к компьютеру, требуется специальная программа. Эта программа, называемая драйвером устройства, часто пишется производителем устройства и распространяется на компакт-дисках вместе с самим устройством. Поскольку для каждой операционной системы требуются специализированные драйверы, производители обычно поставляют драйверы для нескольких наиболее популярных операционных систем.

Чтобы драйвер имел доступ к аппаратной части устройства, то есть к регистрам контроллера, его традиционно интегрируют в ядро операционной системы. Такой подход обеспечивает максимальную производительность, но минимальную надежность, поскольку ошибка в любом драйвере устройства способна вывести из строя всю систему.

В некоторых операционных системах используется другая модель, например, в MINIX каждый драйвер устройства является отдельным процессом, выполняющимся в пользовательском пространстве.

Вообще говоря, назначение драйвера в том, чтобы воспринимать абстрактные запросы от аппаратно-независимых программ верхнего уровня и сообщать им, что запрос выполнен. Типичный запрос, поступающий драйверу диска, считать заданный блок данных. При этом если в момент передачи запроса драйвер бездействует, он сразу начинает работу. Если же драйвер занят, запрос обычно помещается в очередь и обслуживается по мере возможности.

**21. Управление вводом-выводом. Независимый от устройств слой операционной системы. Пользовательский слой программного обеспечения.**

**Независимое от устройств программное обеспечение ввода-вывода**

Точную границу между драйверами и независимым от устройств программным обеспечением проводит система, так как некоторые функции, которые можно реализовать независимо от устройств, часто выполняются прямо в драйверах из различных соображений, в том числе с позиций эффективности. Следующие функции обычно реализуются независимым от устройств программным обеспечением:

1. единообразный интерфейс для драйверов устройств;
2. буферизация;
3. сообщения об ошибках;
4. захват и освобождение выделенных устройств;
5. обеспечение аппаратно-независимого размера блока.

Главная цель независимого от устройств программного обеспечения – выполнение функций ввода-вывода, общих для всех устройств, и

предоставление единообразного интерфейса для программ пользовательского уровня.

Буферизация. Буферизация также является важной как для блочных, так и для символьных устройств. Для блочных устройств аппаратное обеспечение обычно требует, чтобы чтение или запись производились большими блоками. Однако для пользовательских программ такого ограничения нет, и они вправе передавать любые объемы информации.

Сообщения об ошибках. В контексте ввода-вывода ошибки — как нигде частое явление. Операционная система должна приложить максимальные усилия к их обработке. Многие ошибки являются специфичными для конкретного устройства и должны обрабатываться драйвером, т.к. только он знает, что делать (например, повторить попытку, игнорировать ошибку или инициировать сбой системы).

Захват и освобождение выделенных устройств. Некоторые устройства, например, привод CD-RW, рассчитаны на монопольное владение процессом в каждый момент времени. Операционная система должна рассмотреть запросы на использование такого устройства и либо принять их, либо отказать в выполнении запроса, в зависимости от доступности запрашиваемого устройства.

Обеспечение аппаратно-независимого размера блока. У различных дисков могут быть разные размеры сектора. Независимое от устройств программное обеспечение должно скрывать этот факт от верхних уровней и предоставлять им единообразный размер блока, например, объединяя несколько физических сегментов в одну логическую сущность. При этом более высокие уровни имеют дело только с абстрактными устройствами, с одним и тем же размером логического блока, не зависящим от размера физического сектора. Некоторые символьные устройства предоставляют свои данные побайтно (например, модемы), тогда как другие выдают их большими порциями (сетевые интерфейсы). Эти различия также могут быть скрыты.

**Пользовательский слой программного обеспечения**

Хотя большая часть программного обеспечения ввода-вывода относится к операционной системе, небольшие его порции состоят из библиотек, скомпонованных с пользовательскими программами, или даже целых программ, работающих вне ядра. Системные вызовы, включая системные вызовы ввода-вывода, обычно собраны из библиотечных процедур, например:

*count = write(fd, buffer, nbytes).*

Если *С++*-программа содержит такой вызов, библиотечная процедура write будет скомпонована с программой и, таким образом, окажется в двоичном коде, загружаемом в память во время выполнения программы. Набор всех этих библиотечных процедур, несомненно, является частью системы ввода-вывода.

Не все программное обеспечение ввода-вывода пользовательского пространства состоит из библиотечных процедур. Другая важная категория — это система спулинга. Спулинг (spooling) представляет собой способ работы с выделенными устройствами в многозадачной системе. Типичным устройством, на котором используется спулинг, является принтер.

