Zenbook. Загрузка и загрузчики

Рубрика: Boot&Init, Zenbook — alv @ 3:39 пп

Автор: Алексей Федорчук

Первое, что следует сделать после установки новой системы — это загрузить её. И потому разговор о ней резонно начать именно с загрузки и загрузчиков. Правда, это не та тема, в которую начинающему пользователю придётся вникать в первую очередь — как говорилось в двух предыдущих главах, загрузчик будет настроен автоматически и, в первом приближении, правильно. То есть, скорее всего, свежеустановленный Zenwalk загрузится без проблем.

Однако:

- во-первых, при установке не исключены неожиданности, которые потребуют ручного вмешательства, и,
- во-вторых, потенциальный пользователь Zenwalk'а устанавливает его (в том числе и) для изучения.

А изучение любой системы начинается с её загрузки. Так что начнём с этого и мы — тем, кто пока не ощутил необходимости разбираться с этой темой, предлагается просто пропустить пока эту главу.

Содержание

О загрузке вообще

Под загрузкой системы принято понимать загрузку ядра — в данном случае Linux'а (смайлики по вкусу), и подлючение к нему необходимых модулей. Роисходит это примерно так.

Перво-наперво после включения питания запускается программа, прошитая в ПЗУ компьютера (BIOS). Она выполняет процедуру POST — Power On Self Test, то есть самотестирование железа, после чего отыскивает носитель, установленный в BIOS Setup как первое загрузочное устройство. Дальнейшие события зависят от того, какое именно устройство выступает в этом качестве — дискета, винчестер, компакт-диск или, в последнее время — носитель с USB-интерфейсом.

В случае с дискетой или компактом сценарий зависит от их содержимого, с USB-носителем (конкретно, обычно, флэшкой) — от того, какую роль она выполняет в текущих установках BIOS. Эти случаи мы здесь рассматривать не будем.

А вот если в качестве загрузочного устройства выступает винчестер, то программа из ПЗУ находит на нем первый физический блок, содержащий ту самую главную

загрузочную запись (MBR), которая вскользь упоминалась в нашем ликбезе, и передаёт ему вахту.

Поскольку MBR занимает первый физический блок, размер которого всегда равен 512 байтам, то она не может превыщать этой величины. Из них последние два байта занимает так называемая сигнатура диска, по которой BIOS узнаёт, что он имеет дело именно с винчестером (если эта сигнатура не соответствует принятым символам — 55h AAh в шестнадцатеричном счислении, загрузка прекращается).

Следующие от конца 64 байта отводятся под таблицу дисковых разделов, состоящую из четырёх записей по 16 байт. Каждая из них описывает один из четырёх теоретически возможных первичных разделов (primary partitions), о которых говорилось в соответствующей главе, вне зависимости от того, сколько их было создано, и были ли они созданы вообще: в принципе, диск может быть использован и без разделения на разделы, как “сырое” (raw) устройство. С винчестерами так почти никогда не поступают, но это было обычной практикой для дискет и, может иметь место и поныне на флэшках — иногда можно увидеть их отформатированными таким образом. Именно потому, что в таблице разделов предусмотрено четыре записи, число первичных разделов на диске ограничено этой цифрой — так повелось с древних времён IBM PC.

Оставшиеся 448 байт содержат некий код, принимающий на себя управление от BIOS по окончании его работы. В стандартном MBR — том, что фабрично записан на “свежевкрученном” винчестере, этот код только и может, что отыскать первый физический раздел диска (точнее, раздел, помеченный как активный — но, если не предпринять специальных действий, эти понятия совпадают) и “по цепочке” передать управление на его загрузочную запись (Boot Record — уже без Master). Чего было вполне достаточно для загрузки операционок вроде DOS или Windows 9X/ME с первого (или единственного) раздела. Но явно мало в любом другом случае — например, если на диске установлено несколько ОС, которые, естественно, не могут уместиться в одном разделе.

А как обстоит дело с загрузкой предмета нашего разговора, то есть с ядра Linux? В принципе, оно способно загрузить себя само, без посторонней помощи. Образ ядра содержит в себе загрузочную запись, аналогичную BR дискового раздела, которая принять управление непосредственно от BIOS компьютера или от кода, записанного в MBR винчестера. Такой процесс самозагрузки ядра называется bootstrapping - поднятие себя на ноги за шнурки от собственных ботинок.

Однако на практике это оказывается почти столь же неудобным, как и процедура подъема вышеуказанным способом. Потому что образ ядра, предназначенный для непосредственной загрузки, должен быть размещен либо на raw-устройстве (это практиковалось во времена загрузочных дискет), либо на специальном дисковом разделе. Причем раздел этот не имеет права нести на себе какую-либо файловую систему: ведь для того, чтобы распознать образ ядра как файл, а не последовательность нулей и единиц, нужно иметь загруженное ядро с поддержкой данной файловой системы, а ядро-то у нас еще не загружено.

Собственно, этим и определяется неудобство такого способа. Во-первых, выделять специальный первичный раздел только под образ ядра представляется слишком жирным — ведь на диске их может быть всего четыре. Во-вторых, при пересборке ядра оно должно каждый раз переустанавливаться на этот raw-раздел, что не позволяет экспериментировать с ядрами различных версий или сборок с разными параметрами. Наконец, такой способ лишает возможности использовать Linux

совместно с какой-либо другой (или другими) ОС.

Так что прямая загрузка образа ядра используется, насколько мне известно, только при изготовлении загрузочных дискет, и с отмиранием флоппи-приводов забудется окончательно. А штатным средством загрузки ядра Linux оказывается специальная программа, которая так и называется - загрузчиком, обычно с эпитетами - начальный, системный или мультисистемный.

Такой настоящий загрузчик также должен включать в себя код, записываемый в MBR диска или BR первичного раздела. Но функции его — более сложные. Во-первых, он обеспечивает выбор между разделами, с которых может быть загружена та или иная ОС. А далее — возможны варианты:

1. передача управления на BR выбранного раздела “по цепочке”, со сложением с себя ответственности за всё остальное;
2. непосредственная загрузка ядра ОС с выбранного раздела;
3. передача управления специальной программе, способной опознавать различные файловые системы, воспринимать желащие на них ядра как файлы и, соответственно, запускать их на исполнение, то есть, своего рода, мини-ОС.

“Цепочечный” вариант работы поддерживается всеми системными загрузчиками, составляя первую стадию их работы. А вот во второй стадии, если на ней не задействется иной загрузчик из BR выбранного раздела, вступает в силу альтернатива между пунктами 2 и 3.

Из двух традиционно используемых в Linux программ загрузки — Lilo или GRUB, первая реализует механизм “цепочки” плюс загрузку ядра, другая же на второй стадии прибегает к опознанию файловой системы и запуску ядра с неё. В Linux возможно применение и более иных загрузчиков, но эти — наиболее распространены. И либо тот, либо другой, безальтернативно или по выбору, устанавливается по умолчанию при инсталляции любого дистрибутива этой ОС.

В дистрибутиве Zenwalk штатно и безальтернативно предусмотрена установка Lilo — с его рассмотрения мы и начнём.

