Министерство высшего образования и науки Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет математики, информационных и авиационных технологий
Кафедра информационных технологий

А.А. Чичев, Е.Г. Чекал

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Часть 4

Учебное пособие

Ульяновск
2019

УДК 004.052
ББК 32.973
Ч-78

Печатается по решению Ученого совета
факультета математики, информационных и авиационных технологий
Ульяновского государственного университета
(протокол №9/19 от 17.12.2019)

Рецензенты:
доцент кафедры «Прикладная математика и информатика»
УлГТУ, к.т.н. *Т.Е. Родионова*;
зав. кафедрой ТТС УлГУ, д.т.н., профессор *А.А. Смагин*

Чичев А.А.

Ч-78 **Операционные системы**. Часть 4 : учебное пособие / А.А. Чичев, Е.Г. Чекал. – Ульяновск : УлГУ, 2019. – 130 с.

Учебное пособие составлено в соответствии с программой дисциплины «Операционные системы» и предусматривает подготовку инженеров и бакалавров по направлениям: 09.03.02 «Информационные системы и технологии», 09.03.03 «Прикладная информатика», 02.03.03 «Математическое обеспечение и администрирование информационных систем», 11.03.02 «Инфокоммуникационные технологии и системы», – и специальности 10.05.01 «Компьютерная безопасность». Может использоваться студентами родственных специальностей и направлений.

Данное пособие носит характер дополнительной части пособия по операционным системам и состоит из нескольких разделов. В них приведены вопросы по операционным системам и ответы на них. Вопросы и ответы сгруппированы по темам. Вопросы и ответы носят практический характер, то есть предполагается, что студент знаком с практической реализацией соответствующей темы. Ответы даются по возможности краткие и вне контекста. Преподаватель может легко выяснить, реально ли студент знает материал или просто заучил ответы, выяснив контекст.

Пособие предназначено для практического руководства при проведении преподавателями лабораторных занятий и самостоятельного выполнении заданий студентами указанных направлений и специальностей всех форм обучения, а также как сборник дополнительных вопросов для зачётов и экзаменов.

УДК 004.052
ББК 32.973

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Исторически сложилось так, что в настоящее время самой быстро развивающейся, самой функционально продвинутой, самой сложной алгоритмически, самой большой по объёму и вообще самой-самой операционной системой стала операционная система Linux. Возможно, кому-то это высказывание не нравится, но, увы, оно истинно.

Именно в развитие этой операционной системы вкладываются усилия тысяч высококвалифицированных разработчиков, как частных лиц, так и сотрудников фирм. Объём проекта (Linux-4.x) приближается к 20 (двадцати!) млн. строк кода — ещё никогда человечество не разрабатывало столь сложные и объёмные программные системы.

Именно эта операционная система обеспечивает функционирование самых мощных ЭВМ нашего мира — согласно списка Top-500 её используют более 95% суперЭВМ [5].

Именно над развитием этой операционной системы работают сотрудники крупнейших корпораций мира — IBM, HP, Oracle, Intel, Google, Samsung и других, и даже Microsoft (!). Ещё совсем недавно, в 90-ые годы, основной вклад в развитие Linux делали частные лица. Однако, в последние десятилетия в крупнейших ИТ-корпорациях мира появились подразделения, специализирующиеся на разработке в Linux и на её совершенствовании.

Именно эта операционная система совместно с другими Unix'ами является основой Интернета, а «Интернет — это наше всё».

Именно эта операционная система совместно с другими Unix'ами и Unix-подобными ОС является базой для построения высоконадёжных высокодоступных (24x7) информационных систем.

Именно эта операционная система наряду с другими Unix'ами используется на серверах корпораций, банков, бирж, является основой систем управления в энергетике (в том числе АЭС), на транспорте (диспетчерские системы), в связи (АТС, провайдеры Интернета и сотовой связи), в государственных структурах.

То есть, зависимость современной экономики от Unix'овых операционных систем (в том числе, Linux) не просто большая, а основополагающая: если вдруг завтра проснёмся, а Unix/Linux исчез (вот только вчера был — и нет его . . .), то мало не покажется — в 2008 году только один банк «лопнул» и это привело к мировому экономическому кризису, а если все и всё?

Специфической особенностью Linux является открытость исходных текстов ОС — лицензия GPL, по которой распространяется Linux, запрещает закрывать исходники. А это значит, что любой желающий разобраться в том, как устроена эта высокотехнологичная, высоконадёжная ОС, может это сделать. Тем более, что исходники Linux хорошо документированы. Конечно, разобраться в миллионах строк кода — это задача очень нетривиальная. «Порог вхождения» в квалификацию очень высокий. Но возможность-то предоставляется!

По лицензии GPL распространяется не только сама ОС Linux, но и практически всё системное и прикладное ПО дистрибутивов Linux. А это десятки тысяч пакетов программ. Пример: репозитарий ALTLinux (Россия) содержит около 40 тысяч пакетов по состоянию на 2019 год.

А поскольку исходные тексты ОС открыты, то отсюда следует, что эта ОС сама по себе может являться учебным пособием по дисциплинам программирования, причём, учебным пособием очень продвинутым, написанным профессионалами. Аналогичными учебными пособиями являются и десятки тысяч пакетов, распространяемых также по лицензии GPL в составе дистрибутивов. И практически все программы и библиотеки имеют документацию — таково правило включения пакета в дистрибутив. Хотите стать профессиональным программистом? Нет проблем! Открывайте исходники linux-овых программ и самой ОС linux — и учитесь у профессионалов. То есть, действует основополагающий принцип педагогики: «Делай — как я!». Иначе говоря, для целей образования ОС Linux подходит очень хорошо: выполняется основной принцип образовательного процесса: «Делай как я! — делай лучше меня!». Реализуется основополагающее положение обучения по образцу.

Также следует сказать о том, что дистрибутивы Linux распространяются практически бесплатно — по лицензии GPL. То есть, реализуется иная модель бизнеса, нежели при распространении коммерческих дистрибутивов: заработок появляется не при продаже (перепродаже (спекуляции!) — у дистрибьютора) программы, а при сопровождении проданного, а это существенно более высокотехнологичное действие, требующее и гораздо больших знаний и квалификации, и большего количества людей. Сопровождение сложного ПО — это высокие и очень высокие технологии. Сопровождать сложное ПО — это намного сложнее, чем быть просто программистом. Научится программировать не сложно, программистов миллионы, а разобраться в сложной большой программе — о-о! это нужна квалификация, это не каждый может. Недаром существует у программеров поговорка: «Чем в этой проге разобраться — легче новую написать!».

Следовательно, использование Linux способствует развитию высокотехнологичных отраслей экономики.

И, наконец, очень важный момент: если используется Российский дистрибутив Linux, то и деньги за его разработку, использование и сопровождение остаются в России, то есть, у нас с Вами.

Поэтому в данном пособии рассматривается в основном операционная система Unix (и прежде всего — Linux), а остальные упоминаются по мере необходимости в целях сравнения. И основным дистрибутивом, рассматриваемом в пособии, является ALTLinux.

Пособие состоит из четырёх частей. В первой части («Операционные системы. Ч. 1. Работа с операционной системой» — [1]) приведены краткие сведения о работе основных подсистем операционных систем, а также методические указания к лабораторным работам по установке, конфигурированию и эксплуатации операционных систем.

В первом разделе этой части описываются:

— общие сведения о дисциплине «Операционные системы», о её месте в квалификации специалистов, о роли и значении операционных систем для экономики и общественной среды в целом и конкретных специалистов, в частности,

— определение, состав и структура операционной системы, внешние и внутренние интерфейсы, определено понятие процесса в операционной системе, описаны основные алгоритмы управления процессами и методы взаимодействия процессов,

— определены различные виды памяти вычислительных систем и способы управления памятью, особенности подсистемы ввода/вывода операционной системы,

— понятие оболочки операционной системы и операционной среды,

— введение в системное программирование.

Во втором разделе представлены:

— методические указания к лабораторным работам по установке, конфигурированию и эксплуатации операционных систем.

Во второй части пособия («Операционные системы. Ч. 2. Файловые системы») рассматриваются устройство средств хранения, форматы разбиения, файловые системы на устройствах долговременной памяти и внутреннее устройство конкретных файловых систем: ufs/ufs2, клоны ufs (в т.ч. ext), fat12/16/32, ntfs, iso 9660.

В третьей части пособия («Операционные системы. Ч. 3. Сетевая подсистема ОС») рассматриваются ЭМВОС, сетевые технологии и стандарты на них, стеки сетевых протоколов (SMB, IPX/SPX, TSP/IP,

AppleTalk, SNA), именование сетевых объектов на различных уровнях ЭМВОС, взаимодействие процессов и сервисы.

Четвёртая часть пособия («Операционные системы. Ч. 4. Ответы на вопросы») есть дополнительная часть пособия по операционным системам и состоит из нескольких разделов. В них приведены вопросы по операционным системам и ответы на них. Вопросы и ответы сгруппированы по темам. Вопросы и ответы носят практический характер, то есть, предполагается, что студент знаком с практической реализацией соответствующей темы. Ответы даются по возможности краткие и вне контекста. Преподаватель может легко выяснить, реально ли студент знает материал или просто заучил ответы, выяснив контекст.

Пособие предназначено для практического руководства при проведении преподавателями лабораторных занятий и выполнении заданий студентами указанных специальностей всех форм обучения, а также как сборник дополнительных вопросов для зачётов и экзаменов.

Учебно-методическое пособие составлено в соответствии с программой дисциплины «Операционные системы», и предусматривает подготовку инженеров и бакалавров по направлениям 09.03.02 «Информационные системы и технологии», 09.03.03 «Прикладная информатика», 02.03.03 «Математическое обеспечение и администрирование информационных систем», 11.03.02 «Инфокоммуникационные технологии и системы» и специальности 10.05.01 «Компьютерная безопасность». Может использоваться студентами родственных специальностей и направлений.

(Настоятельно!) рекомендуемый авторами дистрибутив Linux для целей образования (и не только) — это ALTLinux (берётся отсюда: http://ftp.altlinux.ru). Обоснование:

а) это самый хорошо локализованный дистрибутив: не только хорошо локализованы много программ (хорошо локализованный — это переведены и меню программного интерфейса, и программная документация), но и имеется очень много русской документации практически по всем вопросам (например, heap.altlinux.ru или wiki.altlinux.ru); этот пункт, пожалуй, важнейший в обосновании выбора, ибо несмотря на напряжённый труд учителей английского языка в школах и преподавателей кафедр иностранных языков в ВУЗах, знание студентами английского крайне посредственное;

б) самая лучшая поддержка в России, в том числе бесплатная, сообществом ALTLinux на форуме (http://forum.altlinux.org), в рассылках (https://www.altlinux.org/MailingLists), IRC (https://www.altlinux.org/IRC), в социальных сетях (https://telegram.me/alt_linux, https://vk.com/altlinux,

https://vk.com/simplylinux, https://www.facebook.com/groups/136328550579/, https://plus.google.com/communities/108911472444655347698) и прочих сервисах (https://www.altlinux.org/Contacts); среднее время ожидания ответа на вопрос — несколько часов; ни один другой дистрибутив linux, претендующий на «русскость», подобной скоростью похвастаться не может;

в) достаточно хорошая распространённость в России, в том числе, в школах;

г) это реально Российский дистрибутив — разработчики живут в РФ и репозитарий находится в России; причём репозитарий ALTLinux — это настоящий отдельный самостоятельный самодостаточный репозитарий с соответствующей инфраструктурой, а не зеркало какого-то там;

д) включен в Реестр (https://reestr.minsvyaz.ru/) — на 2019 год это дистрибутивы: «АлтОбразование-8», «АлтРабочаяСтанция-8», «АлтСервер-8» (https://reestr.minsvyaz.ru/reestr/?sort_by=date&sort=asc&class%5B%5D=54112&name=alt&owner_status=&owner_name=&set_filter=Y).

е) дистрибутив ALTLinux существует/распространяется как в виде полных сборок (почти всё, что нужно в одном .iso), так и в виде заготовок для создания своих сборок — стартекиты (StarterKit`s), что очень удобно для построения своих оригинальных систем; среди стартеркитов есть сборки и для слабых машин (и даже очень слабых) — https://www.altlinux.org/Starterkits/Memory

ОПЕРАЦИОННЫЕ СИСТЕМЫ

0. Граничные условия

1. Ответ студента — должен быть краткий: от одного до трёх предложений. Если преподаватель не поверит, он задаст дополнительный/уточняющий вопрос.

2. Файловые системы — рассматривается простой случай: одна файловая система — один раздел.

3. RAID — не рассматривается.

…

N. Перечень вопросов постоянно расширяется!

1. Архитектуры информационной системы

Типы архитектур информационных систем (ИС):

— монолит,

— файл-сервер,

— клиент-сервер,

— клиент-сервер-SOA,

— клиент-сервер-распределённая.

Подробнее см. в [2].

2. Архитектура SOA

Что это такое — проще всего показать рисунком. Пример архитектуры SOA на основе веб-сервиса — рисунок 1. Подробнее — смотрите в [2]. ИС с данной архитектурой могут также реализовываться на базе почтового сервера, сервера новостей и др.

2а. На основе каких сервисов может быть реализована ИС с архитектурой SOA?

На первом месте по числу реализаций — конечно, веб-сервис. Ранее, 20-30 лет назад и сейчас ещё встречаются, были распространены ИС на основе почтового сервиса. ИС на базе других сервисов — встречаются крайне редко.

Примечание. В данном вопросе имеются в виду не интегрированные в ИС сервисы, а именно базовые для ИС («несущие» всю конструкцию). То есть, примерно так, как показано на рисунке 1, где в простых случаях ИС — это только один квадратик «Бизнес-логика».

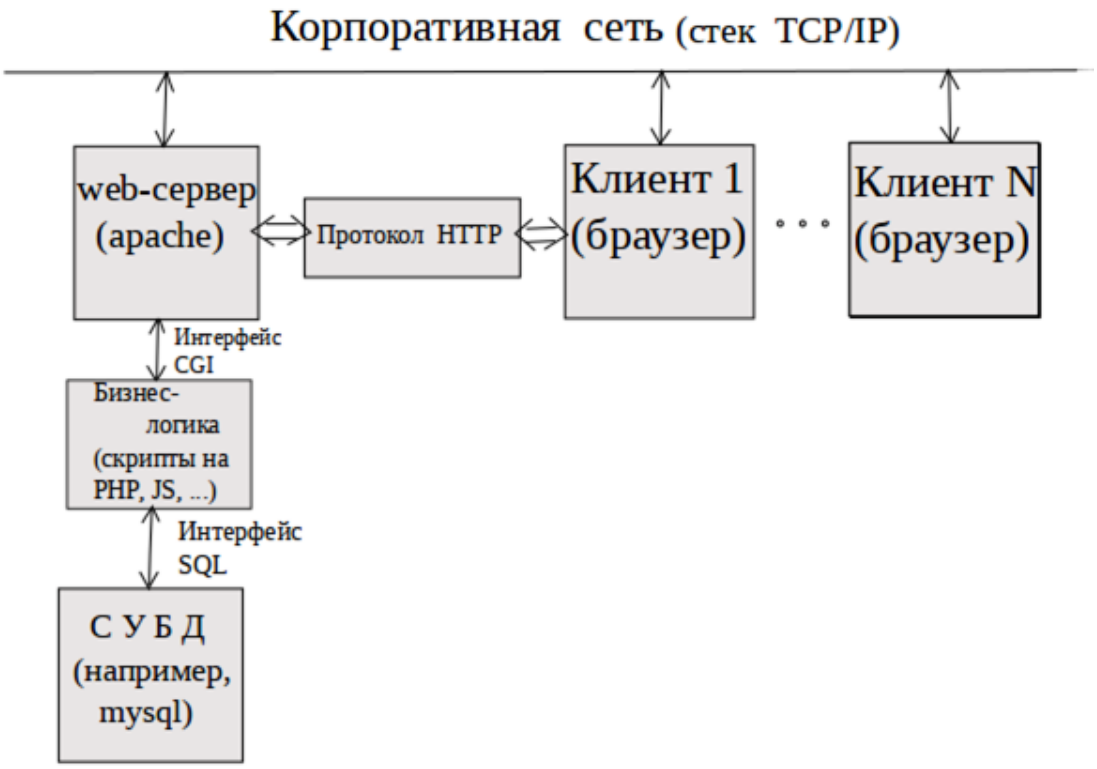

Рис. 1. Архитектура SOA

3. Что такое архитектура фон Неймана?

Ответ, конечно же, проще дать рисунком. На рисунке 2 — общая схема архитектуры фон Неймана, на рисунке 3 — схема более приближена к современным условиям. Управляющее устройство (УУ) и арифметико-логическое устройство (АЛУ) в настоящее время, как правило, существуют в виде одного «камня» CPU.

Рис. 2. Общая схема архитектуры фон Неймана

Рис. 3. Архитектура фон Неймана в более современном представлении

Двухшинная (Гарвардская) архитектура (см. рисунок 4) часто используется во встраиваемых ЭВМ (контроллерах и сигнальных процессорах). «Память команд» и «Память данных» — это ОЗУ, но разная.

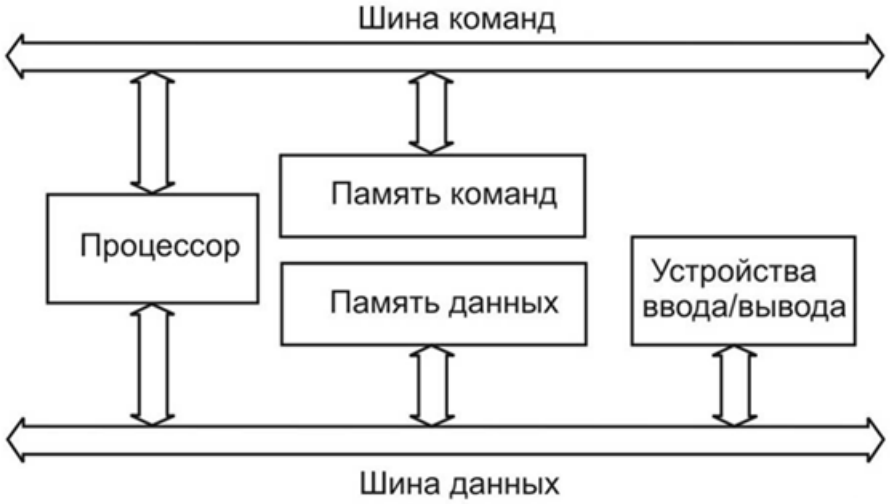

Рис. 4. Двухшинная (Гарвардская) архитектура

4. Что такое машина Поста?

Машина Поста – это абстрактная (несуществующая реально) вычислительная машина, созданная для уточнения (формализации) понятия алгоритма. Предложена в 1936 г. математиком Эмилем Постом. Машина Поста — это система, обладающая алгоритмической простотой и способная определять, является ли та или иная задача алгоритмически разрешимой. Если задача имеет алгоритмическое решение, то она представима в форме команд для машины Поста. Машиной эта математическая конст-

рукция названа потому, что при ее построении используются некоторые понятия реальных машин (ячейка памяти, команда и др.).

“Тезис Поста”: “Всякий алгоритм представим в форме программы для машины Поста”. Этот тезис одновременно является формальным определением алгоритма. То есть, а*лгоритм* (по Посту) — программа для машины Поста, приводящая к решению поставленной задачи. Тезис Поста — гипотеза. Его невозможно строго доказать, потому что в нем фигурируют, с одной стороны, интуитивное понятие “всякий алгоритм”, а с другой стороны — точное понятие “машина Поста”. Для того чтобы опровергнуть гипотезу Поста, необходимо придумать алгоритм, который невозможно записать в виде программы для машины Поста. На сегодняшний день такого алгоритма не существует (не придумано).

Машина Поста состоит из (см. рисунок 5):

— бесконечной ленты, поделённой на одинаковые ячейки (секции). Ячейка может быть пустой (то есть, 0) или содержать метку (например, 1 или любой другой знак V),

— головки (каретки), способной передвигаться по ленте на одну ячейку в ту или иную сторону, а также способной проверять наличие метки, стирать и записывать метку.

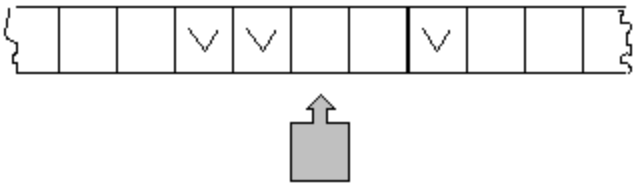

Рис. 5. Машина Поста

Информация о том, какие ячейки пусты, а какие содержат метки, образует ***состояние ленты***. Каретка может передвигаться вдоль ленты влево и вправо. Когда она неподвижна, она стоит против ровно одной ячейки ленты; говорят, что каретка обозревает одну ячейку. За единицу времени (один такт) каретка может совершить одно из трех действий: стереть метку, поставить метку, совершить движение на соседнюю ячейку. ***Состояние машины Поста*** складывается из состояния ленты и положения каретки.

Действия каретки подчинены программе, состоящей из перенумерованного набора команд (команды можно представлять как строки программы). Команды бывают шести типов:

n V i — записать 1 (метку V), перейти к *i*-й строке программы;

n X i — записать 0 (стереть метку V), перейти к *i*-й строке программы;

n ← i — сдвиг влево, перейти к *i*-й строке программы;

n → i — сдвиг вправо, перейти к *i*-й строке программы;

n ! — останов;

n ? i, j — если 0 (пусто), то перейти к *i*, иначе (занято) — перейти к *j*.

Формат команды:

<номер_команды><код_операции><номер_след_команды>

Список недопустимых действий, ведущих к аварийной остановке машины:

— попытка записать 1 (метку) в заполненную ячейку;

— попытка стереть метку в пустой ячейке.

Пример программы для машины Поста:

Задание. На ленте заданы два массива — *m* и *n*, *m* > *n*. Вычислить разность этих массивов. Каретка располагается над левой ячейкой правого массива *n*.

Решение. Запишем решение алгоритма в словесной форме.

1. Делаем шаг вправо и проверяем ячейку. Если она непустая (значит, предыдущая метка не последняя), то возвращаемся назад, иначе – переходим на шаг 6.
2. Стираем метку массива *n*.
3. Идем к правому краю массива *m*, двигаясь справа налево.
4. Стираем правую метку массива *m*.
5. Далее двигаемся слева направо до массива *n* и переходим на шаг 1.
6. Возвращаемся на предыдущую метку.
7. Стираем метку (стерли последнюю метку массива *n*).
8. Идем к правому краю массива *m*, двигаясь справа налево.
9. Стираем правую метку массива *m* и останавливаемся.

Программа:

1 –> 2
2 ? 10, 3
3 <– 4
4 X 5
5 <– 6
6 ? 5,7
7 X 8
8 –> 9
9 ? 8,1
10 <– 11

11 X 12
12 <– 13
13 ? 12,14
14 X 15
15 !

5. Что такое машина Тьюринга?

Интуитивное понимание машины Тьюринга (1936 год [19]) следующее: имеется бесконечная лента, разделённая на клетки. По клеткам ездит каретка. Прочитав букву, записанную в клетке, каретка движется вправо, влево или остаётся на месте, в последнем случае буква заменяется новой. Некоторые буквы останавливают каретку и завершают работу (см. рисунок 6).

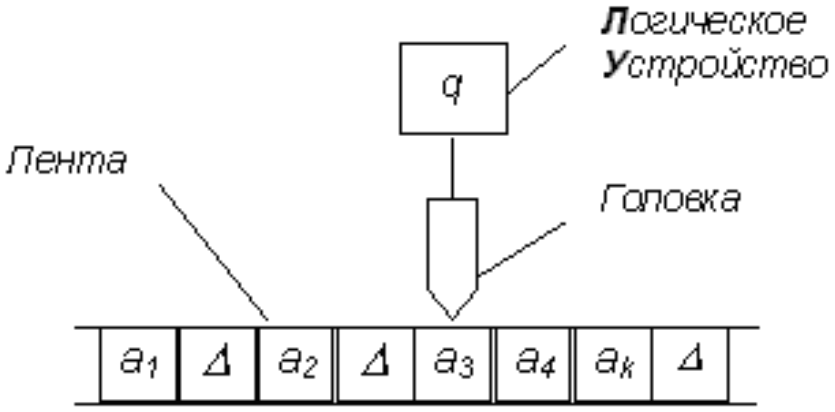

Рис. 6. Машина Тьюринга

Формальное описание

Математически, машину Тьюринга можно описать следующим образом:

Машина Тюринга (МТ)
это система вида $\{A,a0,Q,q1,q0,T,\tau\}$,
где

- A – конечное множество символов алфавита МТ
- $a0 \in A$ – пустой символ алфавита
- Q – конечное множество состояний МТ
- $q1 \in Q$ – начальное состояние МТ
- $q0 \in Q$ – конечное состояние МТ (состояние останова)
- $T=\{Л,П,Н\}$ – множество сдвигов МТ
- τ – программа МТ, то есть функция, задающая отображение $\tau: A \times Q \setminus \{q0\} \rightarrow A \times T \times Q$

Рассмотрим пример программирования машины Тьюринга.

Задание: Дана строка из букв “*a*” и “*b*”. Разработать машину Тьюринга, которая переместит все буквы “*a*” в левую, а буквы “*b*” — в правую части строки. Автомат в состоянии $q1$ обозревает крайний левый символ строки.

Возможние варианты:

aaa —> выходное слово совпадает с входным, просматриваем входное слово до тех пор, пока оно не заканчивается.

a —> выходное слово совпадает с входным, просматриваем входное слово до тех пор, пока оно не заканчивается.

bbb —> выходное слово совпадает с входным, просматриваем входное слово до тех пор, пока оно не заканчивается.

b —> выходное слово совпадает с входным, просматриваем входное слово до тех пор, пока оно не заканчивается.

ab —> выходное слово совпадает с входным, просматриваем входное слово до тех пор, пока оно не заканчивается.

Набросок алгоритма: Машина Тьюринга должна “понимать”, по цепочке каких букв она идет, т.е. у нее должно быть как минимум два состояния. Пусть состояние q_1 — движение по цепочке из букв “*a*”, а q_2 — состояние движения по цепочке из букв “*b*”. Заметим, что цепочка может состоять и из одной буквы. Если мы дошли до конца строки в состоянии q_1 или q_2, т.е. встретили a_0, машина должна остановиться, мы обработали всю строку.

Более сложные варианты:

bba —> abb

bbbaab —> aabbbb

aabbbaab —> aaaabbbb

Набросок алгоритма: Первый вариант входного слова можно последовательно обработать следующим образом: *bba* —> *bbb* —> вернуться к левому концу цепочки из букв “*b*” —> *abb* (заменить первую букву в этой цепочке на “*a*”). Для выполнения этих действий нам потребуется ввести два новых состояния и, кроме того, уточнить состояние q_2. Таким образом, для решения этой задачи нам нужно построить машину Тьюринга со следующими состояниями:

q_1 — идем вправо по цепочке букв “*a*”. Если цепочка заканчивается a_0, то переходим в q_0; если заканчивается буквой “*b*”, то переходим в q_2;

q_2 — идем вправо по цепочке букв “*b*”, если цепочка заканчивается a_0, то переходим в q_0; если заканчивается “*a*”, то заменяем букву “*a*” на “*b*”,

переходим в состояние $q3$ (цепочку вида $\underbrace{b\ldots ba}_{n}$ заменили на цепочку вида $\underbrace{b\ldots b}_{n+1}$);

$q3$ — идем влево по цепочке букв “b” до ее левого конца. Если встретили $a0$ или “a”, то переходим в $q4$;

$q4$ — заменяем “b” на “a” и переходим в $q1$ (цепочку вида $\underbrace{b\ldots b}_{n+1}$ заменяем на цепочку вида $a\underbrace{b\ldots b}_{n}$).

Программа представлена на рисунке 7.

	a_0	a	b
q_1	$a_0 H q_0$	$a П q_1$	$b П q_2$
q_2	$a_0 H q_0$	$b Л q_3$	$b П q_2$
q_3	$a_0 П q_4$	$a П q_4$	$b Л q_3$
q_4			$a П q_1$

Рис. 7. Программа для машины Тьюринга

6. Как выглядит алгоритм?

Алгоритм может выглядеть как:

а) блок-схема — чаще всего алгоритм нам представляется именно так; пример нахождения наибольшего общего делителя (НОД) на рисунке 8;

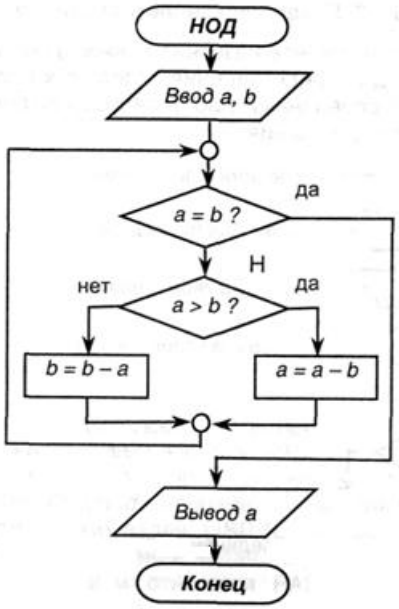

Рис. 8. Графическое представление алгоритма

б) пронумерованный план действий на русском языке (то есть, алгоритм представляет собой «пронумерованную последовательность строк, каждая из которых содержит описания конкретных действий на естественном языке» [1]); пример:

1. Если a = b, результатом считать a; закончить вычисления.

2. Если a > b, найти разность a — b; новым значением a считать значение разности; перейти к п. 1;

3. Если b > a, найти разность b — a; новым значением b считать значение разности; перейти к п. 1;

в) Формула — строчная запись действий (см. рисунок 9), обеспечивающих обработку числовых, символьных или логических данных;

$$y = \frac{a \cdot b^2 + c \cdot \sqrt{d}}{\sin(f) + h}$$

Рис. 9. Алгоритм в виде формулы для «человека»

но если алгоритм должен обрабатываться/исполняться некоторым устройством (например, ЭВМ), тогда вид формулы должна быть изменён к виду, удобному для считывания этим устройством (см. рисунок 10), например, так:

```
y:= (a*b*b + c*sqrt(d))/(sin(f) + h)
```

Рис. 10. Алгоритм в виде формулы для «ЭВМ»

г) Псевдокод — ориентированный на исполнителя «человек», частично формализованный язык, позволяющий записывать алгоритмы в форме, весьма близкой к алголоподобным языкам программирования; пример на рисунке 11:

```
АЛГ Евклид;
    ПОКА a ≠ b
        ПОВТОРЯТЬ
            ЕСЛИ a > b ТО a := a – b
                ИНАЧЕ b := b – a
            ВСЕ
        КЦ;
    ПЕЧАТЬ a
КНЦ.
```

Рис. 11. Алгоритм на псевдокоде

д) исходный текст программы на языке программирования высокого уровня или даже на ассемблере; как это выглядит — знаете;

е) исполняемый файл программы, которую можно запустить на ЭВМ — совершенно нечитабельное представление алгоритма, но таки оно есть; алгоритм можно увидеть командой ***cat <имя_файла> | less***

ж) представление в нотации Наура-Бэкуса;

з) представление алгоритма с помощью метасимволов;

и) представление алгоритма в виде графов — например, «сети Петри» или диаграммы Вирта;

к) представление алгоритма картинками (типа «комикс (см. рисунок 12); это представление нередко используется на упаковках продуктов «быстрого приготовления»: на пакетиках с кофе, лапшой «Ролтон» и пр.;

л) и т. д.