**22. Классификация файловых систем.**

Файловые системы можно классифицировать на нижеследующие категории:

* Для носителей с произвольным доступом (например, жёсткий диск): FAT32, HPFS, ext2 и др. Поскольку доступ к дискам в несколько раз медленнее, чем доступ к оперативной памяти, для прироста производительности во многих файловых системах применяется асинхронная запись изменений на диск. Для этого применяется либо журналирование, например в ext3, ReiserFS, JFS, NTFS, XFS, либо механизм soft updates и др. Журналирование широко распространено в Linux, применяется в NTFS. Soft updates — в BSD системах.
* Для носителей с последовательным доступом (например, магнитные ленты): QIC и др.
* Для оптических носителей — CD и DVD: ISO9660, HFS, UDF и др.
* Виртуальные файловые системы: AEFS и др.
* Сетевые файловые системы: NFS, CIFS, SSHFS, GmailFS и др.
* Для флэш-памяти: YAFFS, ExtremeFFS, exFAT.

**23. Файловые системы. Файл. Каталоги. Файловая система. Принцип хранения данных на жестком диске.**

**Файлы и Каталоги**

Файл — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные, а также собственно хранимые в этой области данные и набор атрибутов, позволяющих ОС манипулировать этими данными.

Каталоги – это особый тип файлов, которые содержат системную справочную информацию о наборе файлов, сгруппированных пользователями по какому-либо неформальному признаку.

**Файловая система**

Файловая система (ФС) —это часть операционной системы, включающая:

* совокупность всех файлов на диске;
* наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;
* комплекс системных программных средств, реализующих различные операции с файлами, такие как создание, уничтожение, чтение, запись, именование и поиск файлов.

Файловые системы поддерживают несколько функционально различных типов файлов, в число которых, как правило, входят:

* обычные файлы;
* файлы-каталоги;
* специальные файлы;
* именованные конвейеры;
* файлы, отображаемые на память.

Обычные файлы, или просто файлы, содержат информацию произвольного характера, которую заносит в них пользователь или которая образуется в результате работы системных и пользовательских программ.

Специальные файлы —это фиктивные файлы, ассоциированные с устройствами ввода-вывода, которые используются для унификации механизма доступа к файлам и внешним устройствам.

Пользователи обращаются к файлам по символьным именам. Однако способности человеческой памяти ограничивают количество имен объектов, к которым пользователь может обращаться по имени. Иерархическая организация пространства имен позволяет значительно расширить эти границы. Именно поэтому большинство файловых систем имеет иерархическую структуру, в которой уровни создаются за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня, при этом частным случаем иерархической структуры является одноуровневая организация, когда все файлы входят в один каталог.

**Принцип хранения данных на жестком диске**

Файл, имеющий образ цельного, непрерывающегося набора байтов, на самом деле очень часто разбросан «кусочками» по всему диску, причем это разбиение никак не связано с логической структурой файла, например, его отдельная логическая запись может быть расположена Внесмежных секторах диска. Логически объединенные файлы из одного каталога совсем не обязаны соседствовать на диске.

Основным типом устройства, которое используется в современных вычисли­ тельных системах для хранения файлов, является дисковый накопитель. Дисковые накопители предназначены для считывания и записи данных на жесткие и гибкие магнитные диски.

Жесткий диск состоит из одной или нескольких стеклянных, или металлических пластин, каждая из которых покрыта с одной или двух сторон магнитным материалом. Таким образом, диск в общем случае состоит из пакета пластин.

На каждой стороне каждой пластины размечены тонкие концентрические кольца - дорожки (traks), на которых хранятся данные. Количество дорожек зависит от типа диска. Нумерация дорожек начинается с 0 от внешнего края к центру диска. Когда диск вращается, элемент, называемый головкой, считывает двоичные данные с магнитной дорожки или записывает их на магнитную дорожку.

**24. Файловая система FAT16 и VFAT.**

**FAT16**

Разработчик – Microsoft;

Полное название - File Allocation Table (Таблица размещения файлов) 16-битная версия;

Представлена – Ноябрь 1987 (MS-DOS 3.31);

Идентификатор тома – 0x04, 0x06, 0x0E ([MBR](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C));

Содержимое каталога – Таблица;

Размещение файлов – Линейный список;

Размер файла – 2 Гбайт;

Количество кластеров – 65 524;

Длина имени файла – 255 символов;

Размер тома – 2 Гбайт.

