МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
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

АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Часть 1. ОБЩИЕ ВОПРОСЫ

Учебно-методическое пособие

Ульяновск
2018

УДК 004.4'6(075.8)
ББК 32.972.1я73
Ч-72

Печатается по решению Ученого совета
факультета математики, информационных и авиационных технологий
Ульяновского государственного университета
(протокол № 5/18 от 22.06.2018)

Рецензенты:

зав. кафедрой информатики ФГБОУ ВО «УлГПУ им. И. Н. Ульянова»,
д.пед.н., к.т.н., профессор *В.Г. Шубович*;

зав. кафедрой ТТС УлГУ, д.т.н., профессор *А.А. Смагин*

Чичев А.А.

Ч-72 **Администрирование информационных систем. Часть 1.** Общие вопросы : учебно-методическое пособие / А.А. Чичев, Е.Г. Чекал. – Ульяновск : УлГУ, 2018. – 156 с.

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

В первой части пособия рассмотрены общие вопросы администрирования: приведены краткие сведения об управлении пользователями и группами пользователей в вычислительных системах, управлении файлами и разграничении доступа к файлам, организации сетей и управлении сервисами в сетях, управлении базами данных.

Специальные вопросы администрирования конкретных информационных систем, в частности ИС «1С», достаточно подробно описаны в другой учебной литературе и в данном пособии не рассматриваются.

Пособие предназначено для практического руководства при проведении преподавателями лекционных и лабораторных занятий со студентами указанных направлений и специальностей всех форм обучения.

УДК 004.4'6(075.8)
ББК 32.972.1я73

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-3.x) превысил 16 (шестнадцать!) 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 90% 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 последнее десятилетие в крупнейших ИТ-корпорациях мира появились подразделения, специализирующиеся на разработке 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?

Поэтому в данном пособии рассматривается в основном вопросы администрирования в операционных системах Unix (и прежде всего – Linux) – общие вопросы администрирования вычислительных систем, вопросы, касающиеся всего программного обеспечения, в том числе и информационных систем. Специальные вопросы администрирования конкретных информационных систем и прежде всего, конечно, 1С (установка, конфигурирование, сопровождение), достаточно подробно рассмотрены в других пособиях (например, [13]).

В первой части пособия рассмотрены сведения об администрировании пользователей и групп пользователей в операционных системах Unix/ Linux, управлении файлами, об организации сетей и управлении сетевыми сервисами, управлении базами данных.

Во второй части пособия приведены методические указания к лабораторным работам по этим темам.

Обращаем внимание, что если говорится об администрировании систем, то очевидно, что речь идёт об администрировании серверных решений.

Первый раздел включает:

- определение операционной системы, понятие оболочки операционной системы (в том числе, графической) и операционной среды, и их свойства,
- введение в системное программирование и технологические особенности системного программирования,
- основные сведения об информационных системах, в том числе, корпоративного уровня и их архитектуре;
- управление пользователями и группами пользователей;
- управление файлами и разграничение доступа к ним;
- эталонная модель взаимодействия открытых систем (ЭМВОС), сетевые технологии и стандарты на них, стеки сетевых протоколов (SMB, IPX/SPX, TSP/IP, AppleTalk, SNA), именование сетевых объектов на различных уровнях ЭМВОС,
- сервисы и администрирование сервисов,
- администрирование Баз Данных.

В втором разделе представлены:

- методические указания к лабораторным работам по перечисленным темам (установке, администрированию и эксплуатации информационных систем).

Пособие предназначено для практического руководства при проведении преподавателями лабораторных занятий и выполнении заданий студентами указанных специальностей всех форм обучения.

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

1

ОСНОВНЫЕ ПОНЯТИЯ ОПЕРАЦИОННЫХ СИСТЕМ

1.1. Общие сведения о вычислительных системах

1.1.1. Виды вычислительных систем

ПЭВМ – персональная ЭВМ: desktop, ноутбук, нетбук, iPad, iFon или просто смартфон. То есть, это то, чем мы пользуемся повседневно. Сюда же относятся не сильно большие серверы.

Встраиваемые ЭВМ – embedded, blade, ТЭЗ (Типовой Элемент Замены – старое советское название встраиваемой ЭВМ)). Пример: встраиваемые ЭВМ стиральных машин, холодильников, мультиварок, автомобилей, самолётов, автоматических межпланетных станций и другой бытовой и небытовой техники. Этот тип ЭВМ тоже широко используется.

СуперЭВМ – это большая редкость. Они бывают «классические», то есть, собранные из компонентов, специально разработанных для каждой конкретной суперЭВМ, и кластеры – собранные из обычных ПЭВМ или встраиваемых ЭВМ. Сюда же можно отнести и большие серверы, которые, вроде как ещё и не «супер», но уже архитектурно построены иначе, нежели ПЭВМ. В последнее десятилетие появился новый вид суперЭВМ – распределённые суперЭВМ на основе grid-сетей или «облаков», см. подробнее в [1].

1.1.2. Обеспечения вычислительных систем

Любая вычислительная система (или просто ЭВМ) состоит как минимум из двух «обеспечений» [1]:

1) технического обеспечения – аппаратура;

2) программного обеспечения – системное и прикладное ПО.

Программное обеспечение вычислительных систем обычно распространяется в виде дистрибутивов – комплектов ПО (ОС + системные + прикладные программы), который, как правило, именуется по имени (и даже версии) операционной системы, входящей в этот набор ПО, но (!!!) ОС –

только малая часть (всего лишь одна) из программ, входящих в дистрибутив.

1.2. Определение операционной системы

Операционная система это программа, которая выполняет функции управления вычислениями (вычислительными процессами) в компьютере, распределяет ресурсы вычислительной системы между различными вычислительными процессами и образует ту программную среду, в которой выполняются прикладные программы пользователей.

Определение. Операционная система – это (одна) программа, которая выполняется на вычислительной системе (почти) с момента включения (вычислительной системы) и до её выключения и которая выполняет следующие основные функции [1]:

1) предоставляет процессам «абстрактную» вычислительную машину,

2) управляет ресурсами (доступом к ресурсам),

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

4) учитывает процессы, ресурсы и пользователей.

1.3. Оболочки операционной системы

1.3.1. Определение оболочки

Оболочка операционной системы (*shell* – «оболочка») – это **программа**, реализующая функцию интерпретации команд операционной системы, обеспечивающая интерфейс для взаимодействия пользователя с функциями системы – тот самый «интерфейс человек-ЭВМ». Программа-оболочка необходима, поскольку напрямую (прямо с клавиатуры или мышкой) вызвать функцию операционной системы, передать ей параметры и заставить что-то сделать никак нельзя. А внутри самой (программы) ОС организовать интерфейс с пользователем невозможно.

В общем случае, различают оболочки с двумя типами интерфейса для взаимодействия с пользователем: текстовый пользовательский интерфейс (TUI – «командная строка») и графический пользовательский интерфейс (GUI). Графические оболочки требуют для своей работы всегда гораздо большей вычислительной мощности (в десятки раз) по сравнению с текстовыми командными оболочками.

Оболочку с текстовым пользовательским интерфейсом называют также командным интерпретатором – shell, который используют для обеспече-

ния интерфейса командной строки с операционными системами. Командой оболочки («командой в Linux») является либо имя некоторой утилиты (то есть, программы!), либо оператор языка shell [1]. Оболочки командной строки достаточно сложны в использовании, требуют запоминания достаточно большого числа команд и их параметров и потому пользоваться ими могут «не только лишь все», однако обладают важной особенностью: они предоставляют практически полный доступ к вычислительной системе и поэтому используются специалистами.

Графические оболочки позволяют существенно упростить интерфейс человек-ЭВМ настолько, что пользоваться вычислительными системами могут «все», в том числе, пользователи-непрофессионалы. Однако, они существенно ограничивают возможности человека по управлению вычислительной системой и при этом резко (на порядки) увеличивается и объём дистрибутива, и требуемая вычислительная мощность системы.

Обратите внимание: программы-оболочки – это всегда **отдельные программы** (отдельные от операционной системы) и они **никогда** не входят в состав ОС.

1.4. Введение в системное программирование

1.4.1. Определение

Системное программирование – это процесс разработки системных программ, управляющих и обслуживающих, то есть, операционной системы и утилит.

Совокупность системного ПО – это весьма сложная, существенно иерархически упорядоченная и тесно взаимодействующая совокупность операционной системы и большого набора утилит.

ОС Linux почти вся написана на C и совсем чуть-чуть на ассемблере. И, следовательно, системное программирование – это в основном программирование на C.

Такое положение (что системное программирование – это программирование на C) сложилось в основном за последние два десятилетия и связано с распространением Unix и Unix-подобных ОС, прежде всего Linux. Ранее, в 60/70/80-ые годы положение было иным: операционные системы писались почти всегда на ассемблере соответствующего процессора и, соответственно, под системным программированием понимали разработку программ на этом языке.

1.4.2. Технологические особенности системного программирования

Основные инструменты системного программиста:

- текстовый редактор – необходим для ввода исходных текстов и другой документации;

- транслятор – необходим для преобразования исходного текста в команды ЭВМ;

- отладчик – необходим для отладки программы;

- другие вспомогательные средства, как то: профилировщики текста, трассировщики выполнения, трассировщики системных вызовов и др.

Как правило, в качестве транслятора в настоящее время используется gcc (GNU C Compieler – см. [1]). IDE (интегрированные среды разработки) используются нечасто и, если используются то: а) для разработки системного ПО (не операционной системы), б) только те IDE, которые позволяют разработку консольных программ. Эти ограничения обусловлены тем, что во-первых, для написания исходных текстов модулей ОС используется почти всегда только обычный текстовый редактор и весьма специфические средства отладки, а во-вторых, системное ПО для Unix/Linux – это консольные программы, именно они чаще всего и являются «командами» Unix/ Linux.

Технология разработки системных утилит выглядит следующим образом:

1) в текстовом редакторе пишется исходник программы, некоторые файлы file.h и file.c (возможно даже несколько файлов); файлы помешаются в некоторый каталог ~/myproga;

2) с помощью текстового редактора создаётся make-файл в каталоге myproga, то есть, в make-файл пишутся строки, которые обеспечат трансляцию и последующую сборку разрабатываемой программы;

3) make-файл запускается, тем самым, транслируется и собирается разрабатываемая программа; загрузочный файл программы должен появиться в этом же каталоге /myproga;

4) запускается программа и наблюдаются результаты плодотворной работы;

5) если нужные результаты не наблюдаются, то идет возврат на пункт 1, исправляются ошибки, и продолжается работа дальше.

В итоге должна получиться некоторая (системная) программа (утилита), с помощью которой можно в дальнейшем выполнять нужную обработку данных.

Если мы хотим, чтобы наши усилия не канули в лету, а начали распространяться в составе некоторого дистрибутива Linux, то необходимо дополнительно сделать следующее:

- написать документацию по созданной программе – как минимум, man, а если программа достаточно сложная, то, кроме того, более подробную эксплуатационную документацию;

- определить зависимости нашей программы, то есть, выявить те библиотеки или программы, которые должны быть предварительно установлены в системе, для того, чтобы наша программа заработала правильно;

- создать инсталлер нашей программы, который будет обеспечивать проверку зависимостей, и если нужно, то инициировать установку отсутствующего, производить установку программы, конфигурационных файлов, документации в нужные каталоги системы;

- оформить нашу разработку в форме пакета нужного формата (.rpm, .deb, .tgz, .pet или какого иного, используемого в нужном нам дистрибутиве): причём, пакеты должны быть двух видов: tarboll – пакет только с исполняемыми файлами, без исходников, и пакет с исходными текстами программы;

- присоединиться к группе разработчиков соответствующего дистрибутива и положить созданный шедевр в репозитарий этого дистрибутива.

Естественно, предполагается, что наша разработка будет относиться к OpenSource и распространяться по какой-либо свободной лицензии.

1.4.3. Транслятор языка C

Язык программирования C был создан Д.Ритчи в 1972 г. специально для написания операционной системы Unix, и с тех пор и "каноническая" ОС Unix, и все ее клоны пишутся на языке C. Поэтому во всех версиях Unix и Unix-подобных систем транслятор языка C входит в комплект поставки системы.

Одним из первых программных продуктов, созданных в рамках проекта GNU, также явился транслятор языка C с открытым кодом. Этот транслятор включается в поставку всех версий ОС Linux.

Транслятор языка C выполняет как собственно трансляцию – перевод исходного текста на машинный язык, – результатом чего является объектный модуль, так и редактирование связей – сборку из нескольких объектных модулей (в том числе, и библиотечных) исполняемого модуля.

Файлы с исходными текстами программ на языке C должны иметь расширение .c, например: hello.c. Результатом трансляции является файл, со-

держащий объектный модуль, его имя совпадает с именем исходного модуля, а расширение – .o, например: hello.o. Для файла, содержащего исполняемый модуль стандартного расширения не существует. При трансляции программы, состоящей из единственного исходного модуля, объектный модуль автоматически удаляется после создания транслятором исполняемого модуля.

Общий формат команды вызова транслятора имеет следующий вид:

gcc [опции] [выходной_файл] файл1 [файл2 :]

У транслятора gcc очень много опций и наиболее часто употребляемые опции следующие:

-c	Подавляет фазу редактирования связей, создает объектный модуль для каждого исходного модуля из перечисленных в параметрах вызова. Выходной_файл с этой опцией не задается. Опция может применяться вместе с опцией -I
-o	Компиляция и редактирование связей. Создает объектный модуль для каждого исходного модуля из перечисленных в параметрах вызова и имеющих расширение .c. Файлы .c рассматриваются как исходные модули и компилируются; файлы, имеющие расширение .o, рассматриваются как объектные модули и подключаются при редактировании связей. Параметр выходной_файл задает имя файла исполняемого модуля. Опция может применяться вместе с опциями -L, -l, -I.
-Lкаталог	Добавить каталог в список каталогов, которые содержат объектные библиотечные модули.
-lбиблиотека	При трансляции подключить модули из библиотеки.
-lбиблиотека	При редактировании связей подключить модули из библиотеки.
-Iкаталог	Искать файлы-хидеры (#include), имена которых не начинаются с «/», сначала в указанном каталоге, а лишь затем – в стандартных каталогах для файлов-хидеров.
-E	Выполнить обработку указанных исходных модулей только препроцессором, результат направляется в стандартный вывод. Выходной_файл с этой опцией не задается. Опция может применяться вместе с опцией -I.
-w	Подавить выдачу предупреждающих сообщений.
--save-temps	Сохранить все временные файлы, это бывает очень интересно. Обратите внимание: у этой опции два дефиса (!).

Примеры:

gcc hello.c

Компиляция исходного модуля hello.c с выдачей сообщений об ошибках на стандартный вывод. Файл объектного модуля не создается. Исполняемый файл – a.out.

gcc -c hello.c

Компиляция исходного модуля hello.c с выдачей сообщений об ошибках на стандартный вывод. При успешной компиляции объектный модуль записывается в файл hello.o.

*gcc -o hello hello.*o

Редактирование связей для объектного модуля hello.o, исполняемый модуль записывается в файл hello.

gcc -o hello hello.o hello1.c

Создание исполняемого модуля в файле hello из объектного модуля hello.o и модуля hello1.c (последний модуль является исходным, он предварительно компилируется.

gcc -o hello hello.o hello1.o -l hellolib

Создание исполняемого модуля в файле hello из объектных модулей hello.o и hello1.o с подключением объектных модулей из библиотеки hellolib.

gcc -o hello --save-temps hello.c

Сохраняются все временные файлы. Можно посмотреть, во что преобразуется программа.

2

ОСНОВНЫЕ ПОНЯТИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

2.1. Определение информационных систем

2.1.1. Определение системы

Система – любой объект, который одновременно рассматривается и как единое целое, и как объединенная в интересах достижения поставленных целей совокупность разнородных элементов.

С точки зрения внутреннего содержания, система – это множество объектов, возможно не одинаковых, с определёнными на них отношениями. Системы значительно отличаются между собой как по составу, так и по главным целям.

2.1.2. Определение информационной системы

Определение абстрактное краткое: **Информационная система (ИС)** – это система, предназначенная для сбора, обработки и распространения информации, целью функционирования которой является информационное обслуживание или обеспечение основной деятельности вышестоящей надсистемы системы информационного обмена.

Определение абстрактное более полное: **Информационная система** – это взаимосвязанная совокупность элементов ввода, обработки, переработки, хранения, поиска, вывода и распространения информации, цель функционирования которой состоит в информационном обеспечении эффективной деятельности организационной системы, подсистемой которой она является.

Следствие из этого определения: Поскольку ИС определена как система, то она естественно обладает основными свойствами систем, такими как иерархичность, централизация и децентрализация, целостность и независимость.

Определение конструктивное: **Информационная система** – это взаимосвязанная совокупность **средств, методов и персонала**, используемых для ввода, хранения, обработки и выдачи **информации** в интересах **достижения поставленной цели**.

В этом определении конкретно указывается состав ИС (средства – аппаратная часть, методы – программная часть, персонал – люди, которые её используют и обслуживают), что является выходной продукцией (информация) и для кого она предназначена – для людей, принимающих решения. То есть, здесь ИС определяется как человеко-компьютерная система обработки информации, причём «намекается» на то, как её построить.

Определение по 24-ФЗ от 20.02.95: **Информационная система** – организационно упорядоченная совокупность документов (массивов документов) и информационных технологий, в том числе с использованием средств вычислительной техники и связи, реализующих информационные процессы. А информационные технологии (по этому же ФЗ) – это методы и способы, использующие компьютерные программно-технические средства, информационные процессы и операции, предназначенные для достижения поставленных целей. Тоже хорошее определение, но не столь точно указывающее состав.

2.1.3. Обеспечения ИС

Техническое (аппаратное) обеспечение – ну, это понятно, – вся совокупность оборудования, на котором ИС функционирует, а также соответствующая документация на эти средства и технические процессы.

Информационное обеспечение – совокупность единой системы классификации и кодирования информации, унифицированных систем документации, схем информационных потоков, циркулирующих в организации, а также методология построения баз данных.

Математическое обеспечение – совокупность средств моделирования процессов управления, постановок типовых задач управления, методы математического программирования, математической статистики, теории массового обслуживания и других прикладных математических методов.

Программное обеспечение – совокупность алгоритмов и программ (общесистемных и специальных, в том числе, самой ИС) и технической документации для реализации целей и задач информационной системы, а также нормального функционирования комплекса технических средств.

Организационное обеспечение – совокупность методов и средств, регламентирующих взаимодействие работников с техническими средствами и между собой в процессе разработки и эксплуатации информационной системы.

Правовое обеспечение – совокупность правовых норм, определяющих создание, юридический статус и функционирование информационных

систем, регламентирующих порядок получения, преобразования и использования информации.

Обеспечение безопасности – методы и средства защиты элементов ИС и информации от случайных или преднамеренных воздействий естественного или искусственного характера, способных нанести ущерб владельцам и пользователям информации и поддерживающей её структуре.

Также ещё три определения, связанные с доступом в систему:

Аутентификация – метод независимого от источника информации установления подлинности информации на основе проверки подлинности её внутренней структуры («*это тот, кем назвался*?»).

Авторизация – в информационных технологиях это предоставление определённых полномочий лицу или группе лиц на выполнение некоторых действий в системе обработки данных («*имеет ли некто право выполнять данную деятельность*?»), то есть, определяются права доступа к ресурсам.

Идентификация – это метод сравнения предметов или лиц по их характеристикам, путём опознавания по предметам или документам, и определения полномочий, связанных с доступом лиц в помещения, к работам и документам и т. д. («*это тот, кем назвался и имеет право выполнять данную деятельность*?»).

2.1.4. Состав и структура ИС

Функциональные задачи ИС (функции) объединяются в модули в соответствии с основными бизнес–процессами: управление производством, управление логистикой, управление трудовыми ресурсами, управление финансами. Интеграция модулей отражает их информационную взаимосвязь: модуль управления производством непосредственно связан с модулем управления логистикой (по входу и выходу), а оба модуля связаны с модулем управления трудовыми ресурсами. Все модули имеют информационную связь с модулем управления финансами (рис. 1).

Модуль управления производством основывается на разнообразных нормативах, определяемых особенностями конструкторских спецификаций и технологических операций. Этот модуль позволяет организовать и спрогнозировать выпуск продукции на планируемый период; учесть возможные изменения в свойствах продукции и соответственно в производственном процессе изготовления продукции, является базой для разработки производственного плана.

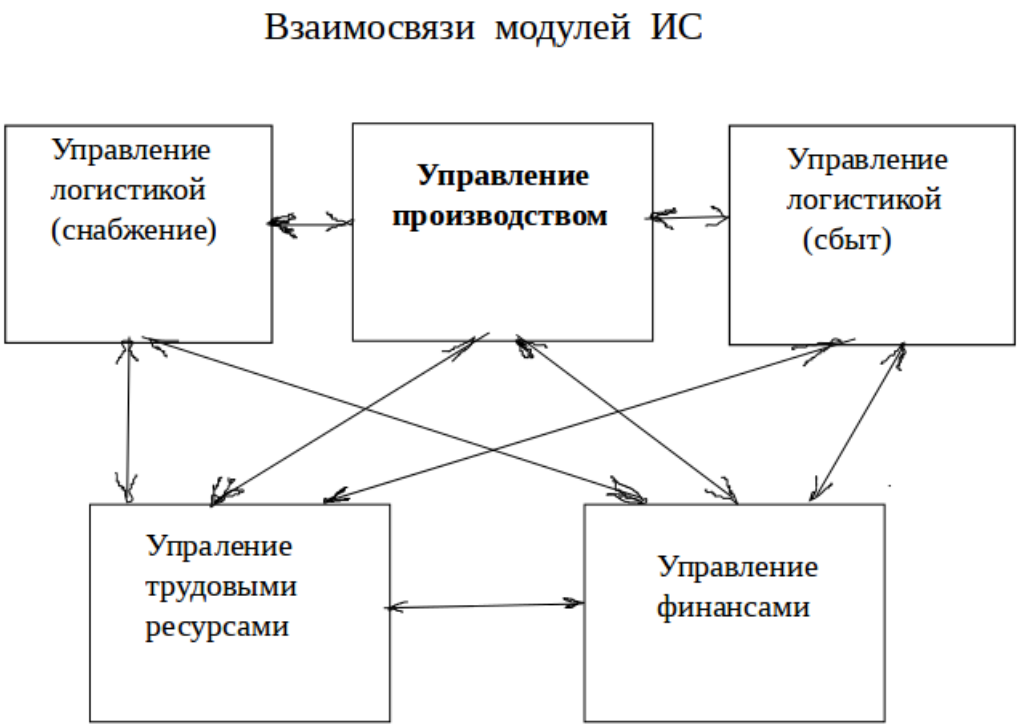


Рис. 1. Взаимосвязи модулей ИС

Модуль управления логистикой обычно делится на две части – снабжение и сбыт. Но иногда выделяется и третья часть: внутрифирменная логистика, или управление складами. Они на основе производственного плана определяют снабжение производства материалами, внутрифирменную производственную логистику и сбыт продукции.

Модуль управления трудовыми ресурсами осуществляет кадровое обеспечение производства, оценку использования рабочего времени сотрудниками и расчет заработной платы.

Модуль управления финансами – расчеты потребности в денежных ресурсах, их использование, позволяет оперативно осуществлять финансово–экономический анализ для оценки деятельности предприятия.

2.2. Классификация (или типология) ИС

В современном мире очень много различных ИС с очень разным предназначением. Причём, среди них, почти никогда нельзя выделить ИС строго определённые, выполняющие какой-либо узкий набор функций, ни с чем непересекающиеся. Как правило, ИС характеризуются множеством существенных параметров, определяющих место конкретной ИС в множестве существующих и потенциально возможных ИС. Отсюда следует, что

любая классификация ИС заведомо неполна, неточна и условна.

Например, можно классифицировать ИС по следующим параметрам:

- масштаб ИС (большая, маленькая и насколько в некоторых, весьма условных единицах измерений («попугаях»); примеры: ИС фирмы «Oracle» – международный уровень; ИС «1С-Предприятие», внедряемая у нас в университете – корпоративный уровень; ИС «1С-Бухгалтерия» для ИП – малая);

- область/отрасль экономики, которую ИС обслуживает, примеры: ИС «Сбербанка» 2017 года внедрения; ИС управления движением в РЖД; ИС «Domino» («Lotus Notes» IBM) аэропорта Домодедово;

- характер решаемых_задач/выполняемых_функций, который никогда не определяется точно и даже в системах, предназначенных по описанию для одной и той же области (для одного и того же набора задач), этот набор функциональности разный; примеры: ИС «Галактика» и ИС «1С-Предприятие»; или ИС «mail.ru» и ИС «yandex.ru»;

- степень автоматизации функций; примеры: ИС «Гефест» – система автоматического сброса неуправляемого боеприпаса для истребителей-бомбардировщиков – участие человека практически не требуется; АСУТП стиральной машинки – загрузить шмотки и нажать кнопочку; ИС «1С-Предприятие», внедряемая у нас в университете, требующая о-очень большого бумажного документооборота – а это всегда ручная работа;

- степень структурируемости обрабатываемой информации; пример: ИС «Библиотека» – документальная, в ней внутреннюю структуру хранимых объектов формализовать в настоящее время невозможно – разума человеческого пока не хватает; ИС типа АСУТП – фактографическая, которая оперирует простыми типами данных, обрабатываемым по жёстким алгоритмам;

- входной язык – степень формализации: жёсткофиксированные запросы или позволяет делать запросы достаточно произвольной формы;

- вид обрабатываемой информации, например, ИС публикуемой информации и ИС непубликуемой информации – это не обязательно «засекреченная» ИС; пример первой: ИС продажи билетов на ЖД транспорте; пример второй: АСУТП хлебопечки или АСУТП ректификационной колонны – даже если вывести на экран информацию, поступающую с датчиков, вы увидите бегущие колонки чисел – вам будет всё понятно?;

- по скорости обработки информации: ИС, работающие по событию – практически все ERP-системы и старше; ИС псевдореального времени – бухгалтерская система iceb и многие ERP-системы и старше, ИС реального

времени – специализированные ИС в энергетике, на транспорте, в военном деле (БИУС «Посейдон» для малых ракетных кораблей), в быту (АСУТП мультиварки), в научной сфере (ИС «синхрофразотрона» типа «андронный коллайдер»);

- и т. д., то есть, параметров, по которым можно группировать ИС – много и, соответственно, много возможных классификаций.

2.3. Архитектуры информационных систем

2.3.1. «Исторические» архитектуры ИС

Монолит. Одна программа как единое целое. Использовалась при «царе Горохе» в незапамятные времена. В настоящее время ИС, построенные по этой архитектуре – экзотика, однако, встречаются. Пример: «Консультант-Плюс» в однопользовательском («настольном») варианте.

Файл-сервер. Появилась и вошла в широкое употребление, когда появились ПЭВМ, которые объединили в локальные сети, то есть, примерно 30-35 лет назад. В ИС, построенной по этой архитектуре, одна из машин сети выделяется (назначается) в качестве центральной (файловый сервер), на которой предпринимаются меры по обеспечению её надёжности. На ней может функционировать совместно используемая СУБД, но чаще эта машина может просто использоваться для хранения и регулярного сохранения файлов с рабочих станций. Все другие компьютеры сети (ПЭВМ) выполняют функции рабочих станций, то есть, на них выполняются некие прикладные программы (Word, T-Flex, Kompas, «Консультант-Плюс», «1С»-однопользовательская или даже «1С»-многопользовательская-файл-серверная и прочие). Пользователи рабочих станций берут файлы с файл-сервера и периодически «сохраняют» свои файлы на файл-сервер. Назначение архитектуры – повысить надёжность хранения информации.

Клиент-сервер – это следующий шаг. Здесь уже решались две задачи:

а) обеспечение надёжности хранения информации – это уже было;

б) создание достаточной вычислительной мощности для обеспечения работы новых ИС с продвинутой обработкой информации.

Кроме того, для функционирования ИС в этой архитектуре **на сеть налагались дополнительные требования по обеспечению доступности и пропускной способности – это очень важное замечание: ИС с архитектурой клиент-сервер работает поверх локальной сети.** Может работать и поверх корпоративной сети, но понятия «корпоративная сеть» в те

времена ещё не было.

В этой архитектуре на выделенном сервере устанавливается специальное программное обеспечение – некоторая часть ИС (говорят, «серверная» часть, ныне это называется «backend»). Иначе говоря, ИС делится на две части: клиентскую и серверную. На сервер выносится значительная часть обработки информации и полностью – функция хранения информации – архив файлов и/или СУБД, на клиенте остаётся некоторая часть обработки информации («толстый» клиент – если оно есть) и взаимодействие с пользователем («тонкий» клиент, если только это). По сети передаются запросы/ответы и пакеты данных. Следовательно, чтобы клиент и сервер понимали друг друга, в состав ИС этой архитектуры **обязательно входит протокол информационного сопряжения и его программная реализация на клиенте и сервере.** В протоколе определяются: формат пакетов, алгоритм обмена и именование взаимодействующих объектов. Если СУБД выносится на отдельный (третий) компьютер, то получается трёхуровневая архитектура «клиент-сервер».

В таком виде архитектура «клиент-сервер» используется и доныне, хотя считается сильно устаревшей. Особенностью разработки ИС в этой архитектуре – большая трудоёмкость и, следовательно, большие сроки разработки, поэтому область применимости данной архитектуры сдвинулась в область специальных ИС.

2.3.2. Архитектура SOA – это наше всё

А потом появились глобальные сети в образе Интернета и технологии сетей с начала 90-ых годов вышли на новый, очень продвинутый уровень развития. В частности, были придуманы сервисы, которые к концу 90-ых годов технологически оформились и получили о-очень большое распространение.

Определение. **Сервис –** это аппаратно-программный комплекс, предназначенный для организации публичного доступа к некоторому ресурсу (контенту, см. ниже главу 6). В обиходе, термин «организация публичного доступа» означает «расшаривание» этого ресурса, а что такое «расшаривание», вы знаете.

Важный абзац. Естественно, это отразилось и на архитектуре ИС – было придумано, как сэкономить деньги на разработке ИС и, в то же время, «клепать» их в «товарных количествах» с достаточным качеством, особо не напрягаясь. В конце 90-ых годов появилась архитектура SOA. И увидели все, что это хорошо:

- и разработчики – стало возможным быстро создавать ИС;

- и потребители – качество оказалось вполне приемлемым, а интерфейсы привычными;

- и «хозяева жизни» – резко ускорился возврат инвестиций.

И теперь почти все ИС в экономике (и не только) создаются в этой архитектуре: форумы, интернет-магазины, порталы, торрент, ICQ, e-mail и прочая – это всё SOA. Практически весь публичный Интернет – SOA.

Определение очень абстрактное: **SOA** – это архитектура ИС, построенной с использованием заменяемых слабосвязанных элементов.

Другое определение, столь же абстрактное. **SOA** – это архитектура ИС, в рамках которой все функции системы являются независимыми сервисами с чётко определёнными интерфейсами, которые можно вызывать в нужном порядке с целью формирования бизнес-процессов.

Определение из Wiki: **SOA** – модульный подход к разработке программного обеспечения, основанный на использовании сервисов со стандартизированными интерфейсами.

Определение OASIS: **SOA** – парадигма организации и использования распределённых информационных ресурсов, таких как приложения и данные, находящихся в сфере ответственности разных владельцев, для достижения желаемых результатов потребителем, которым может быть конечный пользователь или другое приложение.

Определение IBM SOA Foundation: **SOA** – архитектурный стиль для создания ИТ-архитектуры предприятия, использующий принципы ориентации на сервисы для достижения тесной связи между бизнесом и поддерживающими его информационными системами.

Так что же такое SOA? Подход, парадигма, стиль, или что? А, ведь, «тама», в реальной экономике, придётся принять участие в работах по созданию и обслуживанию.

Поэтому дадим определение конструктивное в картинках, см. рис. 2.

Что мы здесь видим:

а) стиль и парадигма – если нарисовать («по-крупному») внутреннюю структуру ПО форумов, интернет-магазинов, порталов и др. современных ИС, мы получим именно эту картинку;

б) модульный подход – да, ясно видны четыре модуля: модуль клиента, модуль сервиса, модуль бизнес-логики, модуль СУБД;

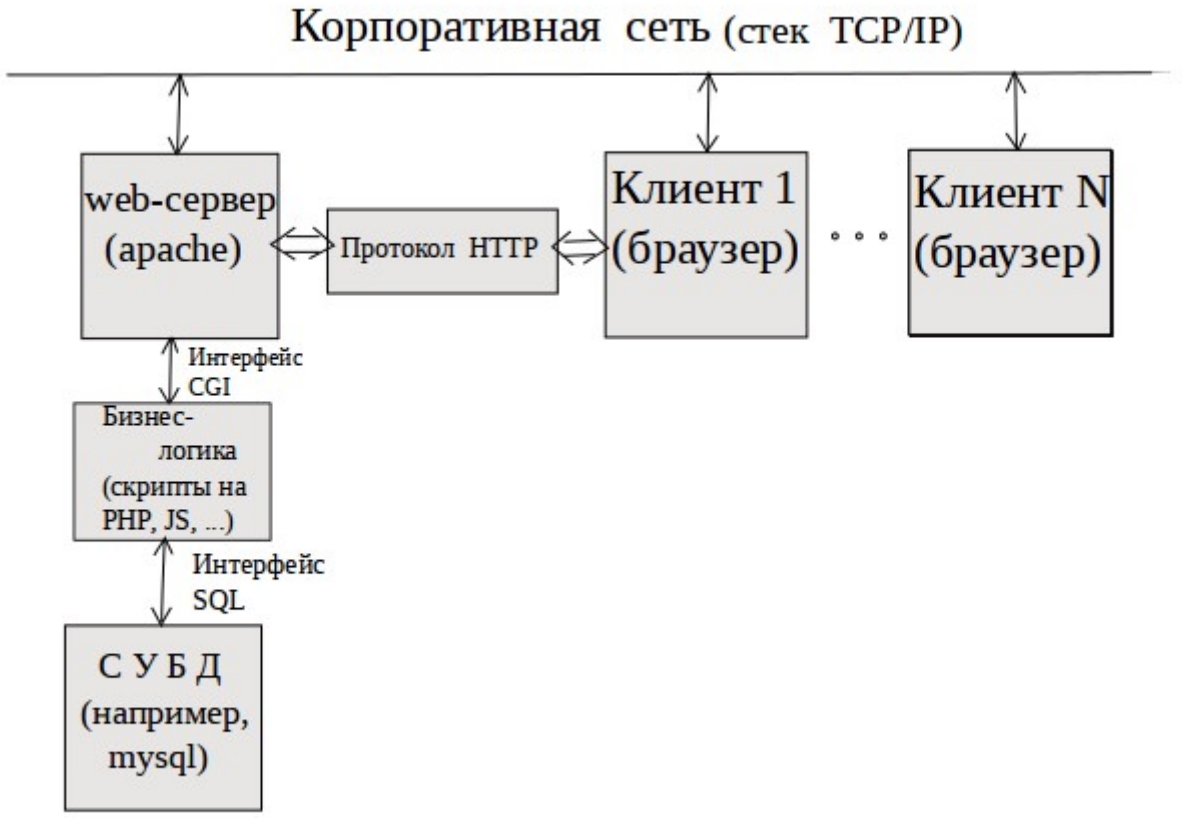


Рис. 2. Архитектура SOA на примере web-сервиса

в) все они достаточно независимы друг от друга – связаны только протоколами и интерфейсами, то есть, «слабосвязаны»;

г) они вполне заменяемы:

- разных браузеров, или клиентов почты, или клиентов торрента, или . . . и т. д., – существуют десятки;

- аналогично реализаций сервисов (например, web-сервисы; apache, IIS, websphera, httplight и т. д.; почтовые сервисы: sendmail, postfix, exim и т. д.) – тоже много;

- различных СУБД: mysql, postgresql, oracle, DB2, linter. . . – не-перечесть;

- реализаций бизнес-логики – это называется «движки» – «движки» форумов, порталов, интернет-магазинов – их можно немало найти в Интернете;

д) протоколы и интерфейсы: все протоколы унифицированы для каждого сервиса (например, неважно, какой клиент мы выберем для торрент – они все поддерживают протокол торрент), с другой стороны, интерфейсы взаимодействия сервиса с бизнес-логикой тоже унифицирован (например, почти все серверы для web-сервиса поддерживают интерфейс CGI, по кото-

рому подключается бизнес-логика); и, наконец, почти все СУБД предоставляют интерфейс, базирующийся на языке SQL;

е) и, последнее, **SOA – это клиент-сервер** или многоуровневый клиент-сервер.

Примечание 1. Кстати, не путайте скрипты бизнес-логики – backend («движки», иногда это даже не скрипты, а настоящие программы) со скриптами, которые передаются в HTML-страничке на клиент – frondend.

Примечание 2. На рисунке 2 СУБД изображена внизу. Так может быть, если web-сервер, бизнес-логика и СУБД установлены на одном компьютере – сервер называется, и это будет двухуровневый клиент-сервер. Однако, в больших ИС, как правило, СУБД запускается на отдельном компьютере и подключен он будет к сети, то есть, на рисунке – на одном уровне с web-сервером (левее его). Тогда это будет трёхуровневый клиент-сервер. Более того, в больших ИС нередко и бизнес-логика (в виде комплекса программ) выносится на отдельный компьютер – получается четырёхуровневая архитектура.

Перечисленные пункты а) – е) точно и конструктивно определяют сущность SOA. То есть, мы действительно наблюдаем унифицированную архитектуру, которая позволяет создавать очень разнообразные ИС и при этом задействуются готовые, заранее разработанные элементы чужого производства. И в результате процесс создания нового изделия (выполнения заказа) нередко заключается всего лишь в должной настройке этих готовых элементов. Это и позволяет легко достигать целей, заявленных в «Важном абзаце» данного пункта: скорость-качество-деньги.

2.4. Модели жизненного цикла

2.4.1. Модели и их применимость

Их две: **каскадная** и **спиральная**. Первая – детерминированная, вторая – свободная. Исторически первой появилась каскадная в те незапамятные времена, когда компьютеры были большими, а пользователей было мало. И эти пользователи были программерами. И потому писали программы сами для себя, не торопясь, «с чувством и наслаждением» и строго последовательно (говорили «поэтапно»: пока один этап не сделали, за следующий не принимались, а этапов было 10-12).

А потом компьютеры пошли в экономику, а в экономике «рулят» деньги, которые всегда считают и требуют вернуть назад и побыстрее. Поэтому процесс разработки резко ускорился, количество этапов сократи-

лось до 3-4: идея → реализация → продажа («внедрение»). Естественно, в продажу поступал достаточно сырой и малофункциональный продукт (версия 1 или даже 0.x), на который поступало масса претензий и замечаний, которые служили основанием для доработки и выходом на новый круг: идея → реализация/доработка → продажа («внедрение»). Появлялась версия 2, которая тоже делалась «быстрей-быстрей» и на которую снова «сыпались» претензии и замечания и т. д. Это мы наблюдаем в жизни – нам постоянно «впариваются» недоделки (см. в лекциях анекдот о двух студентах Коляне и Воване). Однако эта модель обладала очень большим достоинством – она обеспечивала быстрый возврат инвестиций, то есть, ускоренный оборот капитала. А это в экономике чрезвычайно важный фактор, имеющий прямое отношение к росту богатства собственника денег.

Поэтому теперь каскадная модель применяется только в случае создания очень ответственных («критических») систем, да и то в сильно модифицированном виде, разрешающем возвраты назад на доработку и исправление выявленных ошибок. Каскадная модель определена в стандартах серий:

- проектирование: ГОСТ 34.xxx-xx (прежде всего в ГОСТ 34.601-90), ГОСТ 24.xxx-xx;

- этап рабочего проекта: ГОСТ 19.xxx-xx (прежде всего в ГОСТ 19.102-77).