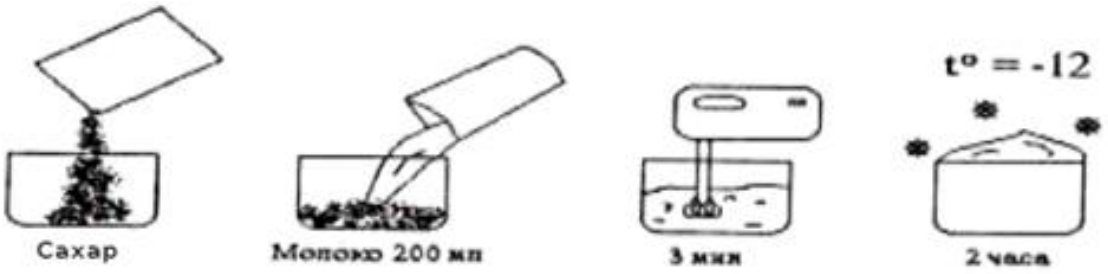

Рис. 12. Алгоритм (рецепт) приготовления мороженого

В целом, на рисунке 13 представлены способы представления алгоритмов.

Рис. 13. Способы представления алгоритмов

7. Лицензии: BSD vs GPL

7.1. Лицензия BSD

Ниже приведена лицензия BSD.

«Copyright 1992-2018 The FreeBSD Project.

1. Распространенные копии исходного кода должны содержать копирайты указанные выше, этот список условий и отказ от ответственности указанный ниже.

2. Распространенные копии бинарного кода должны воспроизводить копирайты указанные выше, этот список условий и отказ от ответственности указанный ниже в документации и/или других материалах поставляемых с распространяемым пакетом.

ЭТО ПРОГРАММА ПРЕДОСТАВЛЕНА БЕСПЛАТНО ДЕРЖАТЕЛЯМИ АВТОРСКИХ ПРАВ И/ИЛИ ДРУГИМИ СТОРОНАМИ "КАК ОНА ЕСТЬ" БЕЗ КАКОГО-ЛИБО ВИДА ГАРАНТИЙ, ВЫРАЖЕННЫХ ЯВНО ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ИМИ, ПОДРАЗУМЕВАЕМЫЕ ГАРАНТИИ КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ. НИ В КОЕМ СЛУЧАЕ, ЕСЛИ НЕ ТРЕБУЕТСЯ СООТВЕТСТВУЮЩИМ ЗАКОНОМ, ИЛИ НЕ УСТАНОВЛЕНО В УСНОЙ ФОРМЕ, НИ ОДИН ДЕРЖАТЕЛЬ АВТОРСКИХ ПРАВ И НИ ОДНО ДРУГОЕ ЛИЦО, КОТОРОЕ МОЖЕТ ИЗМЕНЯТЬ И/ИЛИ ПОВТОРНО РАСПРОСТРАНЯТЬ ПРОГРАММУ, КАК БЫЛО РАЗРЕШЕНО ВЫШЕ, НЕ ОТВЕТСТВЕННЫ ПЕРЕД ВАМИ ЗА УБЫТКИ, ВКЛЮЧАЯ ЛЮБЫЕ ОБЩИЕ, СЛУЧАЙНЫЕ, СПЕЦИАЛЬНЫЕ ИЛИ ПОСЛЕДОВАВШИЕ УБЫТКИ, ПРОИСТЕКАЮЩИЕ ИЗ ИСПОЛЬЗОВАНИЯ ИЛИ НЕВОЗМОЖНОСТИ ИСПЛЬЗОВАНИЯ ПРОГРАММЫ (ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОТЕРЕЙ ДАННЫХ, ИЛИ ДАННЫМИ, СТАВШИМИ НЕПРАВИЛЬНЫМИ, ИЛИ ПОТЕРЯМИ ПРИНЕСЕННЫМИ ИЗ-ЗИ ВАС ИЛИ ТРЕТЬИХ ЛИЦ, ИЛИ ОТКАЗОМ ПРОГРАММЫ РАБОТАТЬ СОВМЕСТНО С ДРУГИМИ ПРОГРАММАМИ), ДАЖЕ ЕСЛИ ТАКОЙ ДЕРЖАТЕЛЬ ИЛИ ДРУГОЕ ЛИЦО БЫЛИ ИЗВЕЩЕНЫ О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.

Взгляды и заключения, содержавшиеся в программном обеспечении и документации, принадлежат авторам и никак не относятся к Проекту FreeBSD.»

7.2. Лицензия GPL

Ниже приведена часть лицензии GPL.

«…

1. Лицензиат вправе изготовлять и распространять экземпляры исходного текста Программы в том виде, в каком он его получил, без внесения в него изменений на любом носителе, при соблюдении следующих условий: на каждом экземпляре помещен знак охраны авторского права и уведомление об отсутствии гарантий; оставлены без изменений все уведомления, относящиеся к настоящей Лицензии и отсутствию гарантий; вместе с экземпляром Программы приобретателю передается копия настоящей Лицензии.

Лицензиат вправе взимать плату за передачу экземпляра Программы, а также вправе за плату оказывать услуги по гарантийной поддержке Программы.

2. Лицензиат вправе модифицировать свой экземпляр или экземпляры Программы полностью или любую ее часть. Данные действия Лицензиата влекут за собой создание произведения, производного от Программы. Лицензиат вправе изготовлять и распространять экземпляры такого произведения, производного от Программы, или собственно экземпляры изменений в соответствии с пунктом 1 настоящей Лицензии при соблюдении следующих условий:

a) файлы, измененные Лицензиатом, должны содержать хорошо заметную пометку, что они были изменены, а также дату внесения изменений;

b) при распространении или публикации Лицензиатом любого произведения, которое содержит Программу или ее часть или является производным от Программы или от ее части, Лицензиат обязан передавать права на использование данного произведения третьим лицам на условиях настоящей Лицензии, при этом Лицензиат не вправе требовать уплаты каких-либо лицензионных платежей. Распространяемое произведение лицензируется как одно целое;

c) если модифицированная Программа при запуске обычно читает команды в интерактивном режиме, Лицензиат обязан обеспечить вывод на экран дисплея или печатающее устройство сообщения, которое должно включать в себя:

- знак охраны авторского права;
- уведомление об отсутствии гарантий на Программу (или иное, если Лицензиат предоставляет гарантии);

- указание на то, что пользователи вправе распространять экземпляры Программы в соответствии с условиями настоящей Лицензии, а также на то, каким образом пользователь может ознакомиться с текстом настоящей Лицензии. (Исключение: если оригинальная Программа является интерактивной, но не выводит в своем обычном режиме работы сообщение такого рода, то вывод подобного сообщения произведением, производным от Программы, в этом случае не обязателен).

Вышеуказанные условия применяются к модифицированному произведению, производному от Программы, в целом. В случае если отдельные части данного произведения не являются производными от Программы, являются результатом творческой деятельности и могут быть использованы как самостоятельное произведение, Лицензиат вправе распространять отдельно такое произведение на иных лицензионных условиях. В случае если Лицензиат распространяет вышеуказанные части в составе произведения, производного от Программы, то условия настоящей Лицензии применяются к произведению в целом, при этом права, приобретаемые сублицензиатами на основании Лицензии, передаются им в отношении всего произведения, включая все его части, независимо от того, кто является их авторами.

…

ОТСУТСТВИЕ ГАРАНТИЙНЫХ ОБЯЗАТЕЛЬСТВ

11. ПОСКОЛЬКУ НАСТОЯЩАЯ ПРОГРАММА РАСПРОСТРАНЯЕТСЯ БЕСПЛАТНО, ГАРАНТИИ НА НЕЕ НЕ ПРЕДОСТАВЛЯЮТСЯ В ТОЙ СТЕПЕНИ, В КАКОЙ ЭТО ДОПУСКАЕТСЯ ПРИМЕНИМЫМ ПРАВОМ. НАСТОЯЩАЯ ПРОГРАММА ПОСТАВЛЯЕТСЯ НА УСЛОВИЯХ "КАК ЕСТЬ". ЕСЛИ ИНОЕ НЕ УКАЗАНО В ПИСЬМЕННОЙ ФОРМЕ, АВТОР И/ИЛИ ИНОЙ ПРАВООБЛАДАТЕЛЬ НЕ ПРИНИМАЕТ НА СЕБЯ НИКАКИХ ГАРАНТИЙНЫХ ОБЯЗАТЕЛЬСТВ, КАК ЯВНО ВЫРАЖЕННЫХ, ТАК И ПОДРАЗУМЕВАЕМЫХ, В ОТНОШЕНИИ ПРОГРАММЫ, В ТОМ ЧИСЛЕ ПОДРАЗУМЕВАЕМУЮ ГАРАНТИЮ ТОВАРНОГО СОСТОЯНИЯ ПРИ ПРОДАЖЕ И ПРИГОДНОСТИ ДЛЯ ИСПОЛЬЗОВАНИЯ В КОНКРЕТНЫХ ЦЕЛЯХ, А ТАКЖЕ ЛЮБЫЕ ИНЫЕ ГАРАНТИИ. ВСЕ РИСКИ, СВЯЗАННЫЕ С КАЧЕСТВОМ И ПРОИЗВОДИТЕЛЬНОСТЬЮ ПРОГРАММЫ, НЕСЕТ ЛИЦЕНЗИАТ. В СЛУЧАЕ ЕСЛИ В ПРОГРАММЕ БУДУТ ОБНАРУЖЕНЫ НЕДОСТАТКИ, ВСЕ РАСХОДЫ, СВЯЗАННЫЕ С ТЕХНИЧЕСКИМ ОБСЛУЖИВАНИЕМ, РЕМОНТОМ ИЛИ ИСПРАВЛЕНИЕМ ПРОГРАММЫ, НЕСЕТ ЛИЦЕНЗИАТ.

12. ЕСЛИ ИНОЕ НЕ ПРЕДУСМОТРЕНО ПРИМЕНЯЕМЫМ ПРАВОМ ИЛИ НЕ СОГЛАСОВАНО СТОРОНАМИ В ДОГОВОРЕ В ПИСЬМЕННОЙ ФОРМЕ, АВТОР И/ИЛИ ИНОЙ ПРАВООБЛАДАТЕЛЬ, КОТОРЫЙ МОДИФИЦИРУЕТ И/ИЛИ РАСПРОСТРАНЯЕТ ПРОГРАММУ НА УСЛОВИЯХ НАСТОЯЩЕЙ ЛИЦЕНЗИИ, НЕ НЕСЕТ ОТВЕТСТВЕННОСТИ ПЕРЕД ЛИЦЕНЗИАТОМ ЗА УБЫТКИ, ВКЛЮЧАЯ ОБЩИЕ, РЕАЛЬНЫЕ, ПРЕДВИДИМЫЕ И КОСВЕННЫЕ УБЫТКИ (В ТОМ ЧИСЛЕ УТРАТУ ИЛИ ИСКАЖЕНИЕ ИНФОРМАЦИИ, УБЫТКИ, ПОНЕСЕННЫЕ ЛИЦЕНЗИАТОМ ИЛИ ТРЕТЬИМИ ЛИЦАМИ, НЕВОЗМОЖНОСТЬ РАБОТЫ ПРОГРАММЫ С ЛЮБОЙ ДРУГОЙ ПРОГРАММОЙ И ИНЫЕ УБЫТКИ). АВТОР И/ИЛИ ИНОЙ ПРАВООБЛАДАТЕЛЬ В СООТВЕТСТВИИ С НАСТОЯЩИМ ПУНКТОМ НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ДАЖЕ В ТОМ СЛУЧАЕ, ЕСЛИ ОНИ БЫЛИ ПРЕДУПРЕЖДЕНЫ О ВОЗМОЖНОСТИ ВОЗНИКНОВЕНИЯ ТАКИХ УБЫТКОВ.

…»

Выводы:

1. Лицензия BSD не является copyleft-лицензией. То есть, она не требует от производного кода быть свободным.

2. Лицензия BSD совместима с GPL-лицензией,

3. GPL и BSD олицетворяют собой две точки зрения на мир свободного программного обеспечения:

- GPL считает, что ПО должно быть свободным и порождать свободное ПО. «Свобода или Смерть» — лозунг Столлмана как нельзя лучше отражает смысл этой точки зрения;

- BSD предполагает, что свобода должна заключаться **и в том**, чтобы делать несвободное ПО. Это более либеральная точка зрения. По сути, здесь возникает философский вопрос: можно ли считать свободой разрешение на ограничение свободы.

А на самом деле хорошо, что большинство разработчиков не увлекаются и подходят к вопросу просто: если они хотят видеть части своего ПО только в подобных свободных продуктах, то они выбирают GPL, если им **все равно**, то они выбирают BSD-лицензии.

8. Основное положение лицензий на ПО (любых)

ДАННОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРЕДОСТАВЛЯЕТСЯ «КАК ЕСТЬ», БЕЗ ЛЮБОГО ВИДА ГАРАНТИЙ, . . .

Смотри этот пункт полностью выше в лицезиях BSD и GPL.

А вот кусочек из коммерческой лицензии на MS Office:

«15.1 Отсутствие гарантийных обязательств. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ WindowsOffice ПРЕДОСТАВЛЯЕТСЯ «КАК ЕСТЬ» БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, . . .»

9. Основные положения коммерческих лицензий на программное обеспечение

В коммерческих лицензиях, прежде всего, определяется то, что Вы **НЕ** имеете право делать. Наиболее полно и подробно это описано в лицензиях Microsoft. Ниже для примера приведены выдержки из лицензии на ОС Windows-7 — это пример лицензии «на компьютер». Вы её найдёте на каждом компьютере с ОС Windows-7.

«…

2. ПРАВА НА УСТАНОВКУ И ИСПОЛЬЗОВАНИЕ.

а. Одна копия на компьютер. Вы можете установить одну копию программного обеспечения на одном компьютере. Такой компьютер называется «лицензированным компьютером».

б. Лицензированный компьютер. С программным обеспечением могут одновременно работать не более двух процессоров на лицензированном компьютере. Если условиями этой лицензии не предусмотрено иное, данное программное обеспечение нельзя использовать на других компьютерах.

в. Количество пользователей. Если условиями этой лицензии не предусмотрено иное, несколько пользователей не могут работать с данным программным обеспечением одновременно.

г. Разные версии. Программное обеспечение может включать несколько версий (например, 32— и 64-разрядную). Вы имеете право установить и одновременно использовать только одну версию.

…

б. Компоненты шрифтов. Во время работы программного обеспечения вы имеете право использовать его шрифты <только> для отображения и печати содержимого.

…

в. Значки, изображения и звуки. Во время работы программного обеспечения вы можете использовать значки, изображения, звуки и элементы мультимедиа, не передавая их третьим лицам. . . .

д. Подключение устройств. К установленному на лицензированном компьютере программному обеспечению могут иметь доступ до 20 других устройств исключительно в целях использования файловых служб, служб печати, служб IIS, телефонных служб и для обеспечения общего доступа к подключению к Интернету

...

Если программное обеспечение не активировано, вы не имеете права его использовать после окончания периода активации. Это необходимо для предотвращения незаконного использования программного обеспечения. **Запрещается обходить процедуру активации каким-либо образом.**

...

а. Функция проверки проверяет, активировано ли программное обеспечение и имеется ли на него лицензия. … **Обходить процедуру проверки запрещено**.

…

Microsoft имеет право <*на вашем компьютере — прим. авторов*>:

— исправлять программное обеспечение, удалять, помещать в карантин или отключать все несанкционированные изменения, которые могут влиять на надлежащее использование программного обеспечения, включая обход функций активации или проверки программного обеспечения;

...

Возможно, вы будете

— вынуждены следовать указаниям Microsoft для получения лицензии на использование программного обеспечения и его повторной активации.

Возможно, вы не сможете: <*если не будете слушаться — прим. авторов*>

— продолжать использовать программное обеспечение или некоторые его функции;

…

8. ОБЪЕМ ЛИЦЕНЗИИ. Программное обеспечение не продается, а предоставляется в пользование по лицензии. . . . Вы не имеете права:

— пытаться обойти технические ограничения в программном обеспечении;

— реконструировать, декомпилировать или дизассемблировать программное обеспечение, …

— использовать компоненты программного обеспечения для работы с приложениями, не предназначенными для работы с этим программным обеспечением;

— создавать больше копий программного обеспечения, чем указано в этом соглашении …;

— публиковать программное обеспечение, предоставляя другим лицам возможность его копировать;

— предоставлять программное обеспечение в прокат, в аренду или во временное пользование;

— использовать это программное обеспечение для предоставления сетевых услуг на коммерческой основе.

….

18. ПЕРЕДАЧА ТРЕТЬЕЙ СТОРОНЕ.

… Первый пользователь данного программного обеспечения может единовременно напрямую передать третьему лицу программное обеспечение и это соглашение путем передачи оригинального носителя, сертификата подлинности, ключа продукта и подтверждения покупки. Первый пользователь должен предварительно удалить программное обеспечение с компьютера, чтобы передать его отдельно от этого компьютера. Первый пользователь не имеет права оставлять у себя какие-либо копии программного обеспечения.

…

28. ОГРАНИЧЕНИЕ И ИСКЛЮЧЕНИЕ ОТВЕТСТВЕННОСТИ ЗА УБЫТКИ И УЩЕРБ. Вы можете взыскать с Microsoft и ее партнеров только прямые убытки в пределах суммы, уплаченной вами за программное обеспечение. Вы не можете взыскать никакие другие убытки, включая косвенные, специальные, опосредованные или случайные убытки, а также убытки в связи с упущенной выгодой.

…

А. ОГРАНИЧЕННАЯ ГАРАНТИЯ. Если на программное обеспечение есть официальная лицензия, и вы следуете инструкциям, программное обеспечение будет по существу работать так, как описано в материалах Microsoft, полученных с программным обеспечением или в его составе.

Б. СРОК ГАРАНТИИ; ПОЛУЧАТЕЛЬ ГАРАНТИИ; СРОК ДЕЙСТВИЯ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ. Ограниченная гарантия на программное обеспечение действует в течение одного года с даты приобретения программного обеспечения первым пользователем. ...

Г. РАЗМЕР КОМПЕНСАЦИИ В СЛУЧАЕ НАРУШЕНИЯ УСЛОВИЙ ГАРАНТИИ. Microsoft бесплатно исправит или заменит программное

обеспечение. Если Microsoft не сможет исправить или заменить программное обеспечение, Microsoft возместит сумму, уплаченную за программное обеспечение, согласно товарному чеку. Microsoft также бесплатно исправит или заменит программное обеспечение, являющееся дополнением, обновлением или заменой. Если Microsoft не сможет исправить или заменить программное обеспечение, Microsoft возместит уплаченную за них сумму (если оплата имела место). Для получения денежного возмещения вы должны удалить программное обеспечение с компьютера и вернуть Microsoft носители и другие соответствующие материалы вместе с документом, подтверждающим покупку…»

Вывод. IMHO, гораздо легче сказать, что «зя», нежели перечислять, что «низя». Особенно любопытны абзацы, следующие после пункта 2.д.

10. Терминал, консоль и командная строка

terminal = текстовая среда ввода/вывода. При «царе Горохе» это были физические устройства, состоявшие из дисплея и клавиатуры и подключавшиеся к ЭВМ через специальные многоканальные интерфейсные устройства. Термин: канал — это линия связи терминал-ЭВМ. Терминалы располагались обычно удалённо, в других комнатах, чтобы позволить многим пользователям одновременно работать на ЭВМ в режиме командной строки. На ЭВМ при этом должна была работать многозадачная и многопользовательская ОС. Со стороны ЭВМ в операционной системе для каждой линии запускался shell — оболочка. В настоящее время такие устройства уже большая редкость (практически не выпускаются), а **terminal** теперь — **это программа**, которая эмулирует тот старый физический терминал и **которая позволяет работать с ЭВМ в режиме командной строки.**

В 80-90-ые годы также выпускались графические терминалы, в состав которых дополнительно входили мышки и которые позволяли работать с ЭВМ в графическом режиме. На ЭВМ при этом должна была быть запущена многозадачная многопользовательская ОС, поддерживавшая работу с таких графических устройств, например, unix с сервисом X.

console = один физический терминал, непосредственно подключенный к системному блоку, в отличие от других терминалов, которые подключаются через интерфейсные устройства; состоит из дисплея, клавиатуры и (в настоящее время) мышки. Это средство управления ЭВМ для системного администратора. Как правило, присутствует во всех ЭВМ, кроме встроенных. Для ПЭВМ console — это дисплей+клавиатура+мышка, под-

ключенные к системному блоку, с помощью которых пользователь работает на ПЭВМ. Для ПЭВМ он же системный администратор.

Командная строка — это интерфейс (типа «человек-ЭВМ»), в котором пользователь вводит команду с клавиатуры и нажимает Enter для выполнения этой команды. Команда здесь — это последовательность символов — имя команды (очень часто это имя программы), за которой следуют через пробел некоторые параметры.

11. Что такое командная оболочка?

Командная оболочка (*shell*) — командный интерпретатор (программа), используемый в операционных системах unix/linux, в котором пользователь может либо давать команды операционной системе по отдельности, либо запускать скрипты, состоящие из списка команд. Часто используется для запуска других программ. В качестве команд интерпретируются вызовы системных или прикладных программ, а также управляющие конструкции языка shell.

12. Что такое команда в ОС Linux?

Команда (текстовой) оболочки **Linux** — это строка символов из имени **команды** и аргументов, разделенных пробелами. Аргументы предоставляют **команде** дополнительные параметры, определяющие ее поведение. Как правило, это запуск другой программы и команда является именем файла этой другой программы. Аргументы или опции (или ключи) команды передаются в запускаемую программу и модифицируют её поведение. Имя программы и её аргументы определяют входной язык программы.

Другой случай: команда является оператором встроенного языка программирования shell — см. руководство по программированию на shell.

Команда графической оболочки Linux — это щелчок мышкой по иконке или выбор мышкой некоторого пункта меню. Как правило, это тоже запуск другой программы. Если мышкой выбирается пункт меню запущенной программы, то это модифицирует работу этой программы. Состав меню программы и действия мышкой в окне программы определяют входной язык программы.

13. Что такое man и как его посмотреть?

man — справочная страница (manual) на какую-либо команду (программу) или функцию системы. Эти справочные страницы поставляются почти со всеми *nix-дистрибутивами. Каждая страница справки является

самостоятельным документом. Так принято в мире unix/linux, что каждая программа или функция библиотеки (например, libc) должна снабжаться справочной страницей — мануалом, руководством по использованию этой программы или функции. Все man'ы складываются в специально определённый для них каталог в файловой системе, обычно /usr/share/man.

man — команда unix/linux, предназначенная для форматирования и вывода справочных страниц по командам и функциям системы. Например, если ввести в терминале:

man xman

то получим вывод в окне терминала справочного руководства по программе xman.

Выход из режима просмотра мануала — «q».

*Примечание***.** Важными признаками хорошей локализации дистрибутива являются:

— наличие в системе достаточного количества переведённых на язык локализации man'ов;

— наличие достаточного количества документации на языке локализации;

— переведённый на язык локализации интерфейс программ (то есть, меню).

14. Как появляются в системе man'ы?

Философия Unix (и, по наследству, Linux) говорит о том, что система должна быть хорошо документирована. Поэтому каждая программа распространяется с соответствующей man-страницей (мануалом) — кратким руководством по использованию. Стиль написания этих руководств очень краткий и достаточно формализован.

И поскольку каждая программа должна иметь краткое руководство, а программы распространяются в пакетах (.rpm, .deb и др.) то, очевидно, что ман’ы в системе появляются при установке программ. То есть, в составе пакета программы, помимо самой программы, её конфигурационных файлов и всего-другого-прочего, должна также присутствовать man-страница, которая помещается при установке пакета в каталог /usr/share/man.

15. Системный вызов

Системные вызовы (system calls) – это метод использования интерфейса операционной системы, предоставляемый ею всем остальным программам, выполняющимся на ЭВМ. Этот интерфейс представляет собой набор функций ОС (ОС — это же программа!). Соответственно, вызов ка-

кой-либо функции ОС из этого набора — есть системный вызов. Системные вызовы создают, удаляют и используют различные объекты, главные из которых – процессы и файлы. Пользовательская программа запрашивает сервис у операционной системы, осуществляя системный вызов. То есть, программа запрашивает обслуживание у ОС, ~~«делает заказ» на доставку пиццы.~~

Как это делается: имеются библиотеки процедур, которые загружают *регистры процессора* определенными параметрами и осуществляют *прерывание процессора*, после чего управление передается обработчику данного *вызова*, входящему в ядро *операционной системы*. Цель таких библиотек – сделать *системный вызов* похожим на обычный *вызов функции*. Обработчик системного вызова проверяет правильность номера системного вызова (разрешён ли?), из регистров процессора берёт параметры системного вызова и формирует вызов функции ОС.

При этом программа переходит в привилегированный режим или режим ядра (*kernel mode*). Поэтому *системные вызовы* иногда еще называют программными *прерываниями*, в отличие от аппаратных *прерываний*, которые чаще называют просто *прерываниями*. Программное *прерывание* – это синхронное событие, которое может быть повторено при выполнении одного и того же программного кода.

В режиме ядра работает код ядра *операционной системы (вызванная функция ОС)*, причем исполняется он в адресном пространстве и в контексте вызвавшей его программы. Таким образом, ядро *операционной системы* имеет полный доступ к памяти пользовательской программы, и при *системном вызове* достаточно передать (в регистрах процессора)

— адреса одной или нескольких областей памяти с параметрами *вызова,*

— адреса одной или нескольких областей памяти для получения результатов *вызова*.

В большинстве *операционных систем системный вызов* осуществляется командой программного *прерывания* (INT). Однако в новых версиях ОС используются более «хитрые» способы создания программных прерываний.

Далее приведены примеры системных вызовов.

Системный вызов OPEN

int open (const char *pathname, int flags, mode_t mode);

Системный вызов OPEN принимает три аргумента:

***pathname** — путь к файлу;

flags — флаги, которые указывают способ открытия файла. Основные флаги (все флаги можно найти в файле ***include/asm-generic/fcntl.h***):

0 (O_RDONLY) — открыть файл только для чтения;

1 (O_WRONLY) — открыть файл только для записи;

2 (O_RDWR) — открыть файл для чтения и записи;

mode — права доступа к файлу, учитываются, если задан флаг O_CREAT. Права доступа задаются для владельца, группы и остальных:

0700 — Владелец файла имеет право на чтение, запись и выполнение;

0400 — Владелец файла имеет право на чтение;

0200 — Владелец файла имеет право на запись;

0100 — Владелец файла имеет право на выполнение;

0070 — Группа файла имеет право на чтение, запись и выполнение;

0040 — Группа файла имеет право на чтение;

0020 — Группа файла имеет право на запись;

0010 — Группа файла имеет право на выполнение;

0007 — Остальные пользователи имеют право на чтение, запись и выполнение;

0004 — Остальные пользователи имеют право на чтение;

0002 — Остальные пользователи имеют право на запись;

0001 — Остальные пользователи имеют право на выполнение.

В **%eax** возвращается файловый дескриптор (целое неотрицательное значение) в случае успеха или отрицательное значение, если произошла ошибка.

```
...
movl $5, %eax          #номер системного вызова (open) в %eax
movl $file_name, %ebx   #путь к файлу (адрес) в %ebx
movl flags, %ecx       #флаги в %ecx
movl mode, %edx        #права доступа к файлу в %edx
int $0x80
...
```

Системный вызов CREAT

int creat(const char *pathname, mode_t mode);

Системный вызов CREAT принимает два аргумента: путь к создаваемому файлу (***pathname**) и права на файл (**mode**). При успешном выполнении возвращает дескриптор созданного файла в регистр **%eax**.

Функция CREAT в действительности представляет из себя обертку для вызова OPEN:

```
fs/open.c:
SYSCALL_DEFINE2(creat, const char __user *, pathname, int, mode)
{
  return sys_open(pathname, O_CREAT | O_WRONLY | O_TRUNC, mode);
}
```

Пример создания файла:

```
1  #creat.s — example of creating a file
2  .section .data
3  fname:
4          .asciz "test.txt"
5  mode:
6          .int 0666
7  .section .text
8  .globl _start
9  _start:
10         movl $8, %eax          #номер системного вызова (creat)
11 в %eax
12         movl $fname, %ebx      #путь к файлу в %ebx
13         movl mode, %ecx        #права доступа в %ecx
14         int $0x80
15         movl $1, %eax
16         movl $0, %ebx
17         int $0x80
```

Системный вызов WRITE

ssize_t write(int fd, const void *buf, size_t count);

Системный вызов WRITE принимает три параметра:
fd — файловый дескриптор:
0 — STDIN (стандартный ввод);
1 — STDOUT (стандартный вывод);
2 — STDERR (стандартный вывод ошибок);
***buf** — указатель на буфер (данные подлежащие выводу);
count — количество символов подлежащих выводу.

И возвращает количество выведенных символов в регистр **%eax**.

Пример использования:

```
1  #write.s — syscall write
2  .section .data
3  output:
4          .ascii "Hello world\n"
5  end:
6          .equ length, end — output
7  .section .text
8  .globl _start
9  _start:
10         movl $4, %eax          #номер системного вызова (write) в %eax
11         movl $1, %ebx          #файловый дескритор (stdout) в %ebx
12         movl $output, %ecx     #указатель на строку в %ecx
13         movl $length, %edx     #количество выводимых символов в %edx
14         int $0x80
15         movl $1, %eax
16         movl $0, %ebx
17         int $0x80
```

Системный вызов READ

ssize_t read(int fd, void *buf, size_t count);

Системный вызов READ аналогично WRITE принимает три аргумента, которые описаны выше. Количество прочитанных символов возвращается в **%eax**. Пример использования:

```
1  #read.s — syscall read
2  .section .bss
3          .lcomm buffer, 1024
4  .section .text
5  .globl _start
6  _start:
7          movl $3, %eax          #номер системного вызова (read) в %eax
8          movl $0, %ebx          #файловый дескритор (stdin) в %ebx
9          movl $buffer, %ecx     #указатель на буфер в %ecx
10         movl $1024, %edx       #размер буфера в %edx
11         int $0x80
12         movl $1, %eax
13         movl $0, %ebx
14         int $0x80
```

Системный вызов LSEEK

off_t lseek(int fd, off_t offset, int whence);

Системный вызов LSEEK предназначен для чтения/записи в произвольное место файла. Он принимает следующие аргументы:

fd — файловый дескриптор;

offset — количество байт, на которые нужно сместиться относительно whence;

whence — устаналивает позицию в файле, принимает следующие значения (определены в **include/linux/fs.h**):

1 (SEEK_SET) — начало файла;

2 (SEEK_CUR) — текущая позиция в файле;

3 (SEEK_END, SEEK_MAX) — конец файла.