**VFAT**

VFAT – это расширение FAT, появившееся в Windows 95. В FAT имена файлов имеют формат 8.3 и состоят только из символов кодировки ASCII. В VFAT была добавлена поддержка длинных (до 255 символов) имён файлов (англ. Long File Name, LFN) в кодировке UTF-16LE, при этом LFN хранятся одновременно с именами в формате 8.3, ретроспективно называемыми SFN (англ. Short File Name). LFN нечувствительны к регистру при поиске, однако, в отличие от SFN, которые хранятся в верхнем регистре, LFN сохраняют регистр символов, указанный при создании файла.

**25. Файловая система FAT 32.**Разработчик – Microsoft;

Полное название - File Allocation Table (Таблица размещения файлов) 32-битная версия;

Представлена – Август 1996 (Windows 95 OSR 2);

Идентификатор тома – 0x0B, 0x0C ([MBR](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C));

Содержимое каталога – Таблица;

Размещение файлов – Линейный список;

Размер файла – 4 Гбайт;

Количество кластеров – 268 435 445;

Длина имени файла – 255 символов;

Размер тома – 2 Тбайт.

**26. Файловая система NTFS.**

Разработчик – Microsoft;

Полное название - New Technology File System (Файловая Система Новой Технологии);

Представлена – Июль 1993 (Windows NT 3.1);

Идентификатор тома – 0x07 ([MBR](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C));

Содержимое каталога – [B+ дерево](https://ru.wikipedia.org/wiki/B%2B_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) (структура данных, представляет собой сбалансированное дерево поиска);

Размещение файлов – [Bitmap](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BA%D0%B0%D1%80%D1%82%D0%B0) (набор последовательно записанных двоичных разрядов, то есть последовательность (массив) битов.);

Размер файла – 264 байт (16 [ЭБайт](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%B0%D0%B1%D0%B0%D0%B9%D1%82)) минус 1 КБ;

Количество кластеров – 4 294 967 295 (232−1);

Длина имени файла – 255 [UTF-16](https://ru.wikipedia.org/wiki/UTF-16) (один из способов кодирования символов из Unicode в виде последовательности 16-битных слов);

Размер тома – 264 − 1 кластер.

**27. Файловая система EXT2, ETX3 и EXT4**

**EXT2**

Разработчик – Реми Кард;

Полное название - Second extended file system;

Представлена – Январь 1993 (Linux);

Идентификатор тома – Apple\_UNIX\_SVR2 (Apple Partition Map)

0x83 (Master Boot Record);

Содержимое каталога – Таблица;

Размещение файлов – Битовая карта (свободное место), Таблица

(метаданные);

Размер файла – 16 ГБ - 2 ТБ;

Количество кластеров – 1018;

Длина имени файла – 255 байт;

Размер тома – 2-32 ТБ.

**EXT3**

Полное название - Third extended file system;

Представлена – Ноябрь 2001 (Linux 2.4.15);

Идентификатор тома – Apple\_UNIX\_SVR2 (Apple Partition Map)

0x83 (Master Boot Record);

Размещение файлов – Битовая карта (свободное место), Таблица (метаданные).

Максимальное число блоков для ext3 равняется 232. Размер блока может быть различным, что влияет на максимальное число файлов и максимальный размер файла в файловой системе.

**EXT4**

Разработчик – Mingming Cao, Andreas Dilger, Alex Zhuravlev (Tomas), Dave Kleikamp, Theodore Ts'o, Eric Sandeen, Sam Naghshineh и другие;

Полное название - Fourth extended file system;

Представлена – 21 октября 2008 (Linux 2.6.28, 2.6.19);

Идентификатор тома – 0x83 (MBR);

Содержимое каталога – [Связный список](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA), B-дерево;

Размещение файлов – [Extents](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D1%82%D0%B5%D0%BD%D1%82)(в файловых системах, непрерывная область носителя информации)/bitmap;

Размер файла – 4 миллиарда (задаётся во время создания ФС);

Количество кластеров – 1018;

Длина имени файла – 255 байт;

Размер тома – 1 [эксбибайт](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B8) (ограничен до 16 [тебибайт](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B8) из-за ограничений [e2fsprogs](https://ru.wikipedia.org/wiki/E2fsprogs) (это набор утилит для поддержания файловых систем ext2, ext3 и ext4)).