Загрузчик Lilo

Lilo (LInux LOader) — традиционный и самый старый из загрузчиков Linux, единственный, который был написан специально для загрузки ядра именно этой операционной системы с жесткого диска. Его функции как мультисистемного загрузчика вторичны. Но, тем не менее, он успешно справляется с загрузкой Windows любого рода, FreeBSD и, вероятно, других BSD-систем (последнее лично мной не проверялось).

Как уже было сказано, в Lilo реальзуется схема прямого обращения к ядру Linux из программы в MBR. Либо, в случае выбора загрузки ОС, отличной от Linux — передача управления на BR несущего её раздела.

То есть в общем случае Lilo записывается в MBR загрузочного диска, откуда при старте машины опознает дисковые разделы — как первичные, так и расширенные. И если раздел несет на себе метку Linux native — он загрузит с него образ ядра Linux (изначально Lilo для этого и предназначался, его функции мультисистемного загрузчика вторичны). Если же тип раздела - иной, Lilo в состоянии “по цепочке” передать управление на его загрузочный сектор, и дальнейшее будет определяться тем кодом, что записан в последнем.

Код Lilo, вместе с конфигурацией его меню, целиком лежит в MBR диска (или BR загрузочного раздела). Так что тип файловой системы для него по большому счету безразличен — то есть он способен загрузить ядро Linux с любой из ее многочисленных нативных файловых систем (и даже с файловой системы DOS, FAT16, за более поздние её модификации не скажу за незнанием).

Для разделов “чуждых” операционок тип их файловой системы для Lilo безразличен тем более — ведь, как уже было сказано, он просто передает управление на их загрузочные сектора. Таким способом - по “цепочке” - Lilo может обеспечить загрузку DOS, Windows 3.X/9X/ME и Windows NT/2000/XP, а также FreeBSD.

Каждая ОС, предусмотренная для “загрузки” через Lilo (вы ведь понимаете, почему теперь я взял это слово в кавычки? - именно, потому что в полном смысле слова оно применимо только к загрузке Linux), представляет собой один из вариантов выбора в меню этой программы. Важно, что таким вариантом может являться не обязательно отдельная операционка, но и разные дистрибутивы Linux, лежащие на собственных разделах, и разные версии ядра одного и того же дистрибутива, и просто разные сборки ядра одной и той же версии. Поэтому Lilo очень часто применяют для тестирования ядер этой системы, скомпилированных с разными опциями, или ядер разрабатываемой ветки.

Варианты загрузки через Lilo описываются в его специальном конфигурационном файле — /etc/lilo.conf. Это простой текстовый файл, доступный для изменения в текстовом редакторе (при наличии полномочий суперпользователя, разумеется). Он содержит несколько секций — общуюg (lobal section), описывающую условия загрузки в целом и отдельные — описание каждого варианта загрузки.

Общая секция обычно содержит такие строки:

- имя устройства, с которого выполняется запуск Lilo (именно Lilo, а не Linux или иной ОС), для случая первого диска это будет выглядеть как boot=/dev/sda;
- предписание вывордить меню выбора вариантов загрузки — prompt;
- время ожидания выбора вариантов загрузки — timeout=##, где ## - время в миллисекундах;
- имя варианта, загружаемого по умолчанию, которое должно соответствовать тому, что далее будет указано в секции этого варианта.

Собственно, обязательной является только первая из них. Если не указать умолчальный вариант, то таковым будет выступать описанный в первой из отдельных секций, без указания времени ожидания выбора он загрузится мгновенно, ну а без строки prompt мы просто не увидим меню выбора вообще.

Кроме указанных, в общей секции может присутствовать и другие строкм, определяющие видеорежим — плотность знаков чисто текстовой консоли или разрешение консоли графической (через VESA framebuffer — ныне последний вариант общеупотребим), а также устанавливающие всякие украшательства — расцветку меню, сплэш-картинку при загрузке, и так далее.

Содержание отдельных секций различается в зависимости от того, предназначен этот вариант для загрузки Linux или иной ОС. В общем случае обязательными здесь будут две строки. Первая — это метка варианта (label), то есть его уникальный идентификатор в виде произвольной, но, желательно, мнемонически прозрачной последовательности символов, например: linux, freebsd, windows. Вторая же - имя устройства, несущего загружаемую ОС или его корневую систему (вроде /dev/sda1,

/dev/sda5, и так далее).

Чтобы читатель мог представить себе это зримо, приведу свой вариант общей секции файла /etc/lilo.conf с некоторыми комментариями:

```
# Конфигурационный файл LILO
# сгенерирован программой 'liloconfig'
#
# Начало общей секции LILO

# Добавление загрузки сплэш-картинки
append="resume=/dev/sda2 splash=silent "

# Определение загрузочного диска
boot = /dev/sda

# Имя файла сплэ-картинки
bitmap=/boot/splash.bmp

# Предписание выводить меню
prompt

# Время ожидания выбора (в секундах)
timeout = 50

# Устанавлиет разрешение и глубину цвета
# графической консоли 1024x768x64k
vga = 791

# Конец глобальной секции LILO
```

К вопросу, каким образом всё это было получено, мы ещё вернёмся.

Субсекция для каждого варианта, загружающего Linux, кроме обязательного идентификатора, должна включать:

- имя файла, содержащего образ ядра, с указанием пути относительно корневого каталога, заданного в одной из следующих строк, что обычно выглядит как image=/boot/bzImage или image=/boot/vmlinuz;
- имя устройства, несущего корневую файловую систему, для случая первого раздела на первом диске это будет root=/dev/sda1;
- указание монтировать корневую файловую систему при загрузке ядра в режиме “только для чтения” read-only; это не значит, что она станет недоступной для записи после загрузки — в процессе инициализации файловые системы будут перемонтированы в соответствие с их описанием в /etc/fstab; да и вообще эта строка не обязательна, и приведена здесь только для порядку.

Кроме того, в Linux-субсекции /etc/lilo.conf могут вноситься и строки, определяющие видеорежим при загрузке (если его не определили в глобальной секции или требуется сделать его иным) и строки, передающие ядру некие параметры, вроде append=”noapic” или append=”noapic”; раньше необходимость в них возникала довольно часто (например, для включения эмуляции протокола SCSI через IDE-шину, что требовалось для записи компакт дисков), но нынче я уже и забыл, когда последний раз приходилось что-то подобное дописывать.

Опять таки для наглядности приведу свои субсекции для разных вариантов загрузки (должен сказать, что у меня их обычно не богато):

```
# Конфигурация загрузочных разделов Linux
```

```
# Загрузка тестовой версии ядра
# Файл образа ядра
image = /boot/vmlinuz-2.6.29-rc3

# Устройство с корневой файловой системой
root = /dev/sda1

# Метка варианта для меню
label = Zen-2.6.29-rc3
read-only

# Загрузка штатной версии ядра
# Файл образа ядра
image = /boot/vmlinuz-2.6.27.10

# Устройство с корневой файловой системой
root = /dev/sda1

# Метка варианта для меню
label = Zenwalk

# Имя инициирующего RAM-диска
initrd = /boot/initrd.splash
read-only
```

В субсекции для загрузки тестовой версии ядра, думаю, всё понятно. А в субсекции для загрузки версии ядра штатной надо обратить внимание на два момента.