Возвращает новую позицию в файле относительно его начала в **%eax**, либо номер ошибки в случае неудачи.

```
...
movl $0x13, %eax
movl fd, %ebx
movl offset, %ecx
movl whence, %edx
int $0x80
...
```

Системный вызов CLOSE

int close(int fd);

Системный вызов CLOSE принимает только один аргумент — файловый дескриптор (**fd**) и возвращает в **%eax** 0 в случае успеха или номер ошибки в случае неудачи.

```
...
movl $6, %eax   #номер системного вызова (close) в %eax
movl fd, %ebx   #файловый дескриптор в %ebx
int $0x80
...
```

Пример вывода текста из файла с использованием системных вызовов OPEN, WRITE, READ и CLOSE:

```
1   .section .bss
2       .lcomm buffer, 1024
3   .section .data
```

```
4   fname:
5           .asciz "test.txt"
6   fd:
7           .int 0
8   .section .text
9   .globl _start
10  _start:
11          movl $5, %eax           #системный вызов open
12          movl $fname, %ebx       #имя файла
13          movl $0, %ecx           #открыть только для чтения (0 —
14  O_RDONLY)
15          movl $0, %edx           #0, т.к. мы не будем создавать файл
16          int $0x80
17          movl %eax, fd           #сохраняем файловый дескриптор
18  read:
19          movl $3, %eax           #системный вызов read
20          movl fd, %ebx           #дескриптор открытого файла
21          movl $buffer, %ecx      #куда сохранять данные
22          movl $1024, %edx        #размер буфера
23          int $0x80
24          cmpl $0, %eax           #остались ли данные для чтения?
25          je close                #если нет, то перейти к закрытию файла
26          movl %eax, %edx         #размер буфера
27          movl $4, %eax           #системный вызов read
28          movl $1, %ebx           #куда осуществлять запись (1 — stdout)
29          movl $buffer, %ecx      #откуда брать данные для записи
30          int $0x80
31          jmp read
32  close:
33          movl $6, %eax           #системный вызов close
34          movl fd, %ebx           #дескриптор файла, который следует закрыть
35          int $0x80
36          movl $1, %eax           #системный вызов exit
37          movl $0, %ebx           #возвращаем 0
38          int $0x80
```

Системный вызов UNLINK

int unlink(const char *pathname);

Системный вызов UNLINK принимает всего лишь один аргумент — путь к удаляемому файлу. В **%eax** возвращается 0 в случае успеха или номер ошибки в случае неудачи.

```
...
movl $0xa, %eax
movl $fname, %ebx
int $0x80
...
```

Системные вызовы TRUNCATE и FTRUNCATE

int truncate(const char *path, off_t length);

int ftruncate(int fd, off_t length);

Системный вызов TRUNCATE принимает два аргумента: путь к файлу (***path**) и новый размер файла (**length**), и возвращает в **%eax** 0 в случае успеха или номер ошибки в случае неудачи.

Создадим файл «**test.txt**» и напишем в нем «**Some text before truncate**». Его размер составляет 26 байт. Теперь выполним следующую программу:

```
1  #truncate.s — syscall truncate
2  .section .data
3  fname:
4          .asciz "test.txt"
5  .section .text
6  .globl _start
7  _start:
8          movl $0x5c, %eax        #номер системного вызова (truncate) в %eax
9          movl $fname, %ebx       #путь к файлу в %ebx
10         movl $9, %ecx           #размер файла в %ecx
11         int $0x80
12         movl $1, %eax
13         movl $0, %ebx
14         int $0x80
```

Программа урезает файл до 9 байт и после ее исполнения в файле останется надпись «Some text». Если указать размер файла больше текущего размера, то в файле образуются дырки. Системный вызов FTRUNCATE

полностью аналогичен TRUNCATE, за исключением того, что вместо пути указывается файловый дескриптор, FTRUNCATE имеет номер 0x5d.

Системный вызов UTIME

int utime(const char *filename, const struct utimbuf *times);

Системный вызов UTIME принимает два параметра: имя файла, которому следует изменить время доступа и модификации, и указатель на структуру **utimebuf**. Структура **utimebuf** включает в себя два поля:

```
1    include/linux/utime.h:
2    struct utimebuf
3    {
4        __kernel_time_t actime;  /* время доступа */
5        __kernel_time_t modtime; /*время модификации */
     };
```

Время доступа и модификации указывается в секундах от 1 января 1970 года. Рассмотрим небольшой пример, создадим файл «**test.txt**», чтобы просмотреть его время доступа и модификации можно воспользоваться командой **ls** (для отображения времени доступа используется параметр **--time=atime**). Изменим эти значения, время доступа выставим равным 1 января 1970 года, а время модификации 1 января 2011 года:

```
 1  #utime.s — syscall utime
 2  .section .data
 3  fname:
 4      .asciz "test.txt"
 5  times:
 6  actime:
 7      .long 0             #0 означает сброс времени на 1 января 1970 года
 8  modtime:
 9      .long 1293830000    #время модификации 1 января 2011 года
10  .section .text
11  .globl _start
12  _start:
13      movl $0x1e, %eax    #номер системного вызова (utime) в %eax
14      movl $fname, %ebx   #путь к файлу в %ebx
15      movl $times, %ecx   #адрес «структуры» в %ecx
16      int $0x80
```

```
17      movl $1, %eax
18      movl $0, %ebx
19      int $0x80
```

После запуска программы можете проверить время доступа и модификации, они должны были измениться. Для того чтобы установить текущее время доступа и модификации необходимо передать 0 (NULL) вместо адреса структуры (**movl $0, %ecx**).

Замечания. В системных вызовах в случае ошибки возвращается отрицательное значение в регистр **%eax**. Например, в системном вызове LSEEK, если задан файловый дескриптор не открытого файла, то в **%eax** будет значение 0xfffffff7, что соответствует -9 (коды ошибок можно посмотреть в файле **include/asm-generic/errno-base.h**), а не 4294967287. Поэтому для обработки ошибок следует использовать условные конструкции для знаковых чисел.

Системные вызовы можно просмотреть используя трассировщик **strace**.

Таким образом, можно проверить себя на правильность заданных аргументов. Если системный вызов возвращает ошибку, то strace отобразит ее в удобочитаемом виде.

Общая концепция работы с системными вызовами, полагаем, должна быть ясна, в **%eax** всегда помещается номер системного вызова, в **%ebx**, **%ecx**, **%edx**, **%esi**, **%edi** с первого по пятый — аргументы соответственно.

Вместо шлюза **$0x80** можно использовать библиотеку языка С и инструкцию **CALL**. У такого подхода есть свои плюсы и недостатки. Главный плюс состоит в том, что будут использоваться разделяемые библиотеки, а минус, как следствие, в увеличении размера исполняемого файла.

Ну и наконец, справку по системным вызовам можно получить на страницах справочного руководства **man** во втором и третьем разделах.

16. Библиотека libc — что это?

Libc — «стандартная библиотека С» — библиотека стандартных функций, которые могут использоваться всеми программами, написанными на С (и, иногда, программами, написанными на других языках).

"Сама по себе библиотека не является частью языка, однако, заложенный в ней набор функций, а также определений типов и макросов составляет системную среду, поддерживающую стандарт Си." ((С)Б. Керниган, Д. Ритчи "Язык программирования С").

В linux библиотека libc определена как **glibc** — GNU C Library (GNU библиотека). Glibc является библиотекой Си, которая обеспечивает системные вызовы и основные функции, такие как open, malloc, printf и т. д. Библиотека C используется для всех динамически скомпонованных программ. Она написана Free Software Foundation для операционных систем GNU. glibc выпущена под лицензией GNU LGPL.

Как правило, функции этой библиотеки описаны в man 3.

17. Библиотека gtk — что это?

Библиотека GTK (ныне используется её более новая версия GTK+) была разработана для использования программой GIMP, отсюда и название —The GIMP Toolkit. Со временем библиотека стала использоваться для разработки других приложений для среды GNOME. Сейчас GTK — это объектно-ориентированный и кросс-платформенный инструмент для создания графического интерфейса приложений, причем созданные с использованием GTK приложения независимы от среды GNOME. Ваше приложение, написанное на GTK, будет отлично работать в KDE или любой другой среде — главное, чтобы библиотека GTK была установлена на компьютере.

Библиотека GTK+ базируется на трёх положениях:

1) библиотека Glib — предназначена для оперирования с различными типами данных. Данная библиотека будет полезной для разработки любых приложений, даже не GTK. Эта же библиотека содержит функции для работы с памятью и для обработки ошибок;

2) библиотека GDK (The GIMP Drawing Kit) — библиотека низкого уровня, ее функции взаимодействуют с функциями оконной системы (X Window); используется для построения графических примитивов;

3) библиотека GTK — используется для создания графического интерфейса — здесь реализованы функции верхнего уровня.

18. Библиотека xlib — что это?

Xlib — X **library**, **это библиотека** нижнего уровня, реализующая интерфейс X протокола к языку C и является основой для всех программ для X Window.

Xlib содержит функции для взаимодействия программ с X-сервером. Библиотека позволяет использовать более высокий уровень абстракции, без знания деталей работы основного протокола системы X Window — протокола X. Появилась в 1985 году.

Некоторые пользовательские приложения используют Xlib напрямую (например, Opera), другие используют специальные инструментарии «виджетов» — «библиотеки-надстройки» над базовой библиотекой Xlib.

Структура графической подсистемы linux приведена на рисунке 14.

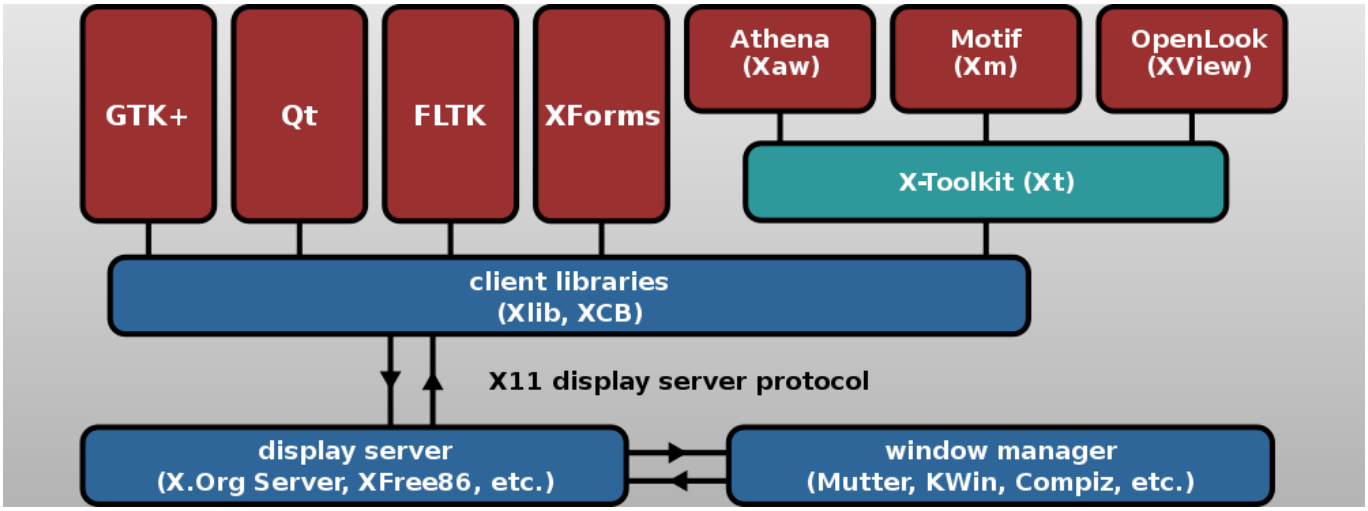

Рис. 14. Структура графической подсистемы linux

ОС И ЗАГРУЗКА

19. Операционная система — определение

Операционная система — это программа, которая выполняется на вычислительной системе с момента включения (вычислительной системы) и до её выключения и которая выполняет следующие основные функции:

1) предоставляет процессам «абстрактную» вычислительную машину,

2) управляет ресурсами (доступом к ресурсам),

3) организует многозадачную многопользовательскую среду и разграничивает процессы и пользователей друг от друга,

4) а также ведёт учёт процессов, ресурсов и пользователей.

20. Загрузка операционной системы

20.1. Если на компьютере «старый» BIOS (стандартные Aword или AMI), винчестер разбит в формате PC BIOS, 32— и 64-разрядные ОС, тогда ОС загружается следующим образом.

Нажатие кнопки питания. По броску питания инициализируются процессоры CPU в контроллерах устройств. Выполняются тесты самотестирования. На CPU начинает работать BIOS, который, просматривая статусные вектора устройств, собирает информацию о состоянии устройств.

Если где-то ошибка, то компьютер пищит n-ое количество раз (видео ещё не работает и по другому BIOS ничего сообщить не может).

Если ошибок нет, тогда BIOS смотрит таблицы CMOS: с какого устройства будем грузиться. Грузит с этого устройства 1-ый сектор и проверяет: последние два байта равны AA55? Если не равны, то выдаётся сообщение: «Не системный диск. Вставьте системный диск» и переходит в режим ожидания нажатия Enter. Если последние два байта равны AA55, то BIOS считает, что это сектор MBR и передаёт управление на первую команду MBR.

Алгоритм MBR: просматривает 1-ый байт главной таблицы разделов, находит активный раздел на устройстве; активный раздел помечен 0x80; из первого сектора активного раздела (из загрузочной области раздела) загружает вторичный загрузчик и передает ему управление.

***)** Вторичный загрузчик загружает ядро ОС, распаковывает его и передаёт ему управление. Ядро выполняет некоторые подготовительные операции и начинает обрабатывать свой главный конфигурационный файл rc (скрипт rc), в котором описано, как ядро должно искать и инициализиро-

вать аппаратное обеспечение компьютера. После завершения этой большой работы ядро определяет уровень, на котором оно будет работать — defaultlevel (файл /etc/inittab) и переходит к обработке соответствующего уровневого каталога, в котором описано, как подготовить операционную среду для пользователя на этом уровне. По завершении обработки уровневого каталога выдаётся приглашение ввести login и пароль. После ввода пользователем требуемого, обрабатываются конфигурационные файлы профиля пользователя из домашнего каталога пользователя. На экране появляется фоновая картинка, панель, меню, иконки. Система переходит в режим ожидания.

20.2. Если на компе новый UEFI и винчестер разбит в формате GPT. Этот алгоритм загрузки работает в основном для 64-разрядных ОС. Тогда ОС грузится так.

Нажатие кнопки питания. По броску питания инициализируются процессоры CPU и в контроллерах устройств. Выполняются тесты самотестирования. На CPU начинает работать модуль UEFI, который, просматривая статусные вектора устройств, собирает информацию о состоянии устройств.

Если где-то ошибка, то компьютер пищит n-ое количество раз (видео ещё не работает и по другому UEFI ничего сообщить не может).

Если ошибок нет, тогда UEFI смотрит таблицы CMOS: с какого устройства будем грузиться. На этом устройстве должен существовать раздел типа ef (EFI (FAT-12/16/32)) по порядку первый или второй размером 200-500 Мб с файловой системой FAT. На этом разделе должен быть каталог EFI, в котором должен находиться каталог BOOT, в котором должен лежать загрузчик bootx64.efi. Этот раздел и его содержимое создаётся автоматически инсталлятором ОС при установке ОС: Windows — при обнаружении на компе UEFI и диске, разбитом в формате GPT; Linux — при обнаружении на компе UEFI, диске, разбитом в формате GPT и выборе при установке опции «Подготовить винчестер автоматически». В иных случаях этот раздел нужно создавать вручную.

UEFI с выбранного устройства **из раздела типа ef с файловой системой FAT** грузит файл EFI/BOOT/bootx64.efi и передаёт ему управление. Загрузчик bootx64.efi обрабатывает свой конфигурационный файл (если он есть). Затем, если на диске установлена одна ОС, то загрузчик сразу же её грузит — ядро этой ОС может лежать здесь же в каталоге EFI/BOOT; иначе, если установлено несколько ОС, то загрузчик выдаёт на экран меню для выбора пользователем нужной ОС, после чего грузит вторичный за-

грузчик указанной ОС из загрузочной области раздела, в котором эта ОС установлена, а вторичный загрузчик загружает ядро этой ОС.

Дальнейшие действия такие же, как в пункте 20.1, абзац, помеченный *).

Если у Вас 32-разрядное «железо» и, соответственно, Вы можете грузить только 32-разрядную ОС, то загрузчик должен называться bootia32.efi.

Определённо можно отметить, что вариант загрузки ОС по пункту 20.2 является менее надёжным и способен «окирпичить» Ваш компьютер. В очередной раз разработчики «железа» и BIOS`о-писатели (о, не так: UEFI-писатели) сделали пользователям гадость. В данном случае постаралась intel.

Примечание 1. В качестве загрузчика bootx64.efi может быть использован любой загрузчик, знакомый с UEFI, как то: grub, systemd-boot, refind, lilo, wind`овый из windows-10 и прочая. Файл загрузчика должен быть переименован в bootx64.efi и положен в указанное место.

Примечание 2. UEFI (Unified Extensible Firmware Interface, Единый расширяемый интерфейс прошивки) разрабатывался компанией Intel как замена BIOS (Basic Input Output System).

21. Конфигурационные (загрузочные) скрипты ОС Linux

Конфигурационные (загрузочные) скрипты ОС — это скрипты, расположенные (как правило) в каталоге /*etc/*rc.d/. Первым обычно исполняется rc.sysinit, затем rc и последним rc.local. В этих скриптах находится алгоритм поиска оборудования (на каком «железе» система запущена, инициализация этого «железа», поиск и подключение нужных драйверов для этого «железа»), подключение swap-раздела, монтирование и проверка локальных файловых систем. В скрипте rc определяется уровень работы системы и обрабатываются скрипты из уровневых каталогов, обеспечивающие создание и настройку операционной среды для пользователя. В скрипте rc.local завершается конфигурирование операционной среды пользователя.

22. Понятие «уровневый каталог»

В каталоге /etc/rc.d/, помимо основных конфигурационных скриптов rc.*, также находятся каталог init.d, содержащий скрипты запуска различных сервисов, и каталоги rcN.d (N — номера от 1 до 6), в которых находятся символьные ссылки в определенном формате на скрипты в каталоге init.d. По этим ссылкам осуществляется запуск необходимых сервисов для каждого уровня выполнения системы (runlevel). Тем самым формируется

операционная среда пользователя на определённом уровне выполнения системы.

Каталоги rcN.d — уровневые каталоги.

23. Что находится в «уровневом каталоге»?

В уровневых каталогах находятся символьные ссылки в определенном формате на скрипты в /etc/rc.d/init.d — скрипты убиения/запуска сервисов.

Формат ссылок (например, на убиение/запуск сервиса ssh):

— K26sshd — убиение сервиса ssh;

— S54sshd — запуск сервиса ssh.

Пример скрипта запуска сервиса fetchmail, обеспечивающего получение почты из внешнего почтового ящика:

```
#!/bin/sh
#
# chkconfig: 345 80 20
# description: run fetchmail service
#

if [ ! -x /usr/bin/fetchmail ]; then
   exit 1
fi

if [ -x /etc/rc.d/init.d/functions ]; then
  . /etc/rc.d/init.d/functions
fi

RETVAL=0

start () {
   echo "Starting fetchmail"
   daemon /usr/bin/fetchmail
   RETVAL=$?
   [ $RETVAL -eq 0 ] && touch /var/lock/subsys/fetchmail
   echo
   return $RETVAL
}

stop () {
   echo -n "Stopping $prog: "
```

```
    killproc /usr/bin/fetchmail
    RETVAL=$?
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/fetchmail
    echo
    return $RETVAL
}

case $1 in
  start)
    start
    ;;
  stop)
    stop
    ;;
  restart|reload)
    stop
    start
    ;;
  condrestart)
    if [ -f /var/lock/subsys/fetchmail ]; then
      stop
      start
    fi
    ;;
  status)
    status /usr/bin/fetchmail
    ;;
  *)
    echo "Usage: $0 {start|stop|restart|reload|condrestart|status}"
    ;;
esac

RETVAL=$?

exit $RETVAL
```

Первые пять строчек скрипта — заголовок. Он состоит из:

— хешбэнга (в примере **#!/bin/sh**), указывающего, какой интерпретатор использовать,

— служебной информации для утилиты chkconfig, содержащую три числа: первое — список уровней запуска, на которых должен выполняться

скрипт, второе — порядковый номер запуска, и третье — порядковый номер останова (убиения) сервиса; второе и третье числа в сумме должны составлять 100;

— комментария.

Далее следуют проверки существования программ, необходимых для сервиса, и файла с набором функций.

Затем идут функции запуска, останова, рестарта, запроса статуса сервиса и другие.

Для работы со скриптами существуют две утилиты service и chkconfig:

— service просто запускает указанный в аргументах скрипт (с указанными параметрами);

— chkconfig — утилита для управления скриптами загрузки. Она содержит свою базу данных скриптов, автоматически включает и выключает скрипты для определенных уровней запуска, позволяет управлять сервисами демона xinetd.

24. Что такое «стартовый скрипт запуска сервиса»?

См. ответ на вопрос 23.

25. Что содержится в каталоге /etc/rc.d/init.d?

Стартовые скрипты запуска сервисов.

См. ответы на вопросы 22 и 23.

26. Последовательность загрузки ОС.

См. ответ на вопрос 20.

ПРОЦЕССЫ

27. Процесс в ОС — определение и состав

Процесс — объект, создаваемый в ОС при запуске какой-либо программы (подробнее см. в [21]).

Состоит из двух элементов:

— адресного пространства, в которое помещается код, структуры данных, стек;

— PCB (process control blok) — блок управления процессом.

Адресное пространство процесса (виртуальное адресное пространство процесса) всегда имеет фиксированный объём 4 Гб на 32-разрядных системах или 16 Гб на 64-разрядных. В него переписывается информация из процесса-родителя (при создании процесса системным вызовом fork()) или из исполняемого файла (при системном вызове exec).

PCB — этот элемент упрощённо можно представлять как таблицу из нескольких десятков строк, в которой многозадачная ОС описывает процесс. Он необходим **операционной системе** для управления процессами — их же много в многозадачной системе.

См. также ответы на вопросы 30 и 32.

28. Как ОС создаёт процесс?

Почти всегда в linux новый процесс создается уже существующим процессом, который делает в памяти свою точную копию с помощью системного вызова fork(). Созданный (дочерний) процесс получает то же окружение, что и его родительский процесс, отличается только номер ID — PID (process identificator). То есть, fork() делает копию адресного пространства процесса-родителя и копию PCB процесса-родителя. Затем в копии PCB меняется имя процесса PID на новое, свободное в текущий момент, имя. После этого происходит возврат из fork() в процесс-родитель и процесс-потомок.

Исключение: процесс init с ID=1 создаётся по другому — см. ответ на вопрос 20.

Если нужно запустить дополнительно другую программу (другой исполняемый файл — процесс-родитель при этом должен продолжать работать, то есть, в итоге получим два процесса с разными PID), то после возврата из fork() в процессе-потомке делается системный вызов exec, в параметрах которого указывается имя исполняемого файла и, возможно, опции (ключи).

Пример:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
int main(int argc, char * argv[])
{ int pid, status;
if (argc < 2) {
printf("Usage: %s command, [arg1 [arg2]...]\n", argv[0]);
return EXIT_FAILURE;
}
printf("Starting %s...\n", argv[1]);
pid = fork();
if (pid == 0) {
execvp(argv[1], &argv[1]);
perror("execvp");
return EXIT_FAILURE; // Never get there normally
} else {
if (wait(&status) == -1) {
perror("wait");
return EXIT_FAILURE;
}
if (WIFEXITED(status))
printf("Child terminated normally with exit code %i\n",
WEXITSTATUS(status));
if (WIFSIGNALED(status))
printf("Child was terminated by a signal #%i\n", WTERMSIG(status));
if (WCOREDUMP(status))
printf("Child dumped core\n");
if (WIFSTOPPED(status))
printf("Child was stopped by a signal #%i\n", WSTOPSIG(status));
}
return EXIT_SUCCESS;
}
```

Если нужно запустить другую программу так, чтобы «заменить» предыдущую (PID останется тот же), то новая программа (исходный текст находится в файле proga.c) запускается из текущей с помощью системного вызова exec.

Пример:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[], char *envp[]){

(void) execle("/bin/cat", "/bin/cat", "proga.c", 0, envp);

printf("Error on program start\n");
exit(-1);
return 0;
}
```

29. Что такое pid?

PID — process identificator — имя процесса, присваивается ему операционной системой при создании. Тип — unsigned short int, то есть, беззнаковое короткое целое (два байта). Диапазон изменения от 0 до 65535.

То есть, мы, люди, именуем наши программы символическими именами (как файлы; программы — это же файлы!), а когда мы программу запускаем, то система создаёт процесс и присваивает ему цифровое имя — PID, которым и пользуется.

30. Что такое PCB?

PCB — Process Control Block — Блок Управления Процессом — таблица с параметрами процесса, которая хранится в ОС в виде списковой структуры.

Назначение PCB. Создаётся для каждого процесса для того чтобы операционная система могла выполнять операции над процессами. Эта структура содержит информацию, специфическую для данного процесса:

- состояние, в котором находится процесс;
- программный счетчик процесса или, другими словами, адрес команды, которая должна быть выполнена для него следующей;
- содержимое регистров процессора;
- данные, необходимые для планирования использования процессора и управления памятью (приоритет процесса, размер и расположение адресного пространства и т. д.);
- учетные данные (идентификационный номер процесса, какой пользователь инициировал его работу, общее время использования процессора данным процессом и т. д.);

- сведения об устройствах ввода-вывода, связанных с процессом (например, какие устройства закреплены за процессом, таблицу открытых файлов).

Состав и строение PCB зависят, конечно, от конкретной операционной системы. Во многих операционных системах информация, характеризующая процесс, хранится не в одной, а в нескольких связанных структурах данных. Эти структуры могут иметь различные наименования, содержать дополнительную информацию или, наоборот, лишь часть описанной информации. То есть, для любого процесса, находящегося в вычислительной системе, вся информация, необходимая для совершения операций над ним, доступна операционной системе. Блок управления процессом является моделью процесса для операционной системы. Любая операция, производимая операционной системой над процессом, вызывает определенные изменения в PCB .

Информацию, для хранения которой предназначен блок управления процессом, удобно для дальнейшего изложения разделить на две части. Содержимое всех регистров процессора (включая значение программного счетчика) — *регистровый контекст* процесса, а все остальное – *системный контекст* процесса.

Знания регистрового и системного контекстов процесса достаточно для того, чтобы управлять его работой в операционной системе, совершая над ним операции. Однако этого недостаточно для того, чтобы полностью охарактеризовать процесс. Операционную систему не интересует, какими именно вычислениями занимается процесс, т. е. какой код и какие данные находятся в его адресном пространстве. С точки зрения пользователя, наоборот, наибольший интерес представляет содержимое адресного пространства процесса, возможно, наряду с регистровым контекстом определяющее последовательность преобразования данных и полученные результаты. Код и данные, находящиеся в адресном пространстве процесса — *пользовательский контекст*. Совокупность регистрового, системного и пользовательского контекстов процесса для краткости принято называть просто *контекстом* процесса. Таким образом, в любой момент времени процесс полностью характеризуется своим контекстом.

31. Что такое контекст процесса?

См. ответ на вопрос 30.

32. Что такое адресное пространство процесса?

При создании процесса для каждого процесса отводится на 32-разрядных машинах 4 Гб памяти, а на 64-разрядных — 16 Гб памяти. Поскольку в многозадачных ОС процессов десятки/сотни, то отводимое адресное пространство является виртуальным, то есть, условным, несуществующим реально. Распределение этого виртуального пространства в ОС Linux и Windows показано на рисунке 15.

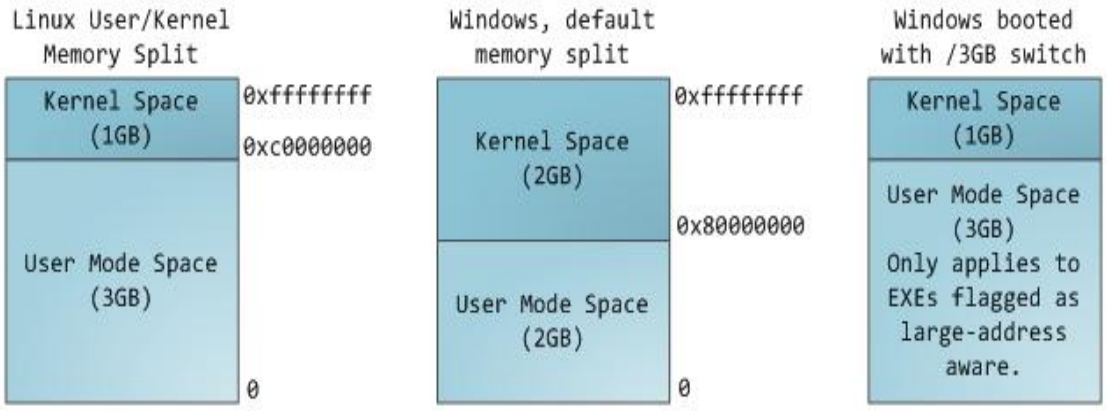

Рис. 15. Распределение виртуального адресного пространства

Конкретно в Linux строение виртуального адресного пространства показано на рисунке 16.

В младших адресах располагается код программы. Затем следует сегмент инициализируемых структур данных, например, так: static char *gonzo = «God's own prototype»;. Затем сегмент неинициализируемых структур данных, например, так: static char *userName;. Затем следует «куча» свободной памяти, из которой программе выдаётся память по запросам alloc для создания динамических переменных. Затем область памяти для отображения файлов в память, например, для библиотек. И почти в самом верху — стек программы. Современные версии Linux, как правило, между сегментами/областями памяти делают смещения случайного размера («дыры», а чтобы хакер не мог так просто вычислить расположение структур программ в памяти).