Основной же в экономике стала спиральная модель в различных вариантах и под разными названиями: «модель ITIL», «модель Oracle», RUP, «модель Microsoft» и т. д. – это всё «ревизии и углубления» обычной спиральной модели. Почему появляются эти варианты? Ну, так, надо же людям деньги как то зарабатывать: одни программируют, другие не могут, поэтому «топчатся» рядом и советуют – пишут всякие рекомендации как делать. Это не шутка. Это метод заработка – «выкачивание» денег из заказчика.

Спиральная модель определена в стандартах:

- ISO/IEC 15288 – процесс проектирования ИС в целом, как «совокупности средств, методов и персонала»;

- ISO/IEC 12207 – процесс проектирования только ПО ИС (только «методов и персонала»).

На оба стандарта есть российские аналоги: ГОСТ Р ИСО МЭК 15288-2005 «Системная инженерия» и ГОСТ Р ИСО МЭК 12207-2010 «Процессы жизненного цикла программных средств».

Эти стандарты определяют «процессы» жизненного цикла ИС. То есть, вместо этапов (стадий) проектирования появился новый термин – «процессы разработки», которые делятся на «работы».

2.4.2. Управление проектом ИС

Определение. **Управление проектом** – создание коллектива разработчиков, планирование и организация работ, контроль за сроками и качеством выполняемых работ.

Техническое и организационное обеспечение проекта – выбор методов и инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки (очень важно), разработка методов и средств испытаний ПО, обучение персонала и т. п.

Обеспечение качества проекта – верификация, проверка (аудит) и тестирование ПО.

Верификация – определение того, насколько текущее состояние разработки, достигнутое на данном этапе, отвечает требованиям этого этапа. Для верификации очень важно предварительно определить методы описания промежуточных состояний разработки (то есть, как будем мерить, с чем сравнивать).

Проверка (аудит) – проверка ПО методами статического и динамического анализа, анализа журналов разработки для выявления уязвимостей скомпилированного и исходного кода.

Тестирование – процесс выполнения ПО системы или компонента в условиях анализа или записи получаемых результатов с целью проверки (оценки) некоторых свойств тестируемого объекта [3].

Для оценки **качества** проекта существует стандарт ГОСТ 28195-89, определяющий контроль качества на всех этапах жизненного цикла ИС. Но сейчас более популярен стек стандартов ISO 9000, хотя он определяет вовсе не качество продукта, а качество технологии его создания, но кто это понимает? То есть, если вы читаете на колбасе «Предприятие сертифицировано на ISO 90XX», то это вовсе не означает, что колбаса съедобная и ее можно покупать; она вполне может быть Галапагосом, но качественно сделанным и красиво оформленным.

Предпроектное обследование осуществляется с целью параметризации проекта ИС: определяется текущее состояние заказчика («как есть»), формулируются предложения «как должно быть», выявляются все материальные, финансовые, людские и временные ресурсы для выполнения необходимых проектных работ, то есть определяются количественные (ну и качественные заодно – надо же порекламироваться?) параметры проекта. На основании выявленных мер разрабатывается технико-экономическое обоснование целесообразности автоматизации. По итогам обследования пишется «Отчёт о предпроектном обследовании» – основа для будущего ТЗ на ИС.

Пилотный проект – некоторая небольшая, но значимая часть ИС, достаточно функционально полная, то есть, которая может эксплуатироваться «почти» независимо от всей остальной ИС, однако дающая представление о «внешнем виде», интерфейсах и особенностях эксплуатации всей ИС.

2.5. Информационные системы уровня предприятия

2.5.1. Определение корпоративной информационной системы

Корпорация – крупная организационная структура, целью функционирования которой является производство товаров и услуг. Исторически корпорация возникла как объединение свободных хозяйственных объектов для достижения экономических целей. Отсюда и современные типы корпораций: в их состав также могут входить подразделения, принадлежащие к разным отраслям экономики, в том числе и к торговле. И, следовательно, КИС может быть объединением нескольких отраслевых ИС.

Или, другими словами: корпорация – стабильная многопрофильная территориально распределенная структура, обладающая всеми необходимыми системами жизнеобеспечения и функционирующая на принципах децентрализованного управления.

Определение абстрактное: **Корпоративная информационная система (КИС) –** управленческая идеология, объединяющая бизнес-стратегию и информационные технологии

Определение конструктивное: **Корпоративная информационная система –** это масштабируемая информационная система, предназначенная для комплексной автоматизации всех видов хозяйственной деятельности больших и средних предприятий, в том числе корпораций, состоящих из группы компаний, требующих единого управления.

В данном определении говорится, что КИС это ИС, функционирующая поверх корпоративной сети. Почти всегда КИС имеют иерархическую структуру из нескольких уровней, причём элементами этой структуры могут быть разные ИС. Для КИС характерны архитектуры клиент-сервер, многоуровневый клиент-сервер и, в последнее время, SOA.

2.5.2. Состав и структура КИС

Современные корпоративные информационные системы являются сложными интегрированными комплексами, которые включают в себя модули, отвечающие практически за все механизмы работы современного

предприятия. Модуль КИС – некоторая функционально-ориентированная система (программный комплекс), возможно, ИС (см. рис. 3).

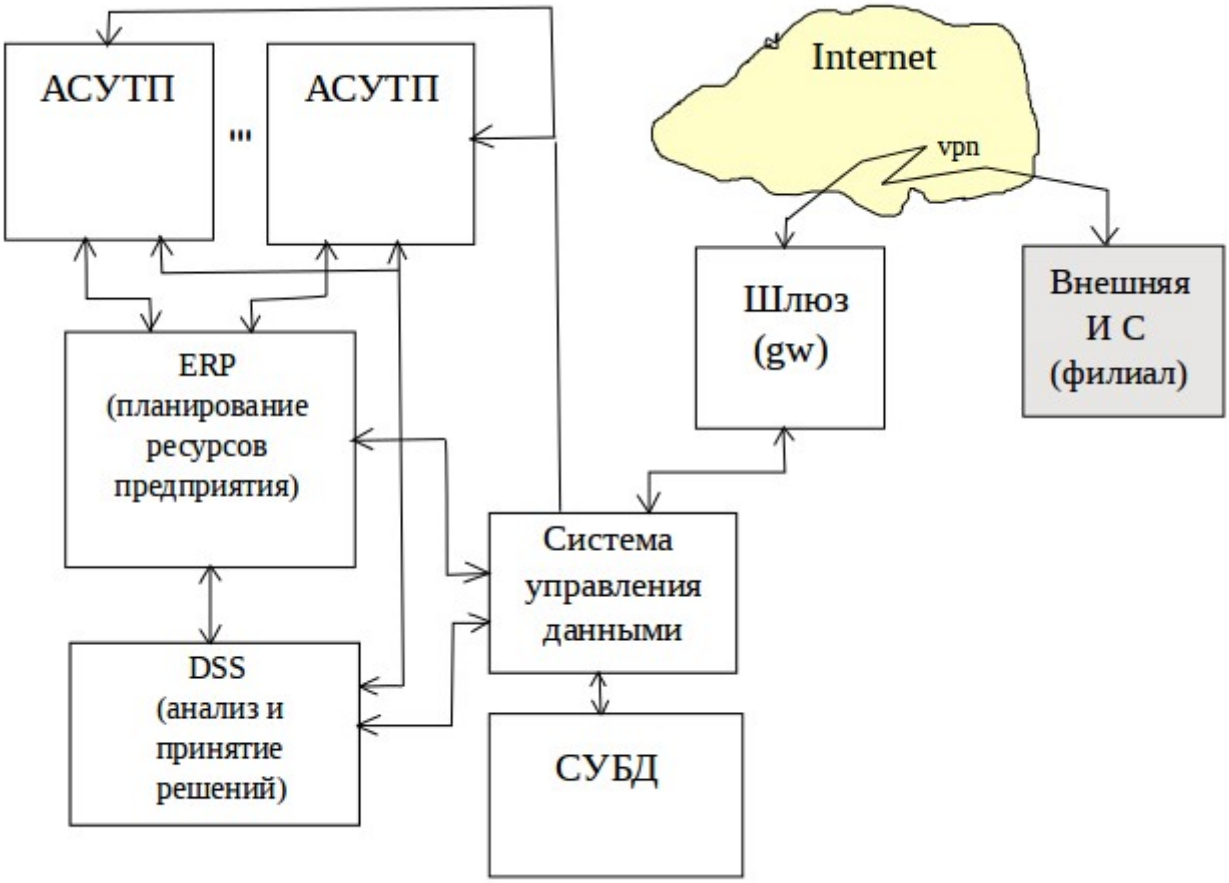


Рис. 3. Структурная модель КИС

Самый нижний уровень – хранилище данных, прямой доступ к которому есть только у системы управления данными (СУД). Остальные модули работают через СУД. СУД обеспечивает создание **единого информационного пространства организации**.

СУД через шлюз имеет выход в Интернет и связана (с помощью vpn) с ИС филиалов организации.

DSS – (Decision Support System – система поддержки принятия решений) – предназначена для поддержки деятельности руководства организации и содержит методы интеллектуального анализа данных.

АСУТП – автоматизированные системы управления технологическими процессами, это ИС, непосредственно связанные с производственными технологическими комплексами. Они получают на входе информацию с датчиков, установленных на оборудовании, и на основании этой информа-

ции вырабатывают команды на управляемое оборудование.

ERP – (Enterprise Resource Planning – планирование ресурсов предприятия) – ИС корпоративного уровня, охватывающие всю финансово-хозяйственную и производственную деятельность предприятия.

Указанные системы могут быть разных производителей и, следовательно, встаёт вопрос об их информационном сопряжении друг с другом, для этого в состав КИС могут входить модули сопряжения (middleware). И отсюда возникает очень важное требование к компонентам КИС – они должны быть открытыми системами.

2.5.3. История вопроса

Прежде всего следует заметить, что корпоративная ИС вовсе не обязательно требует в качестве базы именно корпорацию или иную крупную организацию. Она вполне может работать в сравнительно небольшой фирме с несколькими десятками человек персонала (малый бизнес) и состоять из модуля управления производством, модуля управления логистикой и модуля управления финансами. Управление трудовыми ресурсами в небольшой фирме обычно берёт на себя хозяин фирмы. К этому следует добавить обязательное наличие корпоративной сети в качестве технической базы для функционирования КИС, поскольку именно в рамках настройки корпоративной сети правильно решаются вопросы разграничения доступа к ресурсам сети – прежде всего, к серверам и рабочим станциям.

Вы скажете: «А, вот, мы поверх локальной сети поставим AD и не будем заморачиваться с корпоративной сетью – и так всё будет ОК». Ответ: да, будет, если ваши сотрудники – сплошь «блондинки». Но если среди них заведётся хотя бы один «правильный пацан», то вашему админу лучше не завидовать – последствия не заставят себя ждать. И о причинах банкротства вы можете узнать только постфактум.

Это всё сказано в качестве обоснования положения: **КИС должна всегда работать поверх корпоративной сети.** Потому что, в локальной сети вопросы разграничения доступа если и определяются, то на совершенно недостаточном уровне.

К КИС относятся информационные системы стандартов MPS, MPR , MPR II, ERP, ERP II и CSRP (см. рис. 4).

Первыми в далёкие 60-е годы появились системы **MPS** (Master Planning Scheduling – управление календарным планированием), предназначенные для составления основного плана производства. Основной производственный план составлялся на основании данных о состоянии

спроса или имеющихся и предполагаемых заказах в номенклатуре, то есть, какие и сколько штук изделий нужно произвести. На основании производственного плана в системе **CRP** (Capasity Resours Planning – расчёт производственных мощностей – «задача о назначениях») составлялся план загрузки имеющегося производственного оборудования (станков) и персонала. Эти системы были достаточно простыми в силу небольшой мощности тогдашних ЭВМ.

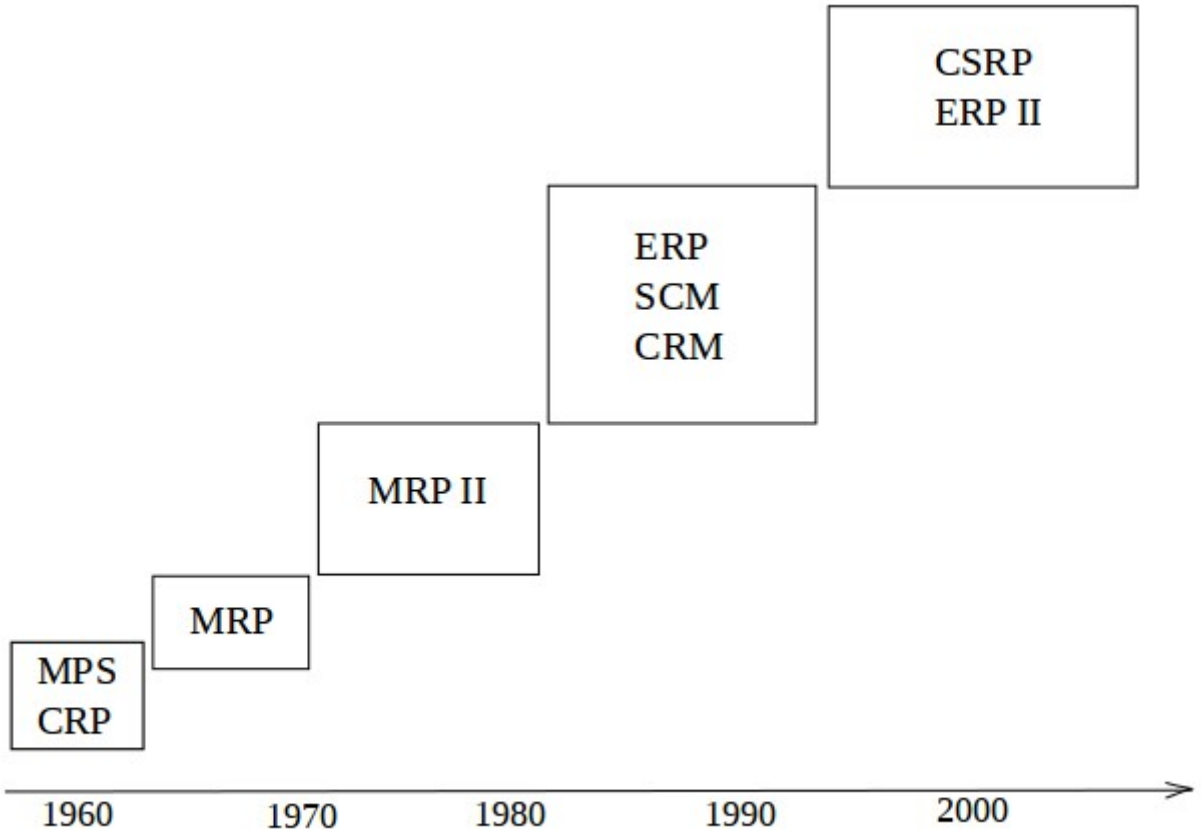


Рис. 4. Развитие стандартов КИС

На рубеже 60-70 годов появились ИС класса **MRP** (Material Requirement Planning – планирования материальных потребностей), которые выполняли уже более продвинутую функцию – подсчёт возможности выполнения нового заказа к нужному сроку при текущей загрузке производства. Если же система CRP показывала невозможность выполнения данного заказа к конкретному сроку, то ИС MRP должна была ответить на вопрос, во что обойдется заказчику выполнение этого заказа, если он все же на сроке настаивает.

Затем развитие вычислительной техники и программирования позволило создать системы класса **MRPII** (Manufacturing Resource Planning – планирование производственных ресурсов), которые обеспечивали прогно-

зирование, планирование и контроль производства по всему циклу, начиная от закупки сырья и заканчивая отгрузкой товара потребителю. Причём они обеспечивали решение задачи планирования деятельности предприятия как в номенклатуре, так и в денежном выражении. Это была новая функциональность, все предыдущие системы работали только в номенклатуре.

В конце 80-х появились по-прежнему большие, но уже достаточно мощные ЭВМ (например, IBM 360), а также в массовых количествах выпускались мини-ЭВМ фирмы DEC. Появились локальные сети и, кроме того, технологический уровень программирования достиг промышленных масштабов – появились первые IDE – интегрированные системы разработки. Это позволило создать системы следующего поколения – **ERP** (Enterprise Resource Planning – планирование ресурсов предприятия). Эти системы уже охватывали всю финансово-хозяйственную и производственную деятельность предприятия. Причём в них реализовывались такие концепции, как «единая точка ввода информации», централизация данных в единой базе данных, режим работы «по событию» (близкий к реальному времени), поддержка территориально распределенных структур.

Это были первые системы, которые стремились унифицировать модели управления предприятиями любых отраслей («под одну гребёнку») и работали на широком круге аппаратно-программных платформ и СУБД. Системы ERP были первыми ИС в архитектуре «клиент-сервер» и в силу необходимости их сопряжения с другими ИС, выполнялись как «открытые системы».

Однако, системы ERP охватывали только деятельность самого предприятия, поэтому для автоматизации взаимоотношений с поставщиками и клиентами были созданы системы SRM (Supplier Relationship Management – управления взаимоотношениями с поставщиками), CRM – (Customer Relationship Management – управление взаимоотношениями с клиентами) и SCM – (Supply Chain Management – управление цепочками поставок и управление запасами). Эти системы получили широкое распространение в торговле и в сфере услуг, где не было необходимости в модулях управления производством.

2.5.4. Современные системы с современной архитектурой

С появлением Интернет и ПЭВМ стало возможным интегрировать в КИС ещё одну важную функцию – взаимоотношения с партнёрами, поставщиками, подрядчиками, дистрибьюторами. Появились системы **ERPII** – (Enterprise Resource and Relationship Processing – обработка данных по ре-

сурсам и взаимоотношениям предприятия). Они уже охватывали всю цепочку управления корпорацией и том числе отношения с поставщиками, подрядчиками и заказчиками.

И, наконец, на рубеже 2000-х годов появилось следующее поколение КИС – системы стандарта **CSRP** – (Customer Synhronized Resource Planning – планирование ресурсов совместно с потребителем). Они интегрировали процессы как внутри одной корпорации, так и за ее пределами. Одним из первых примеров успешного применения информационной системы этого стандарта была организация работ в корпорации Dell.

Это системы текущего дня. И когда вы, купив мультиварку, автомобиль, принтер или что-то ещё, без чего вы никак жить не можете, регистрируетесь на сайте соответствующей фирмы и регистрируете свой продукт (вам же нужны гарантии производителя?) – вы как раз работаете в системе CSRP: вы подключаетесь браузером к веб-серверу корпорации, с него приходит html-страничка с frondend-скриптами (форма ввода), вы вводите информацию, которая через веб-сервер поступает на обработку в бизнес-логику (backend) и СУБД и в прочие модули КИС. Следовательно, CSRP – это есть корпоративная информационная система в архитектуре SOA.

3

УПРАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯМИ И ГРУППАМИ ПОЛЬЗОВАТЕЛЕЙ

3.1. Учётная карточка (user account)

Поскольку Linux – система многопользовательская, то это означает, что в ней должно быть:

- разграничение прав различных пользователей, то есть, система должна уметь контролировать какие файлы пользователи могут читать/писать/изменять, какие программы запускать, в какие каталоги заглядывать;

- и, кроме того, пользователям должно «позволяться» создавать свою «среду обитания» (environment), то есть, иметь свои настройки для программ, свою структуру каталогов, свои настройки терминала, свою настройку графической оболочки, свой набор иконок на столе и т.п.

Для этого в системе должна вестись некоторая база данных, в которой должны сохраняться основные сведения о каждом пользователе этой вычислительной системы.

И такая база данных пользователей действительно в Linux существует, только хранится она не в виде набора таблиц, как в реальных СУБД, а в виде набора файлов, иерархически упорядоченных. Каждая запись в этой БД, содержащая сведения о некотором пользователе, называется **user account**, или по-русски – «учетная карточка пользователя». Таким образом, чтобы допустить нового пользователя в систему, необходимо создать для него этот account, или по-русски – заполнить учётную карточку.

Ниже рассмотрено, как реализовано управление пользователями в ОС Linux, но примерно также оно реализовано и в других сложных программных системах.

3.2. Состав и содержание "учетной карточки"

3.2.1. Именование пользователей в системе

Поскольку пользователей в системе может быть много, то чтобы их различать, нужно их как-то именовать. Мы, люди, именуем друг друга по именам, точнее, по ФИО. Это некоторые текстовые строки – слова нашего языка, которые имеют определённый смысл и используются для реализации очень важной функции в обществе – именования субъектов этого общества.

То есть, имя человека – это некоторая последовательность символов алфавита языка, строка символов, и в программировании (в языке Си) она определяется, например, так:

char s{} = «vasja»;

или так:

char s1[] = «vasilii_sidorovich»;

или даже так:

char s2[] = «vasilii_sidorovich_ivanov»;

В указанном виде – строка символов в одно слово, без пробелов, – эта строка вполне может использоваться в системе для именования пользователей. Почему без пробелов?

Однако, возникает **проблема:** системе достаточно часто приходится выполнять поиск пользователей. А если пользователи именуются таким образом (текстовыми строками), то такой поиск реализуется посредством сравнения строк символов, то есть, для каждого сравнения – цикл посимвольного сравнения до конца строки или до первого несовпавшего символа в строках. А цикл – это всегда долго, это десятки команд, то есть, очень расточительно.

Мы, люди, этого не замечаем: наше мышление устроено так, что символьное именование объектов для нас очень удобно, и операции сравнения имён (идентификации объектов) мозг выполняет очень быстро, ассоциативно, поскольку мозг имеет аппаратную поддержку ассоциативных алгоритмов.

Но система-то не человек, она «железная», работает совсем на других принципах, память в ней линейна (архитектура фон Неймана) и в этих условиях сравнение строк – операция неэффективная.

Чтобы эту проблему решить (повысить скорость выполнения операции идентификации объектов) в системе вводится другое именование

объектов (параллельное!) – числовое, то есть, объекту присваивается некоторое число по определённым правилам – идентификатор объекта (id) и это число для системы выполняет роль имени этого объекта. А числа ЭВМ сравнивает очень быстро – одна команда cmp (см. описание языка assembler).

И это общий принцип: для именования пользователей, программ, файлов, строк в таблицах и др. объектов используются два пространства имён:

- символьное – для удобства человека, именование «со смыслом»;
- числовое – для системы, чтобы быстрее считать, «бессмысленное».

Это верно для любых программ, в том числе, информационных систем. Помните: программа «тупа, как валенок», и интеллекта в ней ровно столько, сколько заложил в неё разработчик и не на крапочку больше – не приписывайте программам «ишкуштвенный интеллект».

3.2.2. Состав «учётной карточки»

«Учётная карточка» состоит из иерархии файлов: на самом верху – файл /etc/passwd, в нём сохраняется «заголовок» учётной записи – самая основная информация о пользователе. Этот файл открыт для чтения всем (потому что, многим программам нужно идентифицировать пользователя), поэтому в целях безопасности, на основе этого файла в системе создаются ещё несколько файлов с дополнительной информацией (например, с паролями), которые уже доступны только администратору системы.

Причём, в разных дистрибутивах linux схема хранения закрытой информации различна: например, в slackware используется схема с «теневыми» паролями – с файлом /etc/shadow, а в дистрибутиве altlinux – более сложная схема: пароли хранятся в индивиуальных для каждого пользователя файлах /etc/tcb/LOGIN/shadow, где LOGIN – Name (login) конкретного пользователя.

Кроме того, для сохранения специфических настроек пользователя, в системе используются ещё несколько десятков файлов. Как правило, эти файлы сохраняются в домашнем каталоге пользователя и «хозяином» этих файлов является сам пользователь, то есть, он их может создавать и редактировать. Именно это он и делает, когда настраивает свою среду.

Как правило, все конфигурационные файлы – обычные текстовые файлы с определённой структурой. В домашнем каталоге пользователя это каталоги и файлы, имена которых начинаются с точки – «скрытые» файлы и каталоги.

3.2.3. Структура файла passwd

Файл passwd – текстовый файл (см. рис. 5), каждая строка которого состоит из отдельных полей, разделённых символом-разделителем – «:». То есть, можно сказать, что по форме информация, содержащаяся в этом файле, представляет собой таблицу. Графы этой таблицы следующие:

Name
Password
user ID
group ID
General information
Home dir
Shell

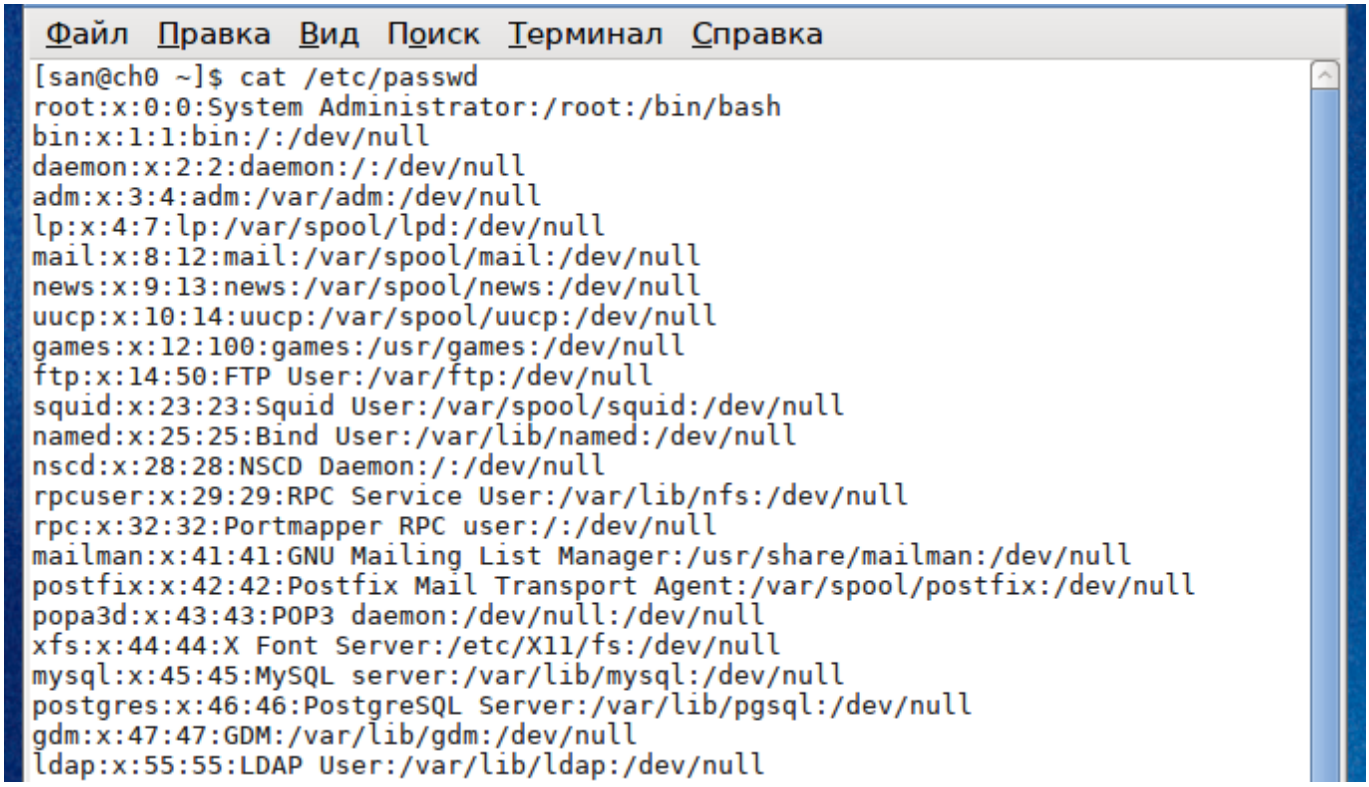


Рис. 5. Начало файла passwd

Иногда кроме этих основных граф, в таблице могут быть ещё дополнительные поля:

Class
Password change time
Account expiration time

Name – уникальное символьное имя пользователя – логин пользователя (login). Это то имя, которым мы, люди, именуем пользователей вычислительной системы, то есть, это «человеческое» имя пользователя. Допустимые символы: [a..z0..9-_], то есть, только малые латинские буквы, цифры, тире и знак подчёркивания и всегда начинается с буквы. Максимальная длина Name обычно определяется в 32 символа. А на самом деле сколько?

Password – пароль пользователя, некоторый набор символов. Кстати, в Linux в пароле могут использоваться почти любые символы, какие могут быть введены с клавиатуры, в том числе «непечатные». В настоящее время, однако, пароль хранится не в passwd, а в другом файле и хранится он в зашифрованном виде, причём для шифрования пароля чаще всего используется алгоритм MD5 – односторонний, то есть зашифровали – и всё. Восстановить уже никак нельзя. Но можно «подобрать» пароль, однако процесс подбора может оказаться очень долгим. Максимальная длина пароля?

User ID – числовое имя пользователя, которым пользуется система – uid. Это некоторое число – беззнаковое целое – в диапазоне от 0 до 65535. Отсюда следует ограничение на количество пользователей в системе. Однако в реальности действуют другие, гораздо более жёсткие, ограничения и поэтому на самом деле максимальное количество пользователей в системе обычно не превышает нескольких сотен. Сколько?

Group ID – числовое имя группы пользователя – gid. У каждого пользователя есть группа. Как правило, каждый пользователь входит в некоторую группу, возможно, не одну.

General information – информация о пользователе, которую заносит сюда Администратор, чтобы не забыть, что за пользователь скрывается за этим логином, если пользователей в системе много. Если пользователь – физическое лицо, то телефон, отдел, номер комнаты, ФИО и т. д. Если юридическое лицо, то название организации, телефон, адрес и др.

Home dir – домашний каталог пользователя. Располагается, как правило, в каталоге /home и обычно называется по имени логина. То есть, пользователь vasja имеет домашний каталог /home/vasja. Здесь сохраняются конфигурационные файлы с настройками среды, именно сюда попадает пользователь после входа в систему и здесь он полный хозяин – все файлы в этом каталоге принадлежат ему и он может делать с ними всё, что хочет. Только не забывайте, если вы всё в домашнем каталоге «снесёте», то можете потом в систему не войти.

Shell – программа, которая запускается для пользователя, когда он

входит в систему. Обычно, это программа-оболочка (в Linux – bash), но здесь может быть указана некоторая другая программа, что часто используется для некоторых задач, причём что-то обязательно должно быть указано.

Class – это поле 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.

Password change time – здесь может быть указан период времени, через который пользователю следует поменять пароль, если требуется повышенная секретность. Для того, чтобы заставить пользователя следовать этому правилу, Администратор задает период времени, после истечения которого система будет предлагать пользователю при входе систему поменять свой пароль.

Account expiration time – здесь Администратор задает дату, после которой доступ пользователю в систему будет запрещен (пример: пользователь в командировке, отпуске). Или Администратор может задать период времени суток, когда пользователь может находиться в системе (например, с 8:00 до 17:00, после 17:00 система при каждом действии пользователя будет напоминать ему, что он должен идти в столовую/кухню ужинать). То есть, карточка пользователя сохраняется (и все его файлы тоже), но войти в систему вне разрешённого периода он не сможет (пока не договорится с Администратором).

3.2.4. Назначение учетной карточки

Основное назначение учетной карточки – зарегистрировать пользователя, чтобы он мог войти в систему и работать в ней. Однако, существует некоторые следствия из этой функции, которые широко применяются администраторами при управлении пользователями.

Пожалуй самое главное следствие – это возможность с помощью определённой настройки учётных карточек поделить пользователей на три категории с разными правами и возможностями в системе:

а) суперпользователь (Администратор) – это, конечно, человек;

б) системные пользователи (или псевдо-пользователи) – «нелюди»;

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»);
- пользователи с «Гостевым входом»;
- автоматический запуск коммуникационных программ.

а) Суперпользователь

Суперпользователь root – Администратор системы, – числится в системе отдельно. Системное (числовое – uid) имя этого пользователя 0 (ноль) и домашний каталог этого пользователя /root находится отдельно от домашних каталогов всех остальных пользователей – непосредственно в корневой файловой системе. Это делается для того, чтобы root мог всегда зайти в систему, независимо от того, что там стряслось с другими пользователями и для него в корневой файловой системе резервируется 4-5% места. Каталог /home с домашними каталогами других пользователей при этом в целях безопасности может быть смонтирован на совсем другой раздел или даже другой диск.

Пользователь root существует в системе исключительно для управления системой – для решения административных задач. Работать под root`ом никак нельзя.

б) Системные пользователи

Это вторая категория пользователей, их системные (числовые – uid) имена начинаются с 1 и до некоторого числа, определённого в файле login.defs в переменной UID_MIN. В altlinux значение этой переменной равно 500 и это означает, что все пользователи с uid'ами от 1 до 499 – системные.

Рассмотрим на примере, что это за пользователи. Прежде всего «огласим» некоторые положения.

1) Пользователь не ручками (или «вооружённый» иголочкой) «ковыряется» в байтах памяти ЭВМ, в секторах на диске, в кадрах сетевой карты, – он это делает (в смысе «ковыряется») с помощью запускаемых им программ. Сейчас, в XXI веке, очень крайне редко бывает по-другому – только в особо специальных случаях.

2) Но когда пользователь запускает программу, то она далее работает с правами и возможностями этого пользователя – «наследует» его права. Если запустил программу суперпользователь, то и права её будут супер-

пользовательские. А это бывает очень опасно.

3) Ниже в главе 6 будут рассматриваться сервисы – объекты общего доступа, в том числе, доступа извне системы. Так вот, сервисы в системе имеет право запускать только суперпользователь.

И возникает задача – как обезопасить систему в этих условиях?

Пример. Вы создаете на своей машине WWW-сервер. Его назначение – выдавать в сеть какие-то документы (файлы *.shtml), специально предназначенные для публичного обозрения. В то же время, не хотелось бы, чтобы он (из-за возможных ошибок конфигурации или злонамеренных "закладок" в программе) мог выдавать наружу другие файлы, находящиеся в системе. Причем, запускаете вы этот WWW-сервер от имени пользователя root.

Для решения этой задачи был придуман способ, как ограничить некоторым программам (например, сервисам ftp-сервер, www-сервер и т.п.) доступ к файлам и директориям.

Способ этот базируется на том, что в Unix/Linux предусмотрена связка прав на доступ к каждому файлу с определенным пользователем. Пользователь может читать файлы, для которых он является "хозяином" или "владельцем". Все остальные файлы можно сделать для него недоступными.

Итак, для решения задачи нужно

- зарегистрировать фиктивного пользователя – например www;
- все файлы, к которым должна иметь доступ программа WWW-сервера, объявить "собственностью" пользователя www (очевидно, что ваши «другие» файлы должны быть недоступны этому самому www);
- объяснить системе, что когда запускается программа WWW-сервера, все должно выглядеть так, как будто его запустил реальный пользователь www.

То есть, для обеспечения безопасности системы задействуется система разграничения доступа Linux. Конечно, при таком подходе никакого реального человека, который скрывается под логином www не должно быть. Чтобы обезопасить себя полностью от появления в системе пользователя-человека с именем www, дополнительно делаем следующее:

- в учётной карточке в поле пароля вместо пароля ставим специальный символ, запрещённый в алгоритме MD5 – «*» или «!»;
- в поле shell ставим имя программы-пустышки или вообще имя «чёрной дыры» (см. рис. 1);
- в поле Home dir указываем путь к «рабочему» каталогу сервиса, то

есть, того места, где лежит «расшариваемый» контент сервиса;

- в поле General information пишем название сервиса.

Например, в altlinux-7.0.5 строчка в файле /etc/passwd, соответствующая сервису WWW, выглядит так:

apache2:x:477:455:Apache2 WWW server:/var/www:/dev/null

а файл */etc/tcb/apache2/shadow*, в котором хранится хэш пароля, выглядит так:

apache2:!!:17194::::::

Аналогичным образом определяются в системе другие системные пользователи – сервисы. То есть, системный пользователь – это «нелюдь», это программа некоторого сервиса.

Замечание. Имена (User Name, login) системных пользователей – зарезервированные имена пользователей в unix/linux. Некоторые системные пользователи создаются уже при установке системы и, значит, нельзя создать обычного пользователя с таким логином. Однако возможен случай, когда вы создали пользователя с зарезервированным именем, а потом ставите сервис, в котором это имя используется – тогда возможны проблемы. Сложность здесь в том, что полного списка системных пользователей, соответственно, зарезервированных имён, – нет.

в) Обычные пользователи – люди

в-1) Полные пользователи

Когда обычный пользователь входит в систему (с физического терминала или по сети), он вводит свое имя (Login Name) и пароль (Password). Соответствующая программа сверяет эти данные с теми, что записаны в учетной карточке и, если они сопадают, "пускает" пользователя в систему. То есть запускается программа, которая указана как Shell в учетной карточке. Пользователь с помощью этой программы может запускать другие программы. При этом ему можно создавать/удалять/писать/читать файлы, которые находятся в его Home dir (который тоже указан в учетной карточке). Кроме того, он может «ходить» по системе (не везде) и даже запускать дозволенные программы из каталогов /bin, /usr/bin и /usr/local/bin. То есть, это нормальный «полный обычный» пользователь. Иногда ему даже предоставляется право переходить в суперпользователя и выполнять команды от его имени – для этого логин обычного пользователя включают в группу wheel.

Диапазон системных (числовых) имён обычных пользователй определён в файле /etc/login.defs в переменных UID_MIN и UID_MAX.

UID_MIN в altlinux равно 500, а UID_MAX = 60000. То есть, всего в системе altlinux может быть 59500 обычных пользователей и ещё несколько сот «необычных».

в-2) пользователи, «поражённые в правах»

Почтовые пользователи

В настоящее время выходит из моды, но ещё широко используется способ доставки почты по протоколу POP. Суть его в том, что почта для конкретного адресата копится на "почтовом сервере" в его личном "почтовом ящике", а сам адресат время от времени запускает на любой машине, имеющей доступ по сети к "почтовому серверу", специальную программу (POP-клиент), которая связывается с сервером и забирает все письма, накопившиеся в "почтовом ящике" – это называется «работа с почтовым сервером в режиме offline».

Если «почтовый сервер» – это ЭВМ с linux, то такой способ работы с почтой реализуется так:

- пользователь регистрируется в системе как обычный пользователь, при этом, для пользователя создаётся «почтовый ящик» – файл в каталоге /var/spool/mail с именем, равным логину пользователя. В этот файл программа «почтового сервера» и будет «складывать» приходящие пользователю письма. Когда пользователь запускает почтового клиента (локально на этой же системе или удалённо на другой системе), чтобы посмотреть свои письма, то почтовый клиент посылает прежде всего запрос «почтовому серверу» на идентификацию пользователя, указывая в запросе его логин и пароль, а «почтовый сервер» сверяет эту информацию всё с той же базой учётных карточек, то есть, также, как при реальном входе пользователя в систему.

Но здесь есть одна особенность: этот человек – «почтовый» пользователь, возможно, больше никакого отношения к этой системе не имеет. Более того, возможно Администратору вовсе и не хочется, чтобы такие «почтовые» пользователи имели бы доступ ещё к чему-либо в системе, кроме своего почтового ящика. И Администратор реализует процедуру превращения таких обычных по регистрации пользователей в ограниченных в правах «почтовых» 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.

Аналогичным образом можно создать «поражённого в правах» пользователя для доступа к другим публичным сервисам.

Анонимные («транзитные») пользователи

Пусть мы к компьютеру подключили модем (несколько модемов (пул) – если у нас несколько телефонных номеров).

Далее, создаём несколько пользователей, например:

UUCPuser – у которого в поле Shell учётной карточки указываем программу /usr/sbin/uucico (из пакета uucp),

PPPuser – для которого запускается демон /usr/sbin/pppd, для связи по протоколу PPP,

SLIPuser – для которого запускается демон /usr/sbin/slipd, для связи по протоколу SLIP.