Во-первых, на строку с указанием файла образа ядра. Обычно здесь фигурирует файл /boot/vmlinuz символическая ссылка на обычный файл вида /boot/vmlinuz-2.6.XX.YY для образа ядра конкретной версии. Однако, как будет сказано в главе о компиляции ядра, при “ядерных” экспериментах возможно переопределение этой ссылки на файл образа новоустанавливаемой версии, и потому лучше всегда задавать его явным образом.

Во-вторых, во второй субсекции можно видеть указание на файл initrd. Это — образ так называемого инициирующего диска в оперативной памяти (RAM-диска), обеспечивающего универсальность загрузки одного и того же ядра на машинах с разными конфигурациями “железа” и разными типами корневой файловой системы. Подробнее этот вопрос также будет рассмотрен в “ядерной” главе.

Субсекции для вариантов загрузки ОС, отличных от Linux, как правило, не содержат ничего, кроме имени устройства несущего раздела и метки варианта:

```
other=/dev/sda1
label=dos
```

Lilo может использоваться и исключительно для загрузки единичной Linux-системы в сочетании с любым мультисистемным загрузчиком, отвечающим за загрузку систем прочих (BSD Loader, GRUB, вплоть до NT Loader). В этом случае в общей секции /etc/lilo.conf в качестве загрузочного устройства следует указать имя корневого раздела Linux-системы — ведь тогда Lilo будет стартовать уже с его загрузочного сектора (а не с MBR диска):

```
boot=/dev/hda1
```

В общей секции же секции можно раз навсегда определить имя устройства корневой файловой системы. А далее возможны и отдельные секции для вариантов загрузки, но они будут описывать уже не самостоятельные ОС или дистрибутивы, а разные версии или сборки ядра, возможно - видеорежимы или параметры ядра.

Как уже было сказано, Lilo напрямую с файловыми системами не работает - даже с файловыми системами Linux. Конфиг же этого загрузчика лежит в каталоге /etc, представляющем собой часть файлового древа инсталлированной Linux-системы, которая при старте машины еще не загружена. Как же загрузчик узнает о собственной конфигурации?

Очень просто, о существовании собственного конфига в момент запуска Lilo и не подозревает. Необходимые данные для опознания загрузочных разделов и ядер на них прошиты в виде бинарного кода в загрузочном секторе - диска или раздела. И делается это одноименной командой - /sbin/lilo. При запуске она обращается к файлу /etc/lilo.conf, исходя из значения строки

```
boot=/dev/имя_устройства
```

в общей его секции, определяет, куда нужно записать данные - в MBR диска или загрузочный сектор раздела, и выполняет запись вариантов загрузки, завершаемую сообщением вроде следующего:

```
Added Zen-2.6.29-3 *
Added Zenwalk
```

где имя_рек - метка (label) добавленного или модифицированного пункта меню.
Из чего становится очевидным, что каждое изменение конфигурации загрузчика должно сопровождаться его переустановкой - перезапуском программы /sbin/lilo.

Теперь пора поговорить о том, откуда берётся файл /etc/lilo.conf и каким образом его можно модифицировать.

Установка Lilo и создание его конфигурационного файла обычно происходит на стадии установки дистрибутива, если его инсталлятор предусматривает использование этого загрузчика, безальтернативно или по выбору (в этом случае обычно в качестве альтернативы выступает GRUB).

Утилита liloconfig

Первый случай, как мы помним, имел место при установке Zenwalk'а. Можно предположить, что он собственно его инсталлятором и реализуется, но это не так. За установку и конфигурирование Lilo в этом дистрибутиве отвечает специальная утилита — liloconfig, унаследованная от Slackware. Она может быть запущена и вне инсталлятора — в любой последующий момент времени. Важно только перед этим выполнить процедуру копирования существующего конфига:

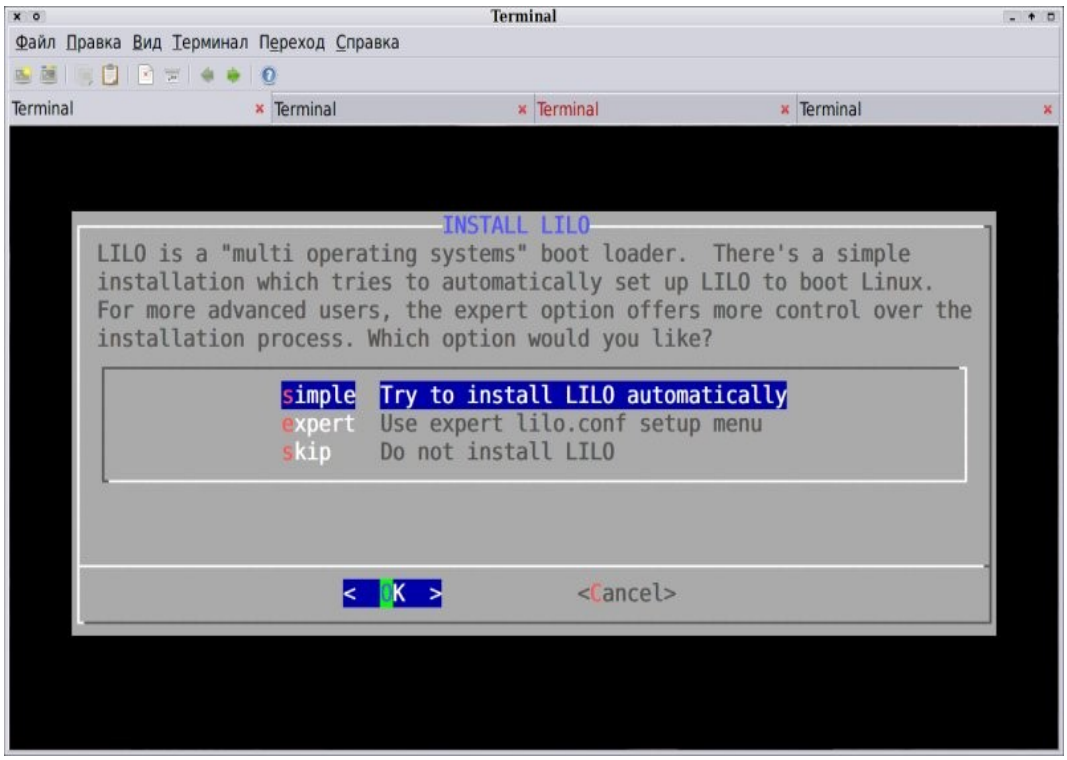
```
# cp /etc/lilo.conf /etc/lilo.old
```

Почему — будет ясно ближе к концу раздела.