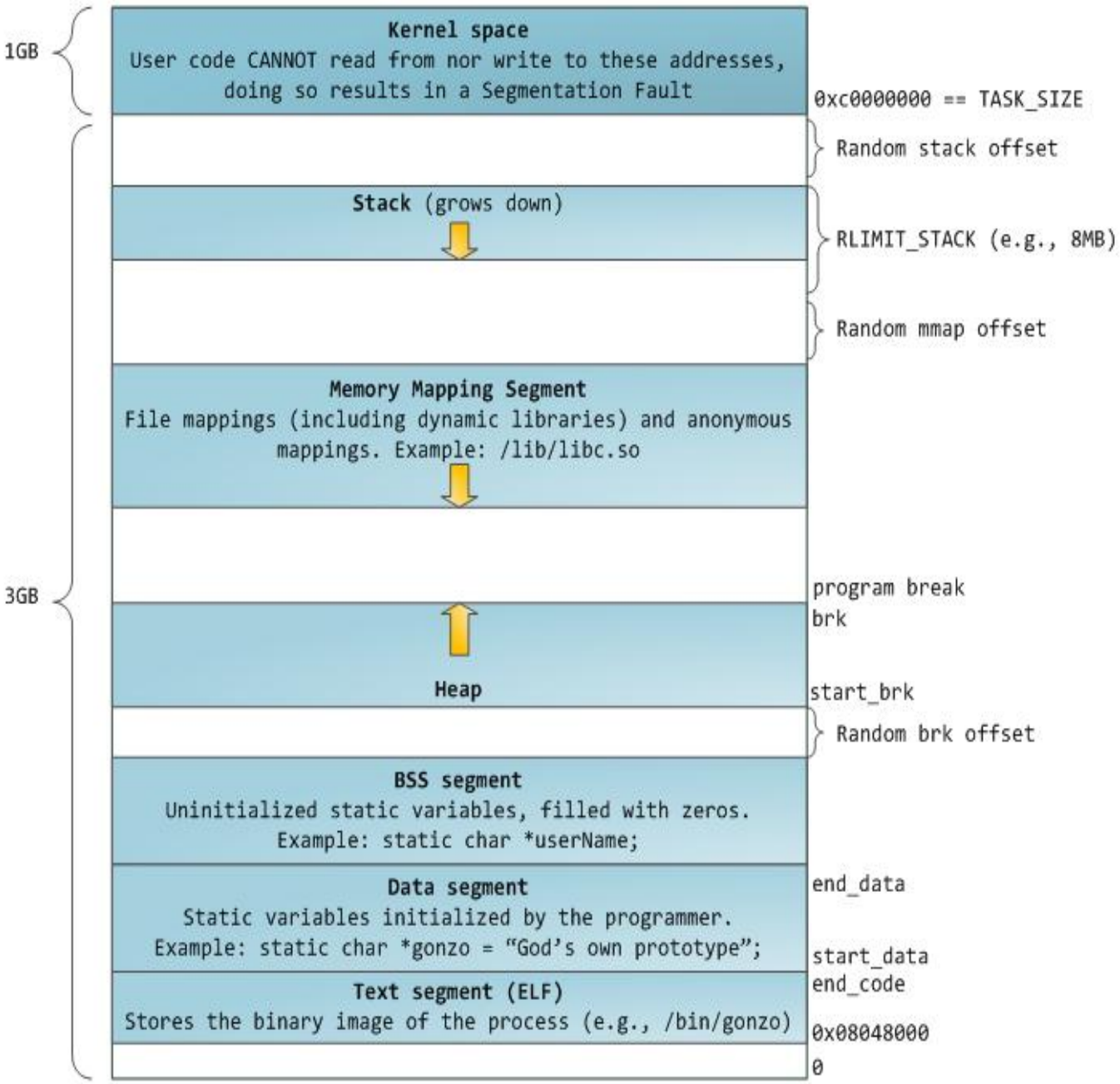

Рис. 16. Адресное пространство процесса. Желтые стрелки показывают направление приращения сегмента

33. Какого объёма адресное пространство процесса?

См. ответ на вопрос 32.

34. Жизненный цикл процесса

Жизненный цикл процесса от рождения (запуска программы) до удаления из системы (завершение программы) показан на рисунке 17.

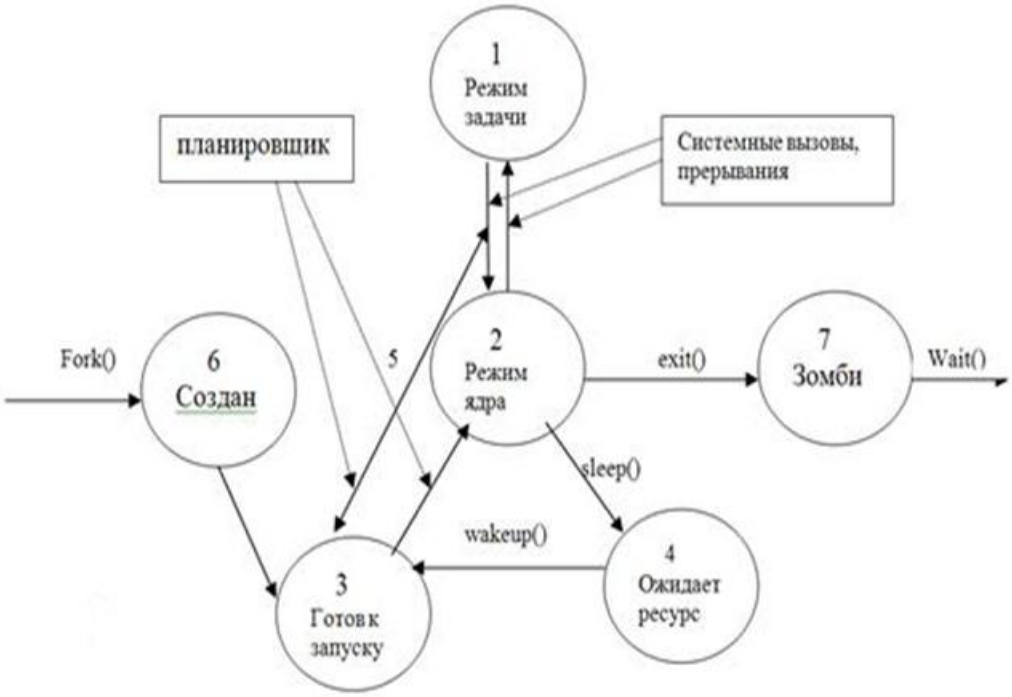

Рис. 17. Жизненный цикл процесса

35. Что такое состояние процесса?

Состояния процесса (в овалах на рисунке 18) и действия/события, приводящие к их изменению, показаны на рис. 18. Совпадает с состояниями жизненного цикла, за исключением промежуточного состояния «выполнение в режиме ядра». В этом состоянии жизненного цикла («выполнение в режиме ядра») процессу уже назначено состояние «running» — «выполнение», но он ещё не выполняется, поскольку ОС занимается подготовительными операциями и проверками.

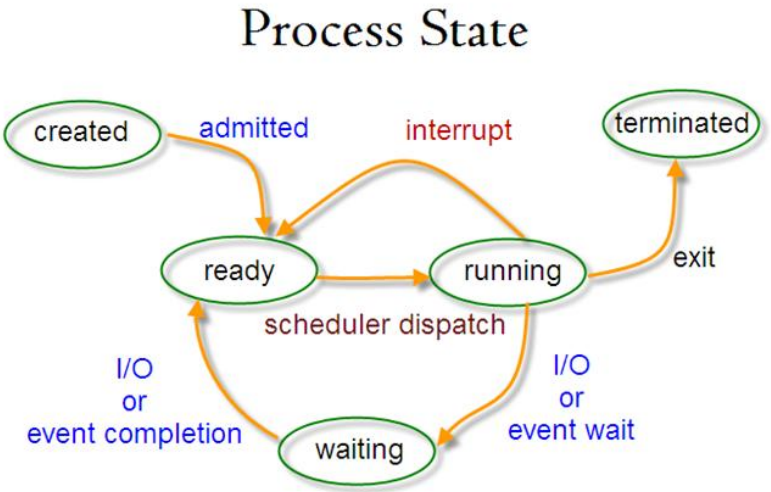

Рис. 18. Состояния процесса

36. Что такое счётчик команд?

Переменная, в которой хранится адрес следующей команды выполняющейся программы.

37. Где хранится счётчик команд?

Счетчик команд хранится:

а) в регистре процессора, когда программа выполняется, то есть, владеет процессором;

б) поле в PCB, в котором сохраняется счётчик команд из регистра процессора в состояниях процесса «ready» и «waiting», то есть, когда программа не владеет процессором.

38. Какого размера (бит/байт/килобайт/мегабайт) счётчик команд?

Размер счётчика команд определяется, как правило, разрядностью процессора. То есть, если процессор 8-разрядный, то счётчик команд 8-разрядный.

Однако, не всегда. Например, на 8-разрядных процессорах счётчик команд нередко 16-разрядный (пример: семейство микроконтроллеров M68HC08 фирмы Motorola.). Аналогично, на некоторых современных 32-разрядных процессорах счётчик команд может быть 64-разрядным.

39. Что происходит со счётчиком команд, когда процесс прерывается?

Счётчик команд запоминается в PCB.

40. Какое отношение имеет счётчик команд к процессу?

Процесс — запущенная на исполнение программа. Её код (её команды, которые сгенерил компилятор при трансляции программы) при выполнении программы находится в сегменте кода адресного пространства процесса. В начальный момент запуска программы счётчик команд содержит адрес фунции .start, с которой начинается выполнение программы. Эту функцию и её параметры компилятор формирует, обрабатывая оператор int main (int argc, char *argv[], char *envp[]). В дальнейшем, по мере выполнения программы, в счётчике команд хранится адрес следующей команды выполняющейся программы.

41. Процесс init: id процесса и назначение этого процесса

В Linux процесс init имеет Process Identificator (PID) =1. В некоторых системах init имеет PID = 0.

init (от англ. *Initialization* — инициализация) — подсистема инициализации в Unix и ряде Unix-подобных систем (например, linux), которая запускает все остальные процессы. Работает как демон.

Процесс init запускается средствами ядра операционной системы. Процесс init исполняет обязанности «приемного» родителя для «осиротевших» процессов.

42. Алгоритм гарантированного планирования с динамическими приоритетами

Алгоритм. Рассмотрим ситуацию, когда процессы в состоянии готовность организованы в 4 очереди. Планирование процессов между очередями осуществляется на основе вытесняющего приоритетного механизма. Чем выше располагается очередь, тем выше ее приоритет. Процессы в очереди 1 не могут исполняться, если в очереди 0 есть хотя бы один процесс. Процессы в очереди 2 не будут выбраны для выполнения, пока есть хоть один процесс в очередях 0 и 1. И, наконец, процесс в очереди 3 может получить процессор в свое распоряжение только тогда, когда очереди 0, 1 и 2 пусты.

Если при работе процесса появляется другой процесс в какой-либо более приоритетной очереди, исполняющийся процесс вытесняется появившимся. Планирование процессов внутри очередей 0-2 осуществляется с использованием алгоритма RR, планирование процессов в очереди 3 основывается на алгоритме FCFS.

Родившийся процесс поступает в очередь 0. При выборе на исполнение он получает в свое распоряжение квант времени размером Q единиц. Если продолжительность его CPU burst меньше этого кванта времени, процесс остается в очереди 0. В противном случае, он переходит в очередь 1. Для процессов из очереди 1 квант времени имеет величину 2Q. Если процесс не укладывается в это время, он переходит в очередь 2. Если укладывается — остается в очереди 1. В очереди 2 величина кванта времени составляет 4Qединицы. Если и этого мало для непрерывной работы процесса, процесс поступает в очередь 3, для которой квантование времени не применяется, и, при отсутствии готовых процессов в других очередях, он может исполняться до окончания своего CPU burst.

Чем больше значение продолжительности CPU burst, тем в менее приоритетную очередь попадает процесс, но тем на большее процессорное время он может рассчитывать для своего выполнения. Таким образом, че-

рез некоторое время все процессы, требующие малого времени работы процессора окажутся размещенными в высокоприоритетных очередях, а все процессы, требующие большого счета и с низкими запросами к времени отклика, — в низкоприоритетных.

Организация перемещения процессов из очередей с низкими приоритетами в очереди с большими приоритетами позволяет более полно учитывать изменение поведения процессов с течением времени,

43. Кооперативный режим планирования процессов

При таком режиме планирования процесс занимает столько процессорного времени, сколько ему необходимо. При этом переключение процессов возникает только при желании самого исполняющегося процесса передать управление, например, при ожидании завершения операции ввода-вывода или по окончании работы.

Этот метод планирования относительно просто реализуем и достаточно эффективен, так как позволяет выделить большую часть процессорного времени для работы самих процессов и до минимума сократить затраты на переключение контекста. Однако при невытесняющем планировании возникает проблема возможности полного захвата процессора одним процессом, который вследствие каких-либо причин (например, из-за ошибки в программе) зацикливается и не может передать управление другому процессу. В такой ситуации спасает только перезагрузка всей вычислительной системы.

«+»:

— высокая эффективность использования вычислительной системы,

— малые накладные расходы на вспомогательные/обслуживающие работы, в том числе, на работу операционной системы.

«-»:

— требует высокой отлаженности и достоверности программного обеспечения,

— система не защищена от зловредного ПО.

44. Вытесняющее планирование процессов

В этом режиме планирования процесс может быть приостановлен в любой момент исполнения.

Всё оперативное время процессора делится на небольшие отрезки времени — кванты. Операционная система устанавливает специальный таймер для генерации сигнала прерывания по истечении кванта. По результатам обработки этого прерывания включается планировщик процес-

сов операционной системы, который организует сохранение контекста выполнявшегося процесса и передачу процессора в распоряжение следующего процесса. Этот следующий процесс выбирается из очереди процессов по определённому алгоритму.

Временные прерывания от таймера помогают гарантировать приемлемое время отклика процессов для пользователей, работающих в диалоговом режиме, и предотвращают "зависание" компьютерной системы из-за зацикливания какой-либо программы.

«+»:

— высокая надёжность функционирования и предсказуемость поведения вычислительной системы,

«-»:

— большие накладные расходы на вспомогательные / обслуживающие работы, например, на работу планировщик ОС.

45. Алгоритм планирования fifo

Это простейший алгоритм планирования процессов, который принято обозначать аббревиатурой FCFS по первым буквам его английского названия – First-Come, First-Served (первым пришел, первым обслужен).

Суть алгоритма. Процессы, находящиеся в состоянии готовность, выстроены в очередь. Когда процесс переходит в состояние готовность, он, а точнее, ссылка на его PCB, помещается в конец этой очереди. Выбор нового процесса для исполнения осуществляется из начала очереди с удалением оттуда ссылки на его PCB. Очередь подобного типа имеет в программировании специальное наименование – FIFO[1], сокращение от First In, First Out (первым вошел, первым вышел). То есть, это обычная очередь, как в магазине.

В простейшем случае, такой алгоритм выбора процесса осуществляет невытесняющее (кооперативное) планирование. Процесс, получивший в свое распоряжение процессор, занимает его до тех пор, пока сам по какой-либо причине не прервётся (например, для осуществления операции ввода/вывода). После этого для выполнения выбирается новый процесс из начала очереди.

«+»:

— лёгкость реализации, алгоритм простой и требует мало ресурсов.

«-»:

— систему легко «подвесить» зациклившимся или «зловредным» процессом,

— непредсказуемость поведения системы.

46. Как при планировании учесть большую/меньшую важность процессов?

Для учета важности процесса ввести приоритет процесса — некоторую числовую переменную, с помощью которой мерить «важность» процесса.

Приоритет может назначаться процессу по некоторым внешним характеристикам процесса — «по политическим соображениям»: например, высокий приоритет может быть присвоен задаче преподавателя или того, кто заплатил $1000 за работу в течение одного часа.

Но может вычисляться и по некоторым внутрисистемным параметрам и оценкам, например, при реализации требования справедливости планирования, или с помощью вычисления оценки работы процесса без прерывания (так называемого, CPU burst) — чем чаще процесс прерывается, тем выше ему назначается приоритет. Или по иным соображениям.

47. Что такое поток?

В Linux для создания дополнительных потоков используются процессы особого типа — обычные дочерние процессы главного процесса, но они разделяют с главным процессом адресное пространство, файловые дескрипторы и обработчики сигналов. Для обозначения процессов этого типа, применяется специальный термин – легкие процессы (lightweight processes, иногда используется термин «клон» — clon). Прилагательное «легкий» в названии процессов-потоков определяется тем, что этим процессам не нужно создавать собственную копию адресного пространства (и других ресурсов) своего процесса-родителя, создание нового легкого процесса (клона) требует значительно меньших затрат, чем создание полновесного дочернего процесса. Поскольку потоки Linux на самом деле представляют собой процессы, в мире Linux не совсем корректно говорить, что один процесс содержит несколько потоков. Наверное, правильней сказать, что такой процесс с потоками является многопроцессным приложением. В общем, толкового термина для этой сущности нет.

В Linux у каждого процесса есть идентификатор — PID. Есть он, естественно, и у процессов-потоков, клонов. Но с другой стороны, спецификация POSIX 1003.1c требует, чтобы все потоки многопоточного приложения имели один идентификатор. Это определяется тем, что для многих функций системы многопоточное приложение должно представляться как один процесс с одним идентификатором.

Эта проблема единого идентификатора решается в Linux следующим образом. Процессы многопоточного приложения группируются в группы

потоков (thread groups). Группе присваивается идентификатор, соответствующий идентификатору первого процесса (основного, главного) многопоточного приложения. Именно этот идентификатор группы потоков используется при «общении» с многопоточным приложением.

Таким образом, поток в linux — lightweight processes (клон), является единицей планирования для планировщика ОС, он реально виден во вне как и обычный процесс.

Совсем иная ситуация в Windows — для неё понятия потока процесса не существует, Windows видит только процессы и только процессы планировщик Windows планирует. А потоки? Не, не слышала.

Так что же тогда это за сущности такие — потоки в Windows? Ведь все о них говорят.

Поток в Windows — это искусственная конструкция в процессе, создаваемая с помощью библиотек. Зачем? А потому что, новый процесс в Windows создаётся тяжело и долго — это затратная операция. И чтобы хоть как-то облегчить/ускорить распараллеливание обработки были придуманы эти искусственные конструкции внутри виндового процесса.

48. Сколько потоков может быть в процессе?

Минимум — один. Это тот самый первый процесс, который создаётся при запуске программы. Он же первый поток.

Максимум — столько, сколько запрограммирует программер. Но возможны ограничения операционной системы — она не безразмерная, это, ведь, тоже программа со своими ограничениями.

49. Как планируется выполнение потоков в linux?

В операционной системе Linux новый процесс создается путем копирования всех атрибутов текущего процесса. Но новый процесс может быть также *клонирован* (cloned); при этом такие ресурсы, как файлы, обработчики сигналов и виртуальная память, используются совместно — не копируются. Если два процесса пользуются одной и той же виртуальной памятью, они функционируют как потоки в рамках одного и того же процесса. То есть, для потоков структуры данных отдельно не задаются. Таким образом, в операционной системе Linux потоки и процессы не различаются. Они являются равнозначными единицами планирования для планировщика ОС, то есть для потоков действуют те же правила планирования, что и для процессов.

ВИНЧЕСТЕР, HARD DISK И HDD

50. Адресация CHS?

CHS (от англ. *Cylinder, Head, Sector* — цилиндр, головка, сектор) — система адресации сектора, как минимальной единицы хранения данных в накопителях на жёстких магнитных дисках, накопителях на гибких магнитных дисках и т.п, основанная на использовании физических адресов геометрии диска.

В этой системе сектор адресуется кортежем из трёх координат: цилиндр-головка-сектор (Cylinder, Head, Sector), именно так, как он физически расположен на диске (см. рисунок 19). При этом цилиндр и головка считаются от нуля, а сектор — от единицы. То есть, первый сектор диска в формате CHS будет иметь адрес (0, 0, 1).

На 4-х пластинах — 8 дорожек, одна точно под другой. Они образуют цилиндр.

Почему счёт цилиндров и головок идёт с нуля, а счёт секторов — с единицы? А потому что сектор 0 — метка начала дорожки.

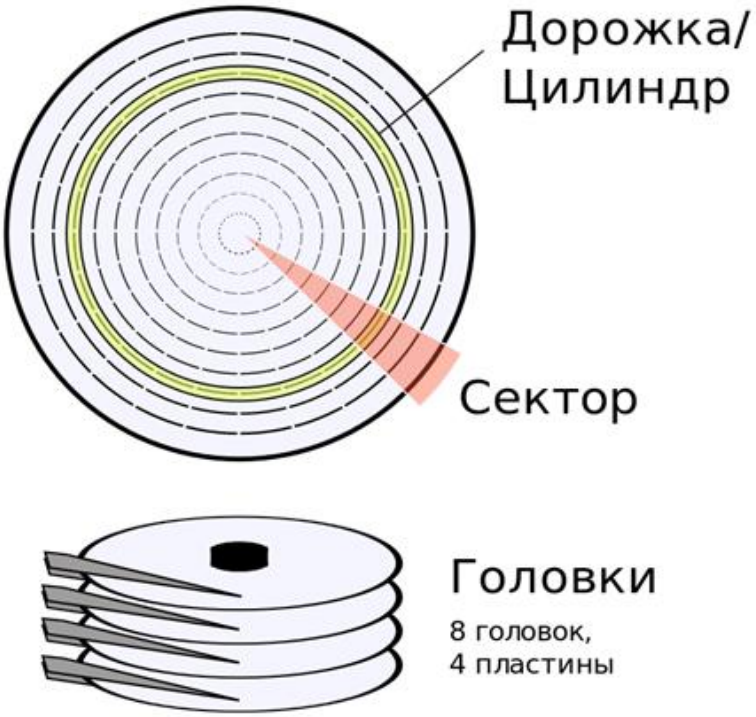

Рис. 19. Адресация CHS

51. Что такое LBA?

LBA (англ. *Logical block addressing*) — механизм адресации и доступа к блоку данных (сектору) на жёстком или оптическом диске, при котором системному контроллеру (который на системной плате) нет необходимости учитывать геометрию самого жесткого диска (количество цилиндров, сторон (головок), секторов на дорожке). Контроллеры современных IDE дисков в качестве основного режима трансляции адреса на внешнем интерфейсе (который смотрит на системную плату, на системный контроллер IDE) используют LBA, а уже внутри себя (на внутреннем интерфейсе при работе с головками и пластинами) адреса LBA пересчитываются в адреса CHS.

Привод, способный поддерживать режим LBA, сообщает об этом в информации идентификации привода.

Суть LBA состоит в том, что каждый блок, адресуемый на жёстком диске, имеет свой номер (порядковый), целое число, начиная с нуля и т. д. (то есть первый блок (сектор) это LBA=0, второй блок LBA=1, ...)

LBA 0 = Цилиндр 0/Головка 0/Сектор 1

Еще одно преимущество метода адресования LBA — то, что ограничение размера диска обусловлено лишь разрядностью LBA. В настоящее время для задания номера блока используется 48 бит, что при использовании двоичной системы исчисления даёт возможность адресовать на приводе (2^{48}) 281 474 976 710 656 блоков (то есть, при блоке в 512 байт, 128 ПиБ).

52. Почему появилось LBA?

Из-за ограниченности программеров BIOS-ов: ну, никак они не ожидали, что прогресс в области винтостроения будет таким быстрым, что за какие-то 40 лет (возможно, ещё живы те, которые писали первые версии BIOS-ов) объём вничестеров с 10-20 Мегабайт (Мега! именно Мега) вырастет до 10-20 Терабайт, то есть, в 1 000 000 (миллион!) раз. В те структуры данных для адресации CHS, которые закладывали BIOS-описатели, новые «размеры» просто не влезают. Пример: число, определяющее количество секторов на новых терабайтных винчестерах, уже лет десять как, не влезает в обычный калькулятор — требуется специальный 64-разрядный, но скоро и в него не будет помещаться.

И такая ситуация «не влезания» с 1990 года по 2002 возникала аж 9 раз! В конце концов у народонаселения терпение лопнуло и адресацию CHS заменили на LBA, то есть, вместо геометрического адреса блока (сектора) начали использовать просто порядковый номер сектора в качестве

его адреса. А пересчёт порядковых номеров секторов в их геометрические адреса на пластинах поручили контроллеру винчестера.

Подробнее — в лекциях,
или здесь: https://www.kv.by/kv_archive/2002/30,
или здесь: https://rom.by/articles/big_HDD/index.htm.

53. Как определить адрес цилиндра-дорожки-сектора, если hdd использует адресацию LBA?

Никак.

Адрес LBA передаётся в винчестер:

CPU → контроллер ATA (PATA или SATA — тот, что на системной плате) → кабель интерфейсный → контроллер винчестера (плата, что снизу на винчестере, обычно зелёного цвета).

Контроллер винчестера пересчитывает адрес LBA по одному ему известному алгоритму в адрес CHS и даёт команду на установку головок. Алгоритм расчёта является оригинальным для каждого экземпляра винчестера, потому что, в алгоритме учитывается HardBadList, а он строго индивидуален. Кроме того, алгоритм зависит от марки винчестера, поколения и других причин. И, естественно, у разных фирм алгоритмы разные.

Контроллер винчестера не сообщает в компьютер вычисленный адрес.

54. Где находится MBR?

MBR находится в первом секторе винчестера.

Конкретно: LBA 0 = Цилиндр 0/Головка 0/Сектор 1.

55. Что содержится в MBR?

MBR — Master Boot Record — Главная Загрузочная Запись, размер = 1 сектор (512 байт).

Структура MBR:

Смещение	Длина, байт	Описание
0000h	446	Код первичного загрузчика
01BEh	16	Раздел 1
01CEh	16	Раздел 2
01DEh	16	Раздел 3
01EEh	16	Раздел 4
01FEh	2	Сигнатура (55h AAh)

Главная таблица разделов

56. Где находится первичный загрузчик?

Первичный загрузчик находится в секторе MBR — первые 446 байт. См. ответ на вопрос 55.

57. Где находится вторичный загрузчик?

Вторичный загрузчик находится в загрузочной области активного раздела (см. лекции).

На рисунке 20 показано начало винчестера в формате разбиения PC BIOS.

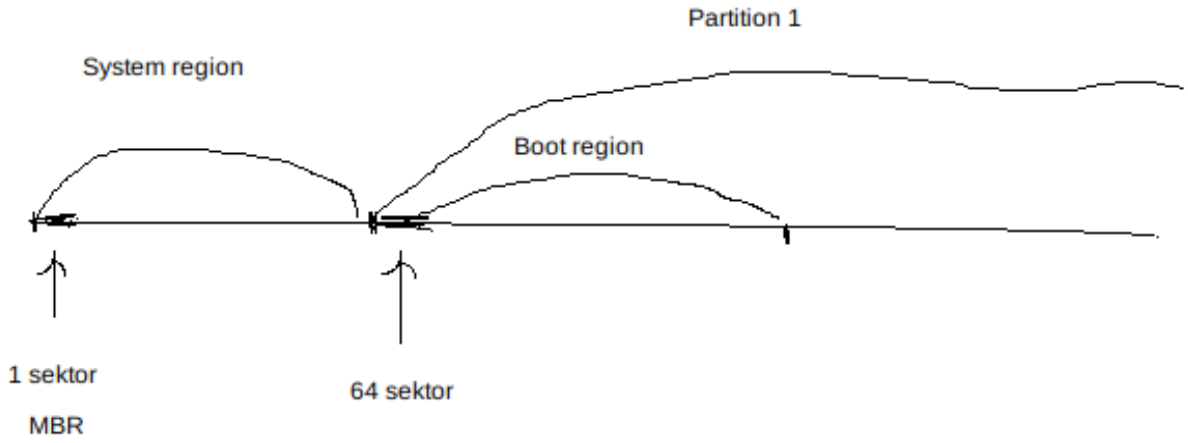

Рис. 20. Начало винчестера

В самом начале — системная область размером 63 сектора. В первом секторе этой области — MBR. Начиная с 64 сектора — первый первичный (primary) раздел. В начале раздела при форматировании раздела почти всегда (во всех файловых системах) создаётся загрузочная область размером от нескольких секторов (в ms dos) до нескольких килобайт (ufs — 8 кб, ext — 4-8 кб, ntfs — 8 кб и т. д.). В эту загрузочную область раздела помещается вторичный загрузчик, если в разделе установлена какая-либо ОС, либо эта область пустая, если на разделе ОС не стоит.

58. А что такое «полуторный загрузчик» и где он находится?

«Полуторный загрузчик» существовал явно, как отдельный файл, в GRUB-1 и помещался в сектора системной области, начиная со второго, то есть, сразу за MBR. Он предназначался для работы с конкретной файловой системой раздела с операционной системой: чтобы опознать её и загрузить с неё вторичный загрузчик. GRUB-2 стал модульным и теперь «полуторный загрузчик» существует в виде модуля GRUB с той же функциональностью. Размещается там же в системной области в секторах со 2-го по 63.

59. Как подготовить hdd к использованию в системе?

1. Правильно подключить диск к системе.

— диск SATA. Здесь всё просто; достаточно правильно разъём вставить;

— диск PATA, в народе чаще называемый IDE. Здесь в три раза всё сложнее.

Во-первых, интерфейсов IDE много, как минимум, штук шесть: от очень древних (PIO и др.), до более современных UDMA-33 (40-pin`овый кабель) и UDMA-66/100/133 (80-pin`овый кабель).

Во-вторых, на hdd , помимо интерфейсной колодки (40-штырьковой) и колодки питания (4-штырьковой), как правило, есть ещё одна колодка на 4/6/8/10 контактов а зависимости от марки hdd и производителя. На этих контактах нужно правильно поставить перемычку (определить статус устройства), иногда, не одну. Как ставить, обычно написано на наклейке на hdd.

В-третьих, на кабеле — три колодки: одна — в системную плату, а на две других можно подключать устройства hdd, DVDROM или др.

То есть, кабелей два (контроллер IDE — двухканальный), а устройств можно подключить всего четыре. Причём так, чтобы статус у всех устройств был разный, иначе компьютер загружаться не будет. Статусы: IDE-0 (первый кабель) — master-1 и slave-1. IDE-1 (второй кабель) — master-2 и slave-2.

Определение статусов. Если интерфейс UDMA-33 (40-pin`овый кабель), то перемычки нужно поставить так, чтобы на каждом кабеле одно устройство было master, а второе — slave. Повторяем: на кабеле два разъёма и на одном разъёме устройство должно быть master, а на втором — slave. И никак иначе. Кто на каком? См. ниже.

Если интерфейс UDMA-66, 100 или 133, то кабель выглядит так: цветной разъём (розовый, синий, жёлтый, «булыжного» цвета) — в системную плату, средний разъём — серый(!) для устройства slave, последний (крайний) разъём — чёрный(!) для устройства master. На обеих (на обеих!) устройствах, подключаемых на один кабель, перемычки ставятся в положение CS — cabel select, то есть, кто есть кто — определяется кабелем, точнее тем, на какой разъём вы подключите устройство.

Именование. В Windows это не столь важно, хотя значение имеет. В linux это имеет большое значение и выглядит так:

Статус устройства	Имя устройства
master-1	sda
slave-1	sdb
master-2	sdc
slave-2	sdd

То есть, как мы подключили устройство — так мы его и именуем. А поскольку имя устройства — элемент полного пути к файлу, то, следовательно, для linux порядок подключения устройств имеет большое значение. С одной стороны — лишняя морока. Но с другой — точно знаешь, на каком hdd лежит файл. Если hdd много. Если один, то, как подключили — значения не имеет, лишь бы статус был определён.