Возможно, убираем в учётных карточках Home dir – это зависит от того, какая коммуникационная программа используется, – некоторым нужно здесь указать рабочий каталог.

Затем в инструкции пользователю оговариваем, что если он, в ответ на запрос Login name введёт, например, PPPuser, то для него на этом конце модемной линии включится демон pppd и он получит IP-доступ по этому протоколу, то есть, пользователь демоном будет перенаправлен на маршрутизатор и т. д. При этом мы не будем знать, какой конкретно пользователь обозвался PPPuser'ом – **он для нас остаётся анонимным** (или транзитным – то есть, он сквозь наше аппаратно-программное обеспечение проходит, например, в Интернет).

Если же мы хотим вести учёт пользователей, то тогда мы должны для каждого пользователя аналогичным образом заводить учётную карточку, в поле Shell которой указывать, например, /usr/sbin/pppd и помесячно обрабатывать системные протоколы.

Примерно 20-25 лет назад именно так работали провайдеры Интернета. Потом, с переходом на технологии xDSL, вместо пула модемов провайдеры стали ставить мультиплексор, который интегрирует входящие соединения по технологии xDSL, выделяет подключения отдельных пользователей и отправляет их на «сервер доступа», который разбирается, что кому принадлежит и что с этим делать.

С точки зрения провайдера, такие пользователи, которых в систему

пускать нельзя, а нужно лишь пропустить в Интернет, являются «транзитными» пользователями. Если вы не имеете почтового ящика на сервере того провайдера Интернета, через которого вы ходите в Интернет, то вы для него «транзитный» пользователь. Провайдер учитывает как минимум следующее: когда вы подключились, когда отключились, сколько мегабайт прокачали туда-сюда. Однако, тенденция такова, что с каждым годом провайдеры всё больше и больше параметров нашей работы фиксируют.

Гости – пользователи

Создаём пользователя-гостя, например, с логином guest:

- в поле Shell, вместо реального shell, указываем некоторую другую программу, которая позволяет посмотреть некоторые документы, предназначенные для публики (доска объявлений BBS, или очень простой web-сервер, который на запрос get будет всегда оправлять post с фиксированным html-файлом, или программа выдачи текущего времени и т. д.);
- вместо домашнего каталога – «чёрная дыра» или рабочий каталог той программы, которая будет выдавать публичные документы, что находятся в этом каталоге;
- пароль убираем – мы же делаем публичную вещь. Или предпринимаем меры, чтобы любой пользователь легко его узнал.

В результате получаем совсем ограниченного в правах пользователя.

3.3. Добавление в систему пользователя

Самый простой способ – Центр Управления Системой.

Немного сложнее – в терминале с помощью программ adduser или useradd (как правило, это одна и та же программа).

Существенно более сложный способ – совсем вручную редактируя файлы «учётной карточки» текстовым редактором.

Второй и, особенно, третий способ требуют знания того, как создаётся пользователь. Кстати, та часть Центра Управления Системой, которая ведает созданием/редактированием пользователей является простой оболочкой над консольными программами adduser, userdel, passwd. Но есть и удобная консольная оболочка на этими программами – vipw (на основе редактора vi), которая после завершения редактирования автоматически синхронизирует изменения во всех файлах базы данных учётных записей – выполняет транзакцию. Vipw особенно удобна для редактирования учётных

записей.

Рассмотрим «ручной» процесс создания пользователя.

1) Вносим новую запись в базу данных учётных записей, например, с помощью vipw. Здесь мы должны сами обеспечить уникальность полей Name и uid нового пользователя.

2) Создаем домашний каталог нового пользователя в /home – имя его мы уже указали. Командой chown делаем хозяином этого каталога нового пользователя, а командой chmod устанавливаем правильный режим доступа к этому каталогу.

3) Копируем в домашний каталог нового пользователя «скелетные» каталоги и файлы из каталога /etc/skel – основа будущего профиля пользователя. Командой chown делаем хозяином этих каталогов и файлов нового пользователя, а командой chmod устанавливаем правильный режим доступа к этим каталогам и файлам.

4) Создаём пустой файл с именем, равным логину нового пользователя, в каталоге /var/spool/mail – это будет почтовый ящик нового пользователя. Командой chown делаем хозяином этого файла нового пользователя, а командой chmod устанавливаем правильный режим доступа к этому файлу.

5) Командой passwd назначаем новому пользователю пароль. Всё.

Через Центр Управления Системой все пять шагов выполняются за два шага.

В терминале это также выполняется за два шага: adduser login, затем passwd login.

Если мы создаём «неполного» пользователя, то соответствующим образом меняем учётную запись. Кстати, ни ЦУС, ни adduser «неполных» пользователей создавать не умеют.

Замечание 1. man useradd, man passwd man 5 passwd, man vipw, man chown, man chmod – команды просмотра справки по указанным командам.

Замечание 2. Конечно же всю работу по созданию и изменению пользователя делает root.

3.4. Внесение изменений в учётную карточку

Пароль пользователя меняется с помощью команды passwd login, все остальные данные можно изменить командой vipw.

Замечание. Если вы делаете всё вручную, то имейте ввиду, что вам

придётся пользоваться тем текстовым редактором, который определён в переменной среды EDITOR, а если эта переменная не установлена, то используется редактор vi.

Есть также интересная команда usermod (см. man), с помощью которой можно поменять почти все поля учётной записи.

Замечание. Естественно, любой пользователь может поменять свой пароль (и даже желательно, чтобы он это делал время от времени) с помощью команды passwd. И больше ничего со своей учётной записью обычный пользователь сделать не может. Потому что управление пользователями – это работа Администратора.

3.5. Как удалить пользователя

Если удаляем пользователя, только что созданного – под новым логином ещё ни разу не заходили в систему, то всё очень просто: удаляем либо с помощью ЦУС, либо командой userdel. При этом будут удалены учётная запись, домашний каталог, почтовый ящик – это минимум того, что нужно удалить.

Однако, если пользователь хотя бы раз заходил в систему, а тем более работал в ней достаточно долгое время, то всё становится совсем не просто. Дело в том, что программы, как правило, не берут на себя ответственности за удаление файлов, не ими созданных, или ими, но кем-то изменённых. Поэтому в этом случае с помощью ЦУС или userdel удаётся удалить только саму учётную запись, а всё остальное нужно удалять вручную. Более того, в зависимости от того, какую роль выполнял пользователь, его файлы могут быть где-то ещё в системе, помимо его домашнего каталога, его имя может встречаться в конфигурационных файлах системы.

Важно при удалении пользователя не забыть удалить почтовый ящик пользователя – файл /var/spool/mail/login_пользователя, даже если он не пустой. Ну, в крайнем случае, перенести (командой mv) в другое место. Иначе могут возникнуть проблемы с почтой у нового пользователя, которому случайно, не преднамеренно, а именно случайно достанутся тот же логин или uid. А также могут возникнуть претензии от старого пользователя – из-за «утечки» информации.

Также файлы пользователя могут оказаться в рабочих каталогах некоторых сервисов, к которым пользователь мог иметь отношение. Например, если пользователь принимал участие в создании контента web-сервера, то его файлы могут оказаться в каталоге /var/www/html – рабочем каталоге

web-сервера.

Все файлы удаляемого пользователя можно найти, например, с помощью команды

find / -user login

где *login* – логин удаляемого пользователя.

Если файлов окажется много и к тому же, некоторые могут оказаться нужными для функционирования системы, то лучше записать список найденного в файл и затем спокойно разобраться, что из найденного нужно и что с этим делать, типа так:

find / -user login > login.txt

Однако, помимо файлов, пользователь может упоминаться в различных конфигурационных файлах системы. Например:

- в файлах */etc/group, /etc/login.access, /etc/ftpusers*;

- в конфигурационных файлах и каталогах почтового сервера */etc/aliases, /etc/postfix* и др.;

- в конфигурационных файлах и каталогах сервера samba;

- если пользователю разрешалось запускать задачи с помощью cron, то пользователь может упоминаться в */etc/crontab* или в */var/cron/allow, /var/cron/deny*, а, также, может существовать его индивидуальная таблица *crontab* в */var/cron/tabs/*;

- если пользователю разрешалось пользоваться "пакетным" выполнением программ, то пользователь может упоминаться в */var/at/at.allow, /var/at/at.den*y;

- если в системе определено "квотирование" дискового пространства, то должна быть quote для этого пользователя;

- если пользователь пользовался IP связью через модем, то он может упоминаться в конфигурационных файлах, лежащих в */etc/sliphome* или */etc/ppp*;

- если пользователь был одним из админов и сопровождал какие-либо сервисы, то его логин может упоминаться в конфигурационных файлах этих сервисов;

- если пользователь был системным пользователем, то от имени пользователя могут запускаться какие-нибудь демоны и тогда имя пользователя может быть в */etc/inetd.conf*, */etc/xinetd.conf*, */etc/rc.local* или каталоге */etc/xinetd.d*.

И это не полный перечень мест, где пользователя можно найти. Поэтому удалить пользователя – совсем не просто.

3.6. Блокировка пользователя

Здесь рассматриваются ситуации, когда пользователю нужно закрыть доступ в систему временно, то есть, пользователь в отпуске, на «больничном», в командировке или каком ином «отгуле», и чтобы под его логином не мог зайти некий иной «друг семьи», его учётную карточку нужно временно заблокировать в целях обеспечения безопасности.

3.6.1. Правильный способ

В учётную карточку в поле Account expiration time поставить прошедшую дату, а в поле пароля в начало пароля добавить запрещённый символ (это или «!», или «*»). Это можно сделать командой:

usermod -e YYYY-MM-DD -L

или

usermod --expiredate YYYY-MM-DD --lock

В этой команде ключ *-e* меняет дату, а ключ *-L* меняет пароль. Обращаем внимание, что вручную (текстовым редактором) ввести правильную прошедшую дату совсем не просто, в связи с тем, что дата в этом поле хранится во внутрисистемном виде (число секунд с 1970 года).

Соответственно, разблокировать пользователя можно командой

usermod -e 99999 -U

или

usermod --expiredate 99999 --unlock

При такой блокировке пользователю при попытке входа в систему будет выдано сообщение: «ваш account истёк».

Рассмотрим, почему нужно делать именно так: одновременным изменением поля Account expiration time и изменением пароля.

Пользователь может войти в систему и/или пользоваться её ресурсами следующим образом:

- непосредственно через терминал – так мы обычно входим, попадая в текстовую или графическую оболочку; при этом работает программа *login*, которая проверяет логин и пароль пользователя;

- по сети с помощью *telnet* или *ssh* – работает программа *login*;

- по сети по модемной линии (PPP или SLIP) – аналогично работает программа *login*;

- пользователь может поиметь доступ к своим файлам по сети через сервис *ftp* (случай «полного» пользователя, а не anonimous) – в этом случае проверкой логина и пароля занимаются демоны ftpd, или *wsftpd*, или

proftpd, или какой-либо иной демон сервиса *ftp*;

- пользователь может поиметь доступ к своей почте через POP- или IMAP-сервер – в этом случае проверкой логина и пароля занимается соответствующий почтовый демон;

- пользователь может поиметь доступ к своим файлам по сети через сервис samba – в этом случае проверкой логина и пароля занимается демон smbd;

- пользователь может запускать программы и копировать туда-сюда свои файлы с помощью RPC-сервиса (rlogin, rsh и др.) – тогда проверкой логина и пароля занимаются соответствующие демоны (rlogind, rshd и др);

- кроме того, есть ещё подключение с помощью удалённого рабочего стола – rdp, подключение через X-сервер, VNC, spice и прочие возможности поиметь систему.

И как видите, в некоторых случаях работает программа login, которая «правильно» обрабатывает учётную карточку, а в некоторых случаях работают совсем иные программы, которые далеко не всегда проверяют всё и полностью и, следовательно, дадут возможность пользователю воспользоваться системой. Чаще всего эти другие программы не обрабатывают поле Account expiration time, поэтому блокирование пользователя только с помощью этого поля не будет полным.

Способ с изменением пароля более полный, но и он не даёт полной блокировки пользователя. Например, если у пользователя в домашнем каталоге в файле .rhosts записаны адреса/имена компьютеров, «пользующихся доверием» и пользователь осуществляет доступ именно с этих компьютеров, то сервис RPC проверяет только адрес компьютера и логин пользователя, а пароль не проверяется.

Аналогичная проблема возникает с сервисом NFS, если в файле */etc/exports* указаны только адреса компьютеров. Но как раз в этом случае (случае RPC-сервисов) изменение поля *Account expiration time* должно сработать.

Таким образом, самый правильный и верный способ блокировки пользователя:

а) воспользоваться программой usermod, как описано выше;

б) при запуске сервисов всегда предпочитать решения из «родных» дистрибутивов (из своего репозитария) – как правило разработчики дистрибутивов проверяют совместимость программ.

3.6.2. Не совсем правильный способ

В поле пароля в начало пароля добавить запрещённый символ (это или «!», или «*») алгоритма MD5. Способ применим, если в качестве алгоритма шифрования паролей используется MD5.

Этот способ очень простой и, потому, очень часто используется. Однако, см. пункт 3.6.1.

3.7. Управление группами пользователей

3.7.1. Определение группы пользователей

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

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

Поскольку на каждую группу заводится учётная карточка, то каждая группа имеет своё символьное имя (group name) и числовое имя (groupID – gid), которые используются в системе аналогично именам пользователя. В учётной карточке группы сохраняется и состав групп (список пользователей, входящих в группу).

Основное назначение групп пользователей – определение прав доступа к различным файлам и каталогам. В unix/linux для каждого файла/каталога существует его владелец (это один из пользователей) и группа "особо допущенных" 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).

Учётные карточки группы сохраняются в файле */etc/group*. Формат этого файла:

group name
group password
group id
group members

Поле *group name* – символьное имя группы (имя для человека), также, как и логин пользователя, это цепочка символов из алфавита [a-z,0-9,-,_] начинающаяся с буквы и длиной до 256 символов.

Поле *group id* – числовое имя группы, принимает значения в диапазоне 0-65535. У первичной группы, как правило, *gid* равно *uid* пользователя. Это числовое имя первичной группы вписывается в учётную карточку пользователя (см. выше п. 3.2).

Поле group password пока не используется и в этом поле обычно ставится символ «*».

Поле group members – список (через запятую) логинов пользователей, входящих в группу.

То есть, для того, чтобы пользователь стал членом какой-либо группы, его логин надо вписать в поле group members файла /etc/group.

Таким образом, членами какой-либо группы являются:

- пользователи, в учетной карточке которых, стоит *group id* этой группы,

- плюс пользователи, перечисленные в файле */etc/group* в поле *group members* этой группы.

См. *man groupadd, man groupdel, man groupmod, man gpasswd.*

3.7.2. Особенности группирования пользователей

Прежде всего, сколько может быть в системе групп пользователей, пользователей в группе, групп у пользователя?

Ответ на этот вопрос может быть разным для разных дистрибутивов и версий unix/linux и определяется конфигурационными файлами дистрибутивов и конфигурационным файлом ядра операционной системы дистрибутива.

Прежде всего, сколько может быть групп в системе. Исходя из определения групп очевидно, что не более 65535. Однако, также как и для максимального количества пользователей в системе, в реальности действуют более жёсткие ограничения и обычно максимальное количество групп определяется в несколько сотен. Сколько?

Сколько может быть пользователей в группе? Здесь ограничением может выступать разрешённая длина строки учётной карточки, то есть, строки файла /etc/group. Очень часто она определяется в 1024 символа. Отсюда следует верхнее ограничение на количество пользователей в группе. Если группа «первичная», то нужно добавить самого пользователя – «хозяина» группы. В некоторых системах допускается слияние строк учётной карточки: то есть, при добавлении логинов пользователей в группу при достижении ограничения 1024 символа создаётся продолжение – следующая строка учётной карточки группы, которая начинается с теми же значениями

полей group name, group password и group id. Таких строк продолжения может быть до 25. Это утверждение требует проверки. Сколько на самом деле может быть пользователей в группе?

Во сколько групп можно включить пользователя? Утверждают, что не более чем в 16 групп, включая свою «первичную». Точнее, системные программы (какие?) будут признавать пользователя только членом только первых 16 групп, хотя вписать пользователя можно во много групп. Как проверить?

Категории групп. Группы делятся на три категории (см. */etc/login.defs, /etc/default/useradd)*:

- группы Администратора – это группы *root* и *wheel*;

- первичные группы системных пользователей, обычно gid = uid системного пользователя;

- остальные группы:

  - первичные группы обычных реальных пользователей, обычно gid = uid пользователя;

  - все прочие («логические») группы, gid которых произвольный (но уникальный!).

Создание и удаление групп. Как сказано выше, первичные группы всех категорий создаются автоматически при создании пользователя и получают при этом соответствующие значения полей group name и group id, а поле group members остаётся пустым. И поскольку файл /etc/group является текстовым и по структуре очень простым, то создание, удаление или изменение групп сложностей не вызывает. При этом нужно следить за оригинальностью имён групп, символьных и числовых. Если в системе групп много (на серверах), то эти операции лучше выполнять с помощью специализированных программ groupadd, groupdel и groupmod, которые «всё делают правильно».

Замечание о безопасности. Также, как и управление пользователями, управление группами пользователей – прерогатива Администратора системы, пользователя root. И хотя в документации на систему декларирована возможность создания «администраторов» групп с некоторыми правами по управлению группами, лучше этого не делать.

Изменение первичной группы пользователя – опасная операция, поскольку при этом необходимо у всех файлов/каталогов пользователя (которые могут оказаться не только в домашнем каталоге пользователя – в зависимости от того, какую роль играет пользователь в системе), также поменять соответствующий атрибут.

Включение или исключение пользователя в группу/из группы – операция более простая и, как правило, легко осуществима вручную текстовым редактором. Кроме того случая, когда управляем сервером корпоративного уровня и на нём числятся сотни и тысячи пользователей – в этом случае, как минимум, нужно воспользоваться программой groupmod.

4

УПРАВЛЕНИЕ ФАЙЛАМИ

4.1. Устройства хранения информации и их содержание

4.1.1. Проблемы с терминологией

Устройство – это нечто целое. Но в вычислительных системах при использовании устройств далеко не всегда устройства используются «в целом», как нечто единое и неделимое. Например, винчестеры («жёсткие диски») как раз тот случай, когда нечто целое для удобства использования предварительно делится на части и затем каждая часть используется достаточно независимо. И при этом возникает путаница с терминологией – части устройства тоже обзываются устройствами. Пока речь идёт об эксплуатации вычислительных систем обыкновенными пользователями («бухгалтерами» или «филологами»), вопрос: является ли «диск D» реально диском или всего лишь разделом диска – не столь важен. Но когда начинаем обсуждать технологическую сторону эксплуатации вычислительных систем – этот вопрос уже приобретает существенное значение.

К сожалению, подобная путаница имеет место быть не только на уровне «вульгарных» описаний работы ЭВМ, но и на уровне технической документации и интерфейсов специальных программ, не предназначенных для обычного пользователя (пример: документация по fdisk/gdisk). И к ещё более великому сожалению исправить данную проблему невозможно, поскольку документация и программы написаны, распространены и используются.

4.1.2. Внутреннее содержание устройства

Рассмотрим кратко среду, в которой хранятся файлы. В настоящее время основным устройством для хранения информации является жёсткий диск (винчестер). Различные USB-устройства имитируют жёсткие диски. Несколько особняком стоят CD/DVD, но основные положения разграничения доступа к файлам на них распространяются тоже.

1. **Разделы**. Прежде всего, устройство – это диск или дископодобное устройство, – разбивается на разделы. Почти всегда. В тех случаях, когда нам кажется, что устройство используется «в целом», чаще всего, на устройстве создаётся один раздел. Так исторически сложилось, по такому пути развивались технологии хранения информации – универсальность и единообразие на почве стандартизации.

Для описания разделов в начале устройств выделяется место («системная область»), в которое помещается «таблица разделов» (partition table),

Здесь возникает первая проблема: в разных операционных системах формат этих таблицы разный (отсюда термин «формат разбиения диска») и, следовательно, диск, разбитый на разделы средствами некоторой операционной системы, будет восприниматься другой операционной системой как диск ещё не подготовленный. И с этим ничего не поделаешь – переписывать операционные системы – не решение. Исключение – ОС linux – «высокообразованная» ОС, которая обучена понимать очень многие форматы разбиения дисков.

Разбиение дисков на разделы реализуется программами fdisk/gdisk. Есть графические аналоги.

2. **Файловые системы**. Внутри разделов создаются файловые системы. Они реализуют некоторые методы управления хранением информации, которые приняты в данной операционной системе. То есть, файловые системы операционнозависимы., в каждой ОС – своя файловая система, реализующая свой метод управления файлами. А метод – это всегда алгоритм.

Здесь возникает вторая проблема: операционные системы не понимают чужие файловые системы. Опять же исключение – ОС linux – «высокообразованная» ОС, которая обучена понимать очень многие файловые системы (более 50).

Процесс создания файловой системы в разделе называется форматированием раздела (именно раздела, а не устройства, форматируется всегда раздел). Это реализуется программой mkfs (см. man mkfs). По завершению работы этой программы раздел устройства готов к использованию, то есть, файловая система раздела может быть смонтирована – подключена к некоторому каталогу (точке монтирования), в котором мы увидим содержание этой файловой системы.

4.2. Файл и файловая система.

В этом пункте мы выясним, то файловая система – это не «набор файлов на диске»((C) Интернет) от слова совсем.

Устройство «диск» – это устройство блокового доступа, то есть, обмен с устройством (чтение/запись) реализуется блоками – порциями информации, которые называются секторами. Сектор – исторически сложившийся термин (по умолчанию равный 512 байт), оставшийся ещё от тех стародавних времён, когда использовалась геометрическая адресация CHS. В настоящее время используется адресация LBA – линейная (порядковая), в которой все сектора устройства по порядку пересчитываются и каждому сектору присваивается его порядковый номер в качестве адреса сектора. То есть, можно говорить про блоковый обмен данными с устройством, но термин «сектор» остался. Следовательно, пространство диска может быть представлено линейно (см. рис. 6).

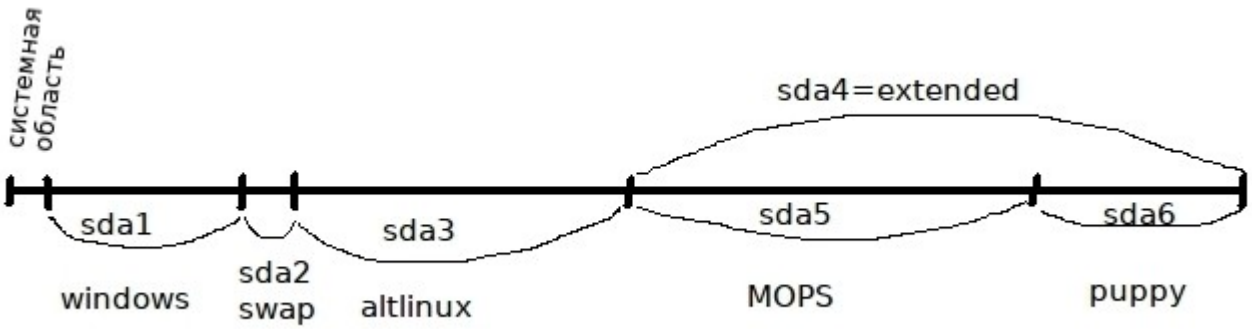


Рис. 6. Пространство устройства, разбитое на разделы программой fdisk в формате PC BIOS

Секторов на современных дисках очень много – счёт пошёл уже на триллионы, и чтобы ускорить работу с дисками, введены более крупные логические единиц: в Windows – кластер, в unix/linux – блок. Смысл терминов кластер и блок почти одинаковый и представляет собой порядковую последовательность секторов (то есть, следующих один за другим без разрыва) в количестве 1, 2, 4, 8, 16, 32, 64 или 128 секторов, то есть, в степени двойки. Ещё раз: термины введены для того чтобы ускорить работу с дисками; попутно решалась задача увеличения максимальных размеров разделов и файлов. Почему работа с дисками в результате ускоряется?

Однако (!) эти новые логические единицы задействуются только в файловых системах, то есть, уже на логическом уровне работы с устройствами. Это означает следующее:

- при разбиении диска на разделы программой *fdisk/gdisk* работа с диском идёт на физическом уровне – используются сектора;

- форматирование: программа *mkfs* (а в Windows – программа format) считывает из таблицы разделов номера первого и последнего сектора указанного раздела и его размер в секторах, преобразует (пересчитывает) раздел в блоки (кластеры), запоминает в суперблоке соотношение «блок/кластер = n секторов» и в дальнейшем уже работает с разделом только с использованием логической единицы блок/кластер; таким образом, формируется некоторое «метрическое» пространство, «мерой» в котором являются уже блоки/кластеры.

То есть, под размещение файла отводятся уже единицы «меры» – блоки/кластеры с некоторыми порядковыми номерами.

О. **Файл** – именованное место в файловой системе. Смысл: берём несколько блоков/кластеров, возможно, не смежных, и присваиваем им имя. Если блоки/кластеры – не смежные, то файл будет «фрагментированным».

О. **Имя файла** – цепочка символов длиной до N символов из алфавита [{a-z}{A-Z}{0-9}{спецсимволы}]. Число N определяется характеристиками файловой системы: в FAT (MSDOS) – 8, в ufs/jfs/xfs/ext (и большинстве других unix/linux) – 256, в ntfs (Windows) – 256.

То есть, пространство имён файлов – пространство символических имён («человеческих» – см п. 3.2.1.). Параллельно («для своего удобства») операционная система присваивает файлам свои имена – числовые – идентификаторы файлов.

Задача 1: Указываем имя файла, например, file.txt; мы знаем, что этот файл где-то там есть; нам нужно получить доступ к данным этого файла.

Решение: прежде всего нужно преобразовать (пересчитать) символическое имя файла в номера блоков, занятых файлом. Затем прочитать блоки с этими номерами.

Задача 2. Современные «универсальные» ОС, в которых мы обычно работаем, многозадачные, а unix/linux даже многопользовательские, то есть, одновременно могут выполняться несколько программ (процессоры же многоядерные!), которые вполне могут «захотеть» поиметь некоторые файлы в файловой системе. Практически одновременно. Как реализовать это так, чтобы процессы поимели свои файлы, не замечая друг друга?

Решение: Нужно «расшарить» файловую систему.

Кстати, «задача 2» – множественный доступ к файлам, возникает даже при последовательной работе: на компьютере поработал vasja – создал

свои файлы, потом поработал koljan – создал свои файлы. Как им потом получить доступ к своим файлам и только к своим файлам?

Эти две задачи как раз и решает файловая система – в этом её смысл и назначение.

Таким образом, файловая система – это метод организации хранения файлов на разделе устройства. Метод – подразумевает наличие алгоритма, реализующего этот метод. Алгоритмы вы все разрабатывали, например, когда писали программы: сначала определяли структуры данных, на которых будет работать алгоритм, а затем писали сам алгоритм – текст программы. Обязательно: сначала структуры данных, потом алгоритм. Точнее, это процесс итеративный, процесс схождения отлаживаемых версий к некоторой «неподвижной точке» – правильной программе.

Важным здесь является то, что алгоритм всегда подразумевает наличие структур данных, на которых он работает – они задают «область определения алгоритма». То есть, не бывает «абстрактных» алгоритмов, алгоритмы всегда конкретны.

Файловая система тоже включает два основных алгоритма, означенных выше: алгоритм преобразования символических имён в адреса блоков и алгоритм «расшаривания» файловой системы. То есть, «алгоритм + структуры данных = программа»((C) Н. Вирт [14]).

«Программа» файловой системы находится внутри операционной системы – это модуль ОС и называется «драйвер файловой системы» (не драйвер устройства – он находится ниже и обеспечивает физический уровень взаимодействия с устройством, а именно драйвер файловой системы – он работает поверх драйвера устройства и обеспечивает логический уровень взаимодействия с устройством [1]). И модули файловых систем в каждой ОС – свои, специфичные, написанные в соответствии с принципами и концепциями этой ОС, отсюда операционнозависимость файловых систем.

А, вот, структуры данных . . . Данные о том, какие файлы и где (в каких блоках) находятся на hdd или флешке, должны, очевидно, храниться на самом устройстве.

Итого, получается: алгоритм файловой системы – общий (алгоритм ext4 – это в linux, алгоритм ufs -это в FreeBSD, алгоритм ntfs – это в Windows), а данные всегда специфичны для каждой флешки, или раздела диска, или . . . на чём там мы можем сохранить файлы с компьютера. Эти данные о файлах записываются в структуры данных (обычно, таблицы), которые создаются на разделе при его форматировании. То есть, **форматирование раздела (работа программы mkfs/format) – это процесс разметки**

раздела и формирование на нём таблиц (пустых!), в которые мы потом будем записывать данные о файлах/каталогах, атрибуты файлов/каталогов.

Точность описания файла/каталога определяется форматом этих таблиц. В некоторых файловых системах таблицы простые (содержат мало граф), в других более сложные. Самая «хитрая» в этом смысле файловая система – ntfs, в ней описание файла вообще не таблица, а списковая структура и она различается для разных файлов и может быть очень сложной и длинной (несколько килобайт), с повторами.

Подробности смотрите в другом учебном пособии [Чичев А.А., Чекал Е.Г. Операционные системы. Часть 2. Файловые системы. – готовится к изданию].

4.3. Разграничение доступа к файлам и каталогам

4.3.1. О пользователях, файлах, процессах и правах

Как сказано выше, unix/linux системы многозадачные и многопользовательские и это означает, что в ней должен быть механизм, ограничивающий доступ программ и пользователей к каталогам и файлам. Здесь «доступ» означает возможность читать, изменять, переименовывать, создавать файлы, менять атрибуты файлов и каталогов (в том числе, те, которые определяют «доступ»), запускать файлы, если являются исполняемыми.

Однако, нужно отметить следующее:

- пользователь вносит изменения в файлы и каталоги не пальчиками и не с помощью отвёртки (иголочки), «ковыряясь» в файловой системе, а всегда с помощью каких-то программ (редакторов, "коммандеров", системных утилит для копирования, удаления файлов и т. п.), которые в момент выполнения являются процессами;

- далеко не все программы запускаются пользователями; достаточно много (демоны) запускаются при старте системы; некоторые могут запускаться в определенные моменты времени (с помощью программы cron), или запускаться по мере необходимости для обслуживания запросов, приходящих по сети (с помощью суперсервера xinetd); кроме того, существует программы, которые для выполнения некоторых вспомогательных действий сами запускают другие программы (в этом случае говорят, что процесс-"родитель" запустил процесс-"потомок"); понятно, что хотелось бы и этим программам (процессам) ограничить доступ к файлам;

- в некоторых случаях полезно, чтобы программа, запущенная пользователем, имела больше прав, чем обычно имеет этот пользователь; например, обычный пользователь не может даже читать файл, в котором "спрятаны" пароли всех пользователей; в то же время, любой пользователь может иметь возможность поменять свой пароль, не обращаясь для этого к администратору; то есть, программа, которая это делает (passwd) в момент выполнения должна иметь права намного большие, чем пользователь, который ее запускает.

Далее отметим следующее (см. также [1]):

- у каждого процесса в PCB запоминается не только его конкретное имя – pid, но и идентификатор пользователя uid, от имени которого он работает; обычно этот uid совпадает с uid'ом пользователя, который запустил этот процесс;

- процессы, которые запустились автоматически, тоже имеют uid, как будто их запустил реальный пользователь; чей именно uid получают эти программы обычно определяется теми программами, которые их стартуют (операционная система, cron, суперсервер и т. д.); в простейших случаях программы-"потомки" просто "наследуют" uid от программы-"родителя", но некоторые "родители" могут запускать программы с другим uid (не совпадающим с собственным);

- некоторые программы в процессе выполнения могут поменять свой uid и, соответственно, получить права, которые сам пользователь, который их запустил, не имел; естественно, для того, чтобы программа могла это сделать, администратор должен "разрешить" ей такое поведение; и кстати, изменение uid делается не только для того, чтобы "расширить" права программы, но и наоборот – "сузить" до прав какого-нибудь конкретного пользователя;

- если процесс может изменять свой uid, то различают "реальный uid" и "эффективный uid" (а есть еще "сохраненный uid"); реальный uid – это идентификатор юзера, который запустил процесс (точнее, это uid процесса-"родителя"); а эффективный – это новый uid, который процесс получил во время выполнения;

- права на файл/каталог определяются по "эффективному uid" процесса; в простейшем случае, когда uid не меняется, "реальный" и "эффективный" uid'ы совпадают и можно говорить просто об uid процесса; или даже просто о правах пользователя (а не процесса) на файл.

4.3.2. Разграничение доступа к файлам и каталогам – основные понятия

С точки зрения файла/каталога все пользователи делятся на три категории:

- владелец (хозяин) этого файла/каталога; хозяин всегда только один;

- группа хозяина файла/каталога (группа «особо лопущенных»; см. п. 3.7.);

- все остальные.

Следовательно, можно определить три различных уровня допуска (набора прав допуска) для каждого файла/каталога: для хозяина, для пользователей из группы хозяина и для прочих – три набора прав допуска. То есть, в описание файла/каталога записываются:

- *uid* пользователя-хозяина и права для него,

- *gid* группы хозяина – определяется группа «допущенных» (причём, хозяин не может произвольно менять состав группы – см. выше п. 3.8.) и права для неё,

- и права на доступ для всех остальных.

Пример. Вводим в терминале команду *ls -l* и видим что-то типа этого (см. на рис. 7):

```
-rw-r--r--  1  student  student   4297  13 мар 21:45 file1
-rw-r--r--  1  student  student   1502  13 мар 22:09 proga.c
-rw-r--r--  1  student  student  51054  12 мар 20:11 proga.odt
-rwxr-xr-x  1  student  student  21230  21 мар 22:12 proga

 \_______/     \_____/  \_____/  \___/  \__________/ \_______/
 "права"     владелец  группа   длина     дата      имя файла
```

Рис. 7. Вывод команды *ls -l*

На рисунке мы видим:

- самая первая колонка (один символ «-») определяет тип файла;

- следующие 9 символов «rw-r—r--» определяют режим лоступа к файлу;

- следующая колонка из единичек – сколько имён у файла; в данном случае у всех файлов по одному имени;

- 4-ая колонка – логин пользователя-хозяина файла;

- 5-ая колонка – логин группы хозяина файла, это и есть группа «особо допущенных»:

- далее длина файла, дата последнего файла и имя файла.

В таком виде («очеловеченном») отображает атрибуты файла команда ls. На самом деле, в описании файла в системе (в inode) атрибуты файла хранятся в «оцифрованном» виде:

- тип файла в виде кода типа файла (6 бит) плюс 9 бит режима доступа;
- далее количество имён у файла;
- далее хранится *uid* хозяина;
- далее *gid* группы хозяина;
- и т. д., все прочие атрибуты.

Рассмотрим подробнее 9 битовое поле режима доступа – их программа ls показывает в виде трёх групп по три бита (см. рис.8).

r	w	x	r	w	x	r	w	x

Рис. 8. Девятибитовое поле режима доступа

Основные биты доступа.

Значение символов: *r* – чтение; *w* – запись/изменение; *x* – выполнение.

Самая правая тройка – режим доступа для всех прочих пользователей. Вторая справа тройка – для группы хозяина. Третья справа тройка – для хозяина файла. Если бит установлен, то программа *ls* показывает соответствующую букву, если бит не установлен, то «-».

Их смысл различается для файлов и каталогов.

Для файлов.

r-- – read, означает, что пользователю разрешается читать содержимое файла, копировать файл. То есть, он может открыть файл редактором, но сохранить файл (записать обратно) под этим же именем не сможет; кстати Writer при открытии файла, который разрешено только читать, предупредит, что файл открыт в режиме «только для чтения». Если файл – программа, то запустить его на исполнение нельзя.

-w- – write, означает, что можно писать в файл, дописать что-нибудь в конец, или вообще затереть содержимое файла, например, командой ls -l > file.txt. Однако, нельзя изменить файл в каком-либо редакторе, так как для этого нужно прочитать файл, а это не разрешено. И разрешение на изменение файла не означает, что можно переименовать или удалить файл (это определяется правами на каталог).

--x – eXecute, означает, что можно запустить на исполнение файл, если он представляет собой программу или скрипт. Этот бит – первый признак по которому система догадывается о том, что файл исполняемый. Но одного лишь установленного бита «x» недостаточно для запуска программы.

Для каталогов.

Каталог, известно, тоже файл [1], а вовсе не «место, где хранятся другие файлы» ((C)Интернет).

r-- – read, – означает, что можно читать этот каталог, то есть, мы можем узнать, какие файлы/каталоги хранятся в этом каталоге, но мы не можем «зайти» в этот каталог или что-то сделать с файлами/каталогами в нём, даже если права самих файлов это позволяют. То есть, мы не можем выполнить команду cd в этот каталог.

-w- – write, означает, что можно изменять каталог: создавать новые файлы в каталоге, копировать файлы в каталог, переименовывать и удалять файлы. Однако, если на каталог установлен только этот бит, то это не значит, что с каталогом можно делать всё, что выше написано.

--x – eXecute, означает, что в каталог можно «войти» («выполнить» каталог). Кстати, даже если внутренние подкаталоги имеют "нормальные" права, а вышестоящий каталог – нет (отсутствует бит "x"), то пользователям не удастся "занырнуть" в подкаталоги, минуя вышестоящую, так как система проверяет полный путь до конечного каталога или файла и, если хотя бы один из компонентов этого пути не имеет соответствующего бита, то пользователю будет отказано в доступе.

Замечание 1. Не путайте права на каталоги и права на файлы. Например, если права на каталог не позволяют пользователю удалить файл, находящийся в нём (нет бита "w"), это не означает, что пользователь не сможет "удалить содержимое" файла. С другой стороны, если пользователь имеет право менять содержимое каталога, то он сможет удалить или переименовать любой, находящийся в нём файл, даже если права на самом файле не позволяют ему не то что писать в файл, но и читать его.

Замечание 2. Для пользователя root (и программ, у которых «эффективный uid» – root) действуют другие правила, на то он и root. Пожалуй, за одним исключением: если на файле программы не стоит бит «x», то даже root не сможет убедить систему, что файл является программой и его можно выполнить (!), пока не установит этот бит.

4.3.3. Разграничение доступа к файлам и каталогам – первый уровень

Комбинации битов – для файлов. Обычно права на файл-непрограмму (например, для "всех прочих") устанавливаются так:

--- никаких прав (нельзя ни читать, ни изменять содержимое),

r-- только чтение,

rw- чтение и запись (изменение) файла.

Если файл является "исполняемым" (программа), то права могут выглядеть так:

--- опять же, никаких прав (читать нельзя, запускать нельзя),

r-x можно запустить файл-программу на выполнение,

rwx можно не только запустить, но и что-нибудь в нём поменять (перетранслировать).

Остальные комбинации (например -w- или --x) кажутся бессмысленными, однако. . .

Права -w- означают, что юзер из соответствующей категории не может прочитать этот файл, но может в него писать. Кажется, что для "исполняемого" файла это сочетание бессмысленно, однако на самом деле смысл этой комбинации – запустить программу нельзя, но "покорежить" что-нибудь внутри – пожалуйста, то есть, смысл вполне определённый. С другой стороны, если этот файл представляет собой "лог" (протокол работы), или почтовый ящик, то такие права имеют вполне здравый смысл. Например, вы допускаете, что другие пользователи (или какие-нибудь программы-демоны) будут дописывать сюда свои сообщения (писать протокол работы), но не хотите, чтобы те же пользователи могли посмотреть – что туда понаписали другие. Правда, при этом никто не мешает этим пользователям просто не читая, записать в этот файл что-нибудь, "похоронив" при этом все старое содержимое файла. Или даже скопировать туда файл нулевой длины (или /dev/null), при этом, естественно, вообще никакого "содержимого" у вашего файла не станет. Решение – поставить на этот файл флаг "только дозапись". Он не помешает вам читать свой файл, а другим пользователям дописывать в него что-нибудь. Правда, не даст даже вам стирать из файла все лишнее или удалить файл, пока вы не "снимите" флаг.