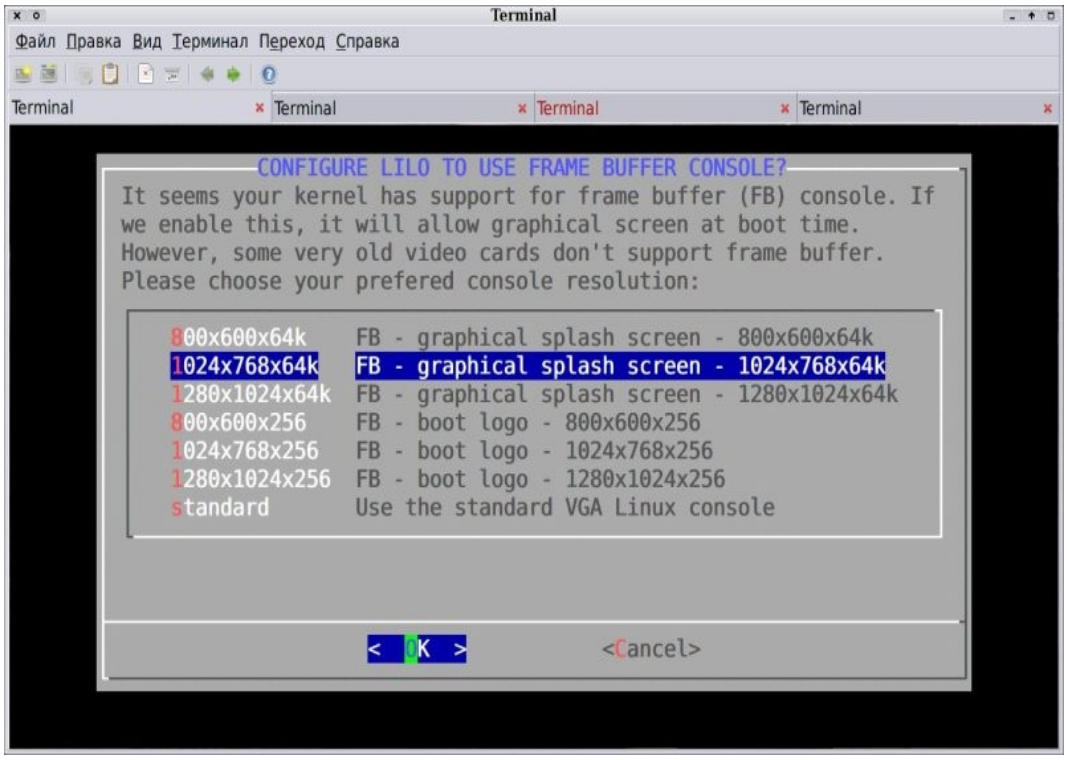
Программа liloconfig запускается от лица суперпользователя одноименной командой:

```
# /sbin/liloconfig
```

после чего демонстрирует свой простой псевдографический интерфейс, основанный на библиотеке ncurces, в характерном для Slackware стиле:



Как видно на рисунке, нам предлагается на выбор два варианта установки Lilo (не считая Skip — то есть отказа от оной): простой (Simple) и экспертный. Первый — тот, к которому мы прибегли при первичной инсталляции. При выбор его у нас будет спрошено только желаемое разрешение консоли; по умолчанию предлагается 1024*768*64K через фреймбуффер, но можно выбрать и иное из списка, а также стандартную текстовую консоль с плотностью символов 80*25.



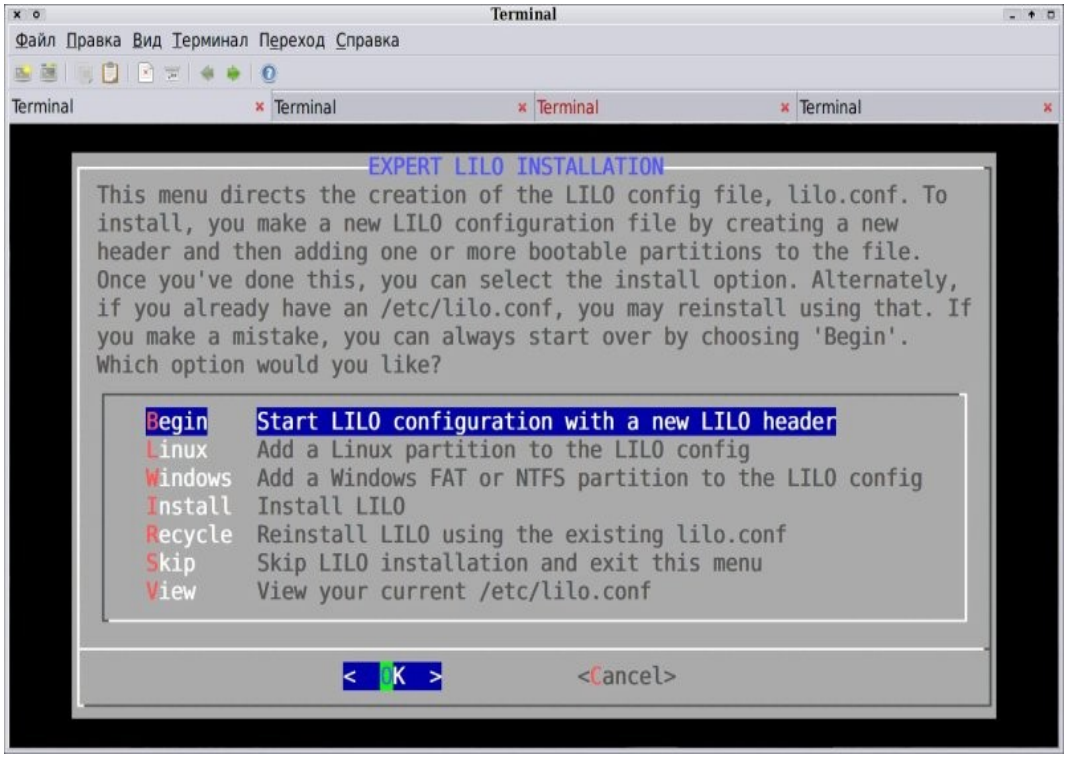
Всё остальное будет проделано автоматически: утилита отыщет загрузочный раздел Linux, а также и Windows, если таковой имеется, внесёт в меню соответствующие им

пункты с некоторыми умолчальными параметрами; в частности, загружаемой системой по умолчанию окажется Windows. После чего незаметно для пользователя будет запущен /sbin/lilo с записью конфигурационных параметров в MBR диска.

Напомню, что утилита liloconfig не слишком “умная”, и, обнаружив первый же раздел с FAT любого рода или NTFS, сочтёт, что он несёт на себе соответствующую ОС, даже если её там и близко не лежала, и, исходя из этого предположения, допишет в меню соответствующий пункт, сделав его умолчальным. Разумеется, загрузить через него не удстся ничего, и исправление этой не вполне удобной ситуации потребует ручного вмешательства.

Кроме того, следует помнить, что “простой” способ приемлем только в том случае, если мы твёрдо решили сделать Lilo первичным мультисистемным загрузчиком, потому что его конфигурация в этом случае безальтернативно будет записана в MBR диска.