2. Разбить hdd на разделы. На hdd должен быть создан хотя бы один раздел. Обязательно. В Windows это делается, как правило, встроенными средствами Windows (в панели управления; можно и вручную программами fdisk или diskpart в PowerShell). В linux делается в терминале с помощью программ fdisk или gdisk (есть в любом дистрибутиве) или gparted.

3. Сформатировать раздел(ы). В Windows — в панели управления в управлении дисками согласиться на предложение системы: «Диск не отформатирован, вам его отформатировать?». Если понимаете, о чём это — соглашайтесь. В linux — в терминале с помощью программы mkfs.

После выполнения этих трёх шагов, hdd (точнее, разделы на нём) готов к использованию.

60. Что такое Partition Table?

Partition Table (PT) — таблица разделов. Таблица определённого формата (отсюда термин — «формат разбиения hdd»), в которой описывается, как разбит на разделы hdd. Таблица определена в ОС, то есть, её «сочинили» программисты, которые писали операционную систему. Как следствие, в каждой ОС — таблицы разделов разные. Исключение — linux. Поэтому появился термин — формат разбиения hdd, который определяется операционной системой.

Самые распространённые из ныне используемых таблицы разделов показаны на рисунке 21, где таблица GPT показана в том урезанном/упрощённом виде, в котором она используется в Windows.

Таблица разделов MBR — старая, разработана вначале 80-х для ПЭВМ PC IBM, на которых устанавливались винчестеры 10-40 Мб. Винче-

стер маленький, на разделы не разбивался — обычно использовался один раздел, соответственно и таблица разделов была простой.

Проблемы возникли, когда винчестеры стали большие. Поэтому была создана новая таблица разделов GPT с существенно лучшими возможностями. Соответственно, появился новый формат разбиения — формат GPT, на который сейчас постепенно переходят.

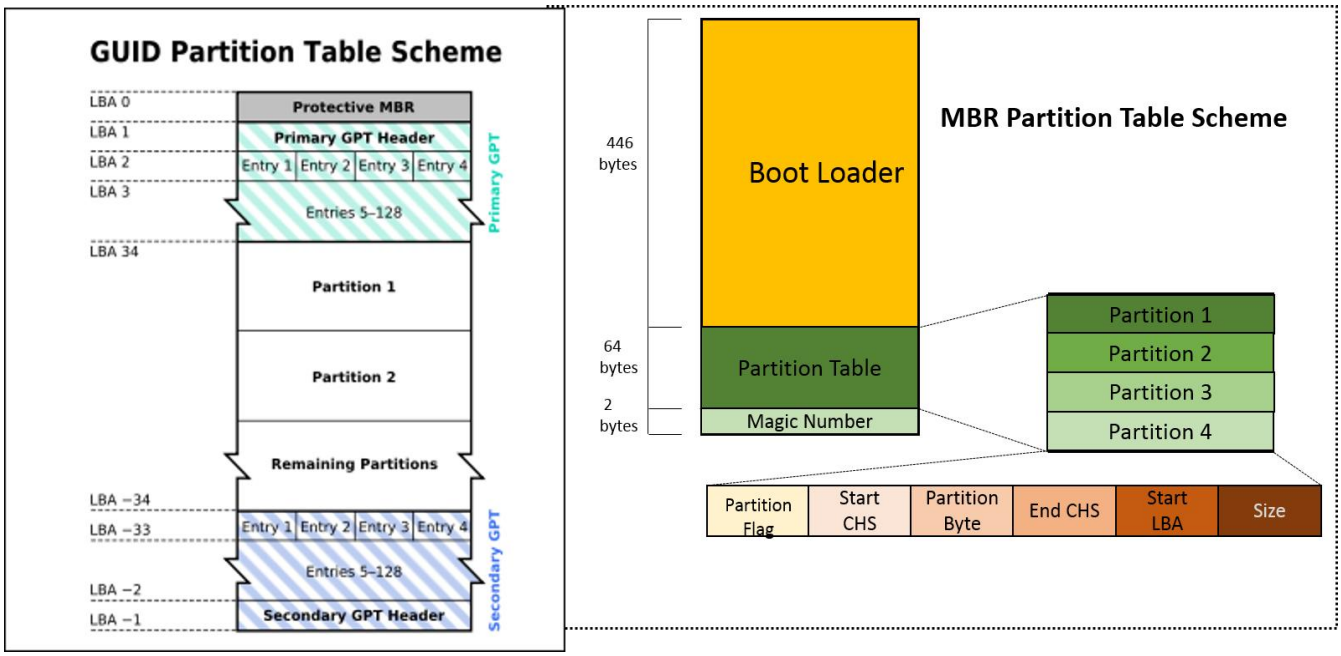

Рис. 21. Таблицы разделов MBR и GPT

61. Какие бывают Partition Table?

Первый ответ на вопрос. В каждой ОС таблица разделов (Partition Table — PT) своя специфическая, несовместимая с таблицами других ОС. Исключение — linux. В программах такая структура данных как таблица описывается как двумерный массив, а иногда как списковая структура (список строк). Определение этой таблицы в программе ОС — формат этой таблицы — определяет формат разбиения винчестера. Отсюда первый ответ на этот вопрос: в каждой ОС свой формат таблицы разделов.

Второй ответ на вопрос. Однако, в каждом формате разбиения hdd таблиц разделов может быть несколько. Это второй ответ на этот вопрос.

Примеры:

— в формате GPT таблиц две — основная в первой (начальной) системной области и резервная во второй системной области (см. рисунок 21);

— в формате PC BIOS (он же формат msdos, он же формат MBR) таблиц разделов может быть от одной, которая в MBR — основная (если

разделов на hdd меньше 4-х), до 1+i, где i — количество логических разделов в расширенном разделе (extended); то есть, каждый логический раздел описывается своей собственной дополнительной логической таблицей разделов; на первую логическую таблицу ссылается строка в главной таблице разделов, которая определяет расширенный раздел; а далее все логические таблицы связаны в списковую структуру;

— аналогично, в других ОС также могут быть основные, резервные и дополнительные таблицы разделов.

Примечание. В формате GPT в настоящее время используется две таблицы разделов (см. выше), но при разработке формата GPT предусматривалось использовать 4 таблицы разделов — две в начале и две в конце винчестера. Однако в ОС Windows реализованы только две таблицы разделов, остальным (прежде всего, linux) пришлось подстраиваться под это решение в целях совместимости, поскольку MS Windows на ПЭВМ — почти монополист пока.

62. Где могут находиться Partition Table?

См. ответ на вопрос 61.

63. Сколько Partition Table может быть на hdd?

См. ответ на вопрос 61.

64. Что такое раздел на hdd?

Неразрывная последовательность секторов на hdd, описанная в таблице разделов как целая сущность.

65. Чем отличается раздел от файловой системы?

Раздел — это . . . см. ответ на вопрос 64.

Файловая система — это метод организации хранения файлов в разделе, то есть, алгоритм, определяющий то, как организовано хранение файлов в некотором разделе. В другом разделе алгоритм может быть другой.

66. Может ли раздел содержать несколько файловых систем?

66.1. Нет и никогда от слова совсем

Файловая система создаётся в разделе программой mkfs (или format), в качестве аргумента которой указывается раздел на hdd. Программа mkfs считывает из таблицы разделов адреса начала и конца раздела (адрес 1-ого

сектора раздела и адрес последнего сектора раздела) и его размер в секторах, пересчитывает сектора в блоки и пишет структуры данных файловой системы в нужные места раздела. То есть, весь раздел занимается целиком.

Примеры:

mkfs -t ntfs /dev/sda1 — отформатировать первый раздел на диске a, создать файловую систему ntfs.

mkfs /dev/sdb2 — отформатировать второй раздел на диске b, файловая система по умолчанию.

mkfs -t fat32 /dev/sdd1 — отформатировать раздел на флешке, создать файловую систему FAT32.

66.2. Может

В linux всё можно. Даже то, что нельзя, но очень хочется. Чтобы продемонстрировать возможности уровня файловой системы Linux (и монтирования), создадим файловую систему в файле, расположенном в существующей файловой системе. Это можно сделать путем создания файла заданного размера с помощью dd (копирование файла из источника /dev/zero) — другими словами, инициализируя файл нулями:

```
$ dd if=/dev/zero of=file.img bs=1k count=10000
10000+0 records in
10000+0 records out
```

Теперь у нас есть файл file.img размером 10 МБ. Свяжем с файлом блочное устройство-заглушку (loop) с помощью команды losetup (чтобы он выглядел как блочное устройство, а не как обычный файл файловой системы):

```
$ losetup /dev/loop0 file.img
```

Теперь, имея файл, который выглядит как блочное устройство (представлен /dev/loop0), создадим на этом устройстве файловую систему с помощью mke2fs. Эта команда создает новую файловую систему ext2 определенного нами размера.

```
$ mke2fs -c /dev/loop0 10000
mke2fs 1.35 (28-Feb-2004)
max_blocks 1024000, rsv_groups = 1250, rsv_gdb = 39
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
```

```
2512 inodes, 10000 blocks
500 blocks (5.00%) reserved for the super user
. . .
```

Теперь файл file.img, представленный блочным устройством (/dev/loop0), может быть смонтирован в точке /mnt/point1 с помощью команды mount.

Обратите внимание, что указанный тип файловой системы — ext2. После монтирования вы можете обращаться к точке монтирования как к новой файловой системе с помощью, например, команды ls.

```
$ mkdir /mnt/point1
$ mount -t ext2 /dev/loop0 /mnt/point1
$ ls /mnt/point1
lost+found
```

Как показано ниже, этот процесс можно продолжить, создавая новый файл в новой файловой системе, связывая его с устройством loop и создавая в нем еще одну файловую систему.

```
$ dd if=/dev/zero of=/mnt/point1/file.img bs=1k count=1000
1000+0 records in
1000+0 records out
$ losetup /dev/loop1 /mnt/point1/file.img
$ mke2fs -c /dev/loop1 1000
mke2fs 1.35 (28-Feb-2004)
max_blocks 1024000, rsv_groups = 125, rsv_gdb = 3
Filesystem label=
...
$ mkdir /mnt/point2
$ mount -t ext2 /dev/loop1 /mnt/point2
$ ls /mnt/point2
lost+found
$ ls /mnt/point1
file.img lost+found
```

Из этого простого примера легко понять, насколько большие возможности предоставляет файловая система (и устройство loop) в Linux. Аналогичным образом с помощью устройства loop можно создавать в файле файловые системы с шифрованием. Это может быть полезно для защиты ваших данных; при необходимости такой файл можно быстро смонтировать с помощью устройства loop.

67. Чем отличается раздел от hdd?

Hdd — всё устройство, винчестер.

Раздел — часть hdd, некоторая неразрывная последовательность смежных секторов на hdd.

68. Что такое «расширенный раздел»?

Это понятие определено только в формате PC BIOS: расширенный раздел в этом формате — контейнер, в котором создаются логические разделы. Поскольку в формате PC BIOS основная таблица разделов содержит только 4 строки — по современным понятиям это очень мало, то при необходимости создания больше 4-х разделов, приходится делать такой «финт»: объявлять один из разделов (конечно же, последний, 4-й) расширенным, почти всегда «на всё оставшееся место», и в нём создавать искусственные сущности — логические разделы, объединяя их в списковую структуру.

Почему раздел extended должен создаваться 4-ым, последним в таблице, разделом? Потому что, хранение описания раздела в Главной Таблице Разделов всё-таки надёжнее, нежели где-то в списковой структуре. Поэтому Главную Таблицу разделов нужно стремиться занять полностью.

См. ответ на вопрос 61.

69. На каких устройствах может быть создан расширенный раздел?

Расширенный раздел может быть создан на блоковых устройствах типа hdd и устройствах, имитирующих hdd, как то: flash, SSD, microSD и др. Если эти устройства разбиваются на разделы в формате PC BIOS.

ФАЙЛОВЫЕ СИСТЕМЫ

70. Как ОС определяет, какая файловая система на разделе?

В таблицах разделов есть графа «идентификатор раздела» — id. Он определяет, под какую файловую систему планируется использовать данный раздел. Программы форматирования должны им руководствоваться. Но, увы, так бывает не всегда.

ОС, при монтировании раздела, считывает суперблок файловой системы или структуру, аналогичную суперблоку, если в файловой системе эта структура по-другому называется. А в суперблоке указано, какая это файловая система.

71. Что находится в первом секторе файловой системы?

Как правило, начальные сектора раздела отводятся под размещение вторичного загрузчика. В разных файловых системах вторичный загрузчик может занимать от одного до нескольких десятков секторов. Если на разделе не установлен вторичный загрузчик (если в этом разделе нет ОС, просто файлы хранятся), то место под него всё равно резервируется: вдруг пользователь осознает свою ошибку?

72. Взаимосвязь между каталогом и индексной таблицей

Каталог — файл специального формата, типа таблица — двумерный массив. Имеет структуру:

I — Индекс файла, порядковый номер строки индексной таблицы, в которой содержатся атрибуты файла	**L — Общая длина строки каталога — длина всех полей строки в байтах**	**l — длина имени файла в байтах**	**Имя файла — до 256 символов**
. . . .	. . .	. . .	. . .

Через первое поле каталога (индекс) осуществляется связь между каталогом и индексной таблицей.

73. Файловая система — определение

Файловая система — это подсистема ОС, управляющая ресурсом «дисковое пространство». Её цель и назначение — эффективное управле-

ние данным ресурсом (https://habr.com/ru/post/108629/). Такое целеполагание означает, что основным содержанием файловой системы являются алгоритмы.

Определение 1. **Фа́йловая систе́ма** — это **метод** организации хранения файлов на разделе устройства хранения (на разделе девайса).

Определение 2. **Фа́йловая систе́ма** — порядок, **определяющий** способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п

Термины «метод», «порядок», «способ» — подразумевают наличие алгоритма, реализующего этот «метод», «порядок», «способ». Естественно, он есть и в файловой системе. Даже не один. Современные файловые системы многозадачных (и даже многопользовательских) операционных систем включают два основных алгоритма:

— алгоритм преобразования имён: символьных «человеческих» в цифровые имена операционной системы и соответствующего устройства хранения информации,

— алгоритм «расшаривания» устройства (точнее, файловой системы (как объекта) на устройстве) между многочисленными программами, выполняющимися в системе, и пользователями, работающими в системе.

Эти алгоритмы в основном реализованы в коде самой операционной системы (иногда бывает и вне, например, в случае использования файловых систем пространства пользователя). Структура реализации файловых систем в ядре linux показана на рисунке 22.

Для программы пользователя библиотека libc предоставляет интерфейс для доступа к системным вызовам.

Интерфейс системных вызовов действует как коммутатор, направляющий системные вызовы из пространства пользователя в соответствующие точки пространства ядра, в том числе, к VFS.

VFS является основным интерфейсом к файловым системам нижнего уровня. Этот компонент экспортирует набор интерфейсов файловых систем и передаёт вызовы в отдельные файловые системы, образ поведения которых может быть весьма различным. Для объектов файловой системы существуют два кэша: кэш узлов inode и кэш каталогов. Каждый из них предоставляет пул недавно использованных объектов файловой системы.

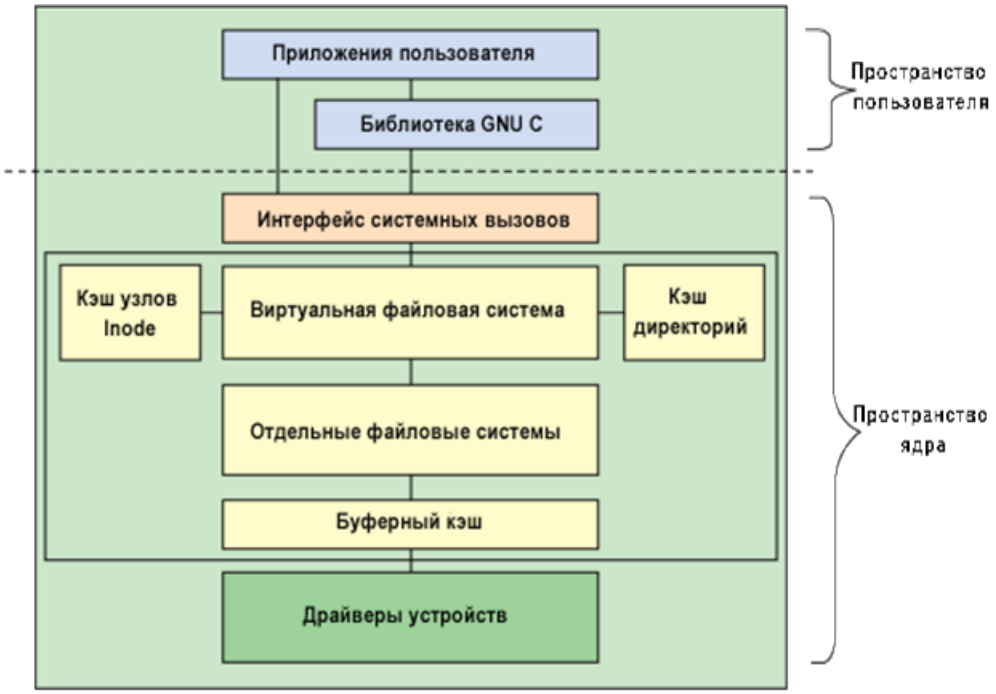

Рис. 22. Подсистема ввода/вывода linux: реализация файловых систем

Реализация каждой файловой системы, например, ext2, JFS, ntfs и так далее, экспортирует общий массив интерфейсов, который используется (и ожидается) VFS. Буферный кэш буферизирует запросы между файловыми системами и блочными устройствами, которыми они могут управлять. Например, через буферный кэш проходят запросы на чтение и запись к драйверам устройств. Это позволяет кэшировать запросы для более быстрого доступа (вместо обращения непосредственно к физическому устройству). Буферный кэш управляется набором списков последних использованных элементов. Обратите внимание, что командой sync можно сбросить буферный кэш на носитель (принудительно отправить все незаписанные данные на драйверы устройств и, в дальнейшем, на устройства хранения).

Параметры создания файловых систем (процесса форматирования раздела) в linux обычно указываются в конфигурационных файлах.

Пример: Файл /etc/mke2fs.conf устанавливает основные параметры файловой системы ext2/3/4 при форматировании раздела.

```
[defaults]
        base_features =
sparse_super,large_file,filetype,resize_inode,dir_index,ext_attr
        default_mntopts = acl,user_xattr
```

```
        enable_periodic_fsck = 0
        blocksize = 4096
        inode_size = 256
        inode_ratio = 16384

    [fs_types]
        ext3 = {
            features = has_journal
        }
        ext4 = {
            features =
has_journal,extent,huge_file,flex_bg,uninit_bg,dir_nlink,extra_isize
            auto_64-bit_support = 1
            inode_size = 256
        }
        ext4dev = {
            features =
has_journal,extent,huge_file,flex_bg,uninit_bg,dir_nlink,extra_isize
            inode_size = 256
            options = test_fs=1
        }
        small = {
            blocksize = 1024
            inode_size = 128
            inode_ratio = 4096
        }
        floppy = {
            blocksize = 1024
            inode_size = 128
            inode_ratio = 8192
        }
        big = {
            inode_ratio = 32768
        }
        huge = {
            inode_ratio = 65536
        }
        news = {
            inode_ratio = 4096
```

```
}
largefile = {
        inode_ratio = 1048576
        blocksize = -1
}
largefile4 = {
        inode_ratio = 4194304
        blocksize = -1
}
hurd = {
    blocksize = 4096
    inode_size = 128
}
```

74. Файл — определение

Файл – это именованная область на диске, размером от 1-ого блока до некоторого предела, устанавливаемого файловой системой. Эта область может быть занята некоторыми данными, а может быть пустой — зарезервированная область под что-то. Именованная область может быть составной (фрагментированной).

75. Сектор на диске и сектор в файловой системе

Вообще говоря, это одно и то же. Но, всё-таки есть отличия.

Сектор в файловой системе — это последовательность байт, длиной 512 байт обычно на старых hdd, и 4 кб на новых, терабайтных hdd (Advanced Format)..

Тот же сектор, записанный на hdd — то же последовательность байт, но длиной гораздо большей. Потому что, хранение информации на hdd — это хранение информации в ненадёжной среде.

Для обеспечения надёжности её восстановления к первоначальному виду информация (сектора данных) кодируется избыточными кодами, в результате этого сектор на hdd размером существенно больше сектора в файловой системе. Насколько больше — зависит от используемого контроллером hdd метода кодирования. Например, коды Рида-Соломона дают избыточность порядка 25%. Это значит, что если контроллер реализует данный вид избыточного кодирования, то размер области данных в секторах составляет примерно 640 байт, а весь сектор вместе со служебной информацией — примерно 700 байт. Однако для обеспечения надёжности иногда используют и более избыточные вилы кодирования.

Но! Кодирование избыточными кодами тщательно скрывается контроллером hdd, то есть, его никак не видно. О нём знает только контроллер hdd, оно реализовано на уровне внутренних алгоритмов контроллера.

Кроме того, в секторе hdd, кроме самих данных, хранятся также коды ECC — коды коррекции ошибок (см. рисунок 23). В отличие от избыточного кодирования коды ECC считаются данными и входят в область данных.

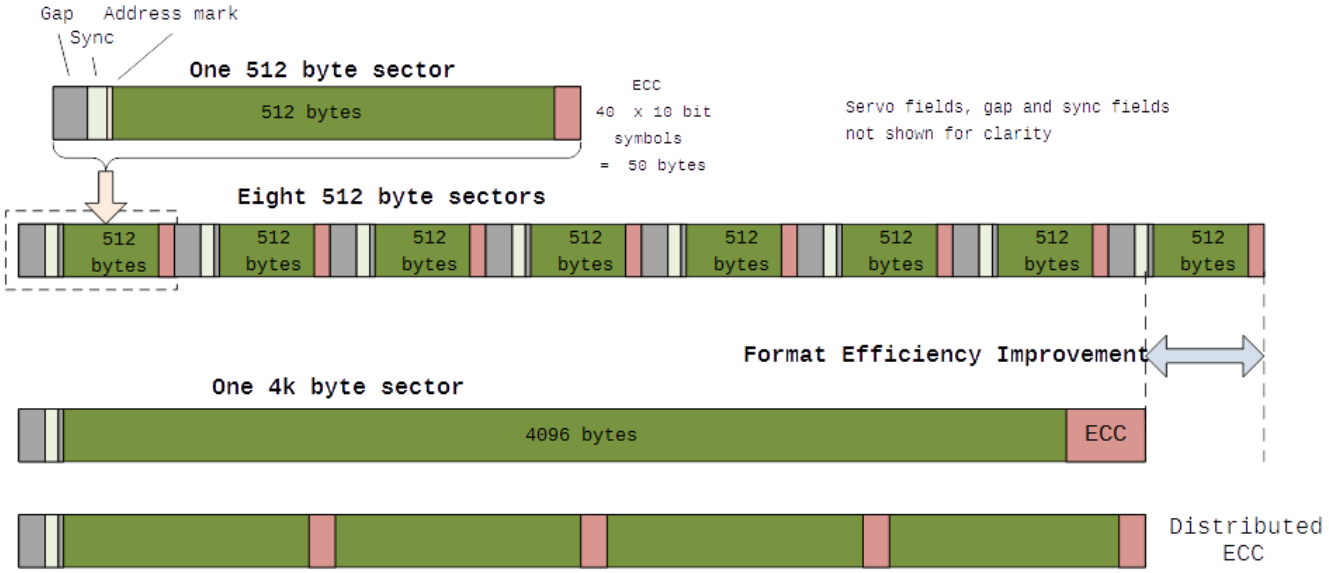

Рис. 23. Форматы секторов hdd: наверху — 512-байтные, внизу — 4-килобайтные

Форматы секторов также влияют на эффективность хранения данных на жёстких дисках:

— разметка с традиционными 512-байтными секторами требует большего количества служебных меток и ECC-данных (их размер в этом случае 50 байт), обеспечивая эффективность форматирования (отношение доступного пользователю объёма накопителя к его фактическому объёму) 87 %;

— разметка по технология 4K (сектора размером 4 килобайта) позволяет уменьшить место, требуемое для служебных меток; ECC-данные (их размер в этом случае 100 байт) могут храниться в конце сектора или распределённо — «distributed ECC», обеспечивая эффективность форматирования до 96 %, то есть, объём доступного пользователю дискового пространства увеличивается на 7-11 %.

76. Файлов2ая система: сектор vs кластер

Кластер — последовательность смежных секторов, длиной 1, 2, 4, 8, 16, 32, 64 и изредка может быть 128 байт.

77. Файловая система сектор vs блок

См. ответ на вопрос 79.

78. Кластер vs блок

Внешне — это почти одно и то же, только кластер — это в ОС Windows, а блок — в ОС linux.

Но внутри есть различия:

— кластер не делится дальше, это минимальная единица хранения информации в виндовых файловых системах;

— блок в unix`овых файловых системах (ufs, ext и некоторых других) может делиться на фрагменты блока. Фрагмент — последовательность смежных секторов в блоке. Фрагментов в блоке может быть 2, 4, 8, 16, 32 — в зависимости от размера блока.

Придуманы фрагменты для того, чтобы уменьшить потери пространства в файловой системе, поскольку последний (или единственный) блок файла обычно неполный. Файловая система может помещать маленькие файлы во фрагменты блока, тем самым экономя пространство.

79. Что такое блок файловой системы?

Блок файловой системы — небольшая последовательность смежных секторов, обычно размером 1, 2, 4, 8, 16, 32, 64 сектора. Изредка используется ещё размер 128 секторов в блоке. Размер блока определяется при форматировании раздела программой mkfs.

В файловой системе ext2/3/4 размер блока в настоящее время (ноябрь 2019 года, версии ядра 4.x, 5.x) определён в файле */etc/*mke2fs.conf и составляет 1, 2, 4 или 8 кб, то есть, 1, 4, 8 или 16 секторов (см. ответ на вопрос 73).

80. Hdd разбит на сектора; а откуда берётся блок файловой система?

Блок — логическая сущность, создаётся в файловой системе при форматировании раздела. Цель:

— уменьшить числа (адреса секторов) с которыми приходится оперировать алгоритмам файловой системы;

— тем самым, ускорить работу файловой системы;

— увеличить размеры обрабатываемых файловой системой разделов, файлов.

81. Где применяются блоки файловой системы, а где кластеры файловой системы?

Блоки — применяются в unix`овых файловых системах, кластеры — в виндовых файловых системах.

82. Что такое директорий, каталог, папка?

Директо́рия — транслитерация с английского *directory* (правильный перевод – **каталог**).

Каталог — объект файловой системы, также является файлом и определяется в системе, как файл. Этот файл имеет специальный формат (типа таблица, но хранящийся, обычно, как список), содержит записи, которые связывают каждое имя файла с номером индекса (порядковым номером строки индексной таблицы), где файл описывается, где хранятся атрибуты файла. Некоторые версии файловых систем дополнительно в каталоге содержат ещё одну колонку — тип файла (обычный файл, каталог, символьная ссылка, устройство, fifo, сокет), чтобы избежать необходимости читать саму индексную таблицу и искать в ней эту информацию.

При форматировании раздела создаваемая индексная таблица делится поровну между всеми группами блоков. А при создании файла, алгоритм файловой системы должен выделить блоки под данные файла в той же группе блоков, в которой находится каталог с именем файла.

Исходная версия Ext2 использовала односвязный список для хранения имен файлов в каталоге; более новые версии могут использовать хэши и двоичные деревья.

По мере роста каталога ему выделяются дополнительные блоки для хранения дополнительных записей файлов, то есть, каталоги растут постепенно по мере заполнения файловой системы. При удалении имен файлов некоторые реализации не освобождают эти дополнительные блоки в целях избегания потерь времени.

Папка (*folder*) — канцелярская папка с арочным механизмом для хранения документов (файлов; см. рисунок 23).

Термин **папка** – *folder,* был введён для представления объектов файловой системы в графическом пользовательском интерфейсе путём аналогии с офисными папками. Он был впервые использован в Mac System Software, предшественнице Mac OS, а в системах семейства Windows — с выходом Windows 95. Эта метафора стала использоваться в большом числе операционных систем: Windows NT, Mac OS, Mac OS X, а также в средах рабочего стола для систем семейства UNIX (например, KDE и GNOME).

! До выхода Windows 95 это понятие – папка — называлось словами каталог или директория.

Папка — регистратор документов

Замок папки-регистратора

Рис. 23. Папка

83. Что такое каталог в файловой системе ext2/3/4?

Каталог в файловой системе ext2/3/4 – это файл типа каталог, имеющий структуру таблицы, но хранящийся как список.

I — Индекс файла, порядковый номер строки индексной таблицы, в которой содержатся атрибуты файла, 4 байта	**L — Общая длина строки каталога — длина всех полей строки в байтах, 2 байта**	**l — длина имени файла в байтах, 1 байт**	**Имя файла, до 255 байт**
. . .	. . .	. . .	. . . .
N = {от 2 до 4 млрд}	15	8	file.txt
. . . .	. . .	. . .	. . .

Структура записи каталога struct ext2_dir_entry_2:

#define EXT2_NAME_LEN 255
__u32 inode — номер inode-а файла
__u16 rec_len — длина записи каталога
__u8 name_len — длина имени файла
char name[EXT2_NAME_LEN] — имя файла

В ext4 в этой таблице может быть ещё одна колонка — тип файла, 1 (или 2) байта.

84. Что такое каталог в файловой системе FAT

Каталог в файловых системах FAT-12/16/32 (см. рисунок 24) является обычным файлом, помеченным специальным атрибутом в поле DIR_Attr — значение 0x10. Формат этого файла — таблица из 12 колонок. Каждая строка таблицы имеет фиксированную длину 32 байта — это файловые записи (записи каталога). Внимание! FAT-32 называется FAT именно 32 вовсе не потому, что длина строки каталога равна 32 байтам.