Права --x – комбинация означает разрешение запустить эту программу, но дело в том, что для того, чтобы программу запустить, её надо «прочесть», а это не разрешено.

Права -wx – для исполняемого файла, пожалуй, смысла не имеет (см. выше комбинацию -w- на исполняемом файле). Вы просто даете возможность изменить содержимое, причем "вслепую" (поскольку посмотреть это

содержимое нельзя). Результатом может быть только порча вашей программы.

Права r-- – комбинация для файлов-программ смысла не имеет. Имейте в виду, что здесь вы даёте возможность скопировать вашу программу.

Комбинации битов – для каталогов. Для каталогов обычно права доступа устанавливаются такие:

--- никаких прав,

r-x нормальные права для "посещения" каталога, но без права изменения (создавать/удалять/переименовывать файлы),

rwx полный доступ (делай в каталоге что хочешь).

Остальные комбинации не всегда бессмысленны.

В остальных комбинациях решающую роль играет наличие бита доступа --x (доступ к содержимому), если его нет, то любая комбинация из двух остальных ничего полезного не даст.

Комбинация r-- даст возможность получить список файлов в этом каталоге, например командой ls, причём, только имена файлов и ничего более. В такой каталог нельзя "войти" командой cd. И, даже если внутри него подкаталоги имеют вполне нормальные права доступа (например r-x), попасть в них будет невозможно.

Комбинации -w- и rw- для каталогов имеют еще меньше смысла. Все равно, изменить файл-каталог с такими правами доступа не удастся.

А, вот, сочетания --x и -wx на самом деле вполне осмысленны.

Таким способом можно сделать "полусекретный" каталог. "Секретность" его заключается в том, что никто посторонний не сможет посмотреть – что за файлы и подкаталоги в нём находятся. Но, если вы своим друзьям сообщите – как называется файл находящийся там, они смогут без проблем его взять оттуда или посмотреть его прямо на месте. То есть, нужно конкретно знать имена содержимого, чтобы этим содержимым воспользоваться (подразумевается, что права на файлы и подкаталоги уже достаточны, чтобы ими воспользоваться).

Конечно, "секретность" в этом случае будет не полной, поскольку, если посторонний каким-то образом узнает правильное название файла или поддиректории, то получит те же возможности, что и "близкие друзья".

Права -wx отличаются от предыдущего тем, что "близкие друзья" могут писать в эту директорию. Они могут скопировать туда файл, удалить его, скопировать оттуда, переименовать (если, конечно, знают его точное

название). Опять же, посторонний сможет разве что записать туда файл, который вы не просили. Удалить там что-нибудь или переименовать, не зная названий файлов (и подкаталогов), он не сможет.

Обычный порядок назначения прав различным категориям пользователей на файл или каталог следующий:

владелец – полные права (rwx),

группе доверенных лиц – без права изменения (r-x),

всем остальным – никаких прав (---).

Однако, поскольку группы составляет root и рядовые пользователи, как правило, не выбирают себе соседей по группе, то их файлы и каталоги имеют одинаковые "допуски" для группы и "всех остальных". Например, для "несекретных" файлов (директорий) – rwxr-xr-x, а те, которые владелец хочет оградить от посторонних rwx------.

При этом, возникает вопрос, что будет, если "всем остальным" дать доступ, пусть и ограниченный, а для группы "особо приближенных" – наоборот убрать, например, так rwx---r-x?

Тогда группа «приближённых» превратится из группы "особо допущенных" в группу "особо нелюбимых". То есть, оформится некий "черный список", куда можно занести (естественно, сделать это может только root) всех тех, кто не пользуется доверием. Обратите внимание, что система, проверяя права конкретного пользователя по отношению к файлу, сначала проверяет – не является ли он хозяином, потом – не входит ли он в группу хозяина, а уже после этого относит его ко "всем остальным". Так что, даже если "все остальные" имеют допуск к файлу, но пользователь имел несчастье попасть в соответствующую группу, к нему будут применяться права доступа этой группы.

4.3.4. Разграничение доступа к файлам и каталогам – второй уровень

Кроме рассмотренных выше битов (чтение, запись и "исполняемость"), которые устанавливаются раздельно по трем категориям пользователей, есть еще три бита доступа, которые можно отнести к файлу/каталогу в целом, поскольку их действие не зависит от того какой пользователь (в смысле из какой категории) пытается обратиться к файлу/каталогу. Смысл этих битов состоит в изменении некоторых свойств файлов/каталогов. На рисунке 9 список битов доступа и флагов показан «рядышком», в одном 16-битном слове, на самом же деле в inode файла основные биты доступа, дополнительные биты доступа и флаги хранятся в разных местах.

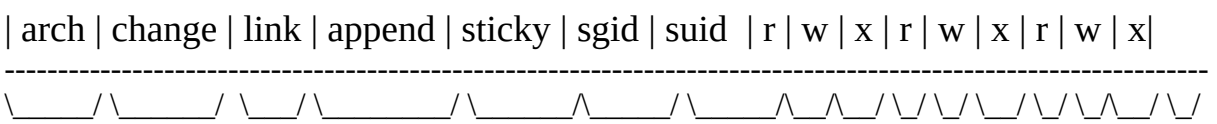


Рис. 9. Полный список битов доступа и флагов.

Бит suid. Расшифровывается как Set user ID, то есть, "установить идентификатор пользователя". Его обычно называют "суидный" бит, а файлы, на которых он установлен – "суидными".

Смысл его в следующем: если он установлен на файле, который является программой, то при выполнении эта программа автоматически меняет "эффективный uid" на идентификатор того пользователя, который является владельцем этого файла. То есть, не зависимо от того – кто запускает эту программу, она при выполнении имеет права хозяина этого файла.

Обычно это делается для того, чтобы пользователь мог выполнить действия, которые требуют привилегий root'a (например, поменять свой пароль). Очевидно, что владельцем такой программы должен быть пользователь root.

И также понятно, что такая программа является "потенциально опасной". В "нормальном" случае она не позволит обычному пользователю сделать то, что выходит за пределы его полномочий (например, программа passwd разрешит пользователю изменить только собственный пароль, но не пароли других пользователей). Но, даже незначительная ошибка в такой программе может привести к тому, что "злоумышленник" сможет заставить ее выполнить еще какие-нибудь действия, не предусмотренные автором программы. То есть, это один из способов "взлома" системы – заставить какую-нибудь из "суидных" программ выполнять действия необходимые "взломщику".

Правах доступа у такого суидного файла выглядят так **s****** (если установлен бит x для пользователя) или **S****** (если соответствующего бита x нет).

Этот бит не несет никакого смысла, если его поставить на каталог, хотя это можно сделать.

Бит sgid. Расшифровывается как Set group ID, то есть, "установить идентификатор группы".

Здесь смысл аналогичен смыслу предыдущего бита. Только меняется не идентификатор пользователя, а идентификатор группы. То есть, при выполнении этого файла он имеет такие права, как будто его запустил кто-то

из группы, которая приписана к этому файлу.

Права доступа у такого файла выглядят так *****s*** (если установлен бит x для группы) или *****S** (если соответствующего бита x нет).

Также как и в предыдущем случае, бит sgid для неисполняемых файлов смысла не имеет.

Однако, для каталогов смысл этого бита следующий: когда файлы создаются в таком каталоге, в их атрибутах проставляется группа та же, что и у каталога. Другими словами, файлы, создаваемые в таком каталоге "наследуют" группу от каталога.

Бит sticky. Никак не расшифровывается, переводится как "липкий".

Выглядит в правах доступа так ********t (если бит x для "всех остальных" установлен) или ********T (если соответствующего бита x нет).

Для каталогов его смысл заключается в том, что удалить файл из такого каталога (или переименовать) может только владелец файла. Выше сказано, что в обычном случае (без такого бита) возможность удалять файлы (как и создавать, переименовывать) определяется правом записи (бит w) на каталоге. То есть, если какой-либо пользователь принадлежит к категории, для которой разрешена запись в директорию, он может удалить в ней любой файл, независимо от атрибутов (владельца, группы, прав доступа) самого файла.

Применяется этот бит, обычно на каталогах, которые являются "публичными" (например, /tmp). 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.

Этот бит может ставиться также на исполняемые файлы. В этом случае он означает, что программа, даже после завершения работы, должна оставаться в памяти (конечно, не в ОЗУ, а в swap'e).

Как сказано в man'e, это полезно для "часто используемых исполняемых файлов общего пользования". Привести пример.

4.3.5. Разграничение доступа к файлам и каталогам – флаги доступа

Кроме уже рассмотренных атрибутов (владелец, группа, биты доступа, дополнительные биты доступа), существует еще один параметр, который ограничивает возможные действия с файлом – "набор флагов" (см. рис. 9).

Замечание. По умолчанию ни команда ls, ни большинство "коммандеров"/"файлменеджеров" флаги не показывают. Для того, чтобы увидеть эти флаги на файлах/каталогах нужно использовать команду lsattr (см. man lsattr).

С помощью флагов можно запретить изменять содержимое файла, его название или и то и другое.

Флаги являются более "сильнодействующим средством", чем права доступа. Они действуют одинаково для всех пользователей (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. Это свойство флагов часто используют в качестве дополнительной меры по усилению безопасности системы.

Флаг «A» – отмена обновления (модификации) записи atime (время доступа к файлу). Это позволяет избежать дополнительных дисковых операций ввода/вывода (например, для систем на портативных компьютерах).

Флаг «a» – разрешено лишь добавлять записи; только суперпользователь может установить или очистить этот флаг.

Флаг «c» – данные файла автоматически упаковываются (сжимаются) на диске ядром операционной системы. Операция чтения информации из этого файла возвращает несжатые данные. Запись информации в такой файл сопровождается предварительной её упаковкой и, наконец, последующим сохранением на диск.

Флаг «D» – устанавливается на каталоги; когда такой каталог модифицируется, внесенные изменения синхронно записываются на диск; это эквивалентно применению опции монтирования `dirsync' к подмножеству файлов.

Флаг «d» – для файла с установленным флагом «d» не выполняется резервное копирование, когда запущена программа dump.

Флаг «E» – используется экспериментальными заплатами сжатия для определения того, что сжатый файл имеет ошибку сжатия.

Флаг «I» – используется кодом для хеш-деревьев (htree), чтобы указать, что каталог находится позади индексированных хешированных дере-

вьев. Это используется в системах индексирования содержимого разделов для ускорения поиска файлов (пример, сервис nepomuc).

Флаг «i» – файл с установленным флагом «i» становится полностью не модифицируемым (недосягаемым): он не может быть удален или переименован, никакие ссылки не могут быть созданы на этот файл и никакие данные не могут быть записаны в него. Только root может установить или очистить такой флаг.

Флаг «j» – данные файла, прежде чем будут записаны непосредственно в файл, сохраняются в журнал ext3. Правда, это происходит только в том случае, если файловая система смонтирована с опциями "data=ordered" или "data=writeback". Когда файловая система смонтирована с опцией "data=journal" все данные файла уже журналируются и этот флаг не имеет никакого эффекта. Только root может установить или очистить этот флаг.

Флаг «s» – при удалении файла с установленным флагом «s» выполняется обнуление его блоков и запись их обратно на диск. То есть, восстановить такой файл будет нельзя от слова совсем.

Флаг «S» – для файла с флагом «S», внесенные изменения в файл синхронно записываются на диск; использование этого флага эквивалентно применению опции монтирования `sync' к подмножеству файлов.

Флаг «T» – каталог с установленным флагом «T» будет считаться, как расположенный на вершине иерархии каталогов с целью использования метода распределения блоков по Orlov (который применяется в системах с Linux 2.5.46 или выше).

Флаг «t» – файл с установленным флагом «t» не будет иметь в конце (в блоке на диске) дописанных (склеенных с ним) частичных фрагментов (хвостов) других файлов (для тех файловых систем, которые поддерживают "склеивание хвостов" файлов). Это необходимо для тех программ (например, LILO), которые непосредственно («напрямую») обращаются к файловой системе и не понимают "склеивание хвостов" файлов. Однако, здесь следует отметить, что файловые системы ext2 или ext3 не поддерживают для файлов "склеивание хвостов" (всё же, кроме некоторых весьма экспериментальных заплат).

Флаг «u» – при удалении файла с флагом «u», его содержимое сохраняется (остается не тронутым) на диске. Это позволяет пользователю в последующем восстановить такой файл.

Флаг «X» – используется экспериментальными заплатами сжатия для определения того, что к необработанному содержимому сжатого файла

можно получить непосредственный доступ.

Флаг «Z» – используется экспериментальными заплатами сжатия для определения того, что сжатый файл является необработанным.

Замечание 1. Перечисленные флаги и их значение имеют место быть для ОС Linux и файловых систем ext2/3/4. Для других unix и unix-подобных ОС и их файловых систем существуют свои наборы флагов доступа к файлам/каталогам, то есть, в отличие от базовых правил разграничения доступа (см. п 4.3.2 – 4.3.4.), которые определяют общие правила, флаги разграничения доступа специфичны для каждой файловой системы.

Замечание 2. Очевидно, что флаги используются не для того, чтобы "защитить" систему от собственного администратора. Он все равно, при желании, сможет убрать эти флаги (и удалить/изменить соответствующие файлы). Правда, для этого ему придется перезагрузить систему, причем возможно с консоли. А вот "взломщик", проникший в систему, даже если каким-то образом и получит root'овские права, не сможет изменить "жизненно важные файлы" (если, конечно, они защищены флагами) или "почистить логи" (если они защищены флагами), чтобы "замести следы" своего пребывания в системе.

Замечание 3. Не для всех флагов есть в linux полная реализация пока.

Замечание 4. Описание флагов в пункте 4.3.5. не является полным.

4.4. Другие вопросы управления файлами

4.4.1. С какими правами файл «рождается»

Как было сказано выше (см. п. 4.3.1.), работа с файлами реализуется через посредство каких-либо программ. В то же время, в системе существуют системные вызовы, которые позволяют менять владельца файла, группу и права доступа. То есть, программа порождающая файл может может создать файл с любыми атрибутами.

Правда, следует заметить, что большинство программ этими функциями не пользуются, поэтому можно сказать, что "в большинстве случаев" атрибуты создаваемых файлов все таки подчиняются нескольким простым правилам.

1. Владелец файла. Владелец файла определяется "эффективным uid'ом" процесса, который его создает. То есть, в большинстве случаев владельцем файла будет тот пользователь, от имени которого работает процесс. Если же программа "суидная", то есть, во время выполнения имеет права того пользователя, которой является владельцем этой программы, то, соответственно, все файлы, порожденные этой программой будут принадлежать хозяину программы, а не пользователю, который ее запустил.

Кстати, если даже в программе используются системные вызовы, которые меняют владельца файла, они сработают только в том случае, если ее "эффективный uid" будет uid root'a. То есть, если ее запустит пользователь root или она является "суидной" и ее владелец root.

Другими словами, какие бы программы не использовал обычный пользователь (если, конечно, они не "суидные") он может создать файлы владельцем которых будет только он. "Подарить" файл кому-нибудь другому обычный пользователь не может.

2. Группа. Аналогично присваивается группа файла, то есть, файлу присваивается «первичная» группа пользователя, создающего файл. Иначе говоря, та группа, от имени которой работает процесс, создающий файл.

Однако, если на каталоге, в котором создаётся файл, стоит бит sgid, то группа создаваемого файла будет наследована от каталога.

3. Права доступа. Они определяются параметром umask среды пользователя. Этот параметр определяет биты прав доступа, которые не надо выставлять у создаваемого файла.

Например, если umask = 0, то

- всем создаваемым файлам назначаются права 666 (rw-rw-rw-);

- всем создаваемым каталогам и исполняемым файлам назначаются права 777 (rwxrwxrwx).

Увидеть текущее значение umask можно с помощью команды

umask<Enter>

Для того, чтобы определить новое значение umask, нужно в качестве параметра этой команды указать число, определяющее биты, которые не надо выставлять у создаваемых файлов. Например, если вы хотите, чтобы у категории "все остальные" вообще не было никаких прав, а "группе допущенных" не ставился бит разрешающий запись, то umask должна выглядеть как 027.

4.4.2. Изменение прав доступа при копировании/перемещении файла

Копирование файлов/каталогов – команда cp, эта команда создаёт новый файл на новом месте.

Перемещение файла/каталога – команда mv, эта команда изменяет место расположения файла и, может быть, имя файла/каталога

Оно так, даже если вы работаете в «коммандере/файл_манагере», потому что, в unix/linux «коммандеры/файл _манагеры» – это, как правило, оболочки над утилитами.

Следовательно, если файл копирует обычный пользователь «по умолчанию» (или, как выражается mc, «с исходным шаблоном»), то действуют те же правила, что и при создании файла (см. п. 4.4.1). Это положение распространяется и на root'a. Однако, root, в отличие от обычного пользователя, может изменить поведение команды cp: у этой команды есть ключ -p – сохранять права, который означает, что надо сохранить все атрибуты (владельца, группу и права) при копировании.

Биты sticky, setuid, setgid при копировании сбрасываются, кроме случая, когда при копировании задан ключ -p. В последнем случае, они сохраняются, если нет конфликта, иначе тоже сбрасываются.

Для обычного пользователя ключ -p может сохранить только основные биты доступа.

Однако, если в месте назначения файл с таким именем уже существует, то новый файл не создаётся, а происходит замещение его содержимого (!), если пользователю разрешено писать в существующий файл. При этом, права доступа, хозяин и группа останутся неизменными, а биты suid и sgid будут сброшены.

При перемещении файла все атрибуты сохраняются, в том числе suid и sgid. Подразумевается, что перемещает файл сам хозяин файла в своём каталожном дереве. А чтобы обычный пользователь мог переместить чужой файл, он должен иметь право записи в оба каталога – исходный и места назначения. А такое возможно?

4.4.3. Корректировка атрибутов файла

1. Владелец (хозяин). Поменять хозяина файла может только root.

2. Группа. Группу может поменять хозяин файла, в том случае, если он является членом этой новой группы. А также root.

Хозяин и группа меняются командой chown («change own», см. man chown). Синтаксис команды:

chown [опции] пользователь[:группа] файл/каталог

В этой команде важной и часто используемой опцией является опция -R – рекурсивное изменение владельца для каталогов и их содержимого.

3. Права доступа. Права доступа может поменять хозяин файла у своего файла. А также root.

Права доступа меняются командой chmod («change mode», см. man chmod).

Синтаксис команды:

chmod [опции] режим_доступа файл/каталог

Опции в команде используются нечасто.

Формат символьного представления режима доступа следующий:

```
  [ u g o a ]   [ [ + – = ]  [ r w x X s t u g o ] ]
  \_______/     \_______/    \_________________/
пользователь   операция     биты доступа
```

пользователь:

u (user) хозяин;

g (group) группа;

o (other) все остальные;

a (all) все три категории;

операция:

+ добавить право;

– удалить право;

= присвоить только эти права файлу/каталогу;

биты доступа:

r – чтение;

w – запись;

x – выполнение (или доступ к каталогу);

X – выполнение, если файл является каталогом или уже имеет право на выполнение для какого-нибудь пользователя;

s – setuid- или setgid-биты;

t – sticky-бит;

u – установка для остальных таких же прав доступа, которые имеет пользователь, владеющий этим файлом;

g – установка для остальных таких же прав доступа, которые имеет группа файла;

o – установка для остальных таких же прав доступа, которые имеют остальные пользователи (не входящие в группу файла).

Формат числового представления режима – строка из не более четырех восьмеричных цифр (от нуля до семи), которые складываются из битовых масок 4, 2 и 1. Любые пропущенные разряды дополняются лидирующими нулями.

Первая цифра выбирает установку идентификатора пользователя (setuid) (4) или идентификатора группы (setgid) (2) или sticky-бита (1).

Вторая цифра выбирает права доступа для пользователя, владеющего данным файлом: чтение(4), запись (2) и выполнение (1).

Третья цифра выбирает права доступа для пользователей, входящих в данную группу, с тем же смыслом, что и у второй цифры.

А четвертый разряд выбирает права доступа для остальных пользователей (не входящих в данную группу), опять с тем же смыслом.

Замечание. При работе с командой chmod настоятельно рекомендуется использовать цифровое представление атрибутов файлов/каталогов. Почему? Так быстрее, а время – деньги.

Пример: Пусть файлы *proga.c* – исходник, *proga.odt* – руководство, а proga – исполняемый файл. Тогда

*chmod 644 proga.**

chmod 755 proga

4.4.4. Установка/изменение флагов доступа к файлу/каталогу

Увидеть установленные на файлах/каталогах флаги можно с помощью команды lsattr (см. man lsattr).

Установить нужные флаги можно с помощью команды chattr (см. man chattr).

Команды похожи на соответствующие команды ls и chmod. У команд lsattr и chattr есть опция -R – рекурсивный обход каталогов.

Синтаксис команды chattr:

chattr [-RV] [-v version] [mode] files

где mode задаётся в символьном виде в формате +-=[AsacDdIijsTtu].

4.5. Заключительные замечания о разграничении доступа к файлам

Задача 1. Наделить разными правами (по отношению к конкретному файлу) разные группы пользователей, причем именно группы, а не "отдельного пользователя" и не "всех зарегистрированных пользователей".

Задача 2. Составлять группы может только root. Поэтому, если владелец файла захочет по своему усмотрению распределить права на свой файл различным пользователям, то оказывается, что он практически не имеет для этого никаких механизмов. Как сделать?

Обе задачи – вполне реальные ситуации при групповой работе над документами.

Пример. Пусть отдел ведёт некоторый проект. В отделе есть сектор программеров и им доступны на запись файлы *.h, *.c, *.so (rw-rw----) и исполняемый файл proga; им же файл proga доступен для исполнения (rwxrwx---). В этом же отделе есть сектор тестирования, которым файл

proga доступен для исполнения (–-r-x---). В этом же отделе есть группа технических писателей, разрабатывающих эксплуатационную документацию. Они имеют право изменять файлы *.odt (rw-rw----). Все другие сотрудники отдела имеют право только читать эти файлы (r—r-----). Сотрудникам из других отделов к файлам этого проекта вообще нет доступа (----).

Решение 1.1 (Паскаль И.). Предлагается распределить права владельца на группу. Обычно, владелец отличается тем, что имеет право записи (модификации файла). Соответственно, "группе допущенных" можно дать права только для чтения (и/или исполнения) файла, а "всем остальным" вообще не давать никаких прав.

Так вот. Если для изменения файла можно использовать только одну определенную программу, то надо сделать эту программу (или ее копию) "суидной", то есть такой, которая при выполнении меняет свой "эффективный userID" на ID владельца. Естественно, владельцем такой программы должен быть тот же пользователь, что и владелец файла, который мы хотим модифицировать, а права на запуск этого файла надо дать только той группе, которой мы хотим дать права на модификацию. Кстати, "владелец" файла может быть и фиктивным пользователем, то есть зарегистрированным в системе только для этой цели и не соответствовать никакому реальному человеку.

Таким образом, нужно создать две разные группы, в первую (writers) внести только тех пользователей, которые имеют право изменять файл, а во вторую (readers) тех, кто может только читать файл (ну и всех из writers, пожалуй, тоже).

Сам файл должен быть открыт для записи только владельцу (это может быть кто-нибудь из группы writers или вообще фиктивный юзер), для чтения – "группе допущенных" и это должна быть группа readers, и, наконец, для "всех остальных" можно смело убирать любые права. Ну, а та программа, которая предназначена для модификации, должна иметь того же владельца, что и исходный файл, группу – writers, быть "суидной" и права на запуск должны быть только у группы и владельца (хотя, если он фиктивный, то ему не обязательно).

Решение 1.2 (Паскаль И.). Второе решение заключается в том, что файл надо "спрятать" в директорию, с соответствующими правами доступа.

Пусть у нас так же существуют две группы readers и writers, причем все члены группы writers входят также и в readers (но не наоборот :-).

Тогда директория, в которую прячется файл должна иметь группу

readers (кто владелец в общем-то – не важно). Причем для группы доступ в директорию открыт, а для "всех остальных" закрыт. Это означает, что в директорию смогут войти только "читатели" и "писатели" (поскольку они входят в группу "читателей").

А сам файл должен иметь группу writers, причем этой группе файл должен быть открыт для записи. А право "только чтение" можно дать категории "все остальные", все равно это будут только те, кто входит в readers, но не попал в writers. Посторонние просто не попадут в эту директорию и даже не увидят, что такой файл существует.

Решение 2.1 (Паскаль И.). Неполное. Проблема даже не в том, что рядовой пользователь не может сам менять состав групп. Эту то проблему можно решить достаточно просто. Можно было бы для каждого пользователя завести отдельную группу (кстати, программа для добавления пользователей adduser, так и предлагает делать) и написать специальную программу (естественно, "суидную"), которая исполнялась бы с правами root'а и позволяла пользователю менять состав группы, но не любой, а только его личной.

Однако, это решение не полное. Проблема в том, что отдельный пользователь может быть полноправным членом только 16-и групп. И если у пользователя vasia окажется слишком много друзей, которые захотят включить его в свои группы, то реально он сможет "дружить" только с шестнадцатью, чьи группы будут первыми в списке групп.

Кроме того, это не решает проблему, если владелец файла захочет дать разные права нескольким разным группам.

Решение 2.2 (Паскаль И.). Полное. Оно описпно в man'ax setfacl, getfacl, acl.

Введение в решение. По умолчанию списки acl для файлов/каталогов выключены в linux. Однако, их можно включить и определить списки доступа (acl) для пользователей и файлов/каталогов. С помощью них можно как расширить права доступа, так и «сузить». И тогда при наличии acl, система определяет права конкретного пользователя по отношению к файлу/каталогу в следующем порядке

- если пользователь есть в списке, то права берутся оттуда;

- если пользователя нет в списке, но есть группа, членом которой он является, то применяются права для этой группы;

- и, наконец, если ни пользователя, ни группы в списке нет, то проверяются обычные права доступа для этого файла как описано выше в этой главе.

Однако, это отдельная тема и об этом потом.

5

ОРГАНИЗАЦИЯ СЕТЕЙ

5.1. Общие сведения о вычислительных сетях

5.1.1. Сети

В экономике (и в жизни вообще) достаточно часто возникает необходимость в передаче информации. Эта задача реализуется с помощью сетей передачи данных.

Определение. Сеть передачи данных – совокупность оборудования, а в современных реализациях, ещё и программного обеспечения, предназначенного для передачи информации.

Сети передачи данных бывают телефонные, телеграфные, вычислительные, сети передачи данных между ЦУП и космическими станциями и другие.

Определение. Вычислительная сеть – совокупность аппаратно-программного обеспечения предназначенного для передачи данных между компьютерами.

Вычислительные сети получили широкое распространение в последние десятилетия. Более того, они своей функциональностью покрывают все остальные виды сетей, то есть, в современной экономике (и в жизни вообще) они становятся основным видом сетей, а для просто сетей передачи данных невычислительного типа – уже даже сложно привести пример их использования.

5.2. Основные определения

5.2.1. Классификации

Всё оборудование сетей передачи данных делится на две категории:

- пассивное – кабели, разъёмы, патч-панели и др., то есть, то оборудование, которое обеспечивает передачу сигнала, несущего информацию;

- активное – сетевые карты, модемы, коммутаторы, мосты, мультиплексоры и др., то есть, то оборудование, которое генерит или принимает сигнал, используемый для передачи информации.

Кроме того, в сетях передачи данных используется различное вспомогательное оборудование, которое не относится к указанным категориям: это устройства бесперебойного питания, устройства кондиционирования воздуха, монтажные стойки, шкафы, кабель-каналы, средства крепления и другое оборудование, которое не участвует непосредственно в передаче данных.

Активное оборудование требует подачи энергии для генерации сигналов: это сетевые карты, повторители, концентраторы, модемы, коммутаторы и др.

Пассивное оборудование подачи энергии не требует: это кабельные системы, соединительные разъемы, коммутационные панели и др.

Обозначения (см. рис. 10):

Рис. 10. Структура линии связи.

DTE – Data Terminal Equipment, оно же ООД – оконечное оборудование данных. Таким оборудованием может являться, например, ПЭВМ – ваш персональный компьютер. Иначе говоря, это то оборудование, которое конкретно пользователь использует для обмена данными с другим пользова-

телем: компьютер, мобильник, телеграфный аппарат Бодо, факсовый аппарат и др.

DCE – Data Circuit terminating Equipment, оно же АПД – аппаратура передачи данных. Таким оборудованием может являться, например, сетевая карта в вашем компьютере или модем. Иначе говоря, это то оборудование, которое непосредственно связывает пользовательское оборудование (ООД), например, компьютеры, с линией связи и является, таким образом, пограничным оборудованием.

Обратите внимание, что чёткой границы между ООД и АПД нет. Например, сетевая карта вроде как принадлежит компьютеру, то есть является частью ООД, но с другой стороны она явно является АПД, так как работает непосредственно с кабелем (с линией связи). Аналогично и модем.

Кабельные системы, иногда говорят «структурированные кабельные системы» (СКС, то есть, кабельные системы специальным образом иерархически организованные) – это кабели, патч-панели, коннекторы, розетки, кабель-каналы и др. элементы, с помощью которых строится кабельная система (включая гвозди, шурупы и дюбели! – см. смету на создание СКС).

Промежуточное оборудование – это либо оборудование линий связи большой протяжённости (маршрутизаторы, мультиплексоры, повторители и др. оборудование и, конечно, кабельные системы), на которых оно используется как раз для обеспечения этой самой «большой протяжённости», либо это оборудование для организации локальной сети (коммутаторы, мосты, концентраторы и, опять же, кабельные системы). Промежуточное оборудование это совокупность активного и пассивного оборудования. Работу промежуточного оборудования пользователь не замечает – см. *прозрачность сети*.

Среда передачи данных – промежуточное оборудование, а также просто пространство, если используются беспроводные технологии (радио, инфракрасная или иная оптическая беспроводная техника и др.).

Вырожденный случай – когда для организации обмена данными промежуточное оборудование не используется, например, если в двух ваших компьютерах стоят беспроводные адаптеры BlueTooth и компьютеры стоят на расстоянии не более десяти метров друг от друга.

Линия связи – АПД + среда_передачи_данных + АПД.

Типы оборудования сетей в терминологии ЭМВОС (см. рис. 11):

1. Конечные системы. ES (End Systems). Являются источниками и/

или потребителями информации (ПК, сетевые принтеры,…)

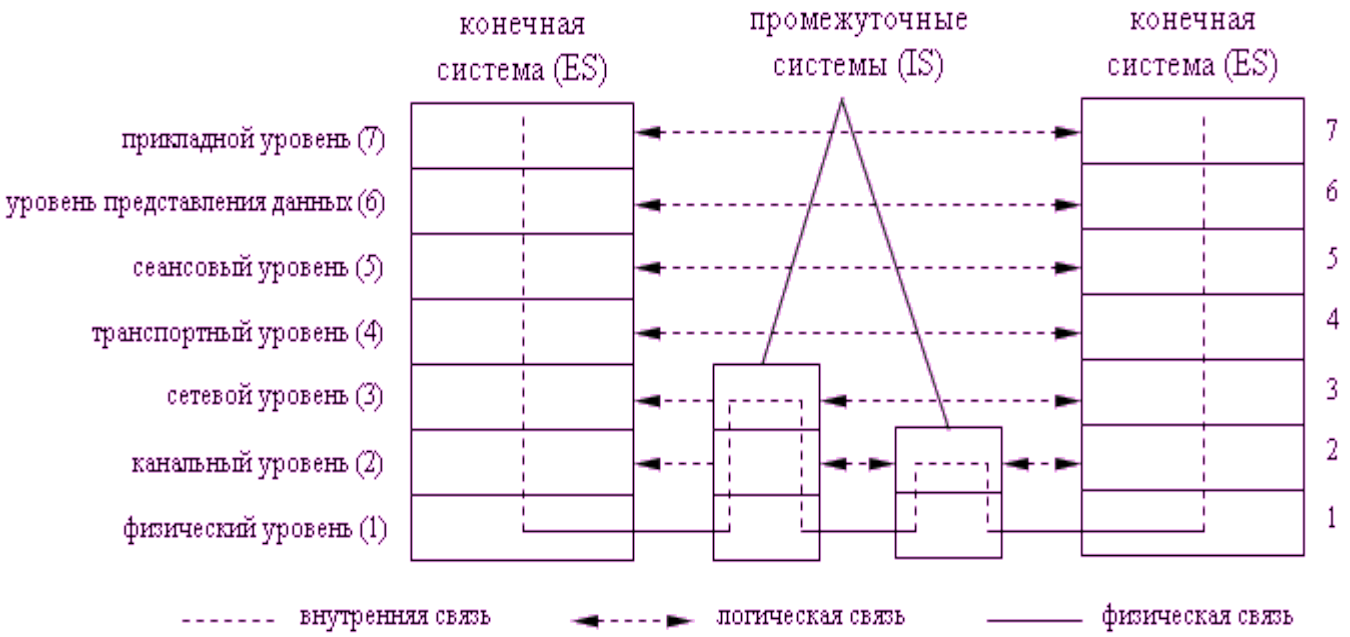


Рис. 11. Структура эталонной модели: внутренние связи (по вертикали) реализуются интерфейсами; логические связи (по горизонтали) реализуются протоколами; физические связи (между системами) реализуются через посредство среды

2. Промежуточные системы. IS (Intermediate Systems). Обеспечивают прохождение информации по сети (концентраторы, маршрутизаторы, модемы, кабельная или беспроводная инфраструктура, соединяющая их).

Сетевой трафик – это поток информации, передаваемый по сети. Трафик кроме полезной информации включает и служебную, необходимую для организации взаимодействия узлов сети.

Пропускная способность сети определяется количеством информации, проходящей через линию связи за единицу времени (бит/сек, кбит/сек, Мбит/сек, Гб/сек), то есть, трафик в единицу времени по всей линии связи.

Производительность сети применима для активного коммуникационного оборудования. Определяется как общее количество неструктурированной информации, пропускаемой оборудованием за единицу времени (бит/сек), то есть, трафик в единицу времени через активное оборудование.

5.2.2. Открытые и закрытые системы

Для организации обмена информацией должен быть разработан комплекс программных и аппаратных средств, распределенных по разным устройствам сети. Поначалу,давным-давно, в 60-70- и даже 80-ые годы каждый разработчик и поставщик сетевых средств сам разрабатывал весь комплекс задач с помощью собственного набора протоколов, программ и

аппаратуры. Это приводило к несовместимости этих наборов у разных поставщиков. Такие системы назывались закрытыми. При соединении нескольких закрытых систем используются частные решения, организующие взаимодействие конкретных закрытых систем, которые работают, но ограничивают возможности пользователей, привязывая их к приложениям или к аппаратному обеспечению определенных производителей.

Открытые системы предполагают открытые стандарты, направленные на обеспечение совместимости между различными системами, то есть, задача построения сетей разбита на несколько взаимосвязанных подзадач с определением правил взаимодействия между ними. Стандартизация этих правил позволяет расширить количество разработчиков аппаратного и программного обеспечения.

5.3. Базовая модель взаимодействия открытых систем (OSI – Open System Interconnection)

ISO – международная организация по стандартизации.

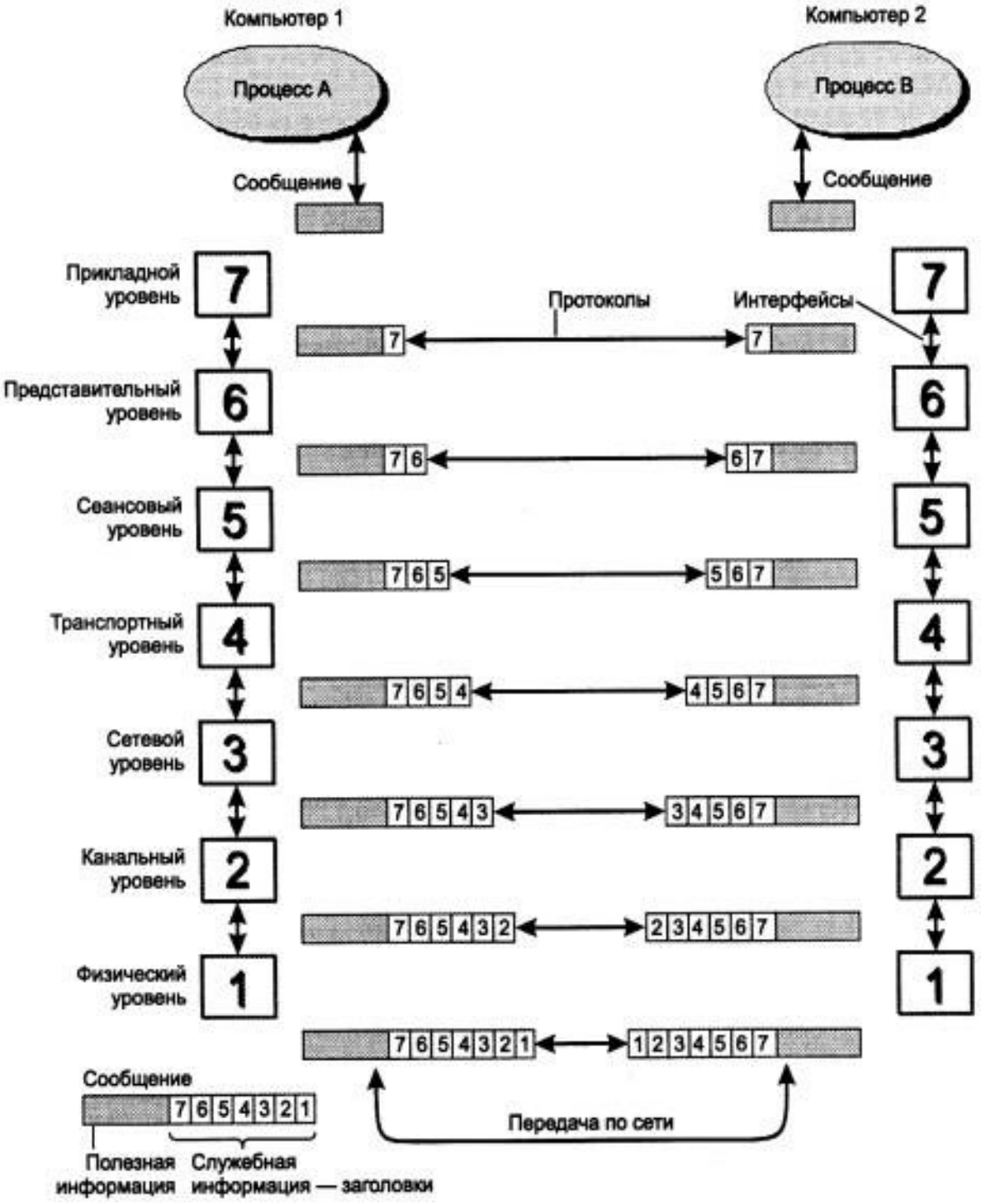
Семиуровневая модель OSI, она же ЭМВОС – Эталонная Модель Взаимодействия Открытых Систем.

Модель разработана в 1977 г. ISO (International Standard Organization). Она стала основой для определения того, как взаимодействуют системы в сети. В 1984 г. была выпущена новая версия, которая стала международным стандартом (см. рис. 12). Сама модель OSI основана на уровневых протоколах, что позволяет обеспечить:

- логическую декомпозицию сложной сети на обозримые части-уровни;
- стандартные интерфейсы между сетевыми функциями;
- симметрию в отношении функций, реализуемых в каждом узле сети;
- общий язык для взаимопонимания разработчиков различных частей сети.