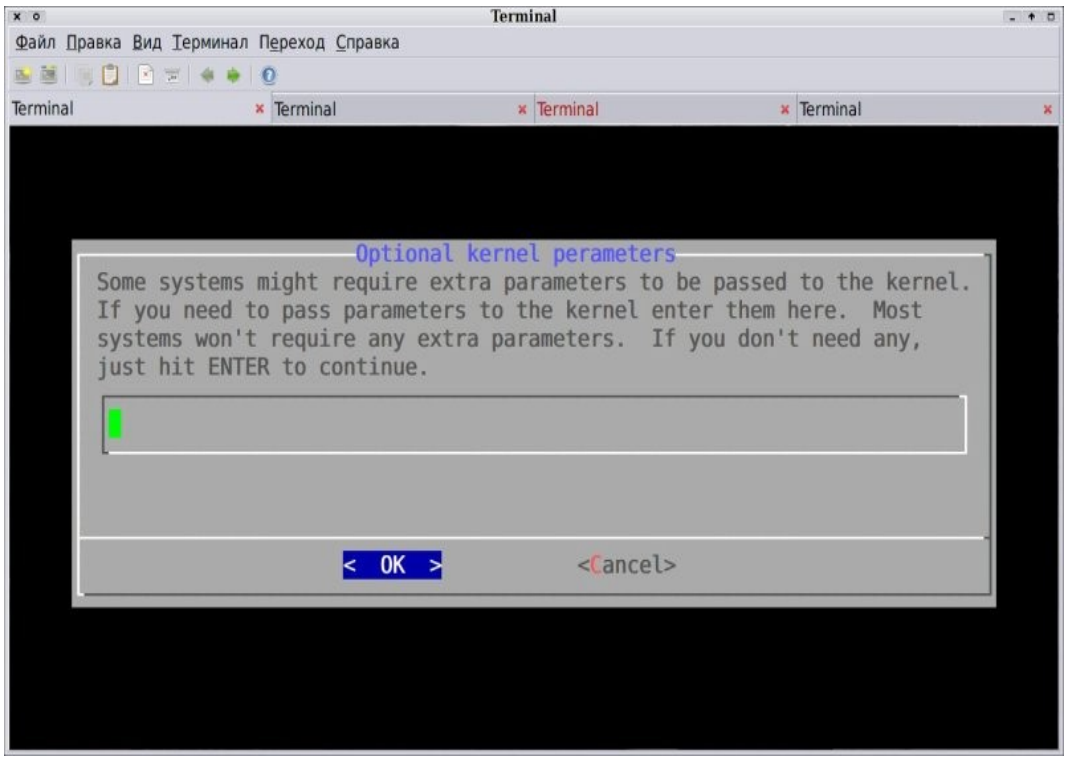
Другое дело — экспертный вариант загрузки: тут пользователю предлагается субменю с рядом пунктов:



И в каждом придётся ответить на ряд вопросов, на основе ответов на которые и будет построен новый файл /etc/lilo.config.

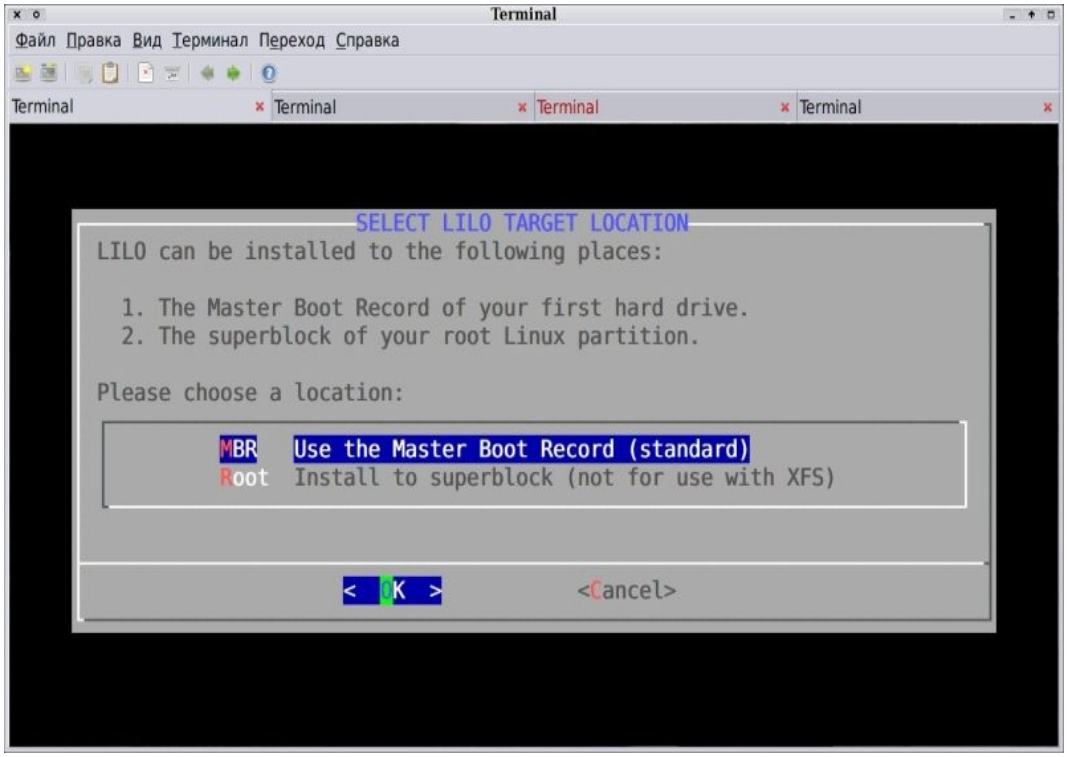
Начнём с первого пункта, Begin, который обеспечивает заполнение глобальной секции этого файла.

Перво-наперво будет предложено задать параметры ядра, те самые, значения которых будут фигурировать в строке append глобальной секции конфига загрузчика:



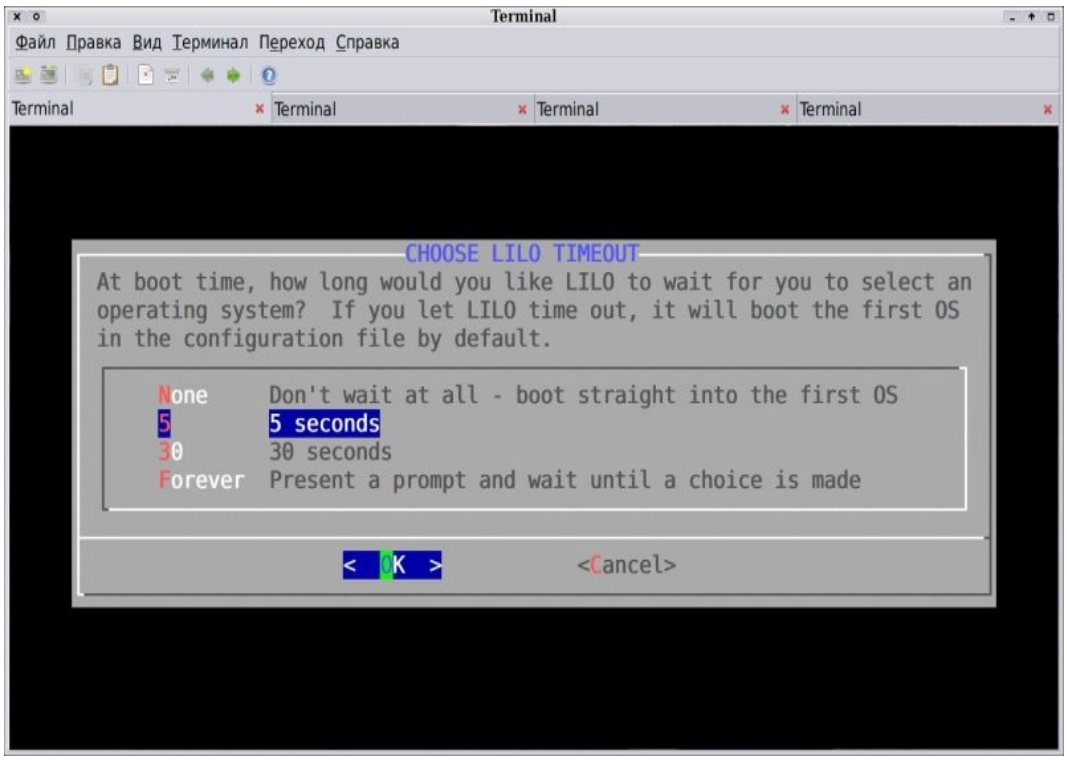
Этот шаг можно спокойно пропустить, если вы точно не знаете, что некие параметры нужны — и точно знаете, какие именно.