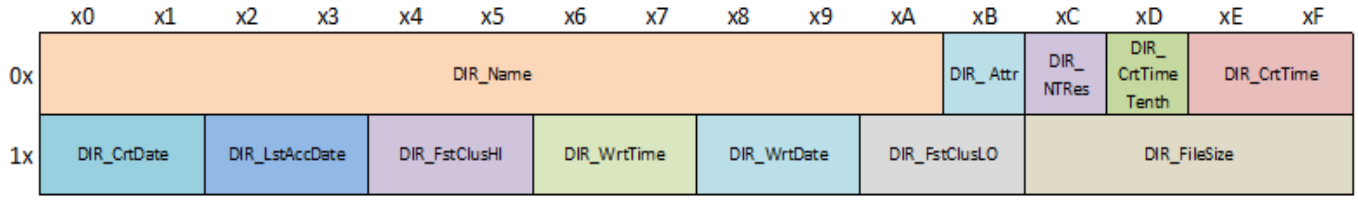

Рис. 24. Строка каталога файловой системы FAT-32

Файловая запись FAT состоит из следующих полей:

— DIR_Name — 11-байтовое поле по относительному адресу 0, содержит короткое имя файла (в рамках стандарта 8.3; для точки здесь места нет, точка добавляется на уровне операционной системы);

— DIR_Attr — байт атрибутов файла;

— DIR_NTRes — байт используется в Windows NT, зарезервирован;

— DIR_CrtTimeTenth — байт счётчика десятков миллисекунд времени создания файла, допустимы значения 0–199. Поле обычно игнорируется;

— DIR_CrtTime — 2 байта определяют время создания файла с точностью до 2 секунд;

— DIR_CrtDate — 2 байта даты создания файла;

— DIR_LstAccDate — 2 байта даты последнего доступа к файлу (то есть последнего чтения или записи — в последнем случае приравнивается DIR_WrtDate). Аналогичное поле для времени не предусмотрено;

— DIR_FstClusHI — 2 байта номера первого кластера файла (старшее слово, в файловых системах FAT12/FAT16 равен нулю);

— DIR_WrtTime — 2 байта определяют время последней записи (модификации) файла, например его создания;

— DIR_WrtDate — 2 байта даты последней записи (модификации) файла, в том числе создания;

— DIR_FstClusLO — 2 байта номера первого кластера файла (младшее слово, используется в FAT-12/16/32);

— DIR_FileSize — 4байта, содержащие значение размера файла в байтах. Фундаментальное ограничение FAT32 — максимально допустимое значение размера файла составляет 0xFFFFFFFF (то есть 4 Гбайт минус 1 байт).

Если первый байт записи FAT (то есть DIR_Name[0]) содержит 0xE5 или 0x05, это значит, что запись свободна (соответствующий файл был удалён). Ноль в DIR_Name[0] означает, что свободна не только эта запись, но и все следующие записи каталога; Windows не анализирует остаток каталога после обнулённой записи.

Если файловая система поддерживает длинные имена (FAT-32/vFAT/exFAT до 255 символов), то для такого файла всё равно заводится обычная каталожная запись с коротким именем, а длинное имя сохраняется в каталоге в записях, последующих после обычной. Количество дополнительных строк каталога определяется длиной имени файла.

85. Что такое каталог в файловой системе ntfs?

О-о! Это очень сложная вещь не только для всех. Каталог в файловой системе ntfs представляется записью MFT (см. рисунок 25), у которой установлен специальный флаг в заголовке записи и в атрибутах $STANDART_INFORMATION и $FILE_NAME, а по содержанию — это таблица, которая содержит следующие графы:

— индекс файла — номер строки MFT, в которой содержатся атрибуты файла,

— длина имени файла — длину «длинного виндового» имени (уточняем, потому что в файловой системе ntfs файл имеет три имени — имя msdos в формате 8.3, имя POSIX — unix-овое с различием регистра символов и имя «виндовое» — в котором большие и маленькие буквы не различаются), она может от 1 до 255,

— само длинное «виндовое» имя файла,

— поле основных атрибутов файла, в число их входят: временные штампы, размеры потоков файла и некоторые флаги.

Однако хранится «таблица» каталога как списковая структура, как последовательность записей переменной длины — ведь, имена файлов разной длины. Для того, чтобы найти конец одной записи каталога и начало другой, используется поле «длина имени файла».

Если каталог маленький (не более нескольких десятков файлов, сколько — определяется версией ntfs), то он хранится «резидентно», то есть, в самой записи MFT, описывающей файл каталога — хотя в этом случае файла каталога как раз может и не быть, всё — в записи (см. рисунок 25). Но поскольку в ntfs «всё есть файл», то в предыдущем предложении противоречия нет.

Рис. 25. Запись MFT, содержащая «маленький» каталог ntfs

Но даже если каталог хранится резидентно, некоторые атрибуты его вполне могут храниться в кластерах где-то в другом месте.

Если каталог большой, то начальная часть его хранится резидентно в MFT, а остальная часть является обычным файлом, помеченным специальным атрибутом в строке MFT.

Большие каталоги MFT хранятся в отсортированном виде — как бинарное дерево, утверждается, что это делается для ускорения доступа к файлам.

В целом, авторы отмечают, что файловая система ntfs крайне запутанная («запущенная»), складывается впечатление, что писали её программисты, не совсем понимавшие, что делают, не имевшие целостного представления о своём предмете труда и налепившие массу «заплат». Кто не верит — пусть проверит. Попробуйте написать программу, которая всего-навсего будет читать указанный каталог ntfs (только каталог) и выдавать информацию из него (все его поля) в виде таблички.

86. Как выглядит таблица файлов в файловой системе FAT?

Таблица файлов в файловой системе FAT выглядит как одномерная таблица, то есть, таблица из одной колонки. Каждое поле — два байта для FAT-12/16, или четыре байта — для FAT-32.

Каждая строка соответствует кластеру в файловой системе. Если в строке 0 (нуль), то кластер свободен; код FFF8 или FFFFFFF8, то это кластер конца файла; если код FFF7 или FFFFFFF7, то сбойный кластер; если число в диапазоне от 3 до максимальное_количество_кластеров, то это указатель на кластер продолжения большого файла.

87. Что такое корневой каталог файловой системы?

Корневой каталог файловой системы – это каталог самого верхнего уровня файловой системы, выше которого ничего нет от слова совсем.

В unix/linux он обозначается «/» (слэшь).

В Windows корневых каталогов столько, сколько «букв дисков». Обозначаются корневые каталоги буквой диска, например, так: D:\

88. Что такое index файла?

В unix/linux индекс файла — это номер строки индексной таблицы, в которой находятся атрибуты файла.

89. Как увидеть index файла?

В unix/linux индекс файла — это номер строки индексной таблицы, в которой находятся атрибуты файла. Увидеть эти номера строк (индексы) можно с помощью команды

```
ls -i -l
```

Вывод этой команды — таблица с атрибутами файлов текущего каталога (см. рисунок 26). Числа в первой колонке это и есть индексы файлов.

90. Что такое жёсткая ссылка (hard link)?

В файловых системах unix/linux атрибуты файлов хранятся в строках индексной таблицы, а имена файлов отдельно в каталогах. В строке каталога вместе с именем файла хранится также номер строки индексной таблицы, в которой файл описан. Этот номер называется индексом файла — см. ответ на вопрос 88. С помощью этого номера осуществляется связь каталога с индексной таблицей, называется эта связь — link или «жёсткая ссылка».

Такой способ определения файлов (атрибуты отдельно в индексной таблице, имя файла отдельно в каталоге) позволяет одному и тому же файлу присваивать несколько имён. То есть, связей link — «жёстких ссылок», у файла может быть много. Другими словами, у файла может быть много имён. В индексной таблице среди атрибутов файла есть параметр — «количество имён файла», значение этого параметра равно количеству имён файла.

Как это увидеть — см. рисунок 26.

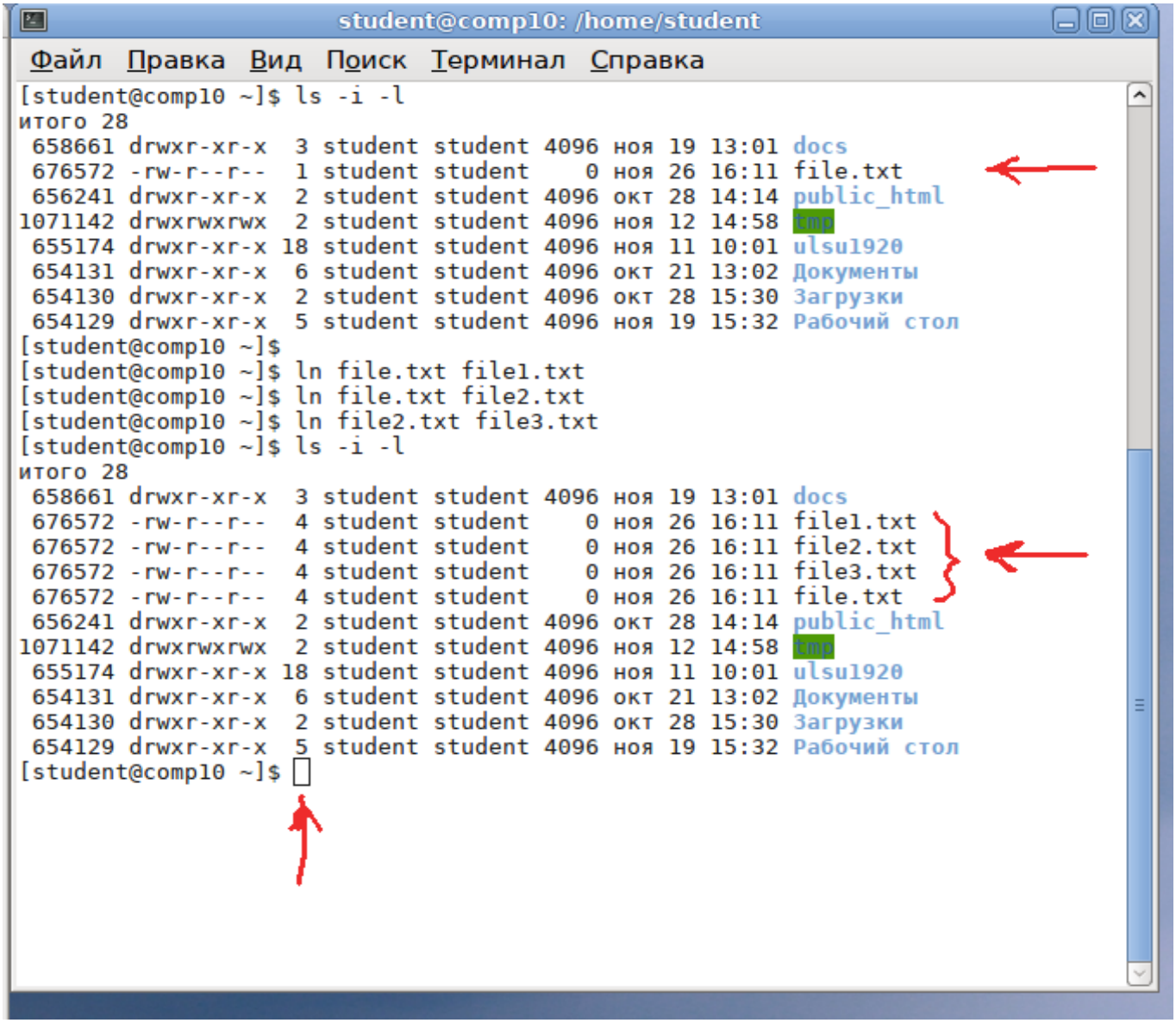

Рис. 26. Hardlinks

Стрелка снизу указывает на колонку «количество имён». Обратите внимание, что файловая система не различает, на какое имя вы создали

очередную hardlinks — все имена для неё одинаковы. Также обратите внимание на индексы файла в первой колонке.

91. Что такое мягкая ссылка (soft link)?

«Мягкая ссылка» — softlinks» — особый тип файла, в котором сохраняется путь к другому файлу. Обычно используется тогда, когда неудобно пользоваться полным или относительным (относительно текущего каталога) именем файла, который лежит где-то там, иногда даже трудно вспомнить, где он там находится и даже как называется. Чтобы с этим не заморачиваться, в своём каталоге создаём мягкую ссылку, в которой это всё запоминается. Создаётся командой:

ln -s <путь_к_файлу_где-то_там> <новое_удобное_имя_здесь>

92. Типы файлов в linux.

В linux определены следующие типы файлов:

— каталог,

— блоковое устройство,

— символьное устройство,

— файл fifo — файл имени именованного канала (именованного pipe),

— файл socket — файл имени сокета,

— символьная (мягкая) ссылка,

— обычный файл.

Что такое каталог — см. предыдущие вопросы.

Файлы блокового и символьного устройств — файлы нулевого размера, в который резидентно хранятся адреса драйверов устройств. В них нет ничего интересного, они находятся в каталоге /dev, ими пользуется операционная система.

Файлы fifo и socket — аналогично файлы нулевого размера, в который резидентно хранятся адреса структур в операционной системе соответственно именованного канала или сокета, то есть, структур для канального механизма взаимодействия процессов. Создаются взаимодействующими процессами и ими же используются для передачи друг другу информации.

Файл символьной ссылки описан в вопросе 90.

Всё остальное — обычные файлы. Как правило, обычный пользователь работает, конечно же, с обычными файлами.

93. Что такое файлы типов fifo, socket, блокового и символьного устройств?

См. ответ на вопрос 92.

94. ISO 9660 — это что?

ISO 9660 — стандарт, выпущенный Международной организацией по стандартизации, описывающий файловую систему для дисков CD-ROM. Также известен как CDFS (Compact Disc File System).

Файловая система CDFS появилась почти при «царе Горохе» в незапамятные времена в конце 70-х годов вместе с изобретением CD-дисков или чуть позже. Первоначально CD-диски нашли широкое применение для замены грампластинок («виниловых»), то есть, для хранения аудиозаписей. А поскольку на грампластинках сохранялось 10-15 записей на двух сторонах, то аналогичные требования были сформулированы и для CD-дисков — хранить 10-20 записей.

Это применение CD-дисков было исключительно популярным в 80-е годы и в 1988 году был согласован и принят международный стандарт ISO 9660, давший файловой системе CDFS международный статус.

С другой стороны, в 80-е годы наблюдалось широкое шествие в народных массах операционной системы msdos с файловой системой FAT. Эти два процесса между собой тесно взаимодействовали, что привело к тому, что файловая система CDFS была создана существенно совместимой с файловой системой FAT и одновременно в ней появились те же минусы, что наличествовали в FAT. В частности: имена файлов ограничены восемью символами и тремя символами расширения; в именах используются только буквы латинского алфавита; фрагментация файлов не допускается, файл может располагаться только в непрерывной цепочке секторов; имена каталогов должны содержать не более 8 символов; максимальная глубина вложенных каталогов — до 8; максимальный размер файла — 2 Гб и др. ограничения.

Поэтому в 90-е годы с появлением и распространением в широких народных массах на ПЭВМ операционных систем Windows, MAC OS, linux и различных версий unix — стандарт был доработан так, чтобы обеспечить совместимость с файловыми системами этих операционных систем — появился стандарт ISO 9660:1999, в котором для обеспечения совместимости со сложными файловыми системами были использованы «расширения» Romeo, Joliet, HPS, Rock Ridge, El Torito, что обеспечивало совместимость со всеми распространёнными файловыми системами. В частности:

— сняты ограничения на длинные имена файлов (разрешено до 255 символов);

— меньше ограничений на использование символов в именах файлов;

— структура каталогов произвольной вложенности;

— для каждого файла записываются атрибуты:

= код типа файла,

= права доступа к файлу, в том числе поля uid и gid,

= количество жёстких ссылок на файл,

= времена создания, модификации, доступа, изменения атрибутов и др;

— поддерживаются специальные файлы:

= разрежённые файлы,

= символьные ссылки,

= файлы устройств,

= файлы сокетов,

= FIFO-файлы;

— код приложения, в котором был создан файл, т. н. Creator code[en], определяющий, в каком приложении будет открыт данный файл;

— флаги и информация для отображения в файловом менеджере Finder;

— и др.

В настоящее время программы записи/создания CD/DVD, как правило, включают по умолчанию все расширения, а расширение El Torito используется в случае создания live-CD/DVD.

Общая структура файловой системы ISO 9660 по стандарту 1988 года. Диск может быть разбит на логические разделы — «сессии», но дальше рассматривается диск с одним разделом.

Запись осуществляется последовательно, по спирали; сектора по 2048 байт.

Порядок записи информации:

1. Каждый CD-ROM начинается с 16 блоков (неопределенных ISO 9660), эта область может быть использована для размещения загрузчика ОС или для других целей.

2. Дальше один блок **основного описателя тома** — хранит общую информацию о CD-ROM, в нее входит:

— идентификатор системы (32байта);

— идентификатор тома (32байта);

— идентификатор издателя (128 байт);

— идентификатор лица, подготовившего данные (128 байт);

— имена трех файлов, которые могут содержать краткий обзор, авторские права и библиографическая информация;

— ключевые слова: размер логического блока (как правило, 2048, но могут быть 4096, 8192 и т.д.); количество блоков; дата создания; дата окончания срока службы диска;

— описатель **корневого каталога** (номер блока содержащего каталог).

3. Могут быть дополнительные описатели тома, подобные основному.

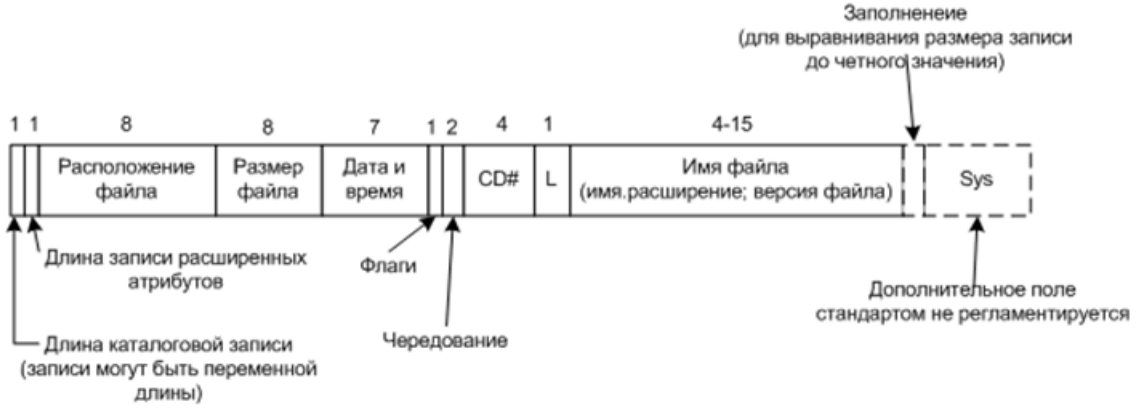

Рис. 27. Каталоговая запись стандарта ISO 9660

Расположение файла — номер начального блока, т.к. блоки располагаются последовательно.

L — длина имени файла в байтах

Имя файла — 8 символов, 3 символа расширения (из-за совместимости с MS-DOS). Имя файла может встречаться несколько раз, но с разными номерами версий.

Sys — поле System use (используется различными ОС для своих расширений).

Порядок каталоговых записей:

— описатель самого каталога (аналог "."");

— ссылка на родительский каталог (аналог "..");

— остальные записи (записи файлов) в алфавитном порядке.

Количество каталоговых записей не ограничено, но ограничено количество вложенности каталогов — 8.

95. Файловая система ext-2/3/4

95.1. Общее

Преданья старины глубокой сообщают нам, что на заре развития Linux использовала файловую систему ОС Minix. Она была довольно стабильна, но оставалась 16-разрядной и, как следствие, имела жёсткое ограничение в 64 Мегабайта на раздел и 14 символов на максимальную длину имени файла. Эти и другие ограничения послужили стимулом к разработке «расширенной файловой системы» (*Extended File System*), которая была представлена в апреле 1992 года. Ext расширила ограничения на размер файла до 2 гигабайт[2] и установила предельную длину имени файла в 255 байт, как в unix-ах.

Тем не менее, оставались ещё проблемы: не было поддержки раздельного доступа, временных меток модификации данных и др. Это послужило причиной для создания следующей версии расширенной файловой системы ext2 (*Second Extended File System*), разработанной в январе 1993 года. В ext2 были также реализованы соответствующие стандарту POSIX списки контроля доступа ACL и расширенные атрибуты файлов.

Граф, описывающий иерархию каталогов файловой системы ext2, представляет собой сеть. Обращаем внимание: именно сеть, а не дерево. Причина такой организации заключается в том, что в файловой системе определено понятие «жёсткой ссылки»: каждое имя файла — это жёсткая ссылка и они могут быть расположены в любом месте файловой системы.

В файловой системе определены:

— имя файла («простое имя файла») — символьное имя длиной до 255 байт, в имени не могут присутствовать символы NULL, слэшь и «;» (точка с запятой»);

— относительное имя — цепочка простых символьных имен всех каталогов, через которые проходит путь от текущего места до данного файла;

— абсолютное имя файла (иногда говорят «полное имя») — цепочка простых символьных имен всех каталогов, через которые проходит путь от **корня файловой системы** до данного файла.

Основные атрибуты файлов (отображаемые командой ls):

— тип файла,

— режим доступа к файлу,

— владелец («хозяин») файла,

— группа хозяина файла,

— количество имён у файла,

— информация о разрешённых операциях (ACL),

— четыре временных штампа: время создания, дата последнего доступа, дата последнего изменения и время последнего удаления файла,

— размер файла,

— адреса занимаемых блоков,

— индекс файла — номер строки индексной таблицы, в которой все эти атрибуты хранятся,

— а также другие атрибуты, командой ls не отображаемые.

Отдельно, в каталоге, хранится имя файла.

95.2. Структура дискового раздела с файловой системой ext2

Всё пространство раздела диска разбивается на блоки фиксированного размера, кратные размеру сектора: 1024, 2048, 4096 или 8192 байт (см. рисунок 28). Размер блока указывается при создании файловой системы на разделе диска. Меньший размер блока позволяет сэкономить место на жёстком диске, но также ограничивает максимальный размер файловой системы. Все блоки имеют порядковые номера. С целью уменьшения фрагментации и количества перемещений головок жёсткого диска при чтении больших массивов данных блоки объединяются в группы блоков.

Суперблок — основной элемент файловой системы ext2. Он содержит общую информацию о файловой системе:

— общее число блоков в файловой системе,

— общее число строк индексной таблицы, строка индексной таблицы — «индексный дескриптор»,

— число свободных блоков в файловой системе,

— число свободных индексных дескрипторов — строк в индексной таблице,

— размер блока файловой системы,

— количество блоков в группе блоков,

— количество индексных дескрипторов в группе блоков,

— размер индексного дескриптора — длина строки индексной таблицы,

— идентификатор файловой системы (магическое число 0xEF53 для семейства файловых систем ext),

— дата последней проверки файловой системы,

— количество произведённых монтирований,

— флаг состояния файловой системы — флаг «чистоты».

Суперблок находится в 1024 байтах от начала раздела.

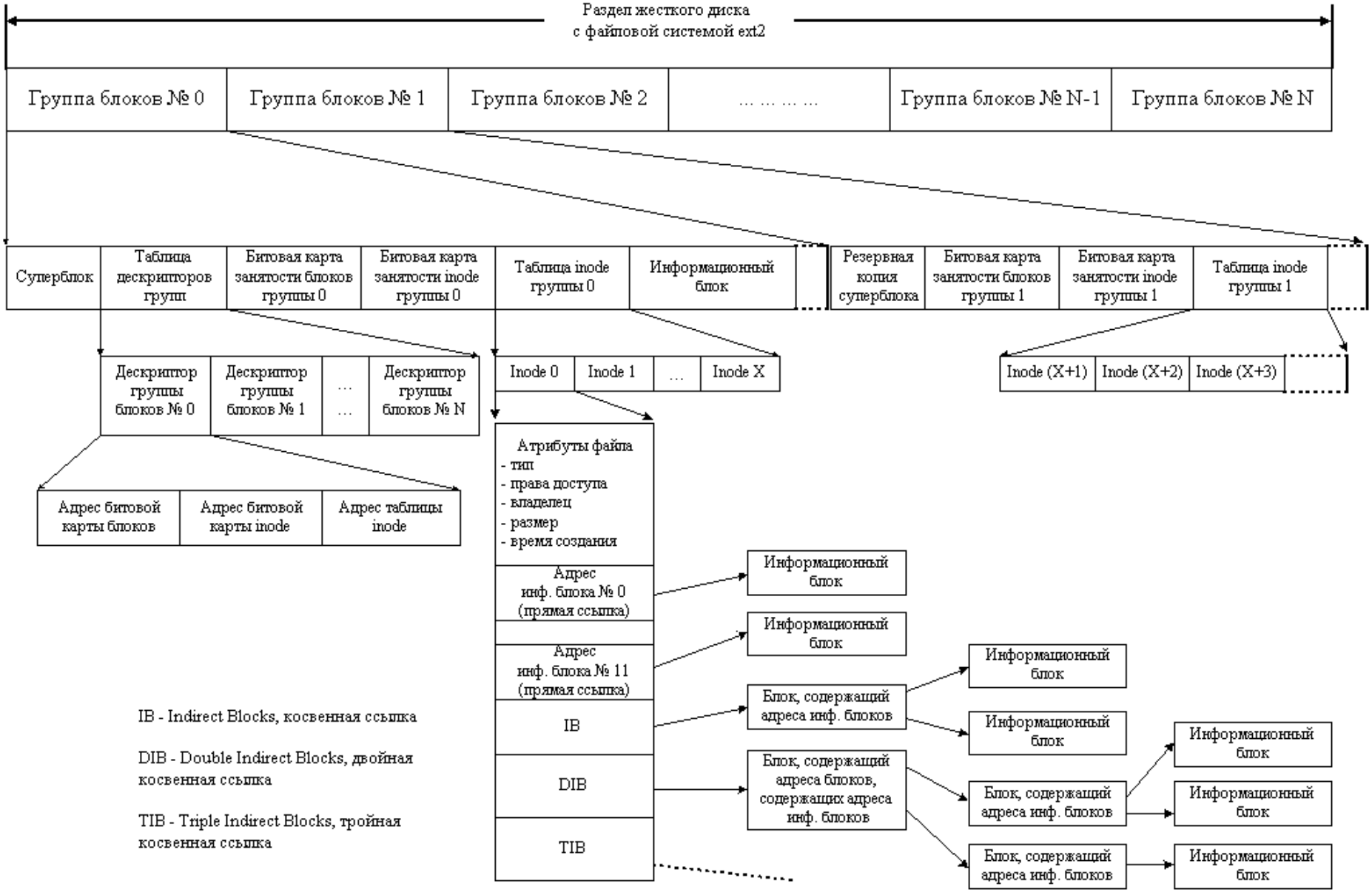

Рис. 28. Структура файловой системы ext-2

В следующем блоке после суперблока располагается **глобальная дескрипторная таблица** (таблица описателей групп блоков) — описание групп блоков, представляющее собой массив (то есть, двумерная таблица с количеством строк, равным количеству групп блоков), содержащий общую информацию обо всех группах блоков раздела.

Далее начинается первая группа блоков.

В первом блоке каждой группы блоков содержится **битовая карта блоков** — структура, каждый бит которой показывает, отведён ли соответствующий ему блок какому-либо файлу. Если бит равен 1, то блок занят.

В следующем блоке содержится **битовая карта индексных дескрипторов**, которая показывает, какие именно индексные дескрипторы заняты, а какие нет.

В следующих нескольких блоках содержится **индексная таблица**, точнее 1/n-ая часть её, где n — количество групп блоков. То есть, вся индексная таблица файловой системы делится поровну между всеми группами блоков и каждая часть помещается в соответствующую группу блоков, сразу после битовых карт.

Ядро Linux пытается равномерно распределить каталоги по группам, а файлы, соответственно, старается по возможности переместить в группу с родительским каталогом.

Все оставшееся место отводится для хранения файлов.

Размер блока файловой системы указывается при форматировании раздела. От его размера косвенно зависят размер группы блоков, количество групп блоков на разделе, размер индексной таблицы и максимальный размер файла.

96. Файловая система ufs

Unix File System (UFS) — файловая система, созданная для операционных систем семейства BSD и используемая в переработанном и дополненном виде на данный момент как основная в операционных системах-потомках (FreeBSD, OpenBSD, NetBSD, DragonFlyBSD).

В ufs используется термин «группа цилиндров» — последовательность смежных цилиндров на hdd (см. вопрос 50). Это обусловлено тем, что ufs возникла давным-давно, в те стародавние времена, когда ЭВМ были большие, а винчестеры маленькие. В смысле, маленькие по объёму дискового пространства, на самом деле, по величине, hdd тоже были большими — размером с письменный стол и весом в сотни килограммов. И в этих маленьких больших hdd использовалась CHS-адресация, отсюда и термин

«группа цилиндров». Каждая группа цилиндров — это, в сущности, маленький логический диск.

При форматировании раздела вначале определяется размер блока файловой системы. Он устанавливается либо по умолчанию, либо указывается в параметрах команды mkfs из ряда 1, 2, 4, 8, 16, 32, 64 кб. В файловых системах ufs размер блока по умолчанию принимался равным 16 кб, а в ufs2 на новых терабайтных hdd (с 4-килобайтными секторами) был увеличен до 32 кб. Существенным отличием блока ufs является использование фрагментов блока для решения проблемы неэффективного использования свободного пространства. Размер фрагмента обычно равен 2 или 4 кб, но есть ограничение: фрагмент может быть равен 1/8, 1/4, 1/2, либо всему блоку, что эквивалентно отказу от функции фрагментирования блока.

Далее вычисляется количество блоков файловой системы.

Затем раздел делится на «группы цилиндров». Размер группы цилиндров в блоках и, соответственно, количество групп цилиндров в разделе, определяется исходя из размеров блока: битовая карта свободных строк в индексной таблице и битовая карта свободных блоков должны занимать либо полный блок, либо место, кратное блоку, либо определённую его часть (половину, четверть, одну восьмую). В начале каждой группы цилиндров резервируется место под вторичный загрузчик. Размер этой загрузочной области в ufs был равен 8 кб (независимо от принятого размера блока), а в UFS2 он стал переменным от 0 до 256 кб. Но помещается вторичный загрузчик только в нулевую группу цилиндров, во всех остальных это место резервируется, но пустует.

Далее после загрузочной области идёт суперблок, затем описатель группы цилиндров, затем две битовые карты, затем часть индексной таблицы.

Нулевая группа цилиндров — особая (см. рисунки 29 и 30): помимо вторичного загрузчика в ней также находится корневой каталог.

В остальных группах цилиндров всё то же самое, только загрузочная область пустая и нет корневого каталога.

Это означает, что структуры данных файловой системы распределены по разделу. В частности, важнейшая структура файловой системы — суперблок, хранится в стольких экземплярах, сколько определено групп цилиндров.

В последних версиях этой файловой системы также может поддерживаться журналируемость.

Рис. 29. Раздел с ufs: вторичный загрузчик есть только в первой группе цилиндров

97. Структура файловой системы ntfs

97.1. Структура

Для ускорения доступа и уменьшения объёма вычислений, сектора раздела объединяются в кластеры — смежные последовательности секторов. Размер кластера определён в диапазоне от 512 байт (1 сектор) до 64 килобайт (128 секторов).

В начале раздела находится загрузочная область (см. рис. 30), в терминологии Windows называемая загрузочной записью раздела (Volume Boot Record). В ней, в самом начале, содержится блок параметров BIOS размером 73 байта — информация о разделе (см. ниже описание этого блока параметров; это аналог суперблока, используемого в unix/linux), а далее до конца загрузочной области идёт код загрузки Windows — вторичный загрузчик. Загрузочная запись занимает обычно 8 КБ (16 первых секторов). Говорят (в Интернете), что загрузочная область дублируется в самом конце раздела. Не проверено.