Функции любого узла сети разбиваются на уровни. Для конечных систем их семь. Внутри каждого узла взаимодействие между уровнями идет по вертикали (по интерфейсам) – см. рис. 12. Взаимодействие между двумя узлами **логически** происходит по горизонтали между соответствующими уровнями (по протоколам). Реально, из-за отсутствия непосредственных горизонтальных связей производится спуск до нижнего уровня в источнике, связь через физическую среду и подъем до соответствующего уровня в приемнике информации. В промежуточных устройствах подъем идет до то-

го уровня, который доступен устройству. Каждый уровень обеспечивает свой набор сервисных функций. Уровень, с которого посылается запрос, и симметричный ему уровень в отвечающей системе формируют свои блоки



данных.

Рис. 12. ЭМВОС – Эталонная Модель Взаимодействия Открытых Систем –

инкапсуляция пакетов.

На каждом уровне определяется свой формат блока данных. Он состоит из заголовка (адрес, управление), поля данных и иногда концевика (например, контрольной суммы по блоку данных. Эти блоки данных по мере спуска вниз по уровням инкапсулируются (вставляются целиком как данные в поле данных – см. рис. 13) в блоки данных нижних уровней. При подъёме происходит обратный процесс – разинкапсуляция.

Рис. 13. Инкапсуляция

Каждый уровень реализует некоторую функциональность. Она определяется возможностями наборов протоколов и интерфейсов уровня, то есть алгоритмами протоколов и интерфейсов и правилами именования сетевых объектов на этом уровне.

Протокол – формализованные правила, определяющие формат сообщений и алгоритм, по которому обмениваются сообщениями сетевые компоненты, лежащие на одном уровне, но в разных узлах. Формально, протокол – это документ. Но программный модуль, реализующий некоторый протокол, часто для краткости также называют «протоколом». При этом соотношение между протоколом – формально определенной процедурой (документом) и протоколом – программным модулем, реализующим эту процедуру, аналогично соотношению между алгоритмом решения некоторой задачи и программой, решающей эту задачу. Полный протокол должен включать разделы:

- описание формата пакетов данных, формируемых на данном уровне,
- описание алгоритма обмена (работы протокола),

- правила кодирования информации в этом протоколе,
- правила именования сетевых взаимодействующих объектов этого протокола.

Конечно, отнюдь не каждый протокол включает все разделы, большинство протоколов разрабатываются в «контексте» стека протоколов (см. рис.14).

Интерфейс – формализованные правила, определяющие формат сообщений и алгоритм, по которому обмениваются сообщениями сетевые компоненты, лежащие в одном узле на соседних уровнях. То есть, интерфейс определяет набор сервисов, предоставляемый некоторым уровнем вышестоящему соседнему уровню. С точки зрения программирования выглядит это как вызов функций некоторой библиотеки. Обратите внимание, что интерфейс всегда предоставляется нижестоящим уровнем вышестоящему уровню. Отсюда такие выражения как «интерфейс человек-ЭВМ» (на самом деле интерфейс предоставляется ЭВМ (нижестоящей «мафыной») вышестоящему человеку). Также взаимодействие сотрудника и начальника определяется интерфейсом сотрудника, который он предоставляет своему начальнику. Этот интерфейс сотрудника определяется «должностной инструкцией», квалификацией и образованием сотрудника.

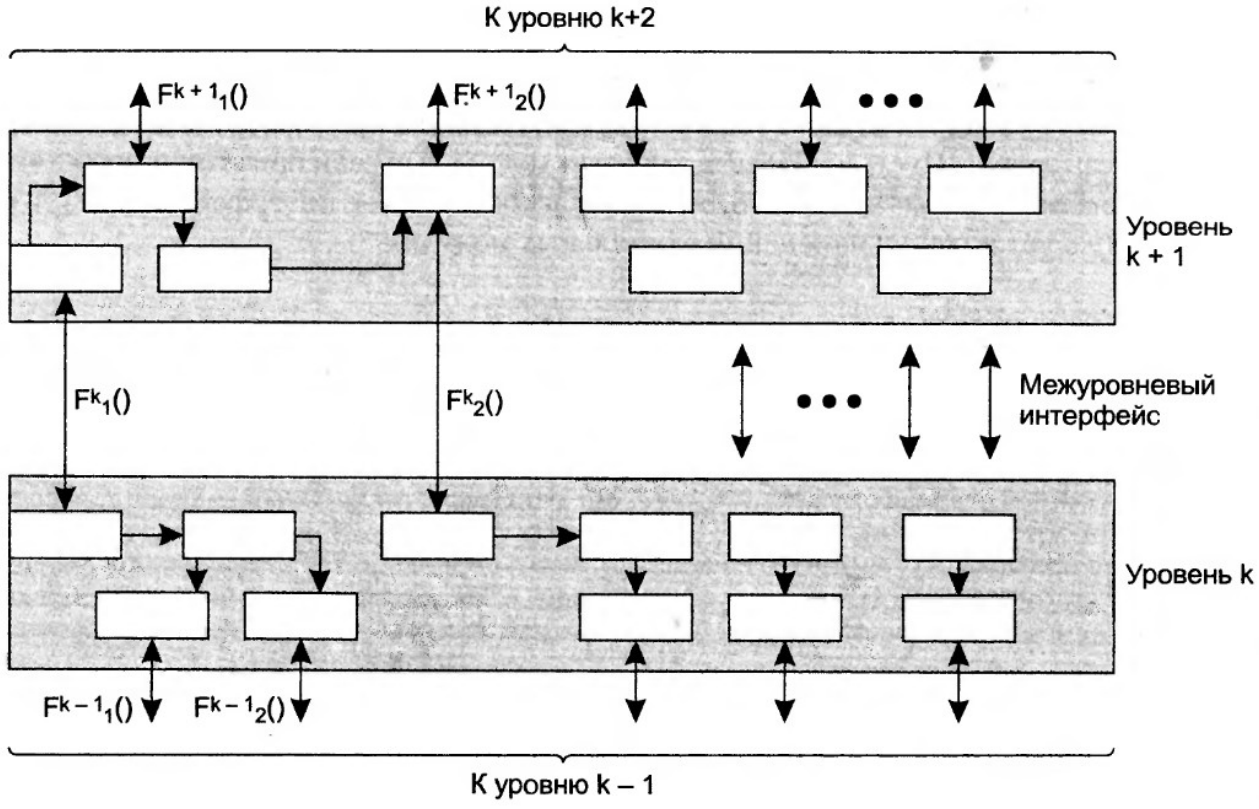


Рис. 14. Стек – совокупность взаимосвязанных и иерархически организованных

протоколов

Стек коммуникационных (сетевых) протоколов – совокупность взаимосвязанных и иерархически организованных протоколов, достаточный для организации взаимодействия узлов в сети. Протоколы стека иерархически организованы – то есть, разбиты на группы-уровни (см. рис. 14), в соответствии со своим предназначением, но, протоколы стека взаимодействуют между собой по вертикали и горизонтали, то есть, они могут обращаться к другим протоколам этого же уровня или к протоколам нижестоящего уровня (по интерфейсу нижестоящего уровня). В некотором смысле каждый протокол стека может рассматриваться как некоторый класс в объектной модели: в протоколе определяются структуры данных (формат пакетов, которыми он обменивается) и алгоритм обмена (что, когда и как должно делаться – методы класса).

5.4. Физический уровень

Физический уровень (Physical layer) реализует передачу битов по физическим каналам связи, таким, например, как коаксиальный кабель, витая пара, оптоволоконный кабель или просто пространство. К этому уровню имеют отношение характеристики физических сред передачи данных, такие как полоса пропускания, помехозащищенность, волновое сопротивление и другие. На этом же уровне определяются характеристики электрических сигналов, передающих дискретную информацию, например, крутизна фронтов импульсов, уровни напряжения или тока передаваемого сигнала, тип кодирования, скорость передачи сигналов и др. Кроме этого, здесь стандартизуются типы разъемов и назначение каждого контакта.

Функции физического уровня реализуются во всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером, последовательным/параллельным портом, контроллером USB и др.

Примером протокола физического уровня может служить спецификация (релиз) 10-Base-T технологии Ethernet, которая определяет в качестве используемого кабеля неэкранированную витую пару категории 3 с волновым сопротивлением 100 Ом, разъем RJ-45, максимальную длину физического сегмента 100 метров, манчестерский код для представления данных в кабеле, а также некоторые другие характеристики среды и электрических сигналов (см. рис. 15).

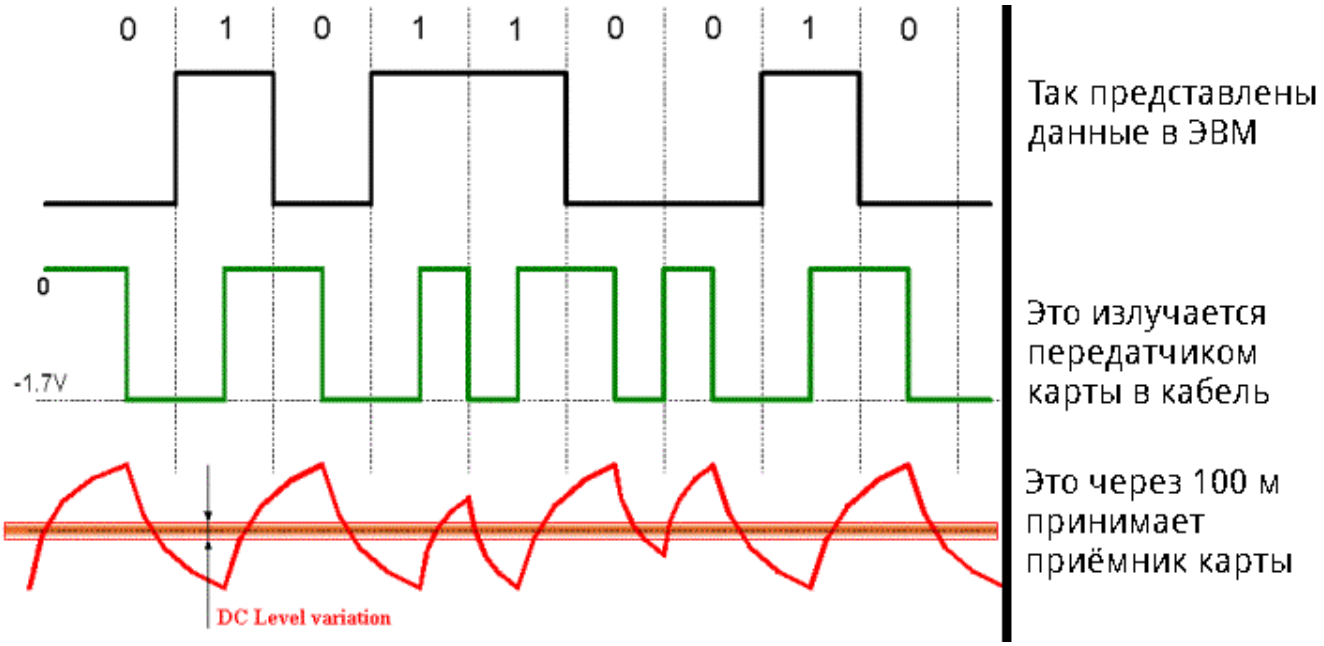


Рис. 15. Что излучает сетевая карта отправителя и что получает сетевая карта получателя – сетевая технология Ethernet, спецификация 10Base-T

5.5. Канальный уровень: подуровень MAC

5.5.1. Подуровень MAC

Канальный уровень делится на два подуровня: нижний подуровень MAC, на котором определяются сетевые технологии, и верхний подуровень LLC, который является единственным и общим (рис. 16). Поскольку сетевых технологий много (несколько десятков), то реализаций подуровня MAC тоже много: каждой сетевой технологии соответствует своя реализация подуровня MAC. В данном пункте рассматривается подуровень MAC.

На физическом уровне просто пересылаются биты. При этом не учитывается, что в некоторых сетях, в которых линии связи используются (разделяются) попеременно несколькими парами взаимодействующих компьютеров, физическая среда передачи может быть занята. Поэтому одной из задач канального уровня (Data Link layer) является проверка доступности среды передачи.

Другой задачей канального уровня является реализация механизмов обнаружения и коррекции ошибок. Для этого на канальном уровне биты группируются в наборы, называемые *кадрами* (*frames* – см. рис. 17). Канальный уровень обеспечивает корректность передачи каждого кадра, по-

мещая специальную последовательность бит в начало и конец каждого кадра, для его выделения, а также вычисляет контрольную сумму, обрабатывая все байты кадра определенным способом и добавляя контрольную сумму к кадру. Когда кадр приходит по сети, получатель снова вычисляет контрольную сумму полученных данных и сравнивает результат с контрольной суммой из кадра. Если они совпадают, кадр считается правильным и принимается. Если же контрольные суммы не совпадают, то фиксируется ошибка.

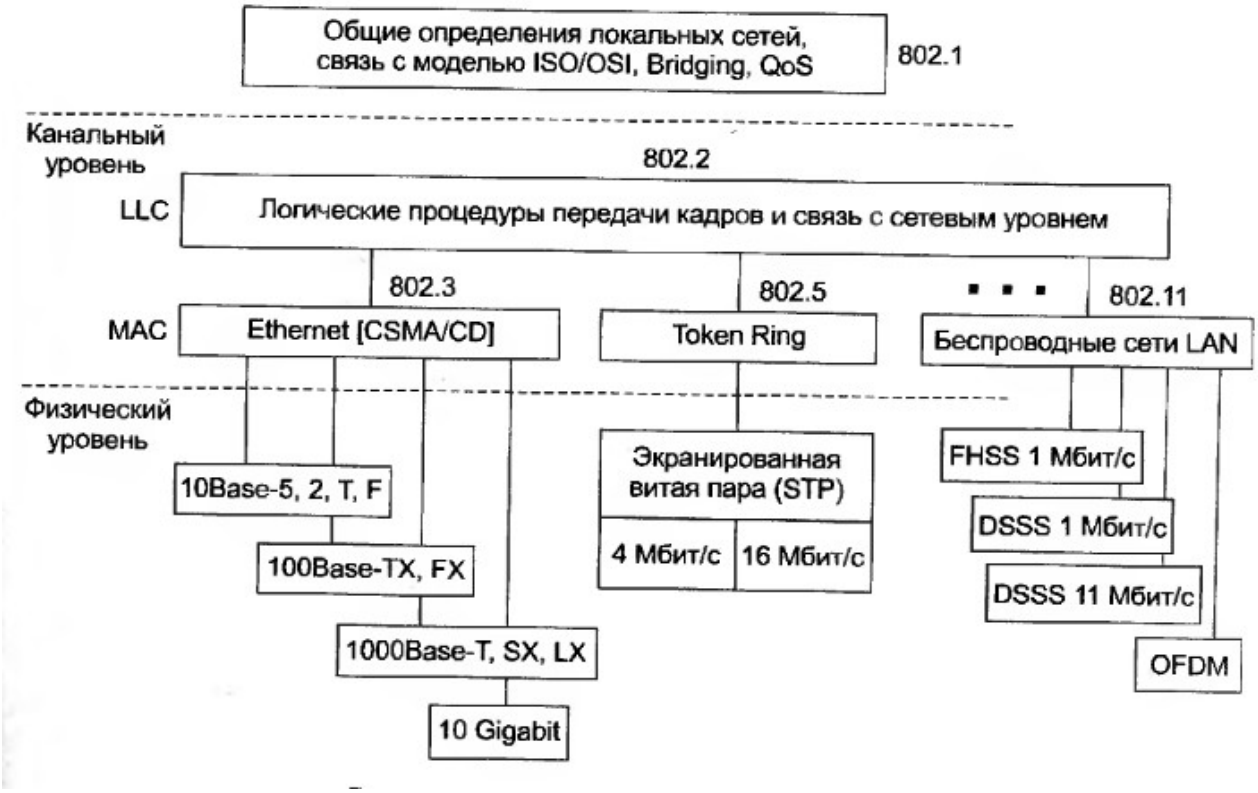


Рис. 16. Структура сетевых технологий.
В каждой сетевой технологии показаны далеко не все релизы

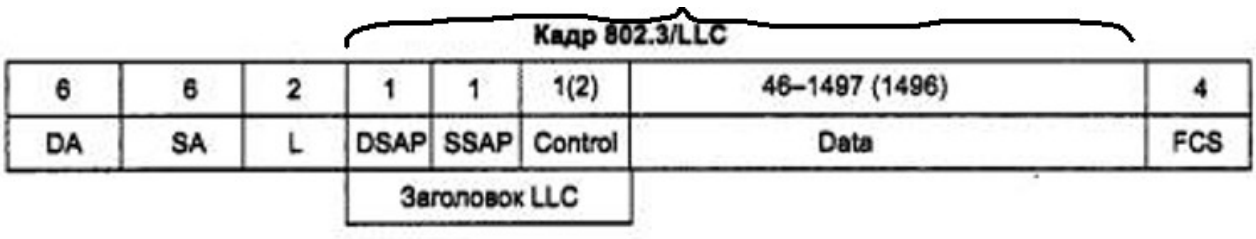


Рис. 17. Кадр MAC. В середине фигурной скобкой выделен инкапсулированный кадр LLC

Канальный уровень может не только обнаруживать ошибки, но и исправлять их за счет повторной передачи поврежденных кадров. Необходи-

мо отметить, что функция исправления ошибок не является обязательной для канального уровня, поэтому в некоторых протоколах этого уровня она отсутствует, например, в Ethernet и frame relay.

В протоколах канального уровня, используемых в локальных сетях, заложена определенная структура связей (топология) между компьютерами и способы их адресации. Хотя канальный уровень и обеспечивает доставку кадра между любыми двумя узлами локальной сети, он это делает только в сети с совершенно определенной топологией связей, именно той топологией, для которой он был разработан (см. п. 5.5.4.). К таким типовым топологиям, поддерживаемым протоколами канального уровня локальных сетей, относятся общая шина, кольцо и звезда, а также структуры, полученные из них с помощью мостов и коммутаторов. То есть, **алгоритм сетевой технологии определяет топологию сети.** Примерами протоколов канального уровня являются протоколы Ethernet, Token Ring, FDDI, l00VG-AnyLAN и др.

5.5.2. Сетевая технология

В середине 90-х годов сформировалось понятие «сетевой технологии» как набора протоколов и интерфейсов физического и канального уровней достаточных для организации взаимодействия узлов в локальной сети. Такими сетевыми технологиями стали Ethernet, Token Ring, FDDI, l00VG-AnyLAN, Frame Relay, ATM и др., в том числе и беспроводные. Такие сетевые технологии иногда называют базовыми сетевыми технологиями.

Каждая сетевая технология определяется некоторым «головным» протоколом канального уровня (Ethernet, Token Ring, FDDI, l00VG-AnyLAN, BlueToosh, MiWi и др.), который определяет основные понятия, основной алгоритм технологии и набор протоколов и интерфейсов, с помощью которых создаются «релизы» этой сетевой технологии (см. рис. 16) – они также называются спецификациями. Например, головной протокол Ethernet, а «под ним» существуют наборы протоколов, интерфейсов, технических требований и даже стандартов, с помощью которых реализуются различные виды сетевой технологии Ethernet: 10Base-5, 10Base-2, 10Base-T, 10Base-F, 100Base-T, 100Base-TX, 1000Base-T и др., всего несколько десятков различных «релизов» Ethernet.

Также можно сказать, что сетевые технологии это разные реализации подуровня MAC и физического уровня.

С технологической точки зрения понятием, тесно связанным с понятием сетевой технологии, является понятие «локальной сети».

5.5.3. Локальная сеть

Обращаите внимание: здесь дано правильное определение локальной сети для ИТ-шников – «технологическое» определение.

Локальная сеть это набор вычислительных систем, сетевое взаимодействие между которыми обеспечивается некоторым релизом некоторой сетевой технологии. Если проще, то это набор компьютеров, объединённых в сеть с помощью некоторой сетевой технологии.

Пример. Предположим, существуют локальные сети по несколько сетевых узлов в каждой: Ethernet (100Base-T), 100VG-AnyLAN и TokenRing. Это разные локальные сети, поскольку в сетевых узлах стоят разные сетевые карты, реализующие разные сетевые технологии, они генерируют кадры данных разного формата, отрабатывают разные алгоритмы и очевидно несовместимы друг с другом.

Сетевые технологии (подуровни MAC), как правило, реализуются аппаратно. Например, в сетевых картах, модемах и других устройствах АПД.

Что нужно для того, чтобы пользователь мог работать в такой локальной сети? Не сильно сложная программа-клиент, которая с верхней стороны обеспечивала бы интерфейс пользователя (реализовывала команды пользователя обмена файлами и сообщениями), а с нижней стороны взаимодействовала бы с сетевыми картами. Когда-то (конкретно, в 70-80-90-ые годы, а также в годы всеобщего засилья MS DOS) так оно и было. Можно на одном из сетевых узлов запустить программу, реализующую функциональность ftp-сервера, тогда получим локальную сеть с файловым сервером. Обратите внимание, при этом мы нигде не выходим за пределы понятия «сетевая технология», то есть используются только протоколы и интерфейсы соответствующей сетевой технологии.

Другое определение локальной сети: это минимальная совокупность аппаратно-программных средств, достаточная для организации сетевого взаимодействия.

Таким образом, локальная сеть это самая простая сетевая структура, реализуемая с минимальным набором задействуемых для её организации средств и ресурсов. Все другие сети состоят из локальных сетей, то есть, корпоративные сети, кампусные сети, интернет состоят из локальных сетей, иначе говоря, представляют собой «сети локальных сетей». Такие «сети сетей» называются «глобальными сетями» и для их организации задействуются протоколы более высоких уровней (см. также рис. 19).

5.5.4. Топология сети

Топология – способ организации связей между элементами. Различают физическую топологию сети и логическую топологию сети.

Физическая топология – это конфигурация графа, вершинам которого соответствуют компьютеры сети (иногда и другое оборудование, например концентраторы), а ребрам – физические связи между ними (например, кабели).

Логическая топология – это конфигурация графа, вершинам которого соответствуют опять же компьютеры сети (иногда и другое оборудование, например, мосты или коммутаторы), а ребрам – логические связи между ними (например, интерфейсы и протоколы).

Компьютеры, подключенные к сети, часто называют *станциями* или *узлами* сети.

Заметим, что конфигурация *физических связей* определяется электрическими соединениями компьютеров между собой и может отличаться от конфигурации *логических связей* между узлами сети (см. рис. 18). Логические связи представляют собой маршруты передачи данных между узлами сети и образуются путем соответствующей настройки коммуникационного оборудования. Если на рис. 18 концентраторы заменить на коммутаторы, то логическая структура сети будет существенно больше похожа на физическую. На логической топологии мы можем не увидеть некоторых физических устройств в силу свойства «прозрачности сетевого оборудования» (см. пункт. 5.2.1).

Типовые топологии (простейшие топологии) – общая шина, звезда, кольцо, точка-точка. Они являются основой локальных сетей и поддерживаются на канальном уровне. Так происходит потому, что основной алгоритм сетевой технологии (алгоритм головного протокола) может работать только в определённых условиях:

- строго определённый способ организации физических связей – топология сети;

- строго определённый способ именования сетевых объектов на этой топологии.

С другой стороны, топология сети определяет особенности алгоритма сетевой технологии: например, может ли линия связи использоваться монопольно или она является разделяемой (shared). Во втором случае возникает комплекс проблем, связанных с её совместным использованием, который включает как чисто электрические проблемы обеспечения нужного качества сигналов при подключении к одному и тому же проводу несколь-

ких приемников и передатчиков, так и логические проблемы разделения во времени доступа к этой линии.

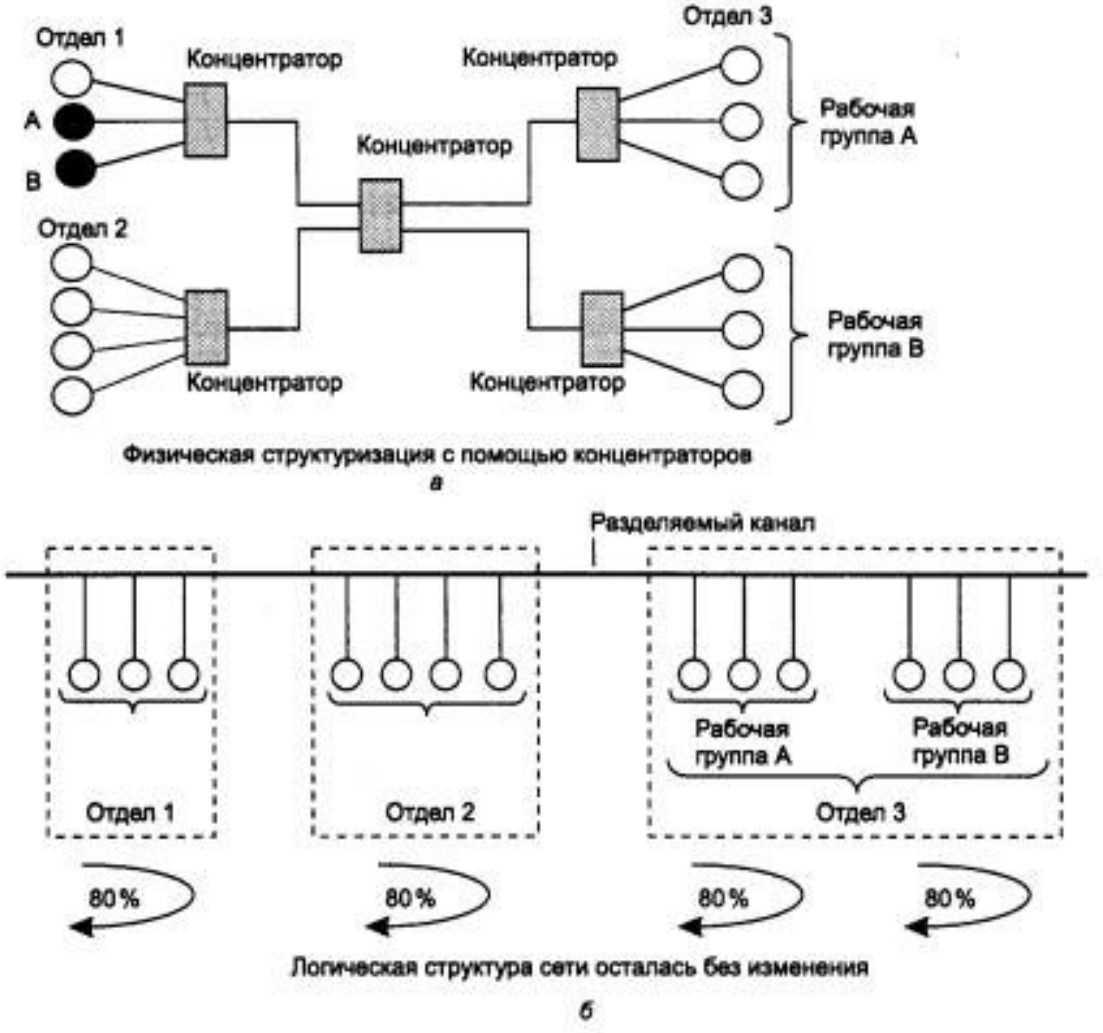


Рис. 18. Физическая и логическая структура сети – две большие разницы. 80% трафика в каждом сегменте являются локальными, но концентраторы транслируют трафик на все компьютеры

Иначе говоря, алгоритм сетевой технологии (определённый в протоколе) и топология сети определяют друг друга (взаимосвязаны).

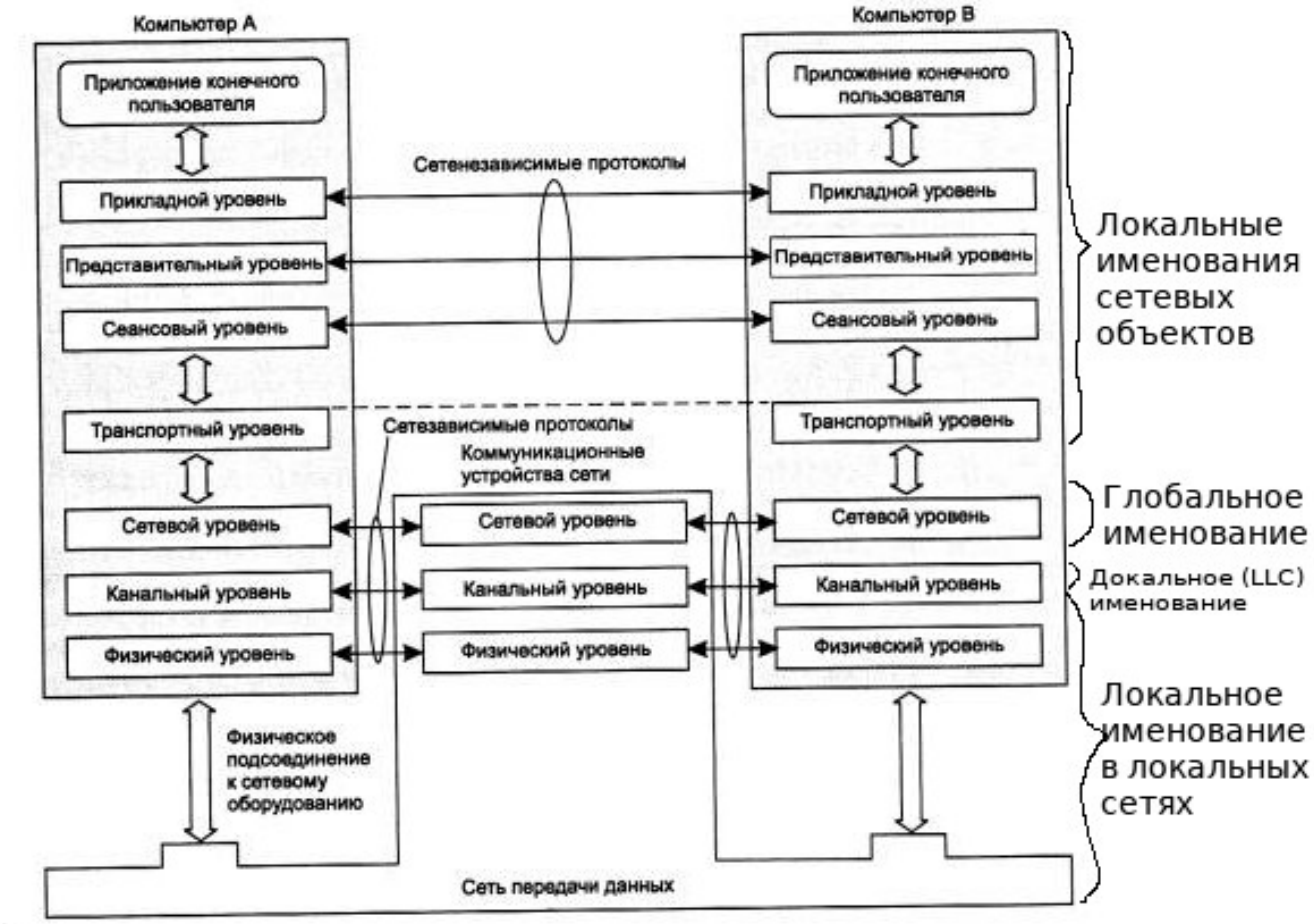
Примеры. Алгоритм сетевой технологии Ethernet – CSMA/CD, может работать только на линейных топологиях – шина или точка-точка и топологиях, построенных из них, при этом ни в коем случае не допускаются кольцевые связи (почему?). Алгоритм сетевой технологии TokenRing – алгоритм «маркерного кольца», может работать только на топологии кольцо. Алгоритм сетевой технологии 100VG-AnyLAN – алгоритм «шины с арбитром», требует наличия на шине специального устройства управления доступом к среде – арбитра. И т. д.

В глобальных сетях, которые редко обладают регулярной топологией, канальный уровень часто обеспечивает обмен сообщениями только между двумя соседними компьютерами, соединенными индивидуальной линией связи – топология «точка-точка». Примерами протоколов для этой топологии могут служить широко распространенные протоколы PPP и LAP-B. То есть, линия используется монопольно и основными вопросами являются: а) надёжность передачи, б) регулирование трафика.

5.5.5. Именование сетевых узлов

С именованием сетевых узлов не возникает проблем только на топологии «точка-точка». Во всех топологиях с разделяемой средой (при подключении к среде передачи трёх и больше компьютеров) возникает проблема их именования – адрес должен быть строго оригинальным (см. рис. 19) в пределах адресуемого (доступного) пространства.

Рис. 19. Именование сетевых объектов на уровнях ЭМВОС



А учитывая, что автономные локальные сети сейчас уже практически

не встречаются, то проблема именования становится глобальной. Поэтому, к адресу сетевого узла и схеме его назначения предъявляются определённые требования [17]:

1) адрес должен уникально идентифицировать компьютер в сети любого масштаба;

2) схема назначения адресов должна сводить к минимуму ручной труд администратора и вероятность дублирования адресов;

3) адрес должен иметь иерархическую структуру, удобную для построения больших сетей; в больших сетях, состоящих из многих тысяч узлов, отсутствие иерархии адреса может привести к большим издержкам – конечным узлам и коммуникационному оборудованию придется оперировать с таблицами адресов, состоящими из тысяч записей;

4) адрес должен быть удобен для пользователей сети, а это значит, что он должен иметь символьное представление например, www.yandex.ru;

5) адрес должен иметь по возможности компактное представление, чтобы не перегружать память коммуникационной аппаратуры – сетевых адаптеров, маршрутизаторов и т. п.

Исторически сложилось так, что в сетевых технологиях (практически во всех – на подуровне MAC канального уровня) используются «аппаратные» адреса активного оборудования – MAC-адреса, поскольку способ их назначения гарантирует их оригинальность для каждого активного сетевого устройства. Такой способ именования удовлетворяет требованиям 1, 2 и 5 и не удовлетворяет требованиям 3 и 4. Однако в локальных сетях (на уровне сетевых технологий) количество сетевых узлов, как правило, ограничено десятками/сотнями, поскольку есть ограничения в сетевых технологиях на количество сетевых узлов в локальной сети, следовательно такое именование вполне удовлетворительно. А учитывая, что протоколы и интерфейсы сетевых технологий, почти всегда реализуются аппаратно, то в некоторой степени выполняется и требование 4 – не так уж часто человеку-пользователю (не админу) приходится иметь дело с физическими адресами.

Пользователь использует для работы в сети коммуникационное программное обеспечение, которое работает поверх сетевых технологий и, следовательно, есть возможность для организации более «человеческого» именования сетевых узлов и сервисов в сети, что обычно и делается.

5.6. Канальный уровень: подуровень LLC

5.6.1. Назначение

Протокол LLC обеспечивает для технологий глобальных сетей нужное качество услуг транспортной службы, передавая свои кадры с помощью какой-либо сетевой технологии либо дейтаграммным способом, либо с помощью процедур с установлением соединения и восстановлением кадров.

5.6.2. Место протокола LLC в иерархии протоколов

Протокол LLC определяется стандартом ISO 8802.2 (IEEE 802.2) и занимает уровень между сетевыми/транспортными/сеансовыми протоколами (в зависимости от вышестоящего стека сетевых протоколов) и протоколами уровня MAC (сетевыми технологиями).

Вышестоящие протоколы какого-либо стека сетевых протоколов передают через межуровневый интерфейс данные для протокола LLC – свой пакет (например, пакет IP, IPX или NetBEUI), адресную информацию об узле назначения, а также требования к качеству транспортных услуг, которое протокол LLC должен обеспечить. Протокол LLC помещает пакет протокола верхнего уровня в свой кадр, который дополняется необходимыми служебными полями. Далее через межуровневый интерфейс протокол LLC передает свой кадр вместе с адресной информацией об узле назначения соответствующему протоколу уровня MAC, который упаковывает кадр LLC в свой кадр (например, кадр Ethernet – рис. 20).

В основу протокола LLC положен протокол HDLC (High-level Data Link Control Procedure), являющийся стандартом ISO. Собственно стандарт HDLC представляет собой обобщение нескольких близких стандартов, характерных для различных технологий: протокола LAP-B сетей X.25 (стандарта, широко распространенного в территориальных сетях), LAP-D, используемого в сетях ISDN, LAP-M, работающего в современных модемах. В спецификации IEEE 802.2 также имеется несколько небольших отличий от стандарта HDLC.

Первоначально в ранних фирменных технологиях подуровень LLC не выделялся в самостоятельный подуровень, да и его функции растворялись в общих функциях протокола канального уровня. Однако, с появлением Интернета и глобальных сетей появилась острая необходимость объединения разношёрстного зоопарка локальных сетей, построенных на основе фирменных сетевых технологий, в сети сетей – корпоративные сети. Из-за больших различий в функциях протоколов фирменных технологий, которые можно отнести к уровню LLC, на уровне LLC пришлось ввести три типа процедур. Протокол вышестоящего уровня может обращаться к одной из этих

процедур.

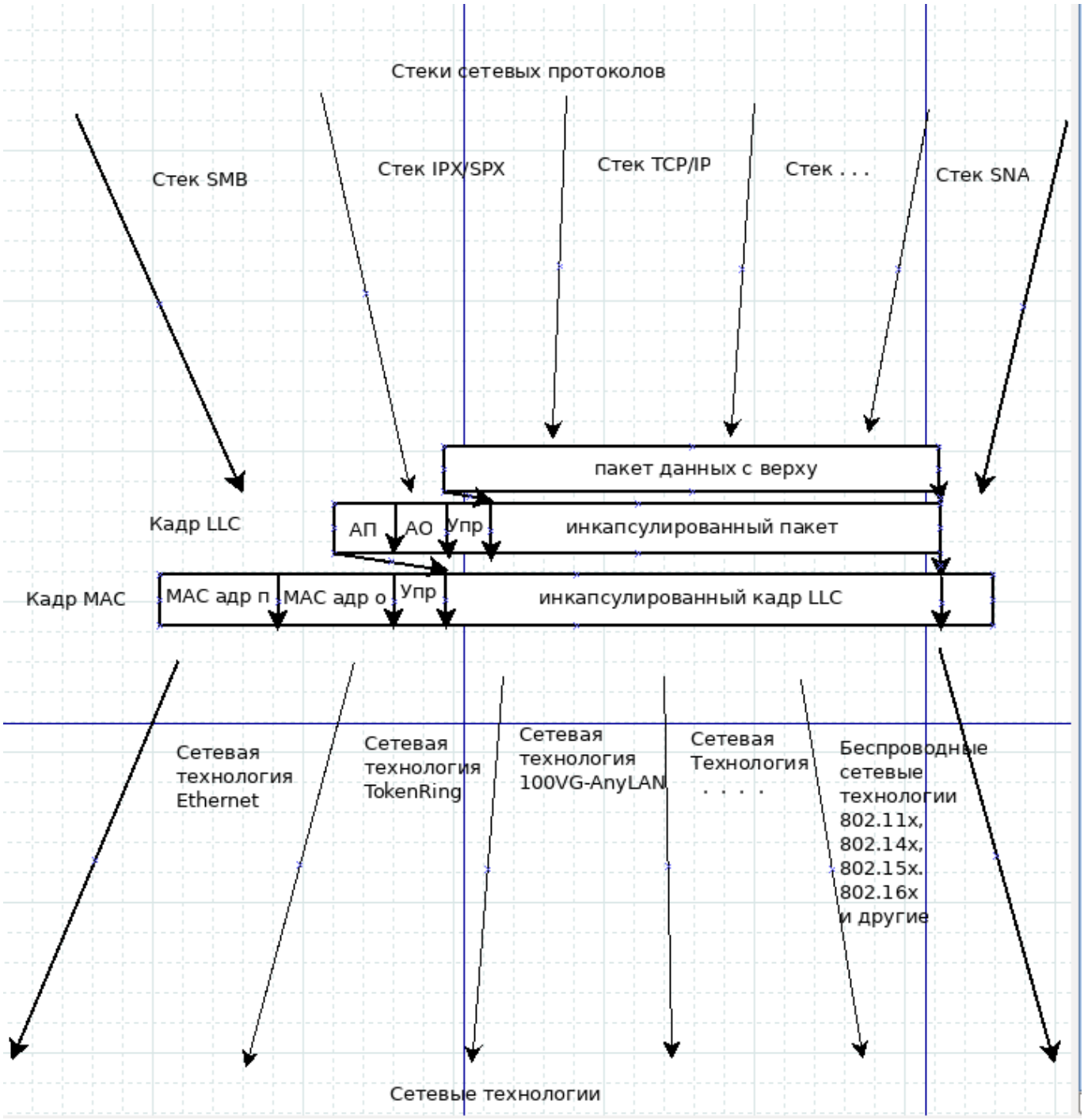


Рис. 20. Место протокола LLC в иерархии протоколов. АП – адрес получателя. АО – адрес отправителя. Упр – байты управления. «MAC адр п» – MAC адрес получателя. «MAC адр о» – MAC адрес отправителя

5.6.3. Три типа процедур подуровня LLC

В соответствии со стандартом 802.2 уровень управления логическим каналом LLC предоставляет верхним уровням три типа процедур:
LLC1 – процедура без установления соединения и без подтверждения;
LLC2 – процедура с установлением соединения и подтверждением;
LLC3 – процедура без установления соединения, но с подтверждением.