Следующим будет вопрос о месте размещения Lilo — в MBR диска или на корневом разделе для Linux-системы:

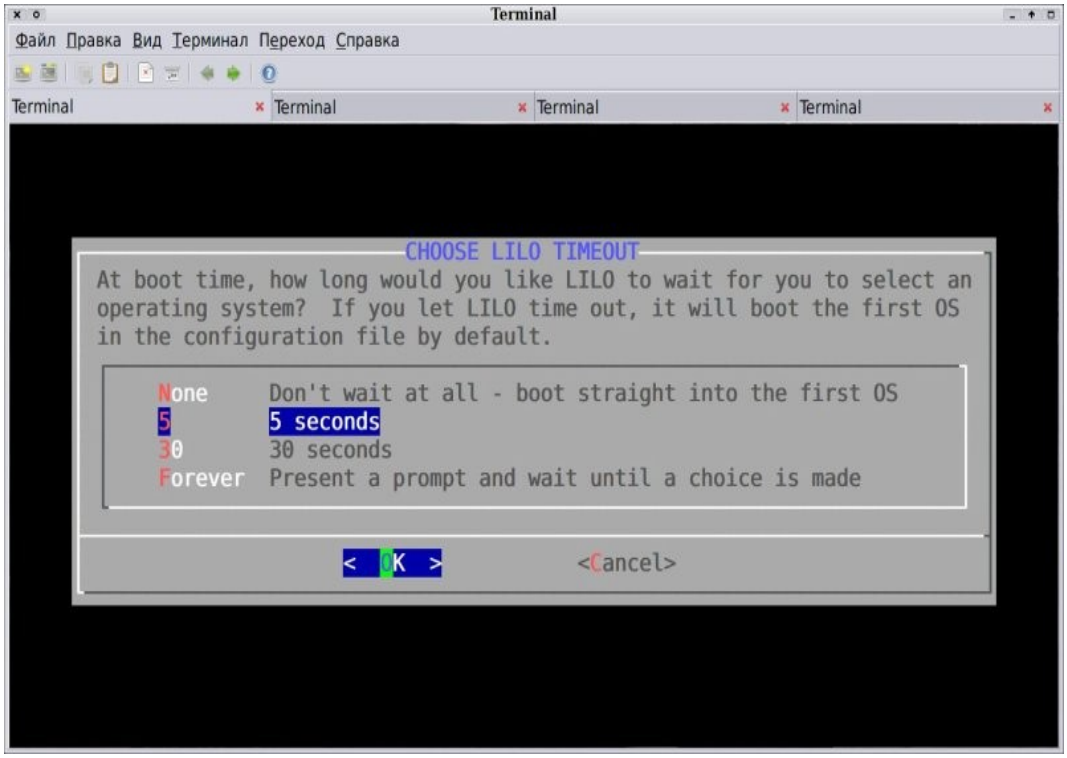


Второй ответ надо выбрать в том (и только в том) случае, если у вас уже имеется какой-либо мультисистемный загрузчик, например, BSD Loader или GRUB. Впрочем, при наличии последнего, как мы увидим в следующем разделе, никакой необходимости в установке Lilo просто нет.

Далее надо установить время, отводимое пользователю на выбор операционки. Варианты — none, то есть мгновенная загрузка (целесообразно, если на машине установлена единственная ОС с единственным же вариантом загрузки), 5 и 30 секунд — выбирать в зависимости от быстроты реакции, и Forever — ожидание выбора до бесконечности.

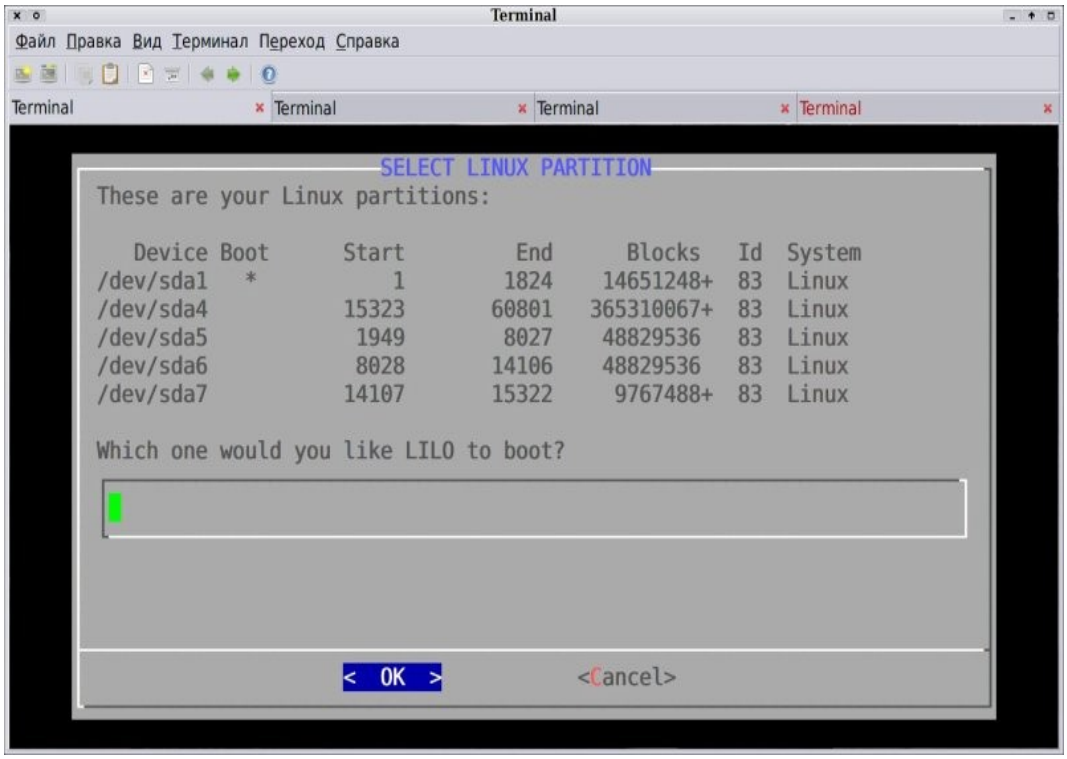


После этого происходит возврат в экспертное меню. Теперь надо выбрать параметры для субсекций, соответствующих отдельны вариантам загрузки. Начнём с загрузки Linux. И для начала выберем раздел, на котором расположена его корневая система, из списка существующих:



В нашем случае это будет /dev/sda1 — его и вписываем в соответствующее поле ввода. Это будет значением параметра root в нашей первой субсекции.