В определенной области раздела (адрес начала этой области указывается в загрузочной записи) расположена основная системная структура NTFS – главная таблица файлов (Master File Table, MFT). В записях этой таблицы содержится вся информация о расположении файлов на трасделе, а данные небольших файлов хранятся прямо в записях MFT — резидентно.

Начало MFT — первый 16 строк, для надёжности дублируются — их копия (файл $MFTMirr) располагается в середине раздела.

Важной особенностью NTFS является то, что вся информация, как пользовательская, так и системная, хранится в виде файлов. Имена системных файлов начинаются со знака "$". Например, загрузочная запись тома содержится в файле $Boot, а главная таблица файлов – в файле $Mft. Такая организация информации позволяет единообразно работать как с пользовательскими, так и с системными данными на томе.

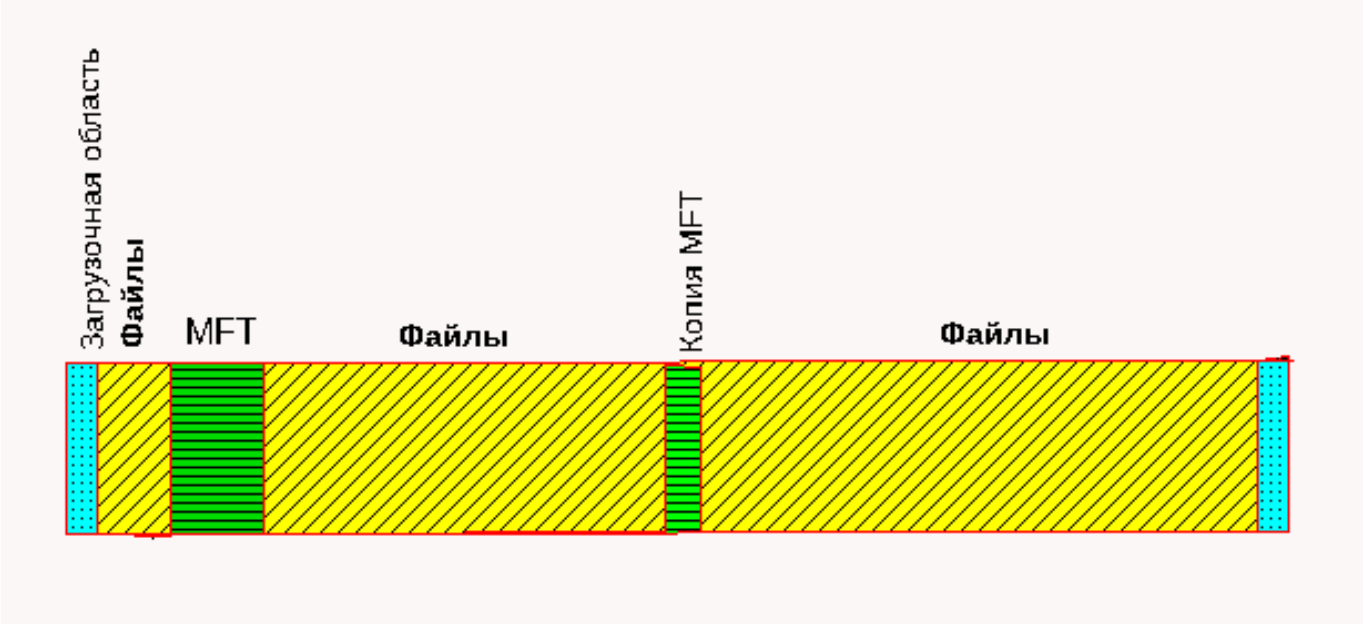

Рис. 30. Структура NTFS

Поскольку MFT является важнейшей системной структурой, к которой при операциях с томом наиболее часто происходят обращения, выгодно хранить файл $Mft в непрерывной области логического диска, чтобы избежать его фрагментации (размещения в разных областях диска), и, следовательно, повысить скорость работы с ним. С этой целью при форматировании тома выделяется непрерывная область, называемая зоной MFT (MFT Zone) размером обычно 12% от объёма раздела. По мере увеличения главной таблицы файлов, файл $Mft расширяется, занимая зарезервированное место в зоне.

Остальное место (88%) на разделе NTFS отводится под файлы – системные и пользовательские.

Но! Если пользователи заполнят своими файлами все предоставленное им 88% пространства раздела, то их файлы начнут создаваться и в зарезервированной MFT Zone. Это означает, что файл $Mft, как и обычные файлы, также может оказаться фрагментированным.

97.2. MFT

Первые шестнадцать элементов главной таблицы файлов MFT зарезервированы для специальных файлов. Пока используются только первые двенадцать элементов. Это скрытые файлы, имена которых расположены в корне раздела. Файлов не видно, но, тем не менее, они существуют. Проверить это можно, попытавшись создать файл с одним из зарезервированных имён в корне раздела.

Список специальных файлов NTFS

0 — $MFT (элемент 0) — Главная таблица файлов. Атрибут данных содержит элементы MFT, а также неиспользуемые растровые атрибуты;

1 — $MFTMirr (элемент 1) — Зеркало (резервная копия) первых шестнадцати элементов MFT;

2 — $LogFile (элемент 2) — Файл журнала, в который записываются все изменения при работе;

3 $Volume (элемент 3) — Атрибут данных $Volume представляет весь раздел. Обращение Win32 по имени «\\.\C:» откроет файл тома на диске C: (предполагается, что диск C: является разделом NTFS), Файл $Volume содержит также имя раздела, информацию о разделе и атрибуты идентификатора объекта;

4 — $AttrDef (элемент 4) — Атрибут данных $AttrDef содержит массив определений атрибутов, который будут использоваться при описании файлов.

```
ypedef struct {
   WCHAR AttributeName[64];
   ULONG AttributeNumber;
   ULONG Unknown[2];
   ULONG Flags;
   ULONGLONG MinimumSize;
   ULONGLONG MaximumSize;
} ATTRIBUTE_DEFINITION, *PATTRIBUTE_DEFINITION;
```

5 — \ (элемент 5) — Корневой каталог файловой системы. Обычно он располагается в пользовательском пространстве раздела, то есть, за пределами 12-процентной зоны;

6 — $Bitmap (элемент 6) — Атрибут данных $Bitmap представляет собой битовую карту кластеров раздела;

7 — $Boot (элемент 7) — Первый сектор $Boot (вторичного загрузчика) является также и первым сектором раздела. Поскольку он используется в самом начале процесса загрузки системы (если раздел является загружаемым), то пространство здесь не нормируется, а хранимые данные не выравниваются по естественным границам. Формат первого сектора можно описать с помощью структуры BOOT_BLOCK.

```
#pragma pack(push, 1)
typedef struct {
   UCHAR Jump[3];
   UCHAR Format[8];
   USHORT BytesPerSector;
   UCHAR SectorsPerCluster;
   USHORT BootSectors;
   UCHAR Mbz1;
   USHORT Mbz2;
   USHORT Reserved1;
   UCHAR MediaType;
   USHORT Mbz3;
   USHORT SectorsPerTrack;
   USHORT NumberOfHeads;
   ULONG PartitionOffset;
   ULONG Reserved2[2];
   ULONGLONG TotalSectors;
   ULONGLONG MftStartLcn;
   ULONGLONG Mft2StartLcn;
   ULONG ClustersPerFileRecord;
   ULONG ClustersPerIndexBlock;
   ULONGLONG VolumeSerialNumber;
   UCHAR Code[0x1AE];
   USHORT BootSignature;
} BOOT_BLOCK, *PBOOT_BLOCK;
#pragma pack(pop)
```

8 — $BadClus (элемент 8) — В атрибуте данных этого файла содержится информация о сбойных кластерах;

9 — $Secure (элемент 9) — Атрибут данных $Secure содержит совместно используемые идентификаторы доступа. $Secure содержит также два индекса;

10 — $UpCase (элемент 10) — Атрибут данных $Upcase содержит эквивалент верхнего регистра всех 65536 символов Unicode. Это нужно

для хранения имён файлов по стандарту POSIX — с различием маленьких и больших букв;

11 — $Extend (элемент 11) — $Extend — это каталог, который содержит специальные файлы, используемые некоторыми дополнительными функциями NTFS. Специальные файлы, хранящиеся в этим каталоге, это: «$ObjId» (поддержка объектных идентификаторов), «$Quota» (поддержка квот), «$Reparse» (данные точек повторной обработки) и «$UsnJrnl» (журнал файловой системы). Начиная с Windows Vista здесь также находится каталог «$RmMetadata» (поддержка транзакций NTFS).

Остальные 4 строки MFT — резерв.

Эти 16 строк — аналог суперблока и в силу их важности для живучести файловой системы, они имеют дубль — файл $MFTMirr в середине раздела.

Далее в MFT идут ещё несколько пустых строк, до 24-ой. Эти строки создаются при форматировании раздела.

А, вот, дальше, начиная с 25-ой строки, строки в MFT создаются динамически при создании файлов пользователем.

Пример строки MFT, определяющей некоторый не очень большой файл, приведён на рисунке 31. В атрибуте «имя_файла» хранится имя файла, несколько атрибутов и ссылка на каталог, в котором файл зафиксирован.

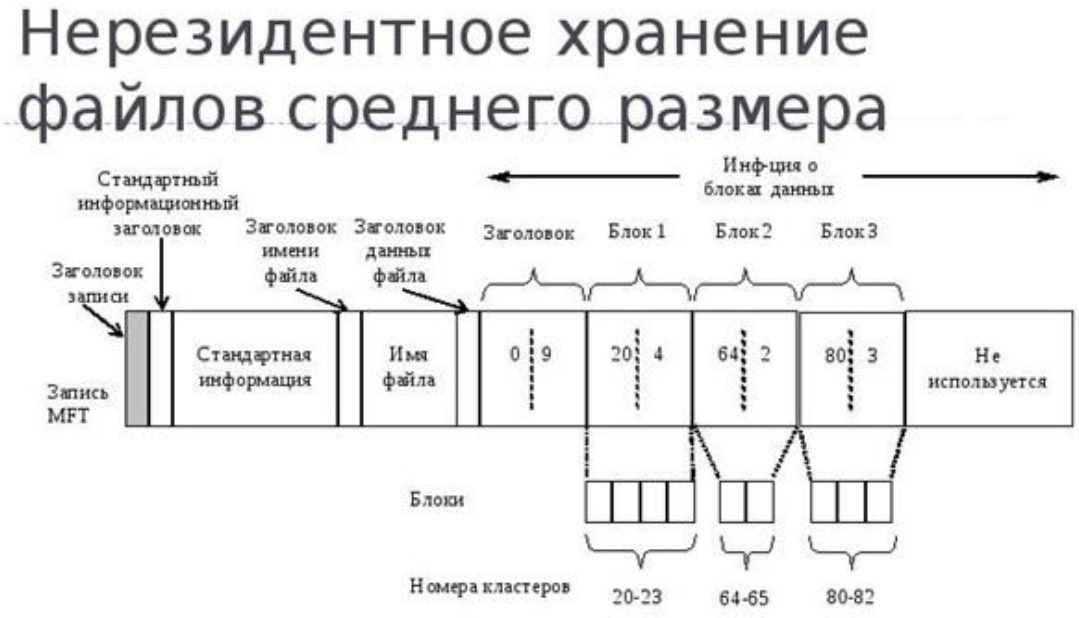

Рис. 31. Строка MFT, определяющая файл среднего размера — от несколько десятков килобайт.

В NTFS есть существенные отличия в хранении малых файлов, средних, больших, сверхбольших и огромных. Также на хранение файла влияет

фрагментированность файла. Например, если файл настолько велик (или сильно фрагментирован, или с файлом часто работали и у него большая история изменений), что его атрибуты не помещаются в одной строке MFT, то эти атрибуты хранятся нерезидентно: из текущей строки MFT заголовок атрибута ссылается на некоторую другую строку MFT, возможно не рядом расположенную, где хранятся данные атрибута. Дополнительных строк у описания файла может быть до 4-х штук. Если и их не хватает, тогда для хранения атрибутов файла привлекаются кластеры на разделе.

На рисунке файл состоит из 3-х серий кластеров: 20-23, 64-65, 80-82. Число таких серий зависит от того, насколько удачно файловая система сумела найти место для хранения файла.

97.3. Каталоги

Небольшие каталоги хранятся резидентно в MFT. Большие каталоги хранятся в кластерах раздела в формате бинарного дерева. Пример хранения резидентно в MFT небольшого каталога показан на рисунке 32. Пример каталога среднего размера (на 9 файлов) показан на рисунке 33. Красные стрелки показывают, как каталожные записи хранятся в алфавитном порядке. Корень B+ дерева находится в атрибуте IndexRoot. Поскольку в строке MFT резидентно девять каталожных записей не помещаются, то для их хранения выделяются два кластера с виртуальными номерами VCN0 и VCN2. Эти кластеры расположены на разделе по адресам, соответственно, 200 и 40.

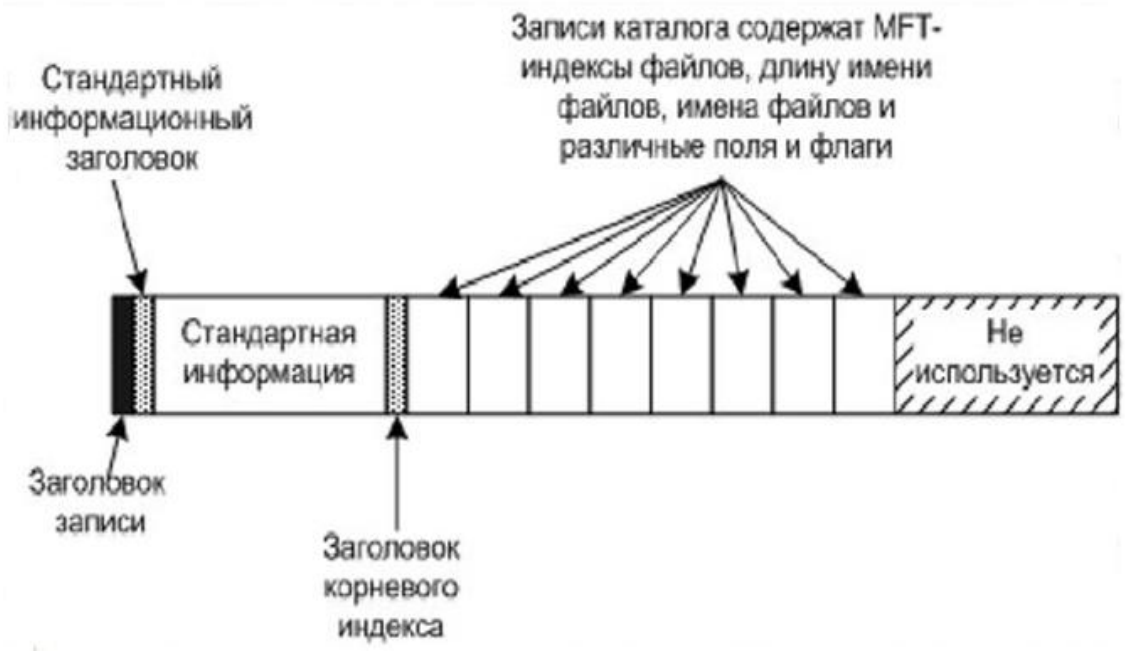

Рис. 32. Строка MFT, содержащая запись о каталоге

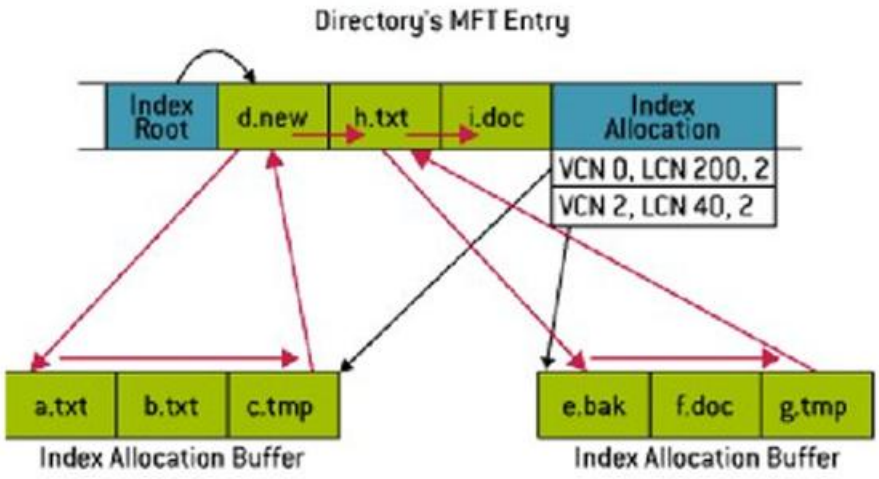

Рис. 33. Хранение в MFT каталога среднего размера.

97.4. Недостатки файловой системы NTFS

а) Авторы полностью согласны с утверждением, файловая система NTFS существенно более надёжна, чем . . . чем что?

Правильно! Чем FAT.

И не более того.

До надёжности файловых систем EXT4, JFS, XFS, reiserfs и тем более ufs — ей как до Альфы Центавра. Это не шутка. Если вы хоть что-то соображаете в теории надёжности, то посмотрите на структуры этих файловых систем и ответьте:

— как может считаться надёжной система, если она свои структуры данных хранит в одном экземпляре?

— во сколько раз различается надёжность хранения данных в одном экземпляре от надёжности распределённого хранения данных?

Журнал, говорите? Так ведь, в EXT4, JFS, XFS, reiserfs и в последних версиях ufs также есть журналы. Причём, их журналы способны работать в режиме journal, про который система журналирования NTFS просто не знает, она работает только в режиме ordered.

б) Запутанность. Это исключительно запутанная система. Складывается впечатление, что её однажды, в самом начале 90-х, сделали специалисты, а потом пришли молодые «временщики» и, не вникая и не разбираясь во всяких там концепциях, принципах, положениях, начали «сопровождать» методом «залепливания дыр», получив в результате «блюдо спагетти», в котором сам чёрт ногу сломит и без . . . не разобраться. Косвен-

ным доказательством этого служит начало разработки файловой системы «нового поколения» ReFS, обратно совместимой с NTFS: похоже «клиент созрел» до состояния, когда сопровождать становится просто тяжко.

Думаете, авторы запугивают? Ну, так попробуйте разобраться в структуре записей MFT о файлах или каталогах. Интернет вам в руки. Крис Касперски разобрался. Его публикации в Интернете доступны. Попробуйте повторить. Как сможете, придёте, поговорим.

в) Пункт б) приводит к тому, что накладных расходов при функционировании файловой системы стало слишком много: и процессорного времени, и памяти. Сама MS не рекомендует использовать NTFS на малых разделах, на которых служебные данные (прежде всего MFT) могут занимать до 25% объёма раздела.

Действительно, сравним:

— ext4, ufs — строка индексной таблицы 256 байт и она не полностью занята, есть место для резидентных файлов,

— NTFS — строка MFT от одного до четырёх килобайт, описание файла может занимать до 4-х строк и причём, очень часто, в строке MFT находится только заголовок атрибута, а сами значения — где-то там в кластерах на разделе.

То есть, в NTFS объём служебной информации о файле в 4-16 и более раз больше, нежели в ext4, ufs. Причём, как правило, информация о файлах хранится в форме «имя=значение», то есть, имя текстом и значение иногда текстом, информация обрабатывается как текстовые строки, что в десятки раз более трудоёмко, нежели обработка чисел.

г) Слабая защита и потенциальная «дырявость» прежде всего из-за реализованной многопоточности файлов — сколько «баз данных» утекло из-за этого в Интернет, целый бизнес на этом построен.

98. Из чего состоит файловая система FAT32?

В начале раздела идут «зарезервированные» сектора — от одного до трёх. В первом из них находится «Блок Параметров BIOS», размером 60 байт — аналог суперблока, и, ниже располагается вторичный загрузчик. Если информация, описывающая раздел не помещается в первом секторе, то задействуются второй и третий зарезервированные сектора.

Далее идут одна за другой две таблицы FAT, в которых строк чуть-чуть больше, чем кластеров на разделе.

Затем идёт корневой каталог.

Далее обычные файлы и каталоги.

99. Команда fsck — назначение и использование.

Команда fsck предназначена для проверки файловых систем и исправления ошибок файловой системы, если они будут обнаружены. Может использоваться для проверки всех файловых систем, которые поддерживаются ядром Linux.

Формат вызова программы следующий (из под root):

fsck [параметры] [файловая система]

Последовательность проверки файловой системы должна быть следующая:

1. Размонтировать файловую систему (!).
2. Запустить fsck для ее проверки.

Например, для проверки файловой системы раздела /dev/hda1 сначала размонтируем его, а потом запустим fsck:

umount /dev/hda1
fsck /dev/hda1

100. Какие символы не могут присутствовать в именах файлов в unix/linux-овых файловых системах и почему?

«/» — слэшь,
NULL — двоичный нуль (0x0),
«;» — точка с запятой,
«*» — звёздочка.
Почему нет? Ну, это же очевидно! Кто не верит — пусть проверит.

101. Что такое суперблок?

В файловых системах ufs, ext и совместимых — суперблок файловой системы или структура, аналогичная суперблоку, это таблица, описывающая всю файловую систему в целом. Размер — 128 байт (см. вопросы 95 и 96).

102. Сколько суперблоков на hdd?

Вопрос немного некорректный. Можно спрашивать, сколько суперблоков в файловой системе. Потому что, суперблок — это элемент файловой системы, а на hdd файловых систем столько, сколько разделов, и в некоторых файловых системах суперблоки присутствуют явно (в unix-овых), а в некоторых неявно (в виндовых: в таковых есть структуры, по назначению выполняющие роль суперблоков, но суперблоками не называющиеся).

Пример: в файловой системе ufs суперблоков столько, сколько в ней определено групп цилиндров. В ext-2/3/4 суперблоков поменьше, поскольку в ней суперблок пишется не в каждую группу блоков, а только в 0, 1, 3, 5, 7, 9, 25, 27, 49, 81, 125, 243, 343, 625, 729 и т. д.

103. Где находится суперблок?

См. ответ на вопрос 102.

104. Какого размера суперблок (байт/кб/мб)?

В ext02 — 128 байт. В ext-4 — 256 байт. В ufs — 1 кб, но занято много меньше половины. В ufs2 — 1 кб, но занято чуть больше половины.

105. Во сколько раз суперблок больше обычного блока файловой системы?

Во-первых, не больше, а меньше.

Во-вторых, обычный блок файловой системы может быть равен 1, 2, 4, 8, 16, 32 и 64 кб — это определяется при форматировании, а суперблок равен — см. ответ на вопрос 104.

106. Что такое битовая карта блоков?

В unix/linux-овых файловых системах в каждой группе_цилиндров / группе_блоков есть по две битовых карты — одна для блоков, другая для строк индексной таблицы. Битовая карта — это битовое поле обычно **размером в один блок**, каждый бит которого соответствует одному блоку в группе блоков, или одной строке в индексной таблице. Если бит = 1, то соответствующий блок или соответствующая строка заняты, если бит = 0, то свободны.

107. Что такое группа блоков?

При форматировании раздела hdd сначала сектора объединяются в блоки (см. ответ на вопрос 79) — более крупные единицы, а затем блоки объединяются в группы блоков — ещё более крупные единицы. То есть, группа блоков — смежная последовательность блоков файловой системы.

Блоки группируются в группы блоков, чтобы:

— повысить надёжность файловой системы за счёт увеличения избыточности,

— увеличить скорость доступа к данным за счёт уменьшения перемещений головок при чтении большого количества последовательных

данных, поскольку позиционирование головок на дорожках — весьма затратная операция.

В первом секторе каждой группы блоков файловой системы располагается суперблок. Во втором секторе — описатель данной группы блоков, дескриптор группы блоков. Затем один блок — битовая карта, показывающая занятость группы блоков файлами. Затем один блок — битовая карта, показывающая занятость индексной таблицы — той её части, что соответствует данной группе блоков. Далее следует блоки, занятые индексной таблицей. Остальные блоки этой группы блоков — блоки данных. Алгоритм выделения блоков пытается выделить блоки данных в той же группе блоков, что и содержащий их индекс.

108. Что такое группа цилиндров?

То же, что и группа блоков, только в файловой системе ufs. В те времена, когда создавалась ufs ещё использовалась на hdd адресация CHS, это и обусловило термин.

109. Что такое индексная таблица?

Индексная таблица, она же таблица индексных дескрипторов, она же таблица **inodes**. Таблица из некоторого, вычисляемого про форматировании раздела, количества строк длиной 128 байт в файловых системах ufs, ext2/3 и совместимых или 256 байт в ufs2, ext4 и совместимых. В каждой строке содержатся атрибуты одного файла. Другое название строки таблицы — inode. Каждая строка, или каждый inode, имеет свой порядковый номер — порядковый номер строки таблицы — О-о! Этот порядковый номер, называемый иногда индексом файла, используется для связи каталога и индексной таблицы. Каталог — это тоже таблица, но об этом почему-то не любят говорить вслух. В первой колонке таблицы каталога как раз и содержится индекс файла, то есть, номер строки индексной таблица с атрибутами файла.

В файловой системе ext-2 индексная таблица выглядит следующим образом (не все поля указаны):

struct ext2_inode
__u16 i_mode — тип файла и права доступа к нему
__u16 i_uid — идентификатор пользователя — хозяина файла
__u16 i_gid — идентификатор группы пользователя — хозяина файла
__u16 i_count — счётчик hardlinks
__u32 i_size — размер в байтах

__u32 i_atime — время последнего доступа к файлу
__u32 i_ctime — время создания файла
__u32 i_mtime — время последней модификации
__u32 i_dtime — время последнего удаления
__u32 i_flags — флаги
__u32 i_blocks — количество блоков, занимаемых файлом
__u32 i_block[EXT2_N_BLOCKS] — адреса информац. блоков (включая все косвенные ссылки)

Тип файла определяют биты 12-15 поля __u16 i_mode:
— 0xC000 — сокет
— 0xA000 — символическая ссылка
— 0x8000 — обычный файл
— 0x6000 — файл блочного устройства
— 0x4000 — каталог
— 0x2000 — файл символьного устройства
— 0x1000 — канал FIFO

110. Что находится в индексной таблице?

См. ответ на вопрос 109.

111. Каков размер индексной таблицы?

Размер индексной таблицы рассчитывается при форматировании раздела и зависит от размера раздела (количества блоков в нём) и задаваемого соотношения количество_блоков/строки_таблицы. По умолчанию это соотношение равно 4:1.

112. Почему индексная таблица называется «индексной»?

Результат кривого перевода. Из названия «индексная таблица» вроде как следует, что в ней хранятся «индексы». На самом деле индексы — это номера строк индексной таблицы, а хранятся в ней атрибуты файлов — см. ответ на вопрос 109. Строки индексной таблицы иногда называют «inode» — индексный узел, совокупность кое-чего, адресуемая индексом.

113. Можно ли создать ссылку на файл, находящийся на hdd другого компьютера?

Можно. Если каталог (или вся файловая система другого компьютера) смонтирована по NFS.

114. Как смонтировать раздел hdd?

Если точно, то монтируется не раздел, а файловая система раздела, и тем не менее, очень часто говорят: «смонтировать раздел». Монтирование файловой системы осуществляется командой:

mount -t <символьное_имя_файловой_системы> <имя_устройства> <точка_монтирования>

где

— символьное_имя_файловой_системы: ufs, ext2, ntfs, xfs, jfs, vfat и т. д.;

— имя_устройства: путь к файлу блокового устройства /dev/sda1 или /dev/sdb2;

— точка монтирования — каталог, к которому подключается монтируемая файловая система.

Пример. Пусть у нас в компьютере один винчестер, тогда он будет называться /dev/sda. Пусть у нас в каталоге mnt есть подкатадлог disk. Мы подключаем к компьютеру флэшку с файловой системой FAT-32. Обычно на флэшках один раздел (это из-за MS, в linux такого ограничения нет). Система видит её как устройство /dev/sdb и раздел на ней /dev/sdb1. Монтируем:

mount -t fat /dev/sdb1 /mnt/disk

Теперь содержимое флэшки видно в каталоге /mnt/disk.

115. Необходимые и достаточные условия монтирования раздела

Устройство, на котором находится раздел, должно опознаваться операционной системой, то есть, в ней должен быть драйвер этого типа устройств.

Если драйвер собран статически (в ядре присутствует), то в каталоге /dev есть файл этого устройства, если, драйвер в модуле, то файл устройства появится при подключении устройства, по этому событию (подключению устройства) ОС подключит драйвер устройства.

На разделе устройства должна быть файловая система.

Файловая система устройства должна операционной системой опознаваться, то есть, в операционной системе должен быть драйвер этой файловой системы.

В соответствующем месте должен существовать каталог — точка монтирования.

Если всё сходится, то тогда и только тогда раздел может быть смонтирован.

116. Можно ли смонтировать каталог?

Можно.

116.1. Локальный (в этой же файловой системе) каталог можно смонтировать в другое место так:

mount --bind <полный_путь_к_каталогу> <точка_монтирования>

Пример 1. Пусть в каталоге /home/student есть подкаталог docs. Тогда после команды

mount --bind /home/student/docs /mnt/sdb1

он становится доступным по пути /mnt/sdb1.

Пример 2. Возможное практическое применение. Предположим, мы, student, являемся админом веб-сервера, рабочий каталог его находится традиционно в /var/www/html. А мы хотим работать с ним непосредственно из своего домашнего каталога. Даём команду (от root`a):

mount --bind /var/www/html /home/student/www

Теперь всё у нас под руками.

116.2. Также, по NFS, как правило, удалённо монтируются каталоги.

117. Что такое блок файловой системы?

Блок файловой системы – это последовательность смежных секторов, размером в 1, 2, 4, 8, 16, 32, 64 сектора.

118. Сколько имён может быть у файла?

В FAT — только одно. В ufs, ext-2/3/4 и аналогичных файловых систем и даже в ntfs — минимум одно, а максимум — сколько надо. А сколько надо — знает пользователь. Ограничений практически нет.

119. Сколько имён может быть у каталога?

Минимум два: «.» и «..» — точка и две_точки. Первое — собственное имя текущего каталога, второе — ссылка на имя данного каталога в вышестоящем (родительском) каталоге.

То есть, в каждом каталоге, даже пустом, первые две строки уже заняты, не забываем, что каталог это таблица.

А максимум — два + <столько_сколько_в_нём_подкаталогов>. То есть, каждый создаваемый подкаталог увеличивает количество имён на единицу.

120. Может ли существовать файл у которого в индексной таблице атрибут «количество имён» равен нулю? То есть, может ли существовать файл без имени?

120.1. Может. Но только временно.