Этот набор процедур является общим для всех методов доступа к среде, определенных стандартами 802.3 – 802.5, стандартом на FDDI, стандартом 802.12 на технологию l00VG-AnyLAN, а также AppleTalk, беспроводные сетевые технологии и др.

LLC1. Процедура без установления соединения и без подтверждения LLC1 дает пользователю средства для передачи данных с минимумом издержек. Это дейтаграммный режим работы. Обычно этот вид процедуры используется, когда такие функции, как восстановление данных после ошибок и упорядочивание данных, выполняются протоколами вышележащих уровней, поэтому нет нужды дублировать их на уровне LLC.

LLC2. Процедура с установлением соединений и подтверждением LLC2 дает пользователю возможность установить логическое соединение перед началом передачи любого блока данных и, если это требуется, выполнить процедуры восстановления после ошибок и упорядочивание потока этих блоков в рамках установленного соединения. Протокол LLC2 во многом аналогичен протоколам семейства HDLC (LAP-B, LAP-D, LAP-M), которые применяются в глобальных сетях для обеспечения надежной передачи кадров на зашумленных линиях. Протокол LLC2 работает в режиме скользящего окна.

LLC3. В некоторых случаях (например, при использовании сетей в системах реального времени, управляющих промышленными объектами), когда временные издержки установления логического соединения перед отправкой данных неприемлемы, а подтверждение о корректности приема переданных данных необходимо, используется дополнительная процедура, называемая *процедурой без установления соединения, но с подтверждением LLC3*.

Использование одного из трех режимов работы уровня LLC зависит от стратегии разработчиков конкретного стека протоколов. Например, в стеке TCP/IP уровень LLC всегда работает в режиме LLC1, выполняя простую работу извлечения из кадра и демультиплексирования пакетов различных протоколов – IP, ARP, RARP. Аналогично используется уровень LLC стеком IPX/SPX.

А вот стек Microsoft/IBM, основанный на протоколе NetBIOS/

NetBEUI, часто использует режим LLC2. Это происходит тогда, когда сам протокол NetBIOS/NetBEUI должен работать в режиме с восстановлением потерянных и искаженных данных. В этом случае эта работа перепоручается уровню LLC2. Если же протокол NetBIOS/NetBEUI работает в дейтаграммном режиме, то протокол LLC работает в режиме LLC1.

Режим LLC2 используется также стеком протоколов SNA в том случае, когда на нижнем уровне применяется технология Token Ring.

5.6.4. Структура кадров LLC

По своему назначению все кадры уровня LLC (называемые в стандарте 802.2 блоками данных – Protocol Data Unit, PDU) подразделяются на три типа – информационные, управляющие и ненумерованные.

- *Информационные кадры (Information)* предназначены для передачи информации в процедурах с установлением логического соединения LLC2 и должны обязательно содержать поле информации. В процессе передачи информационных блоков осуществляется их нумерация в режиме скользящего окна.
- *Управляющие кадры (Supervisory)* предназначены для передачи команд и ответов в процедурах с установлением логического соединения LLC2, в том числе запросов на повторную передачу искаженных информационных блоков.
- *Ненумерованные кадры (Unnumbered)* предназначены для передачи ненумерованных команд и ответов, выполняющих в процедурах без установления логического соединения передачу информации, идентификацию и тестирование LLC-уровня, а в процедурах с установлением логического соединения LLC2 – установление и разъединение логического соединения, а также информирование об ошибках. Все типы кадров уровня LLC имеют единый формат (см. рис. 21).

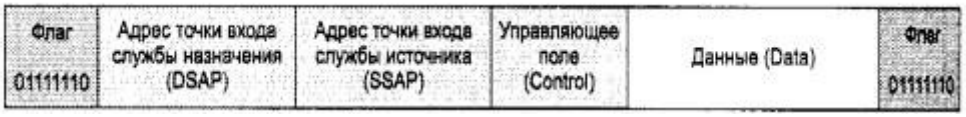


Рис. 21. Формат кадра LLC

Кадр LLC обрамляется двумя однобайтовыми полями «Флаг», имеющими значение 01111110. Флаги используются на уровне MAC для определения границ кадра LLC. В соответствии с многоуровневой структурой протоколов стандартов IEEE 802, кадр LLC вкладывается в кадр уровня

MAC сетевых технологий, то есть, в кадры Ethernet, Token Ring, FDDI и т. д. При этом флаги кадра LLC отбрасываются.

Кадр LLC содержит поле данных и заголовок, который состоит из трех полей:

- адрес точки входа службы назначения (Destination Service Access Point, DSAP);
- адрес точки входа службы источника (Source Service Access Point, SSAP);
- управляющее поле (Control).

Поле данных кадра LLC предназначено для передачи по сети пакетов протоколов вышележащих уровней – сетевых протоколов IP, IPX, AppleTalk, DECnet, в редких случаях – прикладных протоколов, когда те вкладывают свои сообщения непосредственно в кадры канального уровня. Поле данных может отсутствовать в управляющих кадрах и некоторых ненумерованных кадрах. Если пакеты данных, пришедшие от вышестоящих уровней небольшие, то протокол LLC может осуществлять мультипликацию (объединение) пакетов в один кадр LLC, а на принимающем узле, соответственно, демультипликацию (выделение) пакетов данных из кадра (? проверить).

Адресные поля DSAP и SSAP занимают по 1 байту. Они позволяют указать, какой протокол верхнего уровня пересылает данные с помощью этого кадра. Программному обеспечению узлов сети при получении кадров канального уровня необходимо распознать, какой протокол вложил свой пакет в поле данных поступившего кадра, чтобы передать извлеченный из кадра пакет нужному протоколу верхнего уровня для последующей обработки (см. рис. 21).

Для идентификации этих протоколов вводятся так называемые адреса точки входа протокола (Service Access Point, SAP). Значения адресов SAP приписываются протоколам в соответствии со стандартом 802.2. Например, для стека протоколов IBM SNA значение SAP равно 0x4, для протокола IP – 0x6, для протокола NetBIOS (стек SMB) – 0xF0, для стека Banyan – BC, для стека IPX/SPX – E0 и т. д.. Для одних протоколов определена только одна точка входа и, соответственно, только один SAP, а для других – несколько. В большинстве случаев адреса DSAP и SSAP совпадают. Например, если в кадре LLC значения DSAP и SSAP содержат код протокола IPX, то обмен кадрами осуществляется между двумя IPX-модулями, выполняющимися в разных узлах. Но в некоторых случаях в кадре LLC указываются различающиеся DSAP и SSAP. Это возможно только в тех

случаях, когда протокол имеет несколько адресов SAP, что может быть использовано протоколом узла отправителя в специальных целях, например для уведомления узла получателя о переходе протокола-отправителя в некоторый специфический режим работы. Этим свойством протокола LLC часто пользуется протокол NetBEUI.

Таким образом, в качестве имён сетевых узлов на подуровне LLC используются имена вышестоящих протоколов и эти имена определяются тем, какие стеки сетевых протоколов поддерживаются операционной системой, установленной на данном сетевом узле. И, несмотря на то, что имена протоколов (коды протоколов) инвариантны – они определяются централизованно («глобально»), тем не менее, само именование объектов в конкретном сетевом узле (состав протоколов узла) является локальным и зависят от того, какие стеки сетевых протоколов поддерживает ОС данного узла. Пример: на компьютере – Windows, на системной плате – встроенный порт Ethernet, в слот системной платы вставлена карта WiFi-модуля, а в USB-порты – модем GSM и модем BlueTooth. Тогда на компьютере используются как минимум четыре сетевых технологии и два стека сетевых протоколов, то есть, он подключен одновременно к четырём локальных сетям и может быть настроен как маршрутизатор.

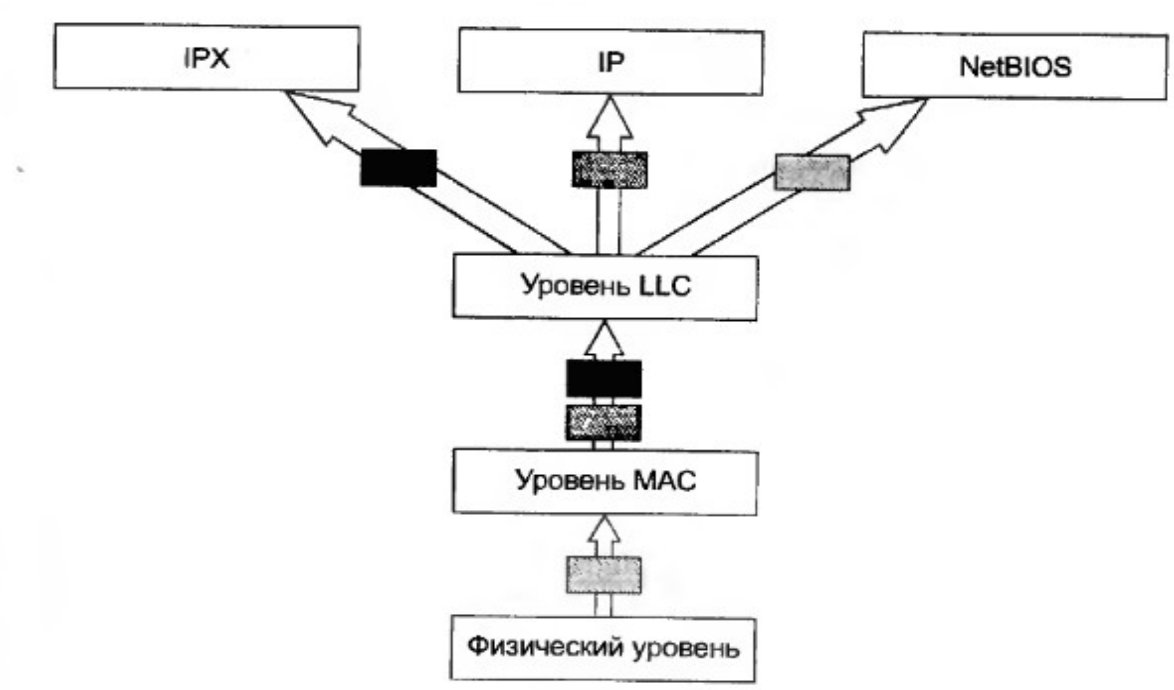


Рис. 22. Разинкапсуляция и демультиплексирование кадров протоколом LLC

Поле управления (1 или 2 байта) имеет сложную структуру при рабо-

те в режиме LLC2 и достаточно простую структуру при работе в режиме LLC1 (рис. 23).

Рис. 23. Структура поля управления кадров LLC.

5.7. Понятие сетевой технологии и примеры

5.7.1. Сетевая технология Ethernet (802.3)

Ethernet – это самый распространенный на сегодняшний день стандарт локальных сетей. Общее количество сетей, работающих по протоколу Ethernet в настоящее время, оценивается в десятки миллионов, а количество компьютеров с установленными сетевыми адаптерами Ethernet – в сотни миллионов.

Когда говорят Ethernet, то под этим обычно понимают любой из вариантов (релизов) этой технологии. В зависимости от типа физической среды стандарт IEEE 802.3 имеет различные спецификации – l0Base-5, l0Base-2, l0Base-T, l0Base-FL, l0Base-FB и др., всего несколько десятков.

Принятые в последующем стандарты Fast Ethernet (100Base-X), Gigabit Ethernet (1000Base-X), 10-ти Гигабитный Ethernet (10GBase-X), во многом не являются самостоятельными стандартами, о чем говорит и тот факт, что их описание являются дополнительными разделами к основному стандарту 802.3 – разделами 802.3u (100Base-T), 802.3z (1000Base-SX/LX/CX), 802.3ab (1000Base-T) и т. д.

Для передачи двоичной информации по кабелю для всех вариантов физического уровня технологии Ethernet, обеспечивающих пропускную способность 10 Мбит/с, используется импульсный манчестерский код, в других стандартах, более скоростных, используются, как правило, разные виды потенциального кодирования, поскольку они позволяют передавать информацию на более низких частотах и тем самым, снизить требования к кабельной системе – релиз 100base-T – кодирование 4B/5B (то есть, байт расширяется до 10 бит), релиз 100Base-T4 – кодирование 8B/6T (то есть,

байт расширяется до 18 бит, и данных передаётся в два раза больше, но при этом снижаются требования к кабелю – полоса пропускания до 35 МГц, а это значит, что можно использовать дешёвый низкоскоростной кабель третьей категории – 4 пары).

Все виды стандартов Ethernet используют один и тот же метод разделения среды передачи данных, называемый методом коллективного доступа с опознаванием несущей и обнаружением коллизий (carrier-sense-multiply-access with collision detection, CSMA/CD).

5.7.2. Алгоритм CSMA/CD

Метод применяется исключительно в сетях с логической общей шиной. Все компьютеры такой сети имеют непосредственный доступ к общей шине, поэтому она может быть использована для передачи данных между любыми двумя узлами сети. Одновременно все компьютеры сети имеют возможность немедленно (с учетом задержки распространения сигнала по физической среде) получить данные, которые любой из компьютеров начал передавать на общую шину (рис. 24). Простота схемы подключения определяет дешевизну реализации метода – эти факторы и определили успех стандарта Ethernet. Говорят, что кабель, к которому подключены все станции, работает в режиме коллективного доступа (Multiply Access, MA).

Чтобы получить возможность передавать кадр, станция должна убедиться, что разделяемая среда свободна. Это достигается прослушиванием основной гармоники сигнала, которая также называется несущей частотой (carrier-sense, CS). Признаком незанятости среды является отсутствие на ней несущей частоты, которая при манчестерском способе кодирования равна 5-10 МГц, в зависимости от последовательности единиц и нулей, передаваемых в данный момент.

Если среда свободна, то узел имеет право начать передачу кадра. Этот кадр изображен на рис. 24 первым. Узел *1* обнаружил, что среда свободна, и начал передавать свой кадр. В классической сети Ethernet на коаксиальном кабеле сигналы передатчика узла *1* распространяются в обе стороны, так что все узлы сети их получают. Кадр данных всегда сопровождается *преамбулой (preamble)*, которая состоит из 7 байт, равных 10101010, и 8-го байта, равного 10101011. Преамбула нужна для вхождения приемника в побитовый и побайтовый синхронизм с передатчиком.

Рис. 24. Метод случайного доступа CSMA/CD

Все станции, подключенные к кабелю, могут распознать факт передачи кадра, и та станция, которая узнает собственный адрес в заголовках кадра, записывает его содержимое в свой внутренний буфер, обрабатывает полученные данные, передает их вверх по своему стеку (см. рис. 22).

Узел *2* во время передачи кадра узлом *1* также пытался начать передачу своего кадра, однако обнаружил, что среда занята – на ней присутствует несущая частота, – поэтому узел *2* вынужден ждать, пока узел *1* не прекратит передачу кадра.

После окончания передачи кадра все узлы сети обязаны выдержать технологическую паузу (Inter Packet Gap) в 9,6 мкс. Эта пауза, называемая также межкадровым интервалом, нужна для приведения сетевых адаптеров в исходное состояние, а также для предотвращения монопольного захвата среды одной станцией. После окончания технологической паузы узлы имеют право начать передачу своего кадра, так как среда свободна. Из-за задержек распространения сигнала по кабелю не все узлы строго одновременно фиксируют факт окончания передачи кадра узлом *1*.

В приведенном примере узел *2* дождался окончания передачи кадра узлом *1,* сделал паузу в 9,6 мкс и начал передачу своего кадра.

Возникновение коллизии. При описанном подходе возможна ситуация, когда две станции одновременно пытаются передать кадр данных по общей среде. Механизм прослушивания среды и пауза между кадрами не

гарантируют от возникновения такой ситуации, когда две или более станции одновременно решают, что среда свободна, и начинают передавать свои кадры. Говорят, что при этом происходит *коллизия (collision),* так как содержимое обоих кадров сталкивается на общем кабеле и происходит искажение информации – методы кодирования, используемые в Ethernet, не позволяют выделять сигналы каждой станции из общего сигнала.

Для возникновения коллизии не обязательно, чтобы несколько станций начали передачу абсолютно одновременно, такая ситуация маловероятна. Гораздо вероятней, что коллизия возникает из-за того, что один узел начинает передачу раньше другого, но до второго узла сигналы первого просто не успевают дойти к тому времени, когда второй узел решает начать передачу своего кадра. То есть коллизии – это следствие распределенного характера сети.

Для корректной обработки коллизии, станции одновременно с передачей также слушают среду и сравнивают передаваемый сигнал с принимаемым. Если передаваемые и принимаемые сигналы отличаются, то фиксируется *обнаружение коллизии (collision detection, CD).* Для увеличения вероятности скорейшего обнаружения коллизии всеми станциями сети станция, которая обнаружила коллизию, прерывает передачу своего кадра (в произвольном месте, возможно, и не на границе байта) и усиливает ситуацию коллизии посылкой в сеть специальной последовательности из 32 бит, называемой *jam-последовательностью.*

После этого обнаружившая коллизию передающая станция обязана прекратить передачу и сделать паузу в течение короткого случайного интервала времени. Затем она может снова предпринять попытку захвата среды и передачи кадра. Случайная пауза выбирается по следующему алгоритму:

*Пауза = L *(интервал отсрочки),*

где интервал отсрочки равен 512 битовым интервалам (в технологии Ethernet принято все интервалы измерять в битовых интервалах; битовый интервал обозначается как bt и соответствует времени между появлением двух последовательных бит данных на кабеле; для скорости 10 Мбит/с величина битового интервала равна 0,1 мкс или 100 нс); *L* представляет собой целое число, выбранное с равной вероятностью из диапазона [0, 2N], где *N* – номер повторной попытки передачи данного кадра: 1,2,..., 10.

После 10-й попытки интервал, из которого выбирается пауза, не увеличивается. Таким образом, случайная пауза может принимать значения от 0 до 52,4 мс.

Если 16 последовательных попыток передачи кадра вызывают коллизию, то передатчик должен прекратить попытки и отбросить этот кадр.

Ограничения алгоритма. Из описания алгоритма видно, что он носит вероятностный характер и вероятность успешного получения в свое распоряжение общей среды зависит от загруженности сети. При значительной загрузке сети (более 30%) коллизии возникают настолько часто, что полезная пропускная способность сети Ethernet резко падает, так как сеть почти постоянно занята повторными попытками передачи кадров. Для уменьшения интенсивности возникновения коллизий нужно либо уменьшить трафик, сократив, например, количество узлов в сегменте, либо перейти на более скоростной протокол.

Следует отметить, что метод доступа CSMA/CD вообще не гарантирует станции, что она когда-либо сможет получить доступ к среде. Конечно, при небольшой загрузке сети вероятность такого события невелика, но при коэффициенте использования сети, приближающемся к 1, такое событие становится очень вероятным. Другие методы доступа – маркерный доступ сетей Token Ring и FDDI, метод Demand Priority сетей 100VG-AnyLAN – свободны от этого недостатка.

Общие ограничения сетевой технологии Ethernet (классической):

Параметр	**Значение**
Битовая скорость	10 Мбит/с
Интервал отсрочки	512 битовых интервала
Межкадровый интервал	9.6 мкс
Максимальное число попыток передачи	16
Максимальное число возрастания диапазона паузы	10
Длина jam-последовательности	32 бита
Максимальная длина кадра (без преамбулы)	1518 байт
Минимальная длина кадра (без преамбулы)	64 байта (512 бит)
Длина преамбулы	64 бит
Минимальная длина случайной паузы после коллизии	0 битовых интервала
Максимальная длина случайной паузы после коллизии	524 000 битовых интервала
Максимальное рассмояние между станциями сети	2500 м
Максимальное число станций в сети	1024

Однако, важно отметить, что каждый вариант (релиз) физической среды технологии Ethernet добавляет к этим ограничениям свои, часто бо-

лее строгие ограничения, которые также должны выполняться. Кроме того, нужно учитывать эксплуатационные ограничения, возникающие при использовании технологии, например, объём трафика, порождаемый сетевыми узлами при использовании конкретных программ.

Производительность сети Ethernet зависит от:

- загрузки сети (количества порождаемых сетевыми узлами пакетов данных в единицу времени),
- размера пакетов:
  ~ (минимальный кадр = 64 байта) + (преамбула = 8 байт), итого 72 байта,
  ~ (максимальный кадр = 1518 байт) + (преамбула = 8 байт), итого = 1526 байт.

Отсюда, учитывая особенности алгоритма Ethernet, несложно подсчитать, что минимальная «чистая» (полезная по данным, за вычетом «накладных» расходов) пропускная способность этой сетевой технологии не превышает 5 Мбит/с при минимальных кадрах и 9 Мбит/с при максимальных кадрах.

5.7.3. Сетевая технология Token Ring

Здесь также используется разделяемая среда передачи данных, которая в данном случае состоит из отрезков кабеля, соединяющих все станции сети в кольцо. Кольцо рассматривается как общий разделяемый ресурс, и для доступа к нему требуется не случайный алгоритм, как в сетях Ethernet, а детерминированный, основанный на передаче станциям права на использование кольца в определенном порядке. Это право передается с помощью кадра специального формата, называемого *маркером* или *токеном (token)*.

Технология Token Ring является более сложной технологией, чем Ethernet. Она обладает свойствами отказоустойчивости. В сети Token Ring определены процедуры контроля работы сети, которые используют обратную связь кольцевой топологии – посланный кадр всегда возвращается в станцию – отправитель. В некоторых случаях обнаруженные ошибки в работе сети устраняются автоматически, например может быть восстановлен потерянный маркер. В других случаях ошибки только фиксируются, а их устранение выполняется вручную обслуживающим персоналом.

Для контроля сети одна из станций выполняет роль так называемого *активного монитора*. Активный монитор выбирается во время инициализации кольца как станция с максимальным значением MAC-адреса, Если активный монитор выходит из строя, процедура инициализации кольца по-

вторяется и выбирается новый активный монитор. Чтобы сеть могла обнаружить отказ активного монитора, последний в работоспособном состоянии каждые 3 секунды генерирует специальный кадр своего присутствия. Если этот кадр не появляется в сети более 7 секунд, то остальные станции сети начинают процедуру выборов нового активного монитора.

5.7.4. Алгоритм маркерного метода доступа

Для обеспечения доступа станций к физической среде по кольцу циркулирует кадр специального формата и назначения – маркер (см. рис. 25). В сети Token Ring любая станция всегда непосредственно получает данные только от одной станции – той, которая является предыдущей в кольце. Такая станция называется *ближайшим активным соседом, расположенным выше по потоку* (данных) – *Nearest Active Upstream Neighbor, NAUN*. Передачу же данных станция всегда осуществляет своему ближайшему соседу вниз по потоку данных.

Start Delimiter	Access Control	End Delimiter
1 байт	1 байт	1 байт

Рис. 25. Формат кадра маркера Token Ring

Получив маркер, станция, при отсутствии у нее данных для передачи, обеспечивает его продвижение к следующей станции. Станция, которая имеет данные для передачи, при получении маркера изымает его из кольца и вместо него выдает в кольцо кадр данных установленного формата (см. рис. 26). Переданные данные проходят по кольцу всегда в одном направлении от одной станции к другой.

Start Delimiter	Access Control	Frame Control	Destination Address	Source Address	Data Данные	CRC Контрольная сумма	End Delimiter
1 байт	1 байт	1 байт	6 байт	6 байт	>0	4 байта	1 байт

Рис. 26. Формат информационного кадра Token Ring

Все станции кольца ретранслируют кадр побитно, как повторители. Если кадр проходит через станцию назначения, то, распознав свой адрес, эта станция копирует кадр в свой внутренний буфер, вставляет в кадр признак распознавания адреса и признак подтверждения приема (флаг приёма) и транслирует кадр дальше. Станция, выдавшая кадр данных в кольцо, при

обратном его получении с установленными флагами изымает этот кадр из кольца и вместо него возвращает в сеть новый маркер для обеспечения возможности другим станциям сети передавать данные. Такой алгоритм доступа применяется в сетях Token Ring со скоростью работы 4 Мбит/с, описанных в стандарте 802.5 (см. рис. 27).

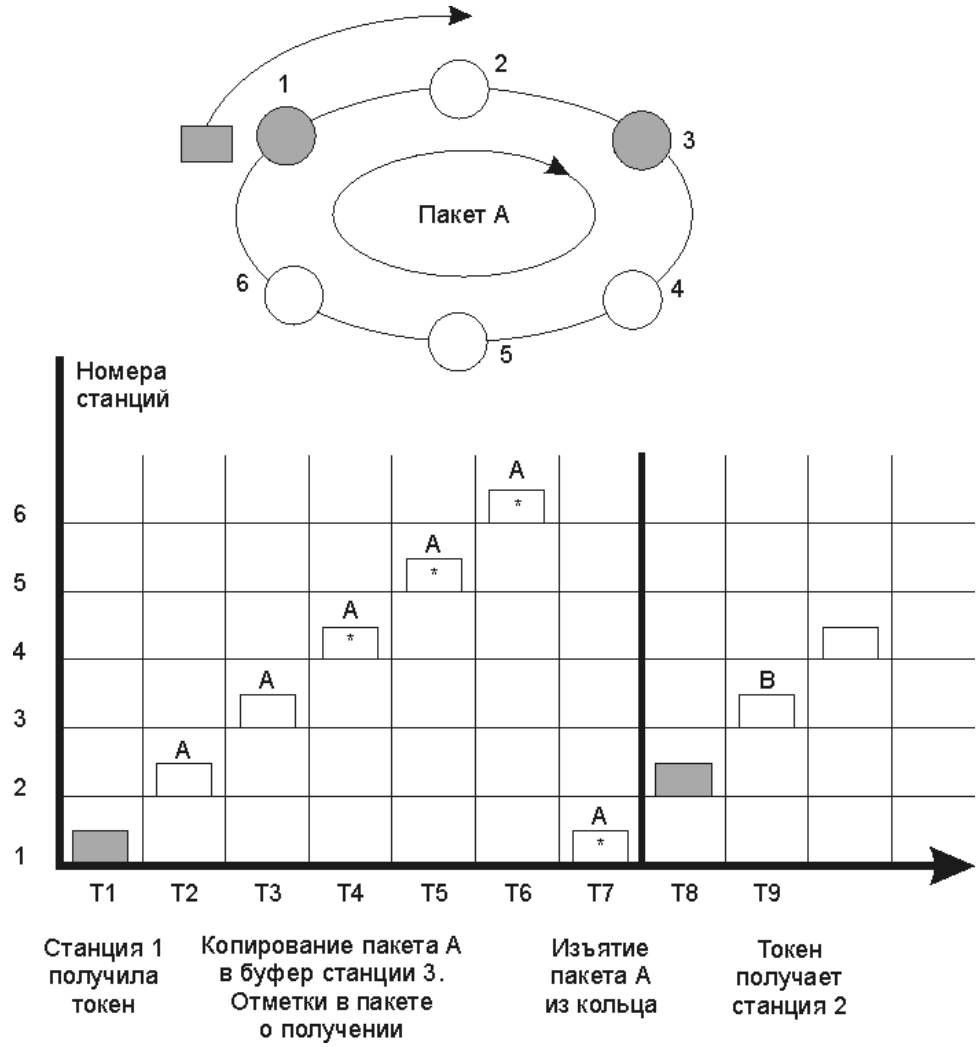


Рис. 27. Принцип маркерного доступа.

Время владения разделяемой средой в сети Token Ring ограничивается *временем удержания маркера (token holding time)*, после истечения которого станция обязана прекратить передачу собственных данных (текущий кадр разрешается завершить) и передать маркер далее по кольцу. Станция может успеть передать за время удержания маркера один или несколько кадров в зависимости от размера кадров и величины времени удержания маркера. Обычно время удержания маркера по умолчанию равно 10 мс, а максимальный размер кадра в стандарте 802.5 не определен. Для сетей

4 Мбит/с он обычно равен 4 Кбайт, а для сетей 16 Мбит/с – 16 Кбайт. Это связано с тем, что за время удержания маркера станция должна успеть передать хотя бы один кадр. При скорости 4 Мбит/с за время 10 мс можно передать 5000 байт, а при скорости 16 Мбит/с – соответственно 20 000 байт. Максимальные размеры кадра выбраны с некоторым запасом.

В сетях Token Ring 16 Мбит/с используется также несколько другой алгоритм доступа к кольцу, называемый алгоритмом *раннего освобождения маркера (Early Token Release)*. В соответствии с ним станция передает маркер доступа следующей станции сразу же после окончания передачи последнего бита своего кадра данных, не дожидаясь возвращения по кольцу этого кадра с битом подтверждения приема. В этом случае пропускная способность кольца используется более эффективно, так как по кольцу одновременно продвигаются кадры нескольких станций. Тем не менее свои кадры в каждый момент времени может генерировать только одна станция – та, которая в данный момент владеет маркером доступа. Остальные станции в это время только повторяют чужие кадры, так что принцип разделения кольца во времени сохраняется, ускоряется только процедура передачи владения кольцом.

Для различных видов сообщений передаваемым кадрам могут назначаться различные *приоритеты*: от 0 (низший) до 7 (высший). Решение о приоритете конкретного кадра принимает передающая станция (протокол Token Ring получает этот параметр через межуровневые интерфейсы от протоколов верхнего уровня, например прикладного). Маркер также всегда имеет некоторый уровень текущего приоритета. Станция имеет право захватить переданный ей маркер только в том случае, если приоритет кадра, который она хочет передать, выше (или равен) приоритета маркера. В противном случае станция обязана передать маркер следующей по кольцу станции.

За наличие в сети маркера, причем единственной его копии, отвечает активный монитор. Если активный монитор не получает маркер в течение длительного времени (например, 2,6 с), то он порождает новый маркер.

5.7.5. Сетевая технология 100VG-AnyLAN

Эта сетевая технология реализует передачу данных со скоростью 100 Мбит/сек по кабелям категории 3. Технология предполагает использование протокола подуровня MAC, который называется DemandPriority (DP) – приоритет требования. Основными её отличиями являются достаточно большая скорость обмена, сравнительно невысокая стоимость аппа-

ратуры (примерно вдвое дороже оборудования наиболее популярной сети Ethernet), централизованный *метод управления* обменом без конфликтов, а также совместимость на уровне *форматов пакетов* с сетями Ethernet и Token-Ring.

5.7.6. Алгоритм метода доступа Demand Priority

Основным компонентом, который обеспечивает выполнения доступа по правилам DP в сетях 100VG-Any LAN, является концентратор-арбитр. Рабочая станция, которая хочет передать кадр данных, первоначально посылает концентратору кадр запроса. Поступившие запросы концентратор обслуживает по принципу циклической очереди. В ответ на запрос концентратор посылает станции кадр разрешения на передачу. Станция формирует информационный кадр и отправляет его концентратору. После получения кадра концентратор направляет этот кадр в порт, к которому подключена станция назначения для данного кадра.

Протокол DP обеспечивает возможность присвоения двух уровней приоритета передаваемым данным:

- Normal Priority (Нормальный приоритет)
- High Priority (высший приоритет)

Запросы нормального приоритета обслуживаются только при отсутствии запросов высшего приоритета. Запросы одного уровня обслуживаются по принципу циклической очереди. Для того чтобы обеспечить выделение гарантированной полосы пропускания для обслуживания запросов нормального уровня протокол DP предполагает использование специального алгоритма манипулирования приоритетами. В соответствии с этим алгоритмом, запросу нормального уровня приоритета, который не был обслужен в течение определенного интервала времени, присваивается высший уровень приоритета.

5.7.7. Сетевая технология ATM

Технология ATM (Asynchronous Transfer Mode) используется как в локальных, так и в глобальных сетях. Основная идея – передача цифровых, голосовых и мультимедийных данных по одним и тем же каналам. До сих пор жесткого стандарта на аппаратуру *ATM* не предполагает. Скорость передачи 155 Мбит/с (для настольных систем – 25 Мбит/с), затем была повышена до 662 Мбит/с, а сейчас ведутся работы по повышению скорости до 2488 Мбит/с. По скорости *ATM* успешно конкурирует с *Gigabit Ethernet*. И кстати, появилась *ATM* раньше, чем *Gigabit Ethernet*. В качестве *среды*

передачи информации в локальной сети технология *ATM* предполагает использование оптоволоконного кабеля и неэкранированной витой пары. Используемые коды – 4B/5B и 8B/10B.

5.7.8. Алгоритм метода доступа ATM

Принципиальное отличие *ATM* от остальных сетей состоит в отказе от привычных пакетов с полями адресации, управления и данных. Вся передаваемая информация упакована в микропакеты (ячейки, cells) длиной 53 байта. Каждая ячейка имеет 5-байтовый заголовок, который позволяет интеллектуальным распределительным устройствам сортировать ячейки и следить за тем, чтобы они передавались в нужной последовательности. Каждая ячейка имеет 48 байт информации. Их минимальный размер позволяет осуществлять коррекцию ошибок и маршрутизацию на аппаратном уровне. Он же обеспечивает равномерность всех информационных потоков сети и минимальное время ожидания доступа к сети.

Заголовок включает в себя идентификаторы пути, канала доставки, типа информации, указатель приоритета доставки, а также контрольную сумму заголовка, позволяющую определить наличие ошибок передачи.

Главный недостаток сетей с технологией *ATM* состоит в их полной несовместимости ни с одной из имеющихся сетей. Плавный переход на ATM в принципе невозможен, нужно менять сразу все оборудование, а стоимость его достаточно высока.

Технология ATM еще в недалеком прошлом считалась перспективной и универсальной, способной потеснить привычные локальные сети. Однако в настоящий момент, вследствие успешного развития традиционных локальных сетей, применение ATM ограничено только глобальными и магистральными сетями и «последней милей» – технология xDSL использует ATM как транспорт.

5.7.9. Сетевая технология PPP

Протокол PPP определяет связь "точка-точка", когда сетевой кабель используется для передачи информации только между двумя компьютерами (или другим сетевым оборудованием), соединенным этим кабелем. Такое соединение характерно при подключении к Internet по телефонной линии, при соединении локальных сетей между собой по выделенным или коммутируемым линиям, а также в сетях X.25, Frame Relay и ATM.

5.7.10. Алгоритм метода доступа PPP

Протокол канального уровня PPP (Point to Point Protocol – протокол точка-точка) позволяет использовать не только протокол IP, но также и другие протоколы сетевого уровня (IPX, AppleTalk и др.). Достигается это за счет того, что в каждом кадре сообщения хранится не только 16-битная контрольная сумма, но и поле, задающее тип сетевого протокола.

Протокол PPP также поддерживает сжатие заголовков IP-пакетов по методу Ван Джакобсона (VJ-сжатие), а также позволяет согласовать максимальный размер передаваемых дейтаграмм, IP-адреса сторон и др. Аутентификация в протоколе PPP является двусторонней, т.е. каждая из сторон может потребовать аутентификации другой.

Процедура аутентификации проходит по одной из двух схем:

а) PAP (Password Authentication Protocol) – в начале соединения на сервер посылается имя пользователя и (возможно зашифрованный) пароль.

б) CHAP (Challenge Handshake Authentication Protocol) – в начале соединения сервер посылает клиенту случайный запрос (challenge). Клиент шифрует свой пароль, используя однонаправленную хэш-функцию (функция у которой по значению Y невозможно определить X) и запрос, в качестве ключа шифрования. Зашифрованный отклик (response) передается серверу, который, имея в своей базе данных пароль клиента, выполняет те же операции и, если полученный от клиента отклик совпадает с вычисленным сервером, то аутентификация считается успешной. Таким образом, пароль по линиям связи не передается. Даже если отклик клиента и будет перехвачен, то в следующий раз использовать его не удастся, т.к. запрос сервера будет другим. Определить же пароль на основании отклика – невозможно, т.к. хэш-функция шифрует данные только "в одну сторону". Для предотвращения вмешательства в соединение уже после прохождения клиентом аутентификации, в схеме CHAP сервер регулярно посылает испытательные запросы через равные промежутки времени. При отсутствии отклика или неверном отклике соединение прерывается.

5.7.11. Сетевая технология 802.11.x – сотовая связь

С конца 90-х годов 20 века наблюдается настоящий бум беспроводных локальных сетей (*WLAN* – Wireless LAN). Это связано в первую очередь с успехами технологии и с теми удобствами, которые способны предоставить беспроводные сети. К настоящему времени, число пользователей беспроводных сетей достигло уже сотен миллионов, и почти все выпускаемые

мобильные компьютеры оснащены встроенными средствами доступа к таким сетям.

В 1997 году был принят стандарт для беспроводных сетей IEEE 802.11. Сейчас этот стандарт активно развивается и включает в себя уже несколько разделов, в том числе три локальные сети (802.11a, 802.11b и 802.11g). Для этих стандартов есть общее название – Wi-Fi (wireless fidelity).

5.7.12. Алгоритм метода доступа 802.11

Оборудование беспроводных сетей включает в себя точки беспроводного доступа (Access Point) и беспроводные адаптеры для каждого абонента.

Точки доступа выполняют роль концентраторов, обеспечивающих связь между абонентами и между собой, а также функцию мостов, осуществляющих связь с кабельной локальной сетью и с Интернет. Несколько близко расположенных точек доступа образуют зону доступа *Wi-Fi*, в пределах которой все абоненты, снабженные беспроводными адаптерами, получают доступ к сети. Такие зоны доступа (Hotspot) создаются в местах массового скопления людей: в аэропортах, студенческих городках, библиотеках, магазинах, бизнес-центрах и т.д.

Каждая точка доступа может обслуживать несколько абонентов, но чем больше абонентов, тем меньше эффективная скорость передачи для каждого из них, поскольку используется метод доступа к сети – CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance). Сеть строится по сотовому принципу. В сети предусмотрен механизм роуминга, то есть поддерживается автоматическое подключение к точке доступа и переключение между точками доступа при перемещении абонентов, хотя строгих правил роуминга стандарт не устанавливает.

Поскольку радиоканал не обеспечивает высокой степени защиты от прослушивания, в сети *Wi-Fi* используется специальный встроенный механизм защиты информации. Он включает средства и процедуры аутентификации для противодействия несанкционированному доступу к сети и шифрование для предотвращения перехвата информации.

5.7.13. Сетевая технология 802.15.1 – BlueTooth

Bluetooth – это технология передачи данных по радио на короткую дистанцию, позволяющая осуществлять связь беспроводных телефонов, компьютеров и различной периферии при отсутствии прямой видимости.

Целью разработки было получение нового радиоинтерфейса с низким уровнем энергопотребления и невысокой стоимостью, который позволил бы устанавливать связь между сотовыми телефонами и беспроводными гарнитурами.

Системе беспроводной связи Bluetooth в спектре радиочастот отведено 79 каналов в полосе 37 МГц (примерно 2 МГц каждый) в диапазоне 2,4465-2,4835 ГГц.

Суть стандарта Bluetooth заключается в оснащении электронных устройств приёмопередатчиками, работающими на частоте 2,45 Ггц, имеющими радиус действия до 10 м и скорость передачи информации до 1 Мбит/с. Возможности применения данных устройств поистине безграничны – беспроводные наушники, мышки, клавиатуры, соединение мобильных телефонов и ноутбуков, обмен информацией между карманными компьютерами.

В состав технологии входят радиомодуль-трансивер, контроллер связи (он же процессор) и управляющее устройство, собственно реализующее протоколы Bluetooth верхних уровней, а также интерфейс с терминальным устройством.

5.7.14. Алгоритм метода доступа 802.15.1

Технология использует скачкообразную перестройку частоты (1600 скачков/с) с расширением спектра. При работе передатчик перескакивает с одной рабочей частоты на другую по псевдослучайному алгоритму. Для разделения приёмного и передающего каналов используется временное разделение. Поддерживается синхронная и асинхронная передача данных. Временные интервалы синхронизированы для передачи пакетов, каждый из которых передаётся на своей частоте радиосигнала.

Потребление мощности устройств Bluetooth должно быть в пределах 0,1 Вт. Каждое устройство имеет уникальный 48-битный сетевой адрес, совместимый с форматом стандарта локальных сетей IEEE 802.

Каждый Bluetooth-модуль содержит формирующую и приёмно- передающую аппаратуру, а также встроенное или «зашитое» программное обеспечение (Firmware). К последнему относится интерфейс хост-контроллера (HCI), менеджер связи (Link Manager), а также контроллер несущей частоты (Baseband). Связь модуля с хостом на физическом и канальном уровнях осуществляется с помощью шин USB, UART, PC Card и соответствующего встроенного программного обеспечения. К физическому уровню относится также радиолиния между модулями.

Модуль поддерживает приём – передачу данных и речевых сигналов. Связь между модулем и хост-контроллером производится с помощью высокоскоростного USB-интерфейса или UART/PCM-интерфейса. Когда используется USB-интерфейс, модуль является USB-ведомым прибором и поэтому не требует ресурсов персонального компьютера.

Интерфейс хост-контроллера (ИХК) в модуле является командным интерфейсом. Хост через ИХК направляет команды, а в ответ принимает от модуля сообщения об их выполнении. Менеджер связи устанавливает необходимую конфигурацию ИХК.

Технология Bluetooth предполагает два вида связи: синхронную – SCO (Synchronous Connection Oriented) и асинхронную – ACL (Asynchronous Connectionless). Первый вид, SCO, рассчитан на установление симметричного соединения «точка – точка» и служит преимущественно для передачи речевых сообщений. Скорость передачи информации SCO равна 64 Кит/с. Второй, ACL, предназначен для пакетной передачи данных. Он поддерживает симметричные и асимметричные соединения типа «точка – много точек». Скорость передачи пакетной информации при ACL составляет порядка 721 Кбит/с. Пакеты данных имеют фиксированный формат. В начале блока находится 72-бит код доступа. Он может применяться, в частности, для синхронизации устройств. За ним следует 54-бит заголовок пакета, содержащий контрольную сумму пакета и информацию о его параметрах (например, о повторной передаче блока данных). Замыкает пакет область, непосредственно содержащая пересылаемую информацию. Размер этой области варьируется от 0 до 2745 бит.

Когда пара любых Bluetooth-устройств соединяется, они образуют пикосеть. Аппарат, инициирующий связь, является ведущим (host, master), а остальные – ведомыми (slaves). Обычно ведущим является тот модуль, который размещён в наиболее мощном устройстве, таком, как персональный компьютер или плата CPU мини-ЭВМ. Число модулей в беспроводном соединении не ограничивается, но в любой момент времени активны должны быть не больше восьми. Не существует разницы как в аппаратной, так и в программной части между ведущими и ведомыми устройствами. Любое из них может быть и тем и другим. Ведущее формирует беспроводное соединение (в каждой сети оно только одно) и полностью контролирует трафик. Ведомые могут отсылать сообщения только в интервале «ведомые – ведущему» после того, как к ним обратился в предшествующий слот «ведущий – ведомым». Если в этом интервале у ведущего нет никакой информации для отправки ведомым, то он передает пакет только с кодом доступа и

заголовком. Если в сети оказывается более 8 устройств, то будет сформирована вторая пикосеть и так далее. Предусмотрена координация трафика и между сетями. Множество пикосетей, способных взаимодействовать друг с другом, формируют распределённую сеть (Scatternet).

5.7.15. Сетевая технология 802.15.4 – MiWi

MiWi является простым протоколом беспроводных сетей, ориентированным на низкие скорости передачи данных, небольшие расстояния и низкую стоимость реализации узла. MiWi основан на спецификации IEEE802.15.4 для беспроводных персональных сетей (WPAN) и является достойной альтернативой существующим стекам протоколов для беспроводных сетей. Он ориентирован на сети небольшого размера с ограниченным количеством маршрутизаторов, построенные на трансиверах MCRF24J40 IEEE 802.15.4-совместимых передатчиках производства фирмы Microchip.

Сеть, построенная на основе протокола MiWi, может иметь до 1024 узлов. В ней могут работать до 8 координаторов, каждый из которых поддерживает до 127 узлов. Существует также ограничение на длину маршрута передаваемого пакета: максимум четыре хопа (прохода через координатор) для конечных узлов и максимум 2 хопа для PAN-координатора.

5.7.16. Алгоритм метода доступа 802.15.4

При описании протокола MiWi используется два термина, заимствованные из стандартов IEEE.

Кластер – группа узлов, формирующая сеть. В рамках MiWi кластер может иметь 3 уровня иерархии. В верхнем уровне находится главный узел PAN-координатор.

Сокет – виртуальное соединение между двумя узлами. В отличие от прямого соединения между двумя узлами с помощью проводов, здесь все узлы имеют виртуальные соединения с использованием одной физической среды передачи данных.

Протокол MiWi базируется на спецификации IEEE 802.15.4 для уровней MAC и PHY и предназначен для построения простых беспроводных сетей диапазона 2,4 ГГц.

В рамках протокола производится формирование сети, подключение новых узлов, маршрутизация. Протокол не описывает такие специфические функции, как регистрация узла в определенной сети, определение обрыва

соединения и частоты обмена информацией между узлами и т. д.

Максимальная длина пакета, включая 16-битную контрольную сумму, 127 байт.

Для контроля подтверждения получения пакета, помимо проверочной контрольной суммы, в рамках стандарта на канальном уровне заложен механизм подтверждений. В заголовке пакета предусмотрен бит ACK: если этот бит установлен, то источник нуждается в подтверждении от приемника. Если источник не получает подтверждения в течение определенного времени, он предпринимает несколько попыток повтора передачи, прежде чем сообщает об ошибке.

Получение подтверждения от приемника говорит только о том, что пакет правильно принят на канальном уровне, но это вовсе не значит, что он был правильно обработан верхними уровнями. Например, если при приеме пакета будет высокая загруженность вычислительных ресурсов приемного узла, аппаратура канального уровня выдаст подтверждение источнику, но пакет не будет обработан корректно. Для сохранения целостности данных необходимо принимать дополнительные меры на верхних уровнях (в пользовательском протоколе).

Протокол MiWi использует адресацию в соответствии с IEEE 802.15.4. Используется три составляющих:

Расширенный уникальный идентификатор (EUI) 8-байтный глобальный идентификатор; каждое производимое в мире устройство, совместимое со стандартом, должно иметь уникальный EUI. Старшие три байта являются идентификатором организации, который выделяет IEEE. Младшие пять байтов пользователь заполняет сам, учитывая уникальность устройств;

Идентификатор PAN (PANID) 16-битный адрес, определяющий группу устройств. Все узлы одной сети имеют один PANID, что и говорит об их принадлежности к конкретной сети;

Короткий адрес устройства, выдаваемый ему при регистрации в текущей сети. Короткий адрес (16 бит) совместно с PANID являются уникальной комбинацией и используются для адресации в конкретной сети. PAN-координатор всегда имеет адрес 0000h.

В сетях MiWi могут использоваться короткие 16-битные адреса.

5.8. Стеки сетевых протоколов

5.8.1. ЭМВОС и структура стеков протоколов

Стеки сетевых протоколов определяются на 3-7 уровнях ЭМВОС. Стеки протоколов появились давно – в 70-80-ые годы XX-го века, поскольку в те далёкие годы вычислительная техника производилась фирмами/организациями в соответствии со своими представлениями о том, что такое сеть и как она должна работать. Ни о какой унификации и совместимости никто не думал. В результате появились десятки реализаций межкомпьютерного информационного взаимодействия – сетей, не совместимых друг с другом и на аппаратном, и на программном уровнях.

Однако в 90-ые годы с переходом от локальных к корпоративным, а затем и к глобальным сетям, появилась острая необходимость в обеспечении совместимости аппаратного и программного обеспечения сетей. Реализовывалась совместимость посредством унификации и стандартизации интерфейсов и протоколов и при этом важную роль играла также цена реализаций и цена обеспечения совместимости.

В итоге к концу 90-ых годов оформилась новая структура сетевого взаимодействия. В основу структуры был положен стек сетевых протоколов OSI, разработанный в 80-ые годы для реализации на мейнфреймах. Поскольку разрабатывался он для больших ЭВМ, то никаких особых ограничений при его разработке не выдвигалось и он получился сложным, но полнофункциональным и хорошо документированным. Но 90-ые годы – это годы бума ПЭВМ, что привело к тому, что мейнфреймы стали экзотикой, а для ПЭВМ стек OSI был слишком сложным для реализации. В результате стек OSI в силу свой полнофункциональности и документированности был определён как Эталонная Модель Взаимодействия Открытых Систем – ЭМВОС.

В соответствии с этой моделью вся структура сетевого взаимодействия делилась на две группы уровней: нижнюю группу образовывали 1-ый и 2-ой уровни Эталонной Модели и в этой группе определялось взаимодействие в локальных сетях, то есть, разнообразные аппаратные реализации, определявшие физическую организацию сетей. Эти нижние 1 и 2 уровни ЭМВОС в 90-ые годы оформились и унифицировались в рамках понятия «сетевые технологии». Они являются базой для понятия «локальная сеть» как минимально необходимой совокупности аппаратно-программного обеспечения для организации сетевого взаимодействия. Алгоритмы на этих уровнях достаточно просты и представимы в виде автоматов, следова-

тельно, могут быть реализованы аппаратно, что и делается в виде различных сетевых устройств: сетевых карт, модемов, мультиплексоров, коммутаторов, концентраторов и т. п. В силу аппаратной реализации эти уровни являются операционнонезависимыми, то есть, для работы с устройствами в ОС необходимы только драйверы, реализующие интерфейс с ними, и более никак эти устройства на работу ОС не влияют.

А, вот, с верхней группой уровней (3-7) ситуация совсем иная. Во-первых, они алгоритмически намного сложнее и в настоящее время не могут быть представлены в виде автоматов, а потому не реализуемы аппаратно. А поскольку они реализуются только программно, то это определяет их зависимость от операционной системы, под которую они написаны. Более того, уровни 3, 4 и 5 для обеспечения эффективности и скорости работы реализуются как модули операционных систем и встроены в ядра ОС и только прикладные протоколы реализуются вне ОС, но как программы под определённую ОС. То есть, стек сетевых протоколов всегда операционнозависим: стек SMB – это Windows, стек IPX/SPX – это NetWare, стек AppleTalk – это Mac OS, стек TCP/IP – это unix, стек SNA – это стек мейнфреймов (OS390) и серверов IBM и т. д.

В результате в начале 90-х годов стеков сетевых протоколов существовало столько же, сколько операционных систем. А потом пришёл Интернет и всех «выгнал», поскольку Интернет базируется на unix-овом стеке TCP/IP, и потому к настоящему времени из всего многообразия стеков остались только:

- стек TCP/IP – unix/linux и основа Интернета;

- стек SMB – в силу распространённости Windows; MS и рада бы уже от него избавиться, но, ведь, он, зараза, встроен в ядро ОС, его просто так не отключишь;

- стек AppleTalk – как тяжёлая наследственность на Mac'ах: после перевода ядра ОС на FreeBSD (Mac OS X – уже TCP/IP) фирме Apple приходится поддерживать стек AppleTalk как наследуемый;

- стек SNA – можно встретить на мейнфреймах IBM – если вы с ними встретитесь.

Все остальные стеки сетевых протоколов «вымерли». Причём, самая интересная ситуация была со стеком IPX/SPX фирмы Novell: в середине 90-х годов почти 70 % всех локальных и корпоративных сетей работали на NetWare, но большая часть – нелегально, поскольку для легального использования NetWare необходимо было регистрироваться в Novell и покупать у неё номера сетей. К сожалению, руководство Novell не осознало, чем оно

владеет, и не сделало регистрацию и выделение номеров сетей бесплатным процессом – и где теперь Novell? Следствие: Интернет базируется на TCP/IP, хотя всё было готово к тому, чтобы он базировался на стеке IPX/SPX.

5.8.2. Отличительные особенности стеков

Как уже было сказано, стеки сетевых протоколов являются операционнозависимыми. Это прежде всего означает, что они разрабатывались и писались в рамках разработки соответствующих операционных систем и в соответствии с теми взглядами и представлениями на сетевое взаимодействие, которое имели разработчики ОС. Как следствие, они получились разными и отличаются друг от друга также, как и сами операционные системы и также несовместимы (см. рис. 28).

<table>
<tr><th>Модель OSI</th><th>Стек SMB
IBM/Microsoft
(Windows, OS/2)</th><th>Стек TCP/IP

(unix, linux и др.)</th><th>Стек IPX/SPX
Novell
(NetWare)</th><th>Стек OSI</th></tr>
<tr><td>7. Прикладной</td><td rowspan="2">SMB</td><td rowspan="2">Telnet, FTP, SNMP, SMTP, POP, IMAP, HTTP, Torrent</td><td rowspan="2">NCP, SAP</td><td>X400, X.500, FTAM</td></tr>
<tr><td>6. Представительный</td><td>Представительный протокол OSI</td></tr>
<tr><td>5. Сеансовый</td><td rowspan="2">NetBIOS/NetBEUI</td><td></td><td></td><td>Сеансовый протокол OSI</td></tr>
<tr><td>4. Транспортный</td><td>TCP, UDP</td><td>SPX</td><td>Транспортный протокол OSI</td></tr>
<tr><td>3. Сетевой</td><td>нет</td><td>IP, RIP. OSPF, ICMP, IGMP, ARP,</td><td>IPX, RIP, NLSP</td><td>ES-ES, IS-IS</td></tr>
<tr><td rowspan="2">2. Канальный</td><td colspan="4">2.1. “Окно”: 802.2 LLC (LLC1, LLC2, LLC3)</td></tr>
<tr><td colspan="4">2.2. Сетевые технологии: 802.3x (Ethernet), 802.5 (Token Ring), FDDI, SLIP, 802.12 (100VG-AnyLAN), X.25, ATM, LAP-B, LAP-D, PPP, 802.11x, 802.14x, 802.15x, 802.16x и другие</td></tr>
<tr><td>1. Физичес–кий</td><td colspan="4">Среда: Коаксиал, экранированная и неэкранированная витая пара, оптоволокно, радиоволны</td></tr>
</table>

Рис. 28. Структура стеков протоколов

То есть, первыми отличиями стеков друг от друга являются:

- состав протоколов стеков;
- их функциональное назначение.

Кроме того, очень большое значение имеют принципы именования сетевых объектов, реализованные в стеках протоколов и прежде всего на третьем (сетевом) уровне. Эти принципы определяют не только алгоритмы

работы протоколов сетевого уровня, но и возможности логической организации сетей в том или ином стеке, то есть, то, как выглядят (как строятся) крупные сетевые структуры. Очевидно, что в реальной экономике в корпоративной среде, а также в иных (неэкономических) крупных организациях возможности по построению крупных сетевых структур играют большую роль и определяют применимость стеков протоколов.

5.8.3. Именование узлов сети в стеке SMB

Физический адрес узла = MAC-адрес, например: 00:04:79:66:0A:46. Почти всегда он представляется в 16-ричном виде. Первые три байта идентифицируют фирму-производителя данного сетевого устройства, а последние три байта – порядковый номер данного вида устройств, выпущенных этой фирмой. Так получается, что каждое устройство имеет свой индивидуальный номер и не существует устройств, имеющих одинаковый MAC-адрес.

Этому физическому адресу в стеке SMB присваивается алфавитно-цифровое (символьное) имя, которое определяется на этапе установки Windows.

Физический адрес (MAC-адрес) предназначен для использования аппаратным и программным обеспечением (когда они формируют пакеты для отправки или обрабатывают принятые пакеты).

Символьное имя этого адреса предназначено для использования человеком.

В стеке SMB символьное имя «привязывается» к MAC-адресу, является его синонимом (алиасом). Это видно здесь:

«Пуск → Панель управления → Сетевые подключения → Подключение по локальной сети»

в новом окне «Подключение по локальной сети – Состояние» на вкладке «Поддержка» выбрать «Подробности» – увидим привязку IP-адреса к MAC-адресу на данном копьютере;

а в новом окне «Подключение по локальной сети – Свойства» на вкладке «Общие» выбрать «Протокол Интернета (TCP/IP)», нажать клавишу «Свойства» – увидим назначение свойств стека TCP/IP на данном компьютере.

А, вот, где имя компьютера привязано к MAC-адресу?

5.8.4. Именование узлов сети в стеке IPX/SPX

Адрес узла сети (или имя узла – в этом стеке адрес и имя не различаются, это одно и то же) в стеке IPX/SPX двухуровневое: на нижнем уровне используется физический адрес узла (MAC-адрес размером 6 байт, например: 00:50:22:B5:7E:93), на верхнем уровне (через точку) – номер сети размером в 4 байта. То есть, в качестве номера сети служит некоторое 16-ричное число, например, A311. Формально, оно должно выдаваться фирмой Novell по запросу системного администратора, а обычно оно выбирается системным администратором по некоторым ему известным критериям.

Физический адрес узла (MAC-адрес) в стеке IPX/SPX служит также именем компьютера.

Надстройкой над этой двухуровневой структурой имён служит NDS (NetWare Directory Service). В сервисе NDS свои правила именования (иерархические), но они распространяются не только на узлы сети, но и на другие объекты распределённой вычислительной системы (пользователей, элементы организационной структуры фирмы/учреждения, сервисы в системе и т. д.). Именование объектов в NDS ориентировано на человека и, как правило, отражает организационную структуру того социально-экономи–ческого объекта, в котором функционирует сеть NetWare. То есть, сервис NDS хотя и привязан к вычислительной сети (узлы сети в нём тоже отражены), но в целом является существенно более абстрактной системой, нежели вычислительная сеть, которая в этом сервисе оказывается очень глубоко скрытой социально-экономическими наслоениями.

Таким образом, в стеке протоколов IPX/SPX числовые адреса узлов сети (например, 00:50:22:B5:7E:93.A311) используются программами и оборудованием (и иногда системными администраторами – когда они знают об их существовании), а символьные имена NDS предназначены для использования человеком.

5.8.5. Именование узлов сети в стеке TCP/IP

В этом стеке протоколов используется двухуровневая система именования узлов сети, которая выглядит как трёхуровневая.

Физическим адресом узла является 6-байтовый MAC-адрес. Этому физическому адресу присваивается имя узла, которое состоит из двух составляющих – символьной (например, comp1.lab213.ulsu.ru) и числовой (например, 192.168.99.11). То есть, в этом стеке имя сетевого узла имеет две формы: символьную и цифровую.

Символьная составляющая, длиной до 255 байт, имеет иерархиче-

скую (многоуровневую) структуру и определяется доменной структурой Internet'a. Она состоит из имени хоста (hostname, например, comp1) и доменной части имени (например, lab213.ulsu.ru). Имя хоста присваивается, как правило, администратором хоста (например, при установке ОС или при настройке сети). Доменная часть имени определяется вышестоящими администраторами соответствующих доменов и регистрируется в InterNIC, международной организации, ведающей доменной структурой Internet'a.

Числовая часть имени, размером 4 байта, имеет двухуровневую структуру – <IP-адрес сети><IP-адрес хоста> и назначается администратором по некоторым правилам назначения IP-адресов.

В целом назначение имени узла в стеке протоколов TCP/IP определяется спецификацией DNS (Domain Name System), определённой стандартами RFC 1034 и 1035. DNS – распределённая база данных, поддерживающая иерархическую систему имён для идентификации узлов в сети для автоматического поиска IP-адреса по известному символьному имени узла сети Internet, предназначенная для автоматического поиска IP-адреса по известному символьному имени узла и обратно, символьного имени узла по известному IP-адресу.

Соответствие между физическим адресом узла сети и именем узла сети, точнее, числовой частью имени узла сети – IP-адресом, обеспечивается специальным сервисом локальной сети – ARP/RARP, работающим на границе канального и сетевого уровней, а также работающими поверх них сервисами BOOTP и DHCP. В каждом узле сети ведётся ARP-таблица соответствий известных данному узлу MAC-адресов соседних узлов и их IP-адресов.

Таким образом, в стеке протоколов TCP/IP числовые адреса узлов сети (MAC-адреса и IP-адреса) используются программами и оборудованием (и иногда системными администраторами – когда они знают об их существовании), а символьные имена предназначены для использования человеком – обычными пользователями.

5.8.6. Следствия из именования

Следствие 1: **Определения локальной сети в стеках протоколов.**

1. Стек SMB: Локальная сеть – это

а) все узлы, что доступны физически (по MAC-адресам).

Примечание 1: Для того, чтобы увидеть «чистую» сеть Windows, обеспечиваемую этим стеком протоколов, необходимо в сетевых настройках компьютеров удалить «протокол TCP/IP»; тогда в «сетевом окружении» увидим «чистую» локальную сеть Windows. Именно локальную сеть – никаких Интернетов, конечно, не будет.

2. Стек IPX/SPX: Локальная сеть – это

а) те узлы, что доступны физически (по MAC-адресам),

б) и только те из них, которые имеют одинаковый номер сети N.

3. Стек TCP/IP: Локальная сеть – это

а) те узлы, что доступны физически (по MAC-адресам),

б) и только те из них, которые имеют одинаковую доменную часть имени (входят в один и тот же поддомен),

в) и только те из них, которые имеют ip-адреса из одной сетки.

Следствие 2: **Понятие корпоративной сети в стеках протоколов.**

1. Стек SMB: Корпоративная сеть – это

- то же, что локальная.

То есть, понятие «корпоративная сеть» формально отсутствует.

2. Стек IPX/SPX: Корпоративная сеть – это

- объединение («прямая сумма») локальных сетей:

N (+) M (+) L (+) K (+) I (+) router's

3. Стек TCP/IP: Корпоративная сеть – это

- объединение («прямая сумма») локальных сетей:

N (+) M (+) L (+) K (+) I (+) router's

Примечание 1. В силу того, что в стеке SMB понятие «корпоративная сеть» отсутствует, но оно необходимо для пользователей, MS вынуждено надстроить над локальной сетью конструкцию «домен+Актив_Директори» для того, чтобы искусственно создать необходимую функциональность.

Примечание 2. Однако, интересно то, что этот надстроенный функционал в своей работе опирается на возможности стека протоколов TCP/IP, включаемого на Windows, и без него самостоятельно работать не может (!).

Вывод, очень интересный и важный: А мир-то реально держится на unix'ax! (!) Windows без unix самостоятельно шагу ступить не может!

6

УПРАВЛЕНИЕ СЕРВИСАМИ

6.1. Определения и содержания понятий

6.1.1. Определение сервиса

Определение. **Сервис –** это аппаратно-программный комплекс, предназначенный для организации публичного доступа к некоторому ресурсу (контенту). В обиходе, термин «организация публичного доступа» означает «расшаривание» этого ресурса. См. понятие «лоббизм» (wikipedia.org).

Определение. **Сервис** – это некоторая функциональность системы, доступная нескольким (всем) процессам/пользователям в многозадачной многопользовательской системе.

Определение. **Сервис локальный,** если он доступен только процессам в локальной системе, то есть, в пределах одной ОС. Обратите внимание: не в пределах одного компьютера, а именно в пределах одной ОС, то есть, виртуальная система – это уже другая ОС.

Определение. **Сервис сетевой,** если он доступен в сети, то есть, с других ЭВМ сети, и неважно, какая это сеть – локальная, или корпоративная, или вообще Интернет.

Следствие. Сервис определяется только для многозадачных систем. В однозадачных системах это понятие существовать не может (причина: последовательное выполнение процессов), поскольку понятие сервиса требует как минимум псевдопараллельности. Если более точно, то на однозадачной системе может создан только один сетевой сервис.

Замечание. Здесь определяются и ниже описываются сервисы, а не какие не службы! Служба – это совсем другое. На примерах: армейская служба; «ходить на службу» (чиновники ходят на службу, на работу); служба в церкви/храме – бывали в церкви? (сейчас это модно), вот, вы ходили на церковную службу и батюшка эту службу вёл, работал; батюшка в церковь ходит на службу (на работу), а вы ходите в церковь совсем за другим. И т. д. И вообще: «служить бы рад, прислуживаться тошно»(С) – здесь

как раз отделяется понятие «служить» (службу, выполнять работу) от понятия «прислуживаться» – предоставлять услугу-сервис, то есть, прислуживающийся предоставляет интерфейс хозяину, по которому хозяин слугу имеет. То есть, сервис – это предоставление услуги, а работа – это то, что потом выполняется демоном и тем, что за ним стоит.

А то, что MS везде использует термин «служба» – это от незнания русского языка.

6.1.2. Состав и структура сервиса

Сервис реализуется некоторой программой или комплексом программ, работающим в «автоматическом режиме» – без диалога с оператором/пользователем (почти всегда). То есть, сервис – это некоторая программа (процесс), запущенная на ЭВМ, и которая по запросу от других программ (процессов), действующих, возможно, по командам пользователей, выполняет некоторые действия, некоторую работу, которая либо недоступна процессам-заказчикам (если сетевой сервис), либо является общей и потому вынесена в отдельный общедоступный процесс (если локальный сервис).

Иначе говоря, сервис всегда работает в архитектуре «клиент-сервер».

На рисунке 29 показана структура сетевого сервиса.

Структура локального сервиса отличается только тем, что протокол сервиса реализуется не через сеть, а через локальные средства межпроцессного взаимодействия: локальный сокет (очень часто), именованный (тоже часто) или неименованный (редко) каналы. То есть, на рисунке вместо слов «Сеть локальная или глобальная» будет написано, например, «Механизм локальных сокетов». На рисунке обведены штриховой линией те элементы, которые составляют непосредственно сервис. Клиенты в состав сервиса не входят, это отдельные процессы, возможно расположенные у чёрта на куличках.

Что мы видим на рисунке:

а) сервис имеет модульную структуру; сложные сервисы – это комплексы программ, как правило, это сетевые сервисы; в простейшем виде сервис состоит (если сервис локальный) только из расшариваемого ресурса, демона и протокола; пример: web-сервис со статическим сайтом – apache + каталог с файлами готовых html-страничек + протокол http;

б) элементы сервиса достаточно независимы друг от друга, то есть связаны только протоколами и интерфейсами

в) реализация каждого сервиса – специфическая и определяется тем, что мы расшариваем и как мы это расшариваем;

Рис. 29. Структура сервиса

г) специфика интерфейсов:

- интерфейс аппаратуры определяется разработчиком аппаратуры,
- интерфейс драйвера – типовой (обычно) и определяется правилами программирования модулей операционной системы,
- интерфейс файловой системы – определяется разработчиками файловой системы, как правило, это те же люди, что разрабатывали ОС,
- интерфейс подключения внешних программ определяется разработчиками демона,
- протокол сервиса специфичен для каждого сервиса и определяется разработчиками демона;

д) и как уже сказано выше, сервис реализуется всегда в архитектуре «клиент-сервер»,

е) из специфичности протокола сервиса следует, что клиенты сервиса также оригинальны для каждого сервиса; однако, бывают исключения: примеры – браузер, putty,

ж) человек (пользователь) – только за клиентом сервиса, в самом сервисе его нет.

Перечисленные пункты а) – ж) точно и конструктивно определяют сущность сервиса.

6.2. Проектирование и программирование сервисов

Разработка сервиса делится на

- разработку демона,
- разработку протокола,
- разработку всего остального.

Прежде всего обо всём остальном:

- реализация клиента – он создаётся по правилам разработки сетевых программ, особой специфики нет;
- реализация «преобразования данных в распознаваемый клиентом вид» – скрипты (как в web-сервисе) или обычные программы командной строки («фильтры»), особой специфики нет;
- разработка драйвера осуществляется по правилам программирования драйверов в данной ОС, особой специфики нет.

Разработка протокола. В нём должно быть определено:

- формат пакетов, которыми будут обмениваться клиент и сервер, с описанием полей пакета;
- алгоритм обмена: как правило, обмен инициирует клиент и простейший алгоритм – «старт-стопный» (запрос – ответ; см. новую технологию создания сервисов REST); если же запросы предполагаются сложными (с уточняющими запросами) и многопакетными ответами, то есть, предполагается «диалог» между клиентом и сервером, то необходимо построить автомат алгоритма, чтобы показать, что отсутствуют возможности для перехода в тупиковые состояния (зависания);
- кодирование информации – это важно для сетевых сервисов, ведь клиенты могут работать в разных операционных системах со своими понятиями о кодировках информации;
- именование взаимодействующих объектов – каким образом клиент найдёт сервис и какая адресная информация должна присутствовать в запросах/ответах и в пакетах данных.

Разработка демона. Прежде всего, определение демона.

Определение. **Демон** – это существо, выполняющим задачи, за которые не хотят браться «боги», то есть, ещё не «бог», но около того, типа «ангела-хранителя».

Определение. **Демон** (daemon, dæmon, др.-греч. δαίμων божество) – компьютерная программа в системах класса UNIX, обычно запускаемая

самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем. Обычно демоны большую часть времени проводят в ожидании некоторого события. Когда это событие происходит, демон активизируется, выполняет свою работу и снова засыпает в ожидании события: как «старик Хоттабыч» – сидит в кувшине и ждёт, пока его потрут. Вывод: найденный кувшин обязательно надо потереть, добрый демон – это голубая мечта любого разгильдяя.

Здесь очень важно уточнение о том, что программа демон не имеет взаимодействия с пользователем (диалога с пользователем), говорят: «не имеет управляющего терминала».

Настройка демона осуществляется двумя способами:

- через ключи (опции), указываемые в строке запуска;

- с помощью конфигурационных файлов, имя которых может быть указано с помощью ключа в строке запуска, либо задаётся по умолчанию – в большинстве случаев.

Очень редко возможности по управлению сервисом включаются в протокол сервиса и тогда интерфейс клиента включает административные возможности.

Конфигурационные файлы демонов обычно помещаются в каталог /etc или в его подкаталоги.

Отсюда следуют первые два требования к этой программе:

а) демон – это программа командной строки и, следовательно, она должна уметь обрабатывать ключи запуска;

б) эта программа в начале своей работы должна уметь прочитать свой конфигурационный файл и настроиться на указанный в нём режим работы.

Кстати, ключи запуска, указываемые в командной строке, более приоритетны и должны переопределять установки из конфигурационного файла.

Следующее требование к разработке демона следуют из самой сущности этой программы:

г) она обрабатывает запросы и должна это делать как можно быстрее; иначе говоря, в ней должно быть выделено то ядро, что будет обрабатывать запросы, и эта часть программы должна быть написана как можно более эффективно с оптимизацией по скорости.

Парадигма демона представлена на рис. 30.

Как видно из рисунка, демон имеет в своём составе ряд действий, которых нет в обычных программах и которые, что бы их правильно сделать, требуют от программера существенного понимания как операционной си-

стемы, так и технологии программирования. Кроме того, центральной частью демона является «вечный цикл» do { . . . } while (1); , выход из которого осуществляется только по получении соответствующих особо ценных указаний.

Так же следует отметить, что программирование демонов несколько различается для разных ядер linux.

```
Начало
Определения типов, переменных, функций;
 . . .
  int main(int argc,char *argv[])
{
 . . .
Обработка конфигурационного файла и ключей;
 . . .
Проверка отсутствия себя;
Определение рабочего каталога и маски;
Переключение в режим демона;
  – Создание родственного процесса;
  – Запись pid-файла;
  – Переопределение стандартных файлов;
  – Если необходимо – переход в системного пользователя;
 . . .
Определение параметров протоколирования;
Определение порядка слежения за процессом демона;
Определение и настройка интерфейсов и протоколов;
. . .
do  {

    Приём запроса;
    Протоколирование запроса;
    Выполнение действий по запросу;
    Формирование и отправка ответа;
    Выполнение административных действий, если сигнал;

}  while (1);

Организация выхода, «приборка мусора» за собой;
return 0;
}
Конец
```

Рис. 30. Парадигма демона

6.3. Методы запуска сервисов

6.3.1. Разовый запуск – «вручную»

Обычно запуск демона «вручную» осуществляется в отладочных или учебных целях. И здесь нужно выделить два случая:

- демон написан правильно (см. п. 6.2.);

- демон ещё не демон или это вообще не демон, но нужно запустить как демон.

Разница этих случаев в том, что в первом случае программа написана правильно и сама всё, что нужно сделает (см. парадигму демона), а во втором случае мы ещё недописали демона или вообще просто хотим запустить некоторую программу как фоновый процесс.

Запуск в первом случае:

```
# progad  <ключи>  <Enter>
```

то есть, никаких особенностей, запускаем как обычную программу, но от root'a.

Запуск во втором случае:

```
$ proga  <ключи>  &  <Enter>
```

то есть, символом «&» специально указываем системе, что мы хотим получить «отсоединённый от терминала» фоновый процесс, потому как наша программа ещё не умеет, или вообще не умеет самостоятельно переходить в режим фонового процесса (демона). И в этом случае, как правило, запуск осуществляется от обычного пользователя. Однако, если мы запускаем в отладочных целях ещё недоделанный демон, то нужно запускать от root'a.

6.3.2. Схема BSD

В этой схеме для запуска демона сервиса и в целом всего сервиса должен существовать «скрипт запуска» сервиса. В этом скрипте выполняется некоторые предварительные действия:

- назначаются нужные переменные среды и проверяются их значения;

- проверяется наличие нужных программ из состава сервиса и пути к ним;

- если необходимо, создаётся структура рабочего каталога сервиса и кладутся туда нужные файлы;

- определяются названия некоторых служебных файлов сервиса и пути к ним;

- определяется название сервиса (как он будет обзываться в системе), оно часто не совпадает с именем демона;

- определяются «точки входа» в скрипт: start/restart/stop/status/reload и др.;

- формируются ключи командной строки для запуска демона;

- если сервис – комплекс программ, то запускаются нужные в определённом порядке;

- проверяются, запущены ли сервисы, от которых зависит данный;

- и т. д., то есть, выполняются те работы, которые нежелательно в целях обеспечения нужной масштабируемости, переносимости и гибкости настройки выполнять непосредственно в демоне.

Этот скрипт помещается в каталог /etc/rc.d/, ему назначается хозяин root и назначается режим доступа 755, то есть, ставятся биты исполнения. Далее, или непосредственно в основном конфигурационном файле rc, или в rc.local ставится вызов этого скрипта. Если система настроена так, что при запуске в конце конфигурирования она просматривает каталог rc.d на предмет поиска исполняемых скриптов и запуска их, то тогда вставлять вызов скрипта в файлы rc нет необходимости, достаточно просто сделать скрипт исполняемым.

Сложность в этой схеме в том, что администратор должен самостоятельно отслеживать правильный порядок запуска сервисов в системе: либо давая соответствующие имена скриптам, чтобы выстроить их в нужном порядке в каталоге /etc/rc.d/ (отсортировать), либо вставить вызов скрипта в файлы rc в нужное место. То есть, к администратору предъявляются повышенные требования – он должен понимать, что делает.

6.3.3. Схема SystemV

В этой схеме также для запуска сервисов используется аналогичные стартовые скрипты, но они помещаются в каталог /etc/rc.d/init.d/ – «свалка» скриптов. Дополнительно в системах, настраиваемых по схеме systenV, создаются «уровневые» каталоги, которые располагаются также в /etc/rc.d/ и называются rc0.d, rc1.d, rc2.d. rc3.d, rc4.d, rc5.d, rc6.d и которые определяют то, как должна быть настроена система при выходе на нужный уровень работы. Сам уровень работы определяется в файле /etc/inittab – называется «The default runlevel»:

0 – выключение системы,

1 – однозадачный однопользовательский режим, обычно для исправления ошибок,

2 – многозадачный многопользовательский, но без поддержки сети (не всегда),

3 – нормальный полнофункциональный режим без графики – серверный,

4 – то же, что третий, считается «промежуточным», обычно не используется,

5 – нормальный полнофункциональный режим с графикой, «десктопный»,

6 – перезагрузка,

В «уровневые» каталоги помещаются ссылки («мягкие ссылки») на скрипты в /rc.d/init.d/, причём ссылки специальным образом именуются:

- первый символ K (kill) или S (start),

- следующие два символа – число в диапазоне [0..99] (порядковый номер), чтобы упорядочить запуск скриптов в нужной последовательности,

- имя скрипта.

При запуске система на последнем этапе конфигурирования заходит в каталог указанного уровня и запускает скрипты по порядку. Поскольку сначала по алфавиту идёт буква K, а потом S, то в этом порядке ссылки и выстраиваются: сначала всё убивается, а потом всё, что нужно для нормальной работы на данном уровне, снова запускается «вчистую». Получается, например, так:

. . .

K90network – 90-стая по порядку операция это «убиение сети»,

K92iptables – выключение fw,

K92messagebus – выключение «шины сообщений»,

S01sysstat – запись в протокол метки «LINUX RESTART»,

S02udevd – запуск демона udevd для формирования каталога /dev,

S05bridge – запуск моста между сетевыми интерфейсами,

. . .

S91nmd – запуск демона именования NetBIOS поверх именования IP,

S95alteratord – запуск демона для Центра управления системой

S99local – выдача приветствия системы «Welcome to hostname»: всё готово, входите.

Для облегчения работы администратора в начале стартовых скриптов помещается строчка

chkconfig: <список_уровней> <порядковый_номер S>
<порядковый_номер K>,

где в «списке уровней» перечисляются те номера «уровневых» каталогов, в которых ссылка для запуска сервиса должна быть сформирована, а поряд-

ковые номера – это те самые числа в диапазоне [0..99], определяющие порядковые номера скриптов, соответственно для старта сервиса и убиения сервиса.

Тем самым, администратору даётся подсказка, в каком порядке запускаются сервисы и в результате существенно снижаются требования к его квалификации – ему уже не надо вникать в суть сервисов и определять взаимозависимости, достаточно открыть скрипт в редакторе / просмотрщике, прочитать эту строчку и создать в «уровневых» каталогах ссылки на стартовый скрипт с указанными номерами. Более того, в некоторых особо «умных» пакетах нужные ссылки создаются при инсталляции пакета.