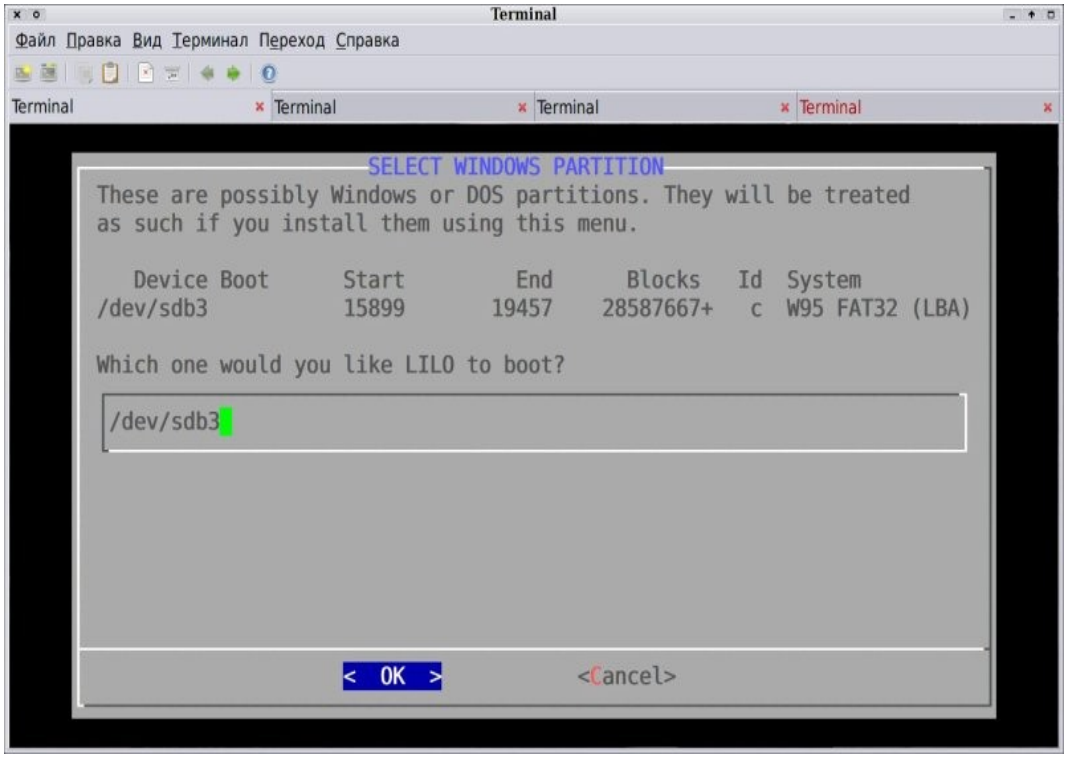
Затем следует задать уникальное имя для субсекции — оно будет значением строки label в конфигурационном файле и названием пункта в меню Lilo при загрузке. Как явствует из рисунка, имя должно быть однословным, но всякого рода цифры, дефисы и подчёркивания в нём допукаются.



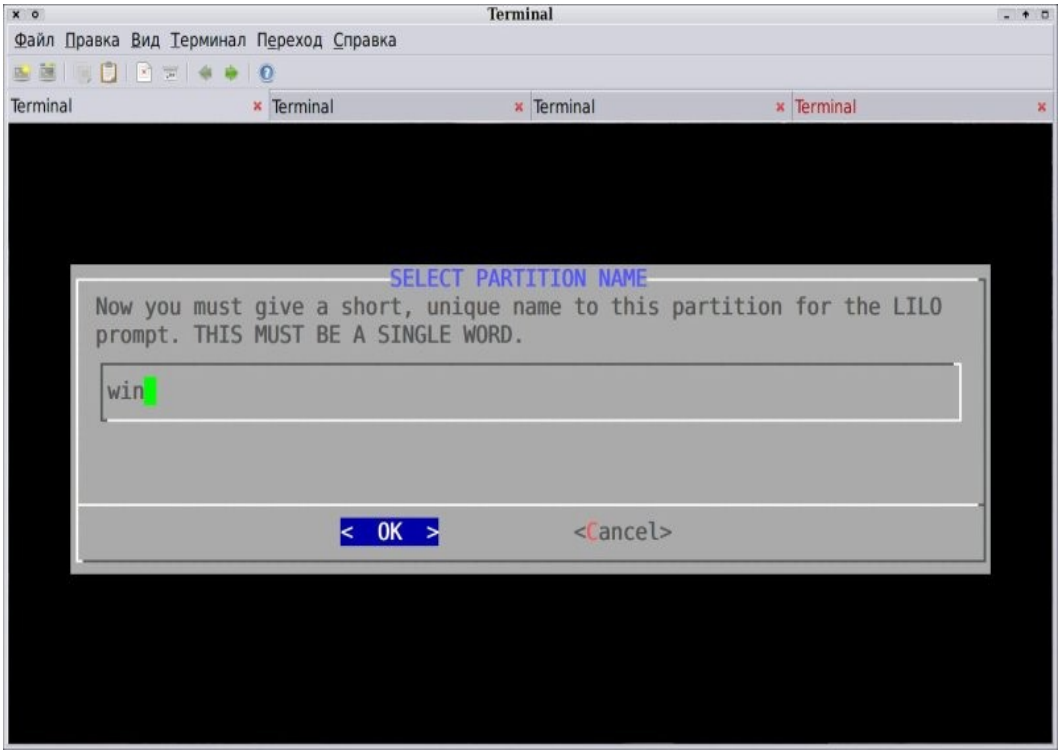
После этого мы опять возвращаемся в меню эксперта, и можем продолжить добавление новых вариантов загрузки. Например, можно добавить возможность

вызова ещё одного дистрибутива Linux'а, лежащего на самостоятельном разделе. Однако сделать несколько вариантов загрузки одной и той же системы не получится: liloconfig автоматически придаст параметру image значение vmlinuz, и никакое другое. Так что для экспериментов, например, с разными сборками ядра или разными параметрами, конфиг Lilo придётся всё равно править вручную.

Однако это компенсируется возможностью добавить вариант загрузки какой-либо Windows. Это делается совсем просто. Сначала выбирается раздел, несущий эту ОС, из списка разделов с подходящими файловыми системами:



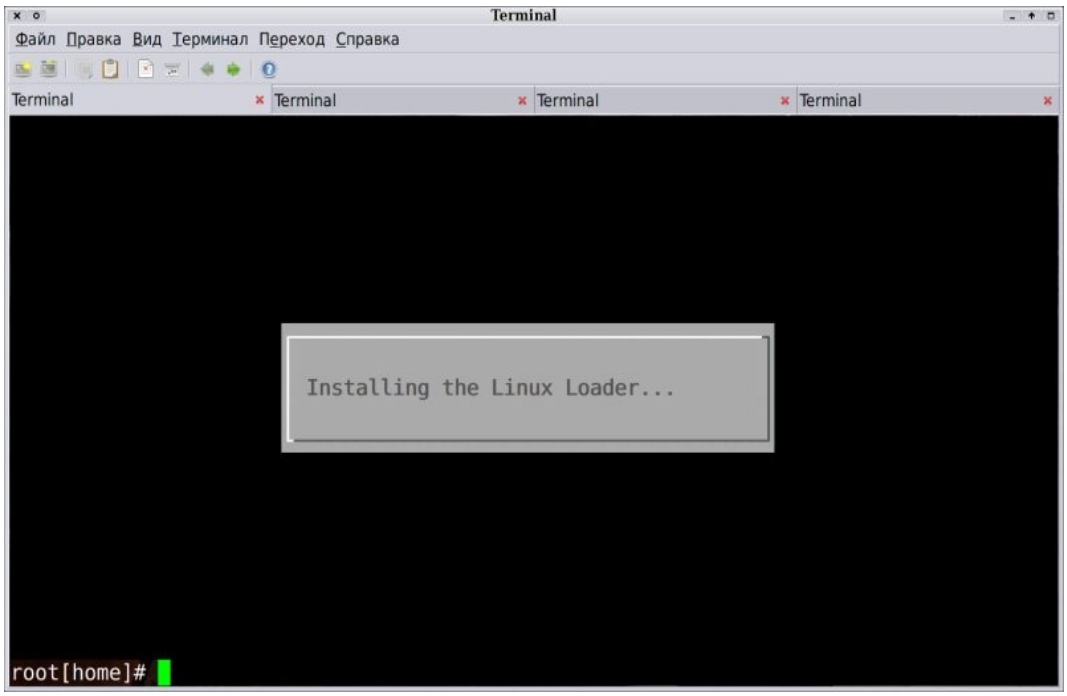
В приведённом примере выбирать нечего. И, к тому же, упаси Бог вас подумать, что на нём действительно стоит Windows — он существует исключительно для обмена данными, но liloconfig об этом не догадывается, и на следующем шаге предлагает задать имя для соответствующего ему пункта:



После чего опять же вернёт нас в меню эксперта.

После того, как мы расправились со всеми пригодными для загрузки наличными операционками, нужно выполнить последнее деяние: увековечить результаты наших действий через пункт меню Install: до сих пор не было выполнено ни одного необратимого действия, и если появились сомнения в правильности настроек загрузчика, можно просто через Cancel (или через комбинацию Control+C) выйти из liloconfig без всяких последствий и без малейшего вреда для существующей конфигурации.

А вот после перехода к пункту Install всё происходит быстро и молча: старый файл /etc/lilo.conf перезаписывается сгенерированным заново, и новые параметры записываются (посредством скрытого исполнения команды /sbin/lilo) туда, куда им было предписано — в MBR диска или BR раздела. После чего liloconfig завершает свою работу с возвратом в командную оболочку, из которой он был запущен:



И теперь изменить что-либо можно, только повторив процедуру конфигурирования. А вернуться к исходной конфигурации загрузчика — выполнив обратное копирование резервной копии старого конфига (мы ведь не забыли её сделать, не так ли?):

```
# cp /etc/lilo.old /etc/lilo.conf
```

Не забыв после этого перезапустить /sbin/lilo.

Я довольно подробно остановился на работе с утилитой liloconfig потому, что она, с одной стороны, снимает у начинающего пользователя страх перед ручной правкой конфига загрузчика, с другой же — способствует понимаю того, что скрывается за строками файла /etc/lilo.conf. Но на практике к ней прибегают достаточно редко (лично я — так ни разу в жизни, за исключением тех случаев, когда она автоматтом вызывается установщиком дистрибутива). Гораздо проще править конфиг Lilo вручную в любом текстовом редакторе — надеюсь, что приведённых в настоящем разделе сведений достаточно для осмысленного выполнения этой процедуры. Самое главное — не забывать после неё про необходимость перезапуска /sbin/lilo.

Можно видеть, что конфигурационный файл Lilo устроен очень просто, а сама загрузочная программа — проста в настройке и использовании (если не забывать про перезапуск /sbin/lilo). Однако у неё есть несколько… ну не столько недостатков, сколько ограничений.

Первое вытекает из положения конфига этого загрузчика внутри какой-либо из нативных файловых систем Linux. Так что для изменения его конфигурации из другой ОС необходимо иметь доступ к той файловой системе, на которой лежит корень инсталлированного Linux’а, причем в режиме записи. И если примонтировать Ext2fs/Ext3fs к файловой системе, например, FreeBSD в режиме read/write можно без проблем, то доступ из любой ОС BSD-семейства к разделам с ReiserFS или XFS в настоящее время возможен только для чтения (и неизвестно, будет ли возможен в обозримом будущем).

Конечно, можно взять за правило размещать корневую систему Linux только на разделе с Ext2fs/Ext3fs. Однако второе ограничение - необходимость перезапуска /sbin/lilo, - этим не обходится, ведь эта программа предназначена для работы в

родной ОС, сиречь Linux (интересно, а пробовал ли кто-нибудь запустить /sbin/lilo под FreeBSD в режиме Linux Compability?). И, соответственно, непременное условие для переконфигурирования Lilo — возможность запуска Linux в каком-никаком виде, хотя бы - с rescue-дискеты или LiveCD.

Есть и третье ограничение, не столь важное: относительно слабые интерактивные возможности. Конечно, Lilo позволяет в режиме командной строки вмешиваться руками в ход загрузки - но только загрузки Linux же (например, передавать параметры ядру). Да и то - в ограниченном объеме. Возможности же вмешательства в ход загрузки чужой ОС вообще заканчиваются в момент выбора соответствующего ей варианта.

И ещё: вся информация о конфигурации загрузчика записывается в MBR, объем которого, как мы помним, далеко не безграничен. И потому втиснуть в меню Lilo можно только конечное число вариантов загрузки, а именно — 16. Впрочем, практическое значение этот лимит имеет только для коллекционеров операционных систем (есть и такое хобби), а для их целей более пригоден будет GRUB.

Раз уж речь зашла об ограничениях Lilo, то, пользуясь случаем, подчеркну: прочие ограничения, на которые можно натолкнуться в литературе, как то: невозможность загрузить ядро Linux с области диска, лежащей за пределами первых 1024 его цилиндров, или с логического раздела в разделе Extended DOS, или с раздела с файловой системой ReiserFS, смонтированной в режиме тайлинга, — давно потеряли силу. И пользователь любого современного дистрибутива о них может смело забыть.

Тем не менее, основная сфера применения Lilo — это работа преимущественно (или исключительно) в Linux, при эпизодическом использовании какой-либо другой ОС. Причем рискну предположить, что под другой ОС будет выступать, скорее всего, какая-либо из версий Windows. Экспериментирование же с многочисленными операционками разных семейств — это, по моему мнению, прерогатива GRUB.

Гранд мира загрузчиков

Если Lilo, как и следует из его названия, создавался в первую очередь для загрузки Linux при сохранении возможности старта Windows, то целью разработчиков GRUB изначально было создание универсального мультизагрузчика, максимально независимого от любой операционной системы. История его началась в 1995 году, когда операционная система GNU Hurd (микроядерная ОС светлого будущего, столь же отдаленного, как и коммунизм в мировом масштабе) достигла той степени развития, что могла уже быть загружена. И встал вопрос - каким же образом это сделать, так как существовавшие тогда загрузчики оказались на это неспособными.

Конечно, с точки зрения идеологии GNU, логично было бы изобрести для этой цели собственный (еще один) загрузчик, в полной мере отвечающий идеалам свободы и демократии. Однако, к счастью для всего прогрессивного человечества, был избран иной путь: Multiboot Specification - универсальный метод загрузки ядер, этой самой спецификации соответствующих. Причем сама спецификация бралась отнюдь не с потолка - в ее основу легли особенности существующих ядер Linux и