Как проверить? Открываем два окна файлового менеджера — второе для контроля, будем в нём напосмотреть. Создаём файл file.txt. Он, естественно, создастся пустой (0 байт). Открываем его в текстовом редакторе — любом. Можно даже в writer`e. Вводим «Фыва олдж — йцукен». Не забываем сделать перевод строки — как правило, в конце файла должен быть перевод строки, так принято в приличных семьях. Сохраняем файл. **Редактор не закрываем!** Видим в файловых менеджерах, что размер файла изменился, уже не нулевой. В первом файловом менеджере удаляем файл. Видим во втором файловом менеджере, что файл исчез. Нажимаем в редакторе пункт меню «Файл». Видим, что подпункт меню «Сохранить» — серый, то есть, недоступно. И не надо. Нажимаем следующий подпункт «Сохранить как». Вводим имя: «file.txt», то есть, то же самое. Сохраняем. Видим, что в файловых менеджерах файл снова появился.

То есть, в редакторе мы некоторое время работали с файлом, у которого не было имени.

120.2. Не может. Операция «удаление файла» в unix/linux реализуется так:

— удаляется имя из каталога (иногда не совсем удаляется, но это на результат не влияет);

— стирается строка индексной таблицы, на которую указывает индекс файла в каталоге; ясно, что теряется информация о номерах блоков, где хранится файл — всё, мы уже не можем найти эти блоки, то есть, данные файла;

— модифицируются битовые карты — обе, появляются нулики в соответствующих местах;

— модифицируется описатель группы_блоков/группы_цилиндров, в котором был файл;

— модифицируется суперблок.

Результат очевиден: «сказали в морг — значит в морг» и система это тщательно и пунктуально выполняет. Перечисленные выше действия — это единая транзакция, если где-то сбой — будет откат, удаление не выполнено, попробуйте ещё раз.

А теперь выполним пункт 119.1 ещё раз, но дополнительно задействуем терминал. В терминале прежде всего выполняем :

cd <каталог,_где_находится_file.txt>

В терминале даём команду:

ls -i -l

Видим индекс файла file.txt в первой колонке. Записываем индекс файла file.txt на промокашку. Вы не знаете, что такое промокашка? Тогда вы очень молодой человек, у вас всё ещё впереди. В смысле, студенты однажды спросили: «А что такое промокашка?».

Далее выполняем те же действия с файлом, что описаны в пункте 119.1 до конца.

В терминале даём снова команду:

ls -i -l

Снова видим индекс файла file.txt в первой колонке. О-о! Он же другой!

То есть, файловая система при удалении файла реально его удалила, но он остался в буфере файловой системы и в памяти редактора. Буфер не очистился, поскольку файл был открыт и из редактора на него была ссылка. Если бы мы редактор закрыли, не сохраняя файл, то он был бы удалён из буфера, а раз нет, значит нет. Когда мы в редакторе дали команду «Сохранить как», файл был снова записан на диск, но уже в новое место. Совсем в новое, даже строка индексной таблицы была задействована новая.

То есть, в редакторе мы некоторое время работали с данными, не являвшиеся файлом. Помните определение файла? «Файл — это именованное место на диске». А у тех данных, с которыми мы работали в редакторе, места на диске не было. То есть, это был не файл.

УПРАВЛЕНИЕ ПАМЯТЬЮ

121. Что такое страница памяти?

На ЭВМ архитектуры intel страница памяти — это минимальный размер памяти, которым может управлять аппаратный менеджер памяти.

Страница памяти = 4 кб.

Причины появления и использования этого понятия:

а) Память мы считаем байтами. На современных компьютерах её гигабайты, то есть, миллиарды байт. И ЭВМ постоянно приходится эти миллиарды байт считать, пересчитывать, числить, адресовать, решать вопросы с доступом (защитой) к этим байтам и с допустимыми действиями с этими байтами. А, ведь, известно, что скорость работы алгоритмов обработки чисел существенно зависят от величины этих чисел. Если вы когда-нибудь умножали/делили/складывали/вычитали «в столбик», то представляете, какой «многоэтажный» может оказаться этот «столбик». Это первая причина работы с памятью страничками, а не байтами. Объявляя страничку равной 4 кб, мы тем самым уменьшаем величину чисел в 4 000 раз (четыре тысячи) и соответствующий «столбик» уменьшается на несколько этажей.

б) Для того, чтобы определить, что «зя» делать с байтами, а чего «низя», или кому доступны байты, а кому нет, нужно для каждого байта хранить атрибуты принадлежности (процессу, пользователю), допустимых действий с байтом («чтение», «запись», «выполнение» и их комбинации), запоминать, менялся ли байт в процессе исполнения программы и давно ли, и вообще, обращались ли к этому байту хоть как-то и когда и прочие атрибуты. То есть, на каждый байт памяти должно храниться несколько десятков байт сопутствующей (служебной) информации. Невероятно и не может быть, но, увы, таки да. Это вторая причина работы с памятью страничками: атрибуты хранятся на целые странички, размер поля атрибутов примерно 40 байт. То есть, примерно один процент памяти используется не по прямому назначению, а на накладные расходы. Один процент — не так уж плохо.

в) Кроме того, на этой идее «использовать для управления памятью достаточно крупную меру — странички», базируются ещё ряд важных возможностей ОС:

— отслеживание «активности» памяти, то есть, связывания виртуальной памяти процесса с реальной физической памятью, ведь, программы могут выполняться, только находясь в реальной памяти;

— ускорение работы подсистемы свопинга при сбросе / восстановлении процессов в раздел свопа на hdd;

— алгоритмы замещения памяти при нехватке оперативной памяти в системе, когда решается вопрос, у кого память отобрать, а кому добавить и как это сделать так, чтобы не было потом мучительно больно за совершённые ошибки;

— возможность аппаратной поддержки управления памятью процессором: по крайней мере, в intel-овских процессорах есть специальные регистры для хранения адресов таблиц страниц, предназначенные для ускорения работы с памятью;

— защита памяти процессов: проверить доступ к целой странице проще, нежели проверять побайтно;

— и др.

122. Страница памяти vs блок файловой системы

Памяти «всегда не хватает чуть-чуть». Даже если её много, всё равно её мало. Для решения проблемы нехватки памяти была придумана подсистема свопинга (swaping), реализующая способ увеличения доступной оперативной памяти за счёт внешней дисковой (на hdd) памяти, которой, как правило, на два-три порядка больше и которая в такой же пропорции дешевле.

На эту внешнюю память, в область свопинга, часть процессов может быть выгружена временно, как правило, те, которые находятся в состоянии «ожидания» — см. «жизненный цикл» процесса. А потом, при наступлении условий активизации некоторого процесса, находящегося в области свопинга на диске, этот процесс перемещается обратно в оперативную память.

То есть, в подсистеме свопинга реализуется взаимообмен между оперативной памятью и hdd. А учитывая, что современные процессоры достаточно быстрые (процессор с тактовой частотой 3 GHz потенциально способен выполнить до 1 (одного!) миллиона команд в секнду), то этот взаимообмен должен быть также достаточно быстрым.

Hdd — блоковое устройство, то есть обмен с hdd идёт блоками, эти блоки называются секторами. Ранее сектора были по 512 байт, на современных hdd — 4 кб. А в самой ОС, опять же, в целях ускорения работы подсистемы ввода/вывода, сектора объединяются в логические блоки/кластеры.

И вот тут возникает вопрос: какого размера должны быть блоки/кластеры и страницы памяти, чтобы взаимообмен с hdd работал максимально быстро?

Именно этот вопрос и определил размер страницы памяти и наиболее часто используемый размер блока/кластера 4 кб — «. . . чтоб никто не

ушёл обиженным»(С). А в тех случаях, когда размер блока/кластера больше или меньше 4 кб, он устанавливается, по крайней мере, кратным 4 кб, чтобы обмен шёл страничками.

123. Что такое сегмент памяти?

Страница памяти является аппаратной мерой памяти, всегда одинаковой для всей аппаратной платформы.

Сегмент — логическая мера памяти, величина переменная и может меняться в процессе выполнения программы.

Далеко не всегда сущности программ (код, данные, стек, вспомогательные структуры) полностью помещаются в страницу памяти. Требуются более объёмные структуры. Таковой является сегмент. Но с учётом того, что страничную организацию памяти никто не отменяет, размер сегмента всегда кратен странице. Причём, если управление памятью страницами пользовательские процессы практически не замечают, поскольку оно реализуется в значительной степени аппаратно и для процессов «прозрачно», то управление сегментами пользовательский процесс использует: например, адрес почти всегда указывается двумя величинами — имя сегмента (код, данные, стек), смещение в сегменте.

Таким образом, назначение сегментов — хранение и защита однородной информации (кода, данных и т.д.). Всё виртуальное пространство процесса разбито на сегменты. И каждый сегмент имеет имя, размер и другие параметры (уровень привилегий, разрешенные виды обращений, флаги присутствия и т. д.). На некоторых архитектурах (intel, например) управление сегментами поддерживается аппаратно: процессор имеет регистры для хранения таблиц описаний сегментов, что ускоряет обработку обращений к памяти.

124. Сколько сегментов памяти содержится в одной странице памяти?

См. ответ на вопрос 123.

125. Сколько байт/килобайт/мегабайт в странице памяти?

См. ответы на вопросы 121 и 122.

126. Что такое аппаратный менеджер памяти и что он делает?

«Аппаратный менеджер памяти» или «контроллер памяти» — цифровая схема, управляющая потоками данных между вычислительной системой и оперативной памятью. Может представлять собой отдельную микросхему или быть интегрирована в более сложную микросхему, например,

в состав северного моста, микропроцессор или систему на кристалле.

Компьютеры, использующие микропроцессоры Intel до 2009 года традиционно имели контроллер памяти, встроенный в чипсет (серверный мост, MCH), но процессоры Intel Core i7 (а также некоторые процессоры AMD и других производителей) имеют интегрированный контроллер памяти, расположенный на том же кристалле, для уменьшения задержки доступа в память.

Контроллер памяти содержит логические цепи, необходимые для проведения операций чтения и записи в DRAM, с соблюдением всех необходимых задержек, например, между чтением и записью. Поток входящих запросов преобразуется в последовательности DRAM команд, при этом отслеживаются различные конфликты по банкам, шинам и каналам. Для увеличения производительности входящие запросы могут буферизоваться и переупорядочиваться.

Также контроллер памяти выполняет периодическое обновление хранимых в DRAM данных. Без периодических обновлений чипы памяти DRAM постепенно теряли бы информацию, так как конденсаторы, хранящие биты, постепенно разряжаются токами утечки Типичное время надежного хранения информации составляет доли секунды, но не менее 64 миллисекунд согласно стандартам JEDEC SDRAM DDR2 и более новым. На более длительных периодах времени информация сохраняется лишь частично. При повышенной температуре (более 85°C) может потребоваться более частое обновление памяти.

Кроме этого контроллер памяти может управлять режимами питания чипов памяти.

127. Что такое виртуальная память?

Виртуальная память (*virtual memory*) — условная, реально (физически) несуществующая, логическая память.

Виртуа́льная па́мять — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (например, жёстким диском). Для выполняющейся программы данный метод полностью прозрачен и не требует дополнительных усилий со стороны программиста, однако реализация этого метода требует как аппаратной поддержки, так и поддержки со стороны операционной системы.

В системе с виртуальной памятью используемые программами адреса, называемые виртуальными адресами, транслируются в физические ад-

реса в памяти компьютера. Трансляцию виртуальных адресов в физические выполняет аппаратное обеспечение, называемое блоком управления памятью. Для программы основная память выглядит как доступное и непрерывное адресное пространство либо как набор непрерывных сегментов, вне зависимости от наличия у компьютера соответствующего объёма оперативной памяти. Управление виртуальными адресными пространствами, соотнесение физической и виртуальной памяти, а также перемещение фрагментов памяти между основным и вторичным хранилищами выполняет операционная система (см. подкачка страниц).

Применение виртуальной памяти позволяет:

— освободить программиста от необходимости вручную управлять загрузкой частей программы в память и согласовывать использование памяти с другими программами

— предоставлять программам больше памяти, чем физически установлено в системе

— в многозадачных системах изолировать выполняющиеся программы друг от друга путём назначения им непересекающихся адресных пространств (см. защита памяти)

В настоящее время виртуальная память аппаратно поддерживается в большинстве современных процессоров. В то же время в микроконтроллерах и в системах специального назначения, где требуется либо очень быстрая работа, либо есть ограничения на длительность отклика (системы реального времени), виртуальная память используется относительно редко. Также в таких системах реже встречается многозадачность и сложные иерархии памяти.

128. Как адресуется память?

По байтно — это основная адресация.

По словно — слово, это 2, 4, 8 байт; используется для удобства программистов и ускорения доступа к командам.

По странично — пользователю это не видно, см. предыдущие вопросы.

По сегментно — если вы знаете ассемблер, то вы знаете, что это такое.

129. Как выглядит адресное пространство процесса, в котором адресуется память?

Образно: отрезок прямой, поделённый на отрезки-сегменты. В пределах отрезка-сегмента память адресуется смещением от начала отрезка.

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

130. Как транслировать (скомпилировать) программу?

Это делается с помощью компилятора gcc (GNU C Compiler).

gcc — это свободно доступный оптимизирующий компилятор для языков C, C++, Ada 95, а также Objective C и других языков. Его версии применяются для различных реализаций unix/linux (а также VMS, OS/2 и других систем), и позволяют генерировать код для множества архитектур.

gcc может использоваться для компиляции программ в объектные модули и для компоновки полученных модулей в единую исполняемую программу. Компилятор способен анализировать имена файлов, передаваемые ему в качестве аргументов, и определять, какие действия необходимо выполнить. Файлы с именами типа name.c, или name.cc, или name.cpp рассматриваются, как файлы на языке C/C++, а файлы вида name.o считаются объектными (т.е. внутримашинным представлением).

Чтобы откомпилировать исходный код C, находящийся в файле F.c, и создать объектный файл F.o, нужно выполнить команду:

gcc -c <compile-options> F.c

Здесь ключ c говорит, что нужно только получить объектный файл, строка compile-options указывает возможные дополнительные опции компиляции.

Чтобы скомпоновать один или несколько объектных файлов, полученных из исходного кода C — F1.o, F2.o, ... — в единый исполняемый файл F, используется команда:

gcc -o F <link-options> F1.o F2.o ... -l<other-libraries>

Здесь ключ o говорит, что имя исполняемого файла должно быть F, строка link-options означает возможные дополнительные опции компоновки, а строка other-libraries — подключение при компоновке дополнительных разделяемых библиотек.

Можно совместить два этапа обработки — компиляцию и компоновку — в один общий этап с помощью команды:

gcc -o F <compile-and-link-options> F.c F1.c ... -l<other-libraries>

После компоновки будет создан исполняемый файл F, который можно запустить с помощью команды:

./F <arguments>,

то есть, точка-слэшь-имя_программы аргументы_программы
где строка arguments определяет аргументы командной строки вашей программы.

В процессе компоновки очень часто приходится использовать библиотеки. Библиотекой называют набор объектных файлов, сгруппированных в единый файл и проиндексированных. Когда команда компоновки обнаруживает некоторую библиотеку в списке объектных файлов для компоновки, она проверяет, содержат ли уже скомпонованные объектные файлы вызовы для функций, определенных в одном из файлов библиотек. Если такие функции найдены, соответствующие вызовы связываются с кодом объектного файла из библиотеки.

Библиотеки обычно определяются через аргументы вида -llibrary-name. То есть, ключ -l, за которым без пробела следует имя библиотеки. Например, -lm определяет библиотеку libm различных математических функций (sin, cos, arctan, sqrt, и т.д.). Библиотеки должны быть перечислены после исходных или объектных файлов, содержащих вызовы к соответствующим функциям.

Среди множества опций компиляции и компоновки наиболее часто употребляются следующие:

-c

Только компиляция. Из исходных файлов программы создаются объектные файлы в виде name.o. Компоновка не производится.

-Dname=value

Определить имя name в компилируемой программе как значение value. Эффект такой же, как наличие строки

#define name value

в начале программы. Часть `=value' может быть опущена, в этом случае значение по умолчанию равно 1.

-o file-name

Использовать file-name в качестве имени для создаваемого gcc файла (обычно это исполняемый файл).

-llibrary-name

Использовать при компоновке указанную библиотеку.

-g

Поместить в объектный или исполняемый файл отладочную информацию для отладчика gdb. Опция должна быть указана и для компиляции, и для компоновки.

-MM

Вывести заголовочные файлы (но не стандартные заголовочные), используемые в каждом исходном файле, в формате, подходящем для утилиты make. Не создавать объектные или исполняемые файлы.

-pg

Поместить в объектный или исполняемый файл инструкции профилирования для генерации информации, используемой утилитой gprof. Опция должна быть указана и для компиляции, и для компоновки. Профилирование — это процесс измерения продолжительности выполнения отдельных участков вашей программы. Когда вы указываете -pg, полученная исполняемая программа при запуске генерирует файл статистики. Программа gprof на основе этого файла создает расшифровку, указывающую время, затраченное на выполнение каждой функции.

-Wall

Вывод сообщений о всех предупреждениях или ошибках, возникающих во время трансляции программы.

-O1

Устанавливает оптимизацию уровня 1. Оптимизированная трансляции требует несколько больше времени и несколько больше памяти для больших функций. Без указания опций -O цель компилятора состоит в том, чтобы уменьшить стоимость трансляции и выдать ожидаемые результаты при отладке. Операторы независимы: если вы останавливаете программу на контрольной точке между операторами, то можете назначить новое значение любой переменной или поставить счетчик команд на любой другой оператор в функции и получить точно такие результаты, которые вы ожидали от исходного текста. С указанием -O компилятор пробует уменьшить размер кода и время исполнения.

-O2

Устанавливает оптимизацию уровня 2. gcc выполняет почти все поддерживаемые оптимизации, которые не включают уменьшение времени

исполнения за счет увеличения длины кода. Компилятор не выполняет раскрутку циклов или подстановку функций, когда вы указываете -O2. По сравнения с -O эта опция увеличивает как время компиляции, так и эффективность сгенерированного кода.

-O3

Устанавливает оптимизацию уровня 3. -O3 включает все оптимизации, определяемые -O2, а также включает опцию inline-functions.

-O0

Без оптимизации. Если вы используете многочисленные -O опции с номерами или без номеров уровня, действительной является последняя такая опция.

Что такое «хидер»?

Если исходник вашей программы находится в файле proga.c, то, как правило, хидер — это файл proga.h.

ДОПОЛНИТЕЛЬНЫЕ ВОПРОСЫ ПО АДМИНИСТРИРОВАНИЮ

131. Accaunt пользователя — содержание

Описание пользователя начинается с файла /etc/passwd, в нём сохраняется «заголовок» учётной записи (accaunt`a) пользователя — самая основная информация о пользователе [2]. В passwd каждый пользователь описывается отдельной строкой. Каждая строка состоит из отдельных полей, разделённых символом-разделителем — «:».

name:password:user_ID:Group_ID:general_information:home_dir:shell

Продолжение описания пользователя находится в других файлах. Самая нижняя часть описания — настройки профиля пользователя, хранится в домашнем каталоге пользователя в файлах и каталогах, имя которых начинается с точки, «скрытых» файлах и каталогах («скрытых», потому что, имя начинается с точки, по умолчанию такие файлы и каталоги файловые менеджеры не отображают).

132. Группы пользователей — что это и для чего?

Все пользователи в unix/linux входят в группы [2]. Как правило, при создании пользователя, одновременно с формированием учётной карточки пользователя, происходит и формирование учётной карточки «первичной» группы пользователя. Каждый пользователь входит по крайней мере в одну группу — как правило, свою «первичную» группу, но он может быть членом нескольких групп.

Определение. Группа — это объединение пользователей с целью наделения их некоторыми правами, возможно отсутствующими в первичной группе.

Основное назначение групп пользователей — определение прав доступа к различным файлам и каталогам. В unix/linux для каждого файла/каталога существует его владелец и группа "особо допущенных" («свои люди — сочтёмся») к этому файлу/каталогу. При этом владелец файла может задать права доступа к нему (чтение, запись и т.п.) разные для себя, группы "допущенных" и для всех остальных (не входящих в эту группу).

133. Классы пользователей — что это и для чего?

Чтобы задействовать возможность управления классами пользователей, в системе должны быть установлены дополнительные пакеты, реализующие это. При этом строка passwd начинает выглядеть так:

name:pwd:user_ID:Group_ID:general_information:home_dir:shell:class:password_change_time:account_expiration_time

Здесь поле **class** определяет принадлежность пользователя к некоторому login-классу. Для всех пользователей в классе определяются некоторые настройки или ограничения, которые начинают действовать при входе пользователя в систему. Обычно определение класса (настройки класса) устанавливается в файле login.conf. Это может быть locale, процент загрузки процессора программами пользователя, объём занимаемой памяти, количество одновременно открытых файлов и др. То есть, для всех пользователей класса вводятся некоторые ограничения.

То есть, в отличие от group, где пользователь, включаемый в группу, наделяется дополнительными правами, включение пользователя в класс наоборот ограничивает его в правах.

134. Какие бывают пользователи?

В системе могут быть следующие виды пользователей [2], кстати, некоторые не в каждой:

а) суперпользователь (root — Администратор) — это, конечно, человек;

б) системные пользователи (или псевдо-пользователи) — «нелюди» (демоны сервисов);

в) обычные пользователи — люди; они делятся на подгруппы:

в-1) полные пользователи;

в-2) пользователи, «поражённые в правах»:

— почтовые и другие пользователи сервисов;

— анонимные («транзитные»), но система всегда знает, кто это;

— пользователи с «Гостевым входом» — совсем никто и звать никак.

135. Где хранится профиль пользователя?

Профиль пользователя хранится в в домашнем каталоге пользователя в файлах и каталогах, имя которых начинается с точки.

136. Как отличить файлы с персональными настройками пользователя?

Это файлы, имя которых начинается с точки, или файлы с обычным именем, но которые лежат в каталогах, имя которых начинается с точки.

137. Пользователь ftp — как создать.

Пользователь сервиса ftp бывает двух видов:

— с доступом в свой домашний каталог,

— без доступа в свой домашний каталог — анонимный пользователь, логин anonimous, пароль любой.

Первые — это обычные «полные» пользователи системы. Если в системе запущен сервис ftp, то они будут, в том числе, и пользователями ftp.

Вторые создаются из первых специальным способом «поражения в правах» обычных пользователей. Для этого

— в учётной карточке в поле Home dir ставится что-то не существующее или, например, «чёрная дыра» /dev/null;

— в поле Shell указывается какая-нибудь программа-пустышка, например /bin/false или тоже «чёрная дыра».

Если пользователь ftp **не должен быть** одновременно пользователем какого либо другого сервиса, то предпринимаются дополнительные меры по недопущению пользователя к другим сервисам системы. Однако, не у каждого сервиса возможны такие тонкие «по-пользовательские» настройки.

138. Как создать почтового пользователя?

Создаётся обычный «полный» пользователь.

Затем реализуется процедура превращения такого обычного пользователя в ограниченного в правах «почтового» пользователей следующим образом:

— в учётной карточке в поле Home dir ставится что-то не существующее или, например, «чёрная дыра» /dev/null;

— в поле Shell указывается какая-нибудь программа-пустышка, например /bin/false или тоже «чёрная дыра».

В результате мы получаем **зарегистрированного в системе, но «поражённого в правах»** почтового пользователя, который к почте своей имеет доступ, а войти в систему — не может.

139. Что может изменить пользователь в своей учётной записи?

Изменить пользователь в своей учётной записи может только пароль.

Если это позволяет делать Администратор системы. Обычно позволяет. Иногда даже принуждает [2].

140. Дискреционный метод разграничения доступа

Реализуется по умолчанию в unix/linux-овых операционных системах. Заключается в том, что централизованно, «декретно» (законом, определяющим общий порядок) утверждён алгоритм наделения пользователей правами. В unix/linux это означает следующее: при создании файла в строке индексной таблицы помимо всяких разных атрибутов файла также запоминается:

— идентификатор пользователя, создавшего файл, uid хозяина файла;

— идентификатор группы пользователя, создавшего файл, gid группы хозяина файла;

— режим доступа к файлу хозяина файла, членов группы хозяина файла и всех прочих.

А в операционной системе реализован (имеется в наличии и всегда выполняется) алгоритм проверки этих атрибутов при каждой попытке поиметь доступ к файлу. А поскольку почти всё в unix/linux системах имеет отношение к файлам, то этот механизм разграничения доступа действует в целом во всей системе.

141. Как временно удалить пользователя.

Временно удалить — это значит удалить, но с возможностью восстановления, когда надо. То есть, не совсем удалить, а заблокировать доступ в систему.

Самый быстрый, но не совсем правильный и не совсем полный способ [2], заключается в добавлении в начало пароля запрещённых (зарезервированных) в алгоритме MD5 символов. Обычно это символы «!» и «*».

142. Флаги доступа к файлам.

Флаги доступа к файлам являются самым сильным средством разграничения доступа к файлам. Они действуют глобально и одинаково для всех пользователей в системе (в том числе, и root'а), не разделяя их на категории. Единственное отличие владельца файла и root'а от прочих пользователей в том, что они могут убрать флаги с файла (и то не всегда) и только потом уже делать то, что им хочется.

С помощью флагов можно запретить изменять содержимое файла, его название (переименование), его удаление и ещё много чего, то есть

можно наделить файлы некоторыми атрибутами, которые существенно изменяют порядок доступа к файлам.

143. Бит suid

Бит режима доступа к файлу, используется для изменения поведения исполняемых файлов — программ: установленный бит разрешает программе изменять свой «эффективный uid» на идентификатор того пользователя, который является владельцем этого файла. Например, если хозяином файла является пользователь vasja, а запускает программу пользователь koljan, то программ, запустившись, может определить, что её хозяин всё-таки vasja и начнёт работать с правами пользователя vasja, а не пользователя koljan.

Смысл. В linux по умолчанию пользователи работают в своих домашних каталогах, домашние каталоги находятся в каталоге /home. То есть, домашний каталог пользователя vasja есть /home/vasja, а домашний каталог пользователя koljan есть /home/koljan. Причём, пользователь не имеет права «заглядывать» в чужой каталог. Но если koljan некоторым образом заимеет программу пользователя vasja, которая что-то там делает с файлами пользователя vasja и установит ей бит suid, то запустив её, он сможет что-то сделать с файлами пользователя vasja.

144. Бит sticky

Обычно устанавливается на каталогах, которые являются "публичными" (расшаренными). Такие каталоги имеют права доступа, позволяющие всем пользователям создавать там свои файлы и удалять их. Однако, было бы неправильно, что любой другой пользователь мог по ошибке или "из вредности" удалять файлы, к которым он не имеет никакого отношения. Для того, чтобы предотвратить возможность удаления файла "посторонним" пользователем, как раз и служит sticky бит.

145. Режим доступа к файлу

Девять бит rwxrwxrwx, определяющих, что может сделать с файлом хозяин файла, члены группы хозяина файла и все прочие пользователи.

ЛИТЕРАТУРА

1. Чичев А.А., Чекал Е.Г. Операционные системы. Часть 1. Работа с операционной системой : учебно-методическое пособие. — Ульяновск : УлГУ, 2015.

2. Чичев А.А., Чекал Е.Г. Администрирование информационных систем. Часть 1. Общие вопросы : учебно-методическое пособие. — Ульяновск : УлГУ, 2018. — 156 с.

3. Чекал Е.Г., Чичев А.А. Надёжность информационных систем. Часть 1 : учебное пособие. — Ульяновск : УлГУ, 2012.

4. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. — 2-е изд. — Спб. : Питер, 2009. — 669 с. : ил.

5. Олифер В., Олифер Н. Компьютерные сети. Принципы, технологии, протоколы : учебник. — 5-е изд. — СПб. : Питер, 2016. — 992 с. : ил. — (Серия «Учебник для вузов»).

ISBN: 9785496019675

6. Таненбаум Э., Уэзеролл Д. Компьютерные сети. — 5-е изд. — Спб. : Питер, 2016.

7. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация. — 3-е изд. — Спб. : Питер, 2007. — 704 с. : ил.

8. Куроуз Д., Росс К. Компьютерные сети: Нисходящий подход. — 6-е изд. — М. : Изд-во «Э», 2016. — 912 с. : ил.

ISBN 978-5-699-78090-7.

9. Куроуз Д., Росс К. Компьютерные сети. Настольная книга системного администратора. — 6-е изд. — М. : Эксмо, 2016. — 912 с. : ил.

ISBN 978-5-699-94358-6, 0132856204

10. Робачевский А. Интернет изнутри. Экосистема глобальной сети. — 2-е изд. — М. : Альпина Паблишер, 2015. — 223 с. : ил.

ISBN 978-5-9614-4803-0, 978-5-9614-5882-4

11. Уэнделл О. Официальное руководство Cisco по подготовке к сертификационным экзаменам CCNA ICND2 200-101: маршрутизация и коммутация, акад / пер. с англ. — М. : ООО «И.Д. Вильямс», 2015. — 736 с. : ил.

ISBN 978-5-8459-1907-6

12. Хант К. TCP/IP. Сетевое администрирование / пер. с англ. — 3-е изд. — СПб. : Символ-Плюс, 2007. — 816 с., ил.

ISBN-10: 5-93286-056-1, ISBN-13: 978-5-93286-056-4

13. Ногл М. TCP/IP : иллюстрированный учебник. — М. : ДМК Пресс, 2001. — 480 с. : ил.

ISBN 5-94074-044-8

14. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. : ил. — (Серия «Классика computer science»).

ISBN 978-5-496-01395-6

15. Таненбаум Э., Остин Т. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2018. — 816 с. : ил.

ISBN 978-5-496-00337-7

16. URL: //top500.org. — дата доступа 07.12.19.

17. Чичев А.А., Чекал Е.Г. Стилевые конфликты и оценка стиля программирования. Сб. трудов XI международной конференции «Информационные технологии в образовании». Ч. 2. — М. : МИФИ, 2001. — С. 151-154.

18. Чичев А.А., Чекал Е.Г. Подготовка ИТ-специалистов в университете. Ученые записки УлГУ, серия «Математика и информационные технологии». – Вып. 1(4). — Ульяновск : УлГУ, 2012. – 226 с. — С. 278-286.

19. Айзерман М.А. Логика, автоматы, алгоритмы / М.А. Айзерман, Л.А. Гусев, Л.И. Розоноэр, И.М. Смирнова, А.А. Таль. — М., 1963.

Учебное издание

А.А. Чичев, Е.Г. Чекал

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Часть 4

Учебное пособие

Директор Издательского центра *Т.В. Максимова*
Подготовка оригинал-макета *М.А. Водениной*

Издается в авторской редакции

Подписано в печать 19.12.2019.
Формат 60×84/16. Усл. печ. л. 7,5.
Тираж 100 экз. Заказ № 186 /

Оригинал-макет подготовлен и тираж отпечатан в Издательском центре
Ульяновского государственного университета
432017, г. Ульяновск, ул. Л. Толстого, 42