6.3.4. Схема с суперсервером

Суперсервер – это сервис запуска сервисов по требованию (on demand). Он работает следующим образом:

- запускается с помощью стартового скрипта по схеме BSD или SystemV в зависимости от системы;

- считывает свой конфигурационный файл /etc/xinetd.conf, настраивается и переходит в режим демона, как описано выше;

- из конфигурационного каталога /etc/xinetd.d/ считывает все файлы, которые там есть; отбирает те, в которых значение переменной disable установлено в no, остальные файлы игнорирует; из выбранных файлов запоминаются значения переменных:

~ type – какой сервис – внутренний (обрабатывает сам xinetd) или внешний,

~ user – пользователь, от имени которого будет запущен сервис,

~ id – имя сервиса,

~ socket_type – тип сокета STREAM, DGRAM или RAW,

~ protocol – протокол, по которому работает данный сервис,

~ port – порт, который использует данный сервис,

~ server – абсолютный путь к исполняемому файлу демона сервиса и, возможно, другие параметры (см. man xinetd);

- всё запомнив, суперсервер слушает указанные порты и ждёт: когда в порт приходит пакет указанного протокола, то суперсервер запускает демон нужного сервиса, дальше обработку делает сам демон сервиса, который завершив обработку и отправив ответ (или завершив сеанс с клиентом), завершается.

- и т. д.

Смысл использования суперсервера в том, что он позволяет не дер-

жать в памяти редко используемые сервисы и, тем самым, в некоторой степени разгрузить систему. Учитывая, что некоторые демоны создают дополнительные родственные процессы и/или потоки для взаимодействия с клиентом, то «разгрузка» системы может быть существенной.

Для того, чтобы для запуска какого-либо сервиса задействовать суперсервер, необходимо создать в каталоге /etc/xinetd.d/ текстовый файл (лучше с именем этого сервиса) определённого формата (см. man xinetd.conf – есть примеры), в котором указать значения переменных для этого сервиса и перезапустить суперсервер.

6.3.5. Схема systemd

Systemd – менеджер системы и сервисов (см. man systemd), обратно совместимый с ранее описанными схемами и предоставляющий такие полезные функции, как параллельный запуск системных сервисов во время загрузки, активацию демонов по требованию, поддержку срезов (снепшотов) состояния системы и логику управления сервисами, основанную на зависимостях. Иначе говоря, решает в одном месте все те вопросы, что ранее реализовывались разными схемами.

Схема systemd появилась и получила распространение уже в нашем веке в связи с появлением в массовых количествах многоядерных процессоров. В старых системах процесс как установки системы (копирование программ, обнаружение устройств, конфигурирование), так и просто очередной загрузки, шёл линейно, то есть, многоядерность, даже если она была, незадействовалась. С появлением многоядерных процессоров естественно возникло желание этот процесс ускорить за счёт распараллеливания работы.

Организует распараллеливание демон systemd, который, запускается как самый первый процесс (pid = 1) и действует как менеджер сервисов. Конфигурационные файлы демона находятся в каталоге /etc/systemd/, а стартовые скрипты запуска сервисов (которые здесь называются юниты – unit) располагаются в каталоге /lib/systemd/. Причём, стартовые скрипты могут организовыватся в группы («target»).

В скриптах указывается взаимозависимость сервисов («Requisite» – какие сервисы нужны данному, «After» – после чего грузить, «Before» – что может грузиться после, «Conflicts» – с какими сервисами конфликтует за ресурсы, то есть, с какими не параллелить).

Соответственно, демон systemd, на основе этой информации, реализует непротиворечивое дерево запуска. Также в скриптах указываются протоколы, по которым работает сервис, сокеты и порты, которые слушает..

Это позволяет демону systemd организовать запуск сервисов по требованию, аналогично сервису xinetd.

Пока эта схема запуска сервисов не является полной альтернативой ранее описанным схемам, то есть, она совместима с ними, работает прежде всего на ранних этапах загрузки системы и определяет состав и порядок запуска «системных» сервисов, то есть, обеспечивающих функционирование системы в целом. В конце процесса загрузки (при запуске «пользовательских» сервисов) всё равно используются вышеописанные схемы, в рамках которых пользователь может организовать запуск нужных уже ему сервисов. Тем не менее, тенденция такова, что схема systemd станет основной.

Для того, чтобы организовать запуск сервиса по схеме systemd необходимо:

а) создать файл запуска сервиса определённого формата (см. man systemd), например, мы разработали свой несложный сервис с демоном abrtd. Тогда создаём примерно такой файл:

```
[Unit]
Description=Daemon to detect crashing apps
After=syslog.target

[Service]
ExecStart=/usr/sbin/abrtd
Type=forking

[Install]
WantedBy=multi-user.target
```

В файле мы видим следующее:

- в секции [Unit] – описание и указание на то, что наш сервис может быть запущен только после того, как будет запущен сервис syslog;

- в секции [Service] – указываем путь к исполняемому файлу нашего сервиса и сообщаем демону systemd, как он узнает, что наш сервис успешно загрузился;

- в секции [Install] – сообщаем демону systemd, что наш сервис надо запускать после того, как будет выполнено все, что определено в цели multi-user.target.

б) созданный файл обзываем так: имя_сервиса.service (то есть, у нас это будет: abrtd.service) и копируем его в каталог /etc/systemd/system/.

в) Затем даём команду на перезапуск демона systemd:

 # systemctl daemon-reload

По этой команде демон systemd перечитает свои конфигурационные файлы, в том числе и нами созданный.

г) После этого можно запустить наш сервис командой

 # systemctl start abrtd.service

Останов сервиса:

 # systemctl stop abrtd.service

Перезапуск сервиса:

 # systemctl restart abrtd.service

Перезагрузка сервиса (сервис перечитывает свои конфигурационные файлы, не прерывая работы):

 # systemctl reload abrtd.service

Плюсом этой схемы являются большая гибкость и большие функциональные возможности – systemd умеет не только сервисы запускать. Но есть и очень существенный минус: в настоящем виде эта схема намного сложнее в использовании любой из предыдущих и, соответственно, выше требования к квалификации администратора. То есть вернулись обратно почти на уровень схемы BSD, когда надо было знать правильный порядок загрузки сервисов. Вероятно, это изменится.

6.3.6. Применимость схем запуска

Схема BSD использовалась в старых версиях FreeBSD – отсюда и её название, а также используется в Linux в дистрибутивах slackware. Однако, в последних версиях наметился отказ от этой схемы, в силу сложности конфигурирования, и переход на схему systemV.

Схема systemV используется в дистрибутивах unix AIX, IRIX, HP-UX, SCO, UnixWare и других, а также в дистрибутивах Linux redhat/fedora, suse, altlinux, debian и других.

Схема с суперсервером используется во всех дистрибутивах.

Схема systemd начала использоваться в последние годы в целях ускорения процесса загрузки и конфигурирования посредством распараллеливания работы. Стимулом для применения этой схемы стало появление многоядерных процессоров.

В целом схемы BSD и systemV используются для запуска высоконагруженных сервисов, когда запросов много, а в тех случаях, когда запросов мало, лучше использовать схему с суперсервером, которая позволяет экономить ресурсы системы.

6.4. Другие вопросы управления сервисами

6.4.1. Почтовый сервис – описание

Это один из самых «капризных» сервисов с очень высокими требованиями к правильной настройке сети. Для его работы необходимо, чтобы стек протоколов TCP/IP был сконфигурирован правильно, и прежде всего, правильно работало разрешение имён – резолвер и DNS: преобразование из символьной формы имени в цифровую и обратно (см. п. 5.8.).

Структура почтового сервиса показана на рисунке 31. Он состоит из следующих элементов:

- MUA (Mail User Agent) – почтовый клиент, например, Thunderberd;
- MTA (Mail Transport Agent) – почтовый транспортный агент, например, Postfix;
- MDA (Mail Delivery Agent) – почтовый агент доставки, например, procmail;

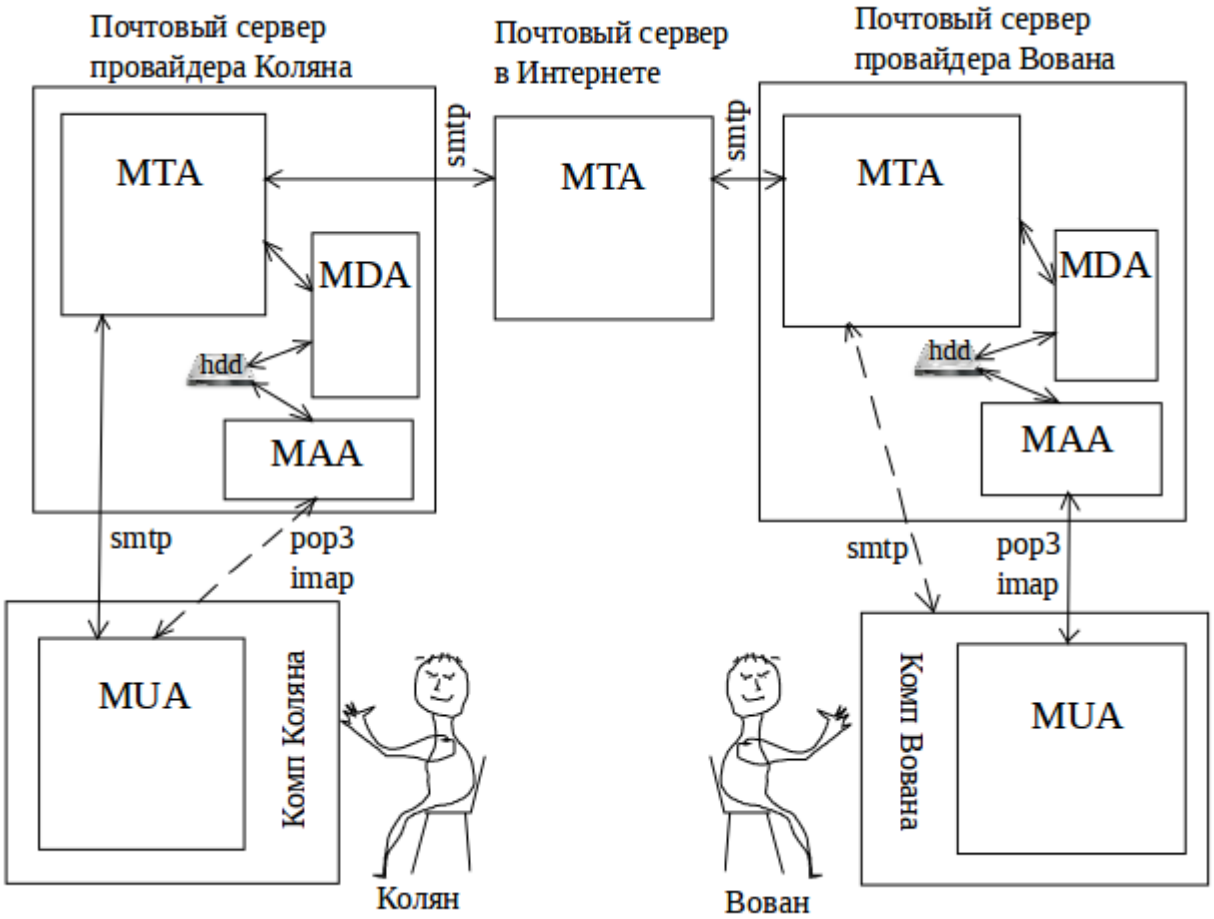


Рис. 31. Колян написал письмо Вовану

- MAA (Mail Access Agent) – агент доступа к почте, например, popper3d, или Dovecot.

Протоколы:

- SMTP (Simple Mail Transfer Protocol) – простой протокол передачи почты;

- POP3 (Post Office Protocol) – почтовый протокол версии 3;

- IMAP (Internet Message Access Protocol) – интернет протокол доступа к сообщениям.

Алгоритм обмена почтовым сообщением: Колян, сидя за своим компьютером, сочинил письмо своему другану Вовану, используя почтовый клиент (MUA) Thunderberd. Пусть почтовый адрес Вована – vovan@mv.ru. Когда Колян нажимает кнопку «Отправить», его Thunderberd формирует письмо (заголовок («конверт») и текст), соединяется с MTA почтового сервера провайдера Коляна по протоколу SMTP (порт 25) и передаёт ему письмо. MTA, получив письмо, обращается к сервису DNS, на предмет поиска записи MX (Mail Exchanger) для домена mv.ru, в котором, наверное, находится почтовый сервер адресата. Если такая запись найдена, то MTA Коляна связывается с MTA найденного почтового сервера, спрашивает, есть ли такой адресат и если есть, то передаёт ему письмо. Если Колян ошибся адресом, то MTA Коляна сформирует письмо Коляну об ошибке, вкладывает в него письмо Коляна и отправит его Коляну, то есть, передаст своему MDA, который положит его в почтовый ящик Коляна (на hdd). Если же всё правильно, то MTA Вована передаст полученное письмо своему MDA, который положит его в почтовый ящик Вована (на hdd). Когда Вован надумает сесть за компьютер и почитать письма, он даст указание своему MUA «получить письма». Тогда MUA Вована соединится с почтовым сервером провайдера, конкретно, с MAA (например, dovecot, по протоколу POP3, порт 110, или по протоколу IMAP, порт 143), который отдаст ему письмо.

На рисунке пунктирами показаны взаимодействия:

- слева: Колян может получить письмо с сообщением об ошибке, если оно будет сформировано MTA;

- справа: Вован может послать ответ Коляну по вышеприведённому алгоритму.

6.4.2. Почтовый сервис – установка и настройка

Рассмотрим установку почтового сервиса на компьютере с установленной системой Альт Линукс 7.0.5 Кентавр. В качестве клиента может быть использован любой другой компьютер (даже Wind'овый) с установленным почтовым клиентом Thunderberd.

Порядок настройки:

1. Компьютеры должны быть объединены в сеть физически, то есть, в разъёмы RJ-45 (сетевые) должны быть вставлены кабели, которые должны подключаться к коммутатору.

2. Должна быть настроена сеть, как минимум так, как описано в приложении к заданию на лабораторную «Настройка сети без DNS» – это будет локальная сеть. Но вы можете аналогичным образом настроить и корпоративную сеть. Обязательно должна быть выполнена проверка правильности настройки сети, как описано в лабораторной: зайти на другой компьютер по краткому символическому имени другого компьютера и вернуться обратно в свой компьютер по краткому имени, а затем тоже самое повторить с полным именем. Хождение с компа на комп по IP-адресу не считается – на такой сети почтовый сервис работать не будет. Пусть мы настроили сеть так, как описано в приложении к лабораторной «Настройка сети без DNS».

3. В качестве MTA будем использовать Postfix. В Кентавре 7.0.5. он уже установлен по умолчанию, но настроен только на работу с интерфейсом lo0, то есть, для передачи писем только между локальными пользователями. Postfix использует для своей работы системных пользователей mail, mailman, postfix, postman и группы mail, mailman, postfix, postman, postdrop. В altlinux все нужные пользователи и группы создаются автоматически при установке пакетов. Также создаются стартовые скрипты запуска почтовой системы в /etc/rc.d/init.d/ и даже в уровневых каталогах присутствует правильная ссылка на запуск postfix и он реально запускается, но . . только для локальной работы.

4. В качестве MDA в altlinux используется procmail, он также уже установлен (вместе с Postfix) и настроен для раскладки приходящих писем по почтовым ящикам всех зарегистрированных в системе пользователей – локальных пользователей.

5. Все конфигурационные файлы Postfix лежат в каталоге /etc/postfix/. Для перенастройки Postfix для работы в сети необходимо поправить его основной конфигурационный файл main.cf следующим образом:

#/etc/postfix/main.cf

mailbox_command = /usr/bin/procmail – $DOMAIN -d $LOGNAME

inet_protocols = ipv4

inet_interfaces = all

myhostname = имя_компа.домен (например, mail.firma.ru – почтовый сервер фирмы)

mydomain = домен (например, firma.ru)

mynetworks = 192.168.0.0/16, 127.0.0.0/8 (все сетки в диапазоне 192.168 и локальная петля)

myorigin = $myhostname (имя сервера, используемое в конверте)

mydestination = $myhostname, $mydomain, localhost.localdomain, localhost.$mydomain, \

mail.$mydomain (адресаты – локальные пользователи сервера и \ пользователи нашей сети)

smtp_host_lookup = hosts (сеть без DNS, поэтому говорим, что разрешение имён – в hosts)

Примечание. Всё, что в круглых скобках, в конфигурационный файл вставлять не надо.

В переменной mynetworks перечисляются обслуживаемые сетки ip-адресов. Если почтовый сервер будет обслуживать только локальную сеть (например, 192.168.99.0), то её и указываем:

mynetworks = 192.168.99.0/24, 127.0.0.0/8

Поскольку у нас сеть без DNS, то переменной smtp_host_lookup назначаем значение hosts, чтобы Postfix не лез за разрешением имён к DNS (по умолчанию значение этой переменной – dns).

После исправлений проверяем правильность настроек:

postfix check

и перезапускаем сервис:

systemctl reload postfix

Тестируем – отправляем письма пользователям с помощью команды mail:

[user@mail ~]$ mail vasja

Subject: Proba

PROBA-точно_проба

Cc: user

Ctrl/D

[user@mail ~]$

Здесь наш компьютер – mail.firma.ru, наш логин – user. Пишем пись-

мо пользователю vasja. В Cc: указываем, что хотим получить себе копию. Завершение письма – Ctrl-D с новой строки.

После отправки писем смотрим, что появилось в почтовых ящиках пользователей в каталоге /var/spool/mail/. Также нелишне заглянуть в протокол /var/log/maillog.

6. **Далее устанавливаем и настраиваем получение почты по протоколам POP3/IMAP**. Для этого с помощью synaptic ставим пакет dovecot, который по умолчанию не устанавливается. После установке в каталоге /etc/rc.d/init.d/ появится стартовый скрипт запуска данного сервиса. Создаём ссылки в каталоге /etc/rc.d/rc5.d/ для убиения и старта сервиса:

```
# ln -s ../init.d/dovecot K54dovecot
# ln -s ../init.d/dovecot S54dovecot
```

Исправляем конфигурационный файл /etc/dovecot/dovecot.conf

```
protocols = imap pop3 lmtp
listen = *   (слушаем только интерфейсы ipv4)
verbose_proctitle = yes   (увеличиваем количество информации о пользователях)
```

Запускаем сервис и проверяем с помощью команды ps:

```
[root@mail etc]# service dovecot start
[root@mail etc]# ps -ax | grep dovecot
2778 ?        Ss     0:00 /usr/sbin/dovecot -F
2781 ?        S      0:00 dovecot/anvil [0 connections]
2782 ?        S      0:00 dovecot/log
2783 ?        S      0:00 dovecot/ssl-params
2784 ?        S      0:00 dovecot/config
2785 ?        RN     0:02 dovecot/ssl-params
2787 pts/0    S+     0:00 grep --color=auto dovecot
[root@mail etc]#
```

7. **Далее настраиваем клиенты для отправки/получения почты**. Как уже было сказано, на компьютерах локальной/корпоративной сети у нас предполагается использовать MUA Thunderberd. Соответственно, в настройках Thunderberd указываем в качестве имени почтового ящика свои логин в формате login@hostname_почтового_сервера и пароль, с которыми мы зарегистрированы на почтовом сервере, то есть, в имени почтового ящика мы указываем свой логин на почтовом сервере и его (почтового сервера) полное имя в локальной/корпоративной сети. Нажимаем OK. При

этом Thunderbird проверяет правильность пары логин-пароль на указанном компьютере сети – почтовом сервере и если на этом компьютере есть такой пользователь с таким паролем и этот компьютер является почтовым сервером, то Thunderbird сообщает: «Конфигурация найдена у провайдера электронной почты» – см. рис. 32. Нажимаем «Готово». Если вам есть письма, то Thunderbird их вам скачает.

Если у вас несколько почтовых ящиков, то далее их можно добавить.

Теперь каждый раз при запуске Thunderbird, он будет автоматически проверять наличие писем вам на почтовом сервере, а также каждый раз при отправке вами письма. То есть, при каждом подключении в почтовому серверу Thunderbird автоматически выполняет полный цикл работ по приёму/отправке писем.

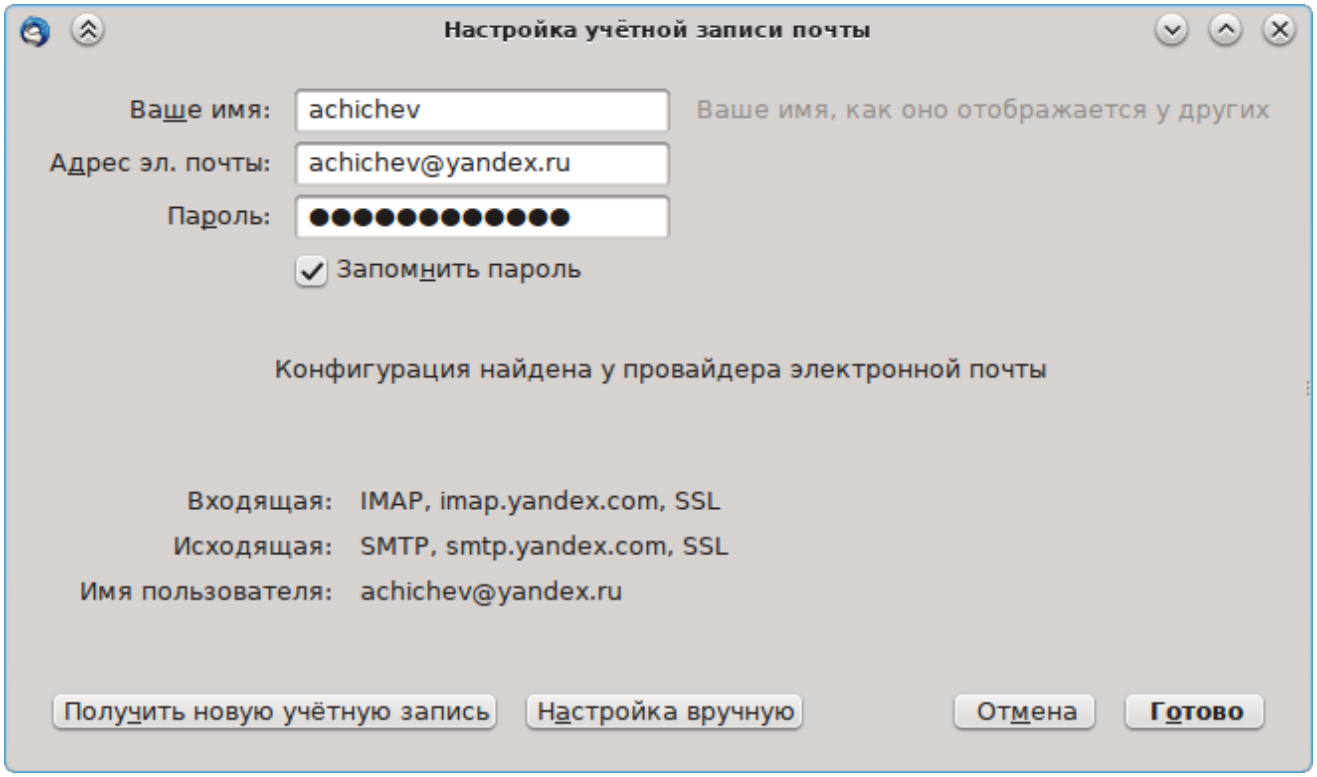


Рис. 32. Настройка Thunderbird

6.4.3. Сервис ftp

В качестве примера рассмотрим установку и настройку сервера wsftp. Снова установка пакета wsftp с помощью synaptic из репозитария altlinux. Напоминаю: если хотите всё установить легко и быстро, то не следует пользоваться чужими пакетами, всегда надо предпочитать пакеты из родного дистрибутива.

Сервис ftp в отличие от многих других «узкопрофессиональных» сер-

висов является многофункциональным. Он реализует следующие фуекции:

- доступ «по чтению» к некоторому расшариваемому каталогу (обычно pub) с файлами;

- доступ в некоторый каталог (по умолчанию – incoming) с возможностью записи в него своих файлов для обмена с другими пользователями сервиса или просто для хранения;

- если пользователь зарегистрированный в системе, то может предоставляться доступ (чтение и запись) в домашний каталог пользователя;

- возможность переброса файлов с одного ftp-сервера напрямую на другой ftp-сервер.

Доступность функций определяется настройками сервиса и они могут предоставляться в различных сочетаниях независимо друг от друга.

Конфигурационный файл wsftp по умолчанию находится в файле /etc/vsftpd.conf

Пример настройки сервиса ftp с доступом только для зарегистрированных пользователей в каталоги pub – чтение и incoming – чтение и запись. В файле vsftpd.conf следует изменить следующие строчки:

```
listen=YES
anonymous_enable=NO
local_enable=YES
write_enable=YES
chroot_local_user=YES
```

Остальные настройки – по умолчанию.

Рабочий каталог сервиса ftp обычно создаётся в /var/ftp. В этом каталоге нужно создать два подкаталога pub и incoming. Права на каталог pub – 755, а на каталог incoming – 777. На каталог incoming необходимо также установить sticky-бит.

При установке пакета wsftp создаётся системный пользователь ftp. Этот пользователь должен быть назначен хозяином каталога /var/ftp и всех его подкаталогов.

Если мы хотим, чтобы ftp-сервисом пользовались не только зарегистрированные пользователи, но и некоторые другие, кого мы известим, но не anonymous!, тогда создаём вспомогательного обычного пользователя, например, ftpuser с домашним каталогом /var/ftp и определяем ему пароль.

О логине-пароле извещаем определённую группу пользователей.

7

УПРАВЛЕНИЕ БАЗАМИ ДАННЫХ

7.1. Задачи администрирования БД

В данном разделе приведены основные вопросы администрирования БД. Более тонкие вопросы можно посмотреть в [19].

Общее администрирование

Общее администрирование рассматривается на примере СУБД MariaDB (прежняя версия называлась MySQL) и включает в себя в основном работу с демоном mysqld, сервером MariaDB и обеспечение доступа пользователей [21].

К наиболее важным задачам общего администрирования относятся следующие:

- запуск и остановка сервера;
- поддержка учетных записей пользователей;
- поддержка регистрационных файлов;
- резервирование и копирование баз данных;
- настройка сервера;
- установка нескольких серверов;
- обновление.

Безопасность

В процессе инсталляции СУБД важно также обеспечить надежную защиту хранимых данных. Администратор СУБД полностью отвечает за предоставление доступа к каталогам данных и серверу, и должен разбираться в следующих вопросах:

- защита файловой системы;
- защита сервера;
- отладка и поддержка баз данных;
- восстановление после сбоя;
- превентивная поддержка.

7.2. Запуск и остановка СУБД

Существуют три основных метода запуска сервера MariaDB:

- непосредственный вызов mysqld;
- вызов сценария safemysqld;
- вызов сценария mysql.server.

Непосредственный вызов mysqld выглядит следующим образом:

/etc/rc.d/init.d/mysqld start

Сценарий safemysqld располагается в подкаталоге bin каталога инсталляции СУБД. Его же можно найти в каталоге scripts дистрибутива MySQL.

Сценарий mysql.server можно отыскать в подкаталоге share/mysql каталога инсталляции или каталоге support-files исходной дистрибуции MySQL. Для использования эти сценарии необходимо скопировать в соответствующие каталоги запуска.

Для самостоятельного завершения работы сервера применяется команда mysqladmin:

mysqladmin shutdown

7.3. Защита новой инсталляции СУБД

На этапе инсталляции MySQL необходимо обязательно установить пароль для MySQL-пользователя root:

mysqladmin -u root password "пароль"

Перезагрузки таблиц разрешений осуществляется командой

mysqladmin -u root reload

7.4. Управление пользовательскими учетными записями

Оператор GRANT создает пользователей MySQL и позволяет настроить их привилегии. Оператор REVOKE удаляет привилегии. Операторы grant и revoke работают с четырьмя следующими таблицами.

Таблица разрешений	Содержимое
user	Подключающиеся к серверу пользователи и все их глобальные привилегии
db	Привилегии уровня базы данных
tables_priv	Привилегии уровня таблицы
columns_priv	Привилегии уровня столбца

Существует еще одна, пятая таблица разрешений (host), однако операторы grant и revoke не в состоянии ее обрабатывать.

Если оператор GRANT запускается для определенного пользователя, в таблице user для него создается новая запись. Если оператор определяет для пользователя какие-либо глобальные привилегии (привилегии администратора или привилегии, применяемые сразу ко всем базам данных), они также записываются в таблицу user. Права обработки базы данных, таблицы или столбца записываются соответственно в таблицы db, tables_priv и column_priv.

Оператор GRANT имеет следующий синтаксис:

GRANT привилегии (список_атрибутов)
ON список_баз данных(таблиц)
TO логин@адрес_машины_пользователя
IDENTIFIED BY "пароль"
WITH GRANT OPTION

Ниже в таблице указаны возможные привилегии.

Спецификатор привилегий	**Разрешенная операция**
user	Подключающиеся к серверу пользователи и все их глобальные привилегии
alter	Изменение таблиц и индексов
create	Создание баз данных и таблиц
delete	Удаление существующих записей из таблиц
drop	Удаление баз данных и таблиц
index	Создание и удаление индексов
insert	Вставка новых записей в таблицы
references	Не используется
select	Извлечение существующих записей из таблиц
update	Изменение существующих записей таблиц
file	Чтение и запись файлов сервера
process	Просмотр информации о внутренних потоках сервера и их удаление
reload	Перезагрузка таблиц разрешений или обновление журналов, кэша компьютера или кэша таблицы
shutdown	Завершение работы сервера
all	Все операции. Аналог – all privileges
usage	Полное отсутствие привилегий

Адрес машины пользователя может задаваться конкретным именем, IP-адресом, маской.

Для отмены привилегий пользователя применяется оператор revoke. Его синтаксис:

REVOKE привилегии (атрибуты)
ON база_данных(таблицы)
FROM логин

Ниже даны ряд полезных команд работы с учетными записями пользователей.

Принудительная перезагрузка таблиц разрешений выполняется так:
FLUSH PRIVILEGES

Посмотреть присвоенные привилегии
SHOW GRANTS FOR логин@адрес_машины_пользователя

Лишить всех прав пользователя
REVOKE ALL ON база_данных(таблица)
FROM логин@адрес_машины_пользователя

Изменить пароль пользователя
UPDATE USER SET PASSWORD=PASSWORD('пароль')
WHERE User = 'логин'

Удалить пользователя:
DROP USER логин@адрес_машины_пользователя

7.5. Подключение к серверу

Консольное подключение к серверу осуществляется с помощью утилиты mysql:

mysql -u логин -p

Далее вводится пароль пользователя и можно приступить к работе.

В некоторых случаях возникает необходимость в самостоятельной перезагрузке сервера из-за невозможности подключения к нему. Подключение к компьютеру localhost обычно осуществляется через сокет ОС UNIX, которым, как правило, является /tmp/mysql.sock. На linux этот сокет обычно находится в каталоге /var/lib/mysql. Удаление этого файла делает невозможным подключение клиентов. Такая ситуация, в свою очередь, может возникнуть после запуска процесса cron, который удаляет временные файлы из каталога /tmp. В этом случае наберите команду:

mysqladmin -p -u root -h имя_сервера shutdown

Если сокет удален в результате работы задания программы остановки, проблема может возникнуть снова. Чтобы избежать этого, настройте программу остановки на использование другого файла сокета. Это можно осуществить с помощью глобального конфигурационного файла. Так, например, если /usr/local/var – каталог данных, для перемещения в него файла сокета достаточно добавить следующие строки в файл /etc/my.cnf:

[mysqld]
socket=/usr/local/var/mysql.sock

[client]
socket=/usr/local/var/mysql.sock

В случае когда администратор не может подключиться к серверу из-за того, что забыл пароль пользователя root или в процессе изменения случайно присвоил ему не то значение, которое предполагал, необходимо восстановить контроль над сервером, чтобы заново установить пароль. Это можно сделать следующим образом:

- завершите работу сервера; авторизовавшись как пользователь root на компьютере с сервером, администратор может завершить работу сервера с помощью команды kill. Используя команду ps, можно отыскать PID процесса сервера. С этой же целью можно просмотреть PlD-файл, который обычно располагается в каталоге /run или /var/run; перед следующим запуском сервера необходимо проверить таблицы с помощью команд myisamchk и isamchk;

- перезапустите сервер с помощью параметра --skip-grant-tables. Это укажет серверу не использовать таблицы разрешений для проверки соединений и позволит подключиться с полномочиями пользователя root без пароля; После удачного подключения измените пароль пользователя root;

- используя команду mysqladmin flush-privileges, укажите серверу снова перезагрузиться, но с применением таблиц разрешений. Если используемая версия mysqladmin не поддерживает опцию flush-privileges, попробуйте воспользоваться командой reload.

7.6. Проверка и восстановление таблиц

Существуют два способа проверки и восстановления таблиц. Первый – с помощью специальных инструкций, второй – с помощью утилиты myisamchk. Соответствующие инструкции называются CHECK TABLE, REPAIR TABLE и OPTIMIZE TABLE. Они достаточно удобны, поскольку

выполняются в рамках серверного процесса. В этом смысле они ничем не отличаются, к примеру, от инструкции SELECT. Утилита myisamchk обладает рядом дополнительных возможностей, которые в ряде ситуаций оказываются весьма удобными. Например,

CHECK TABLE courses
REPAIR TABLE courses
OPTIMIZE TABLE

Таблицы базы данных снабжены флагом, указывающим, изменилось ли содержимое таблицы с момента последней проверки. Инструкция CHECK TABLE пропустит неизмененные таблицы при наличии ключевого слова CHANGED. В утилите myisamchk соответствующий режим включается с помощью опции --check-only-changed. Особым образом помечаются также неправильно закрытые таблицы. Чтобы проверить только их, укажите флаг FAST (инструкция CHECK TABLE) или опцию --fast (утилита myisamchk).

По умолчанию утилита myisamchk ищет повреждения только в индексных файлах. В инструкции CHECK TABLE этот режим включается с помощью флага QUICK. Сама инструкция CHECK TABLE по умолчанию проверяет не только индексы, но и неправильные ссылки на удаленные записи. В утилите myisamchk этот режим включается с помощью опции --medium-check. Расширенный режим проверки задается флагом EXTENDED и опцией --extended-check. В этом случае будут проверяться все индексируемые значения.

Инструкция REPAIR TABLE устраняет повреждения в таблице. То же самое делает утилита myisamchk, при наличии опции —recover.

Инструкция OPTIMIZE TABLE удаляет из таблицы пустые участки и осуществляет пересортировку записей. Аналогичные действия выполняет утилита myisamchk при наличии опции --analyze. Инструкция OPTIMIZE TABLE также сортирует индексы (соответствующая опция утилиты myisamchk называется --sort-index).

7.7. Резервное копирование и восстановление

В СУБД существуют три основных способа архивирования данных. Первый – это копирование табличных файлов, второй – создание SQL-образов таблиц, третий – создание форматированных текстовых файлов. Первый способ является самым экономным и быстродействующим. Но для таблиц тех типов, которые поддерживают транзакции, последние два

способа являются более гибкими. Например, все таблицы InnoDB хранятся в группе больших файлов, поэтому архивы нельзя будет сгруппировать по базам данных или таблицам.

Какой бы метод ни был выбран, не забудьте защитить таблицы от изменений на время резервного копирования. Если копируются табличные файлы, следует остановить сервер. В остальных случаях достаточно поставить блокировки чтения с помощью инструкции LOCK TABLES и выполнить инструкцию FLUSH TABLES. Последняя необходима для того, чтобы все изменения индексов были записаны в таблицы. Наличие блокировок чтения позволит другим процессам параллельно обращаться к таблицам с запросами на выборку.

Для создания sql-образа таблицы предназначена утилита mysqldump. Она записывает текст инструкций в поток stdout, поэтому нужно перенаправить результаты ее работы в файл.

Сделать дамп базы данных:

mysqldump -u root -p --databases имя_базы_данных

--add-drop-table > путь_к_файлу.sql

Не забудьте заблокировать все таблицы для записи, прежде чем запускать утилиту mysqldump. В противном случае целостность результатов не гарантируется.

В случае отсутствия своей базы данных восстановить свою базу данных можно следующей командой:

mysql -u root -p имя_базы < дамп_базы.sql

Примечание. Восстановление базы возможно и так:

- создать только пустую базу данных с именем своей базы данных

CREATE DATABASE имя_базы

- сделать базу данных текущей

USE имя_базы

- восстановить таблицы базы данных

SOURCE дамп_базы.sql

- проверить наличие таблиц

SHOW TABLES

Инструкции BACKUP TABLE и RESTORE TABLE копируют табличные файлы в указанный каталог. Естественно, серверный процесс должен иметь право записи в этот каталог. Программа MySQL копирует туда файлы с расширениями .frm и .MUD. Индексный файл (.MYI) можно воссоздать на основании первых двух, что позволит сэкономить место в ар-

хиве. *Например,*

```
BACKUP TABLE dictionary TO '/tm/backup'
RESTORE TABLE dictionary FROM '/tmp/backup'
```

Инструкция BACKUP TABLE самостоятельно заботится о блокировании таблиц и очистке табличных буферов. Это означает, что, в отличие от других методов резервного копирования, дополнительные инструкции не нужны.

Инструкция RESTORE TABLE копирует архивные файлы в каталог базы данных и перестраивает индексы. Таблица не должна существовать на момент восстановления. В случае необходимости можно удалить ее с помощью инструкции DROP TABLE или же вручную удалить табличные файлы.

Использованная и рекомендуемая литература

1. Чичев А.А., Чекал Е.Г. Операционные системы. Часть 1. Работа с операционной системой : учебно-методическое пособие. – Ульяновск : УлГУ, 2015.
2. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – Издательский дом «Питер», 2001.
3. Чекал Е.Г., Чичев А.А. Надёжность информационных систем. Часть 1 : учебное пособие. – Ульяновск : УлГУ, 2012.
4. Таненбаум Э., Вудхалл А. Операционные системы. Разработка и реализация. 3-е изд. – СПб. : Питер, 2007. – 704 с., ил.
5. Беляков М.И., Рабовер Ю.И., Фридман А.Л. Мобильная операционная система. – М. : Радио и связь, 1991.
6. URL: //top500.org (дата обращения 07.01.14).
7. Робачевский А. Операционная система Unix. – BHV, 1999.
8. Ахо В., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. – «Вильямс», 2001.
9. Боуман И. Концептуальная архитектура Ядра Linux. 1998 г. / пер. Д. Шевченко. – 2006.
10. Керниган Б.В, Пайк Р. Unix – универсальная среда программирования. – М. : Финансы и статистика, 1992.
11. Дейтел Г. Введение в операционные системы. – М. : Мир, 1987.
12. Дунаев С. Unix. System V. Release 4.2. – М. : Диалог МИФИ, 1996.
13. Нечаева Н.Н. Программирование в системе 1С : учебное пособие. – Ульяновск : УлГУ, 2012.
14. Вирт Н. Алгоритмы+структуры данных=программы : пер. с англ. – М. : Мир, 1985. – 406 с., ил.
15. Робачевский А. Операционная система Unix. – BHV, 1999.
16. Цикритис Д., Бернстайн Ф. Операционные системы. – М. : Мир, 1977.
17. Снейдер Й. Эффективное программирование TCP/IP. – Питер, 2001.
18. ГОСТ 19781-90. Обеспечение систем обработки информации программное. Термины и определения. – М. : Изд-во стандартов, 1990.
19. Администрирование MySQL. НОУ «ИНТУИТ». – Режим доступа : https://www.intuit.ru/studies/courses/989/165/info (дата обращения 12.06.2018).

Учебное издание

Чичев А.А., Чекал Е.Г.

АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Часть 1. ОБЩИЕ ВОПРОСЫ

Учебно-методическое пособие

Директор Издательского центра *Т.В. Филиппова*
Подготовка оригинал-макета *М.А. Водениной*

Издается в авторской редакции

Подписано в печать 08.10.2018. Формат 60×84/16.
Гарнитура Times New Roman. Усл. печ. л. 9,0.
Тираж 100 экз. Заказ № 137 /

Оригинал-макет подготовлен и тираж отпечатан в Издательском центре
Ульяновского государственного университета
432017, г. Ульяновск, ул. Л. Толстого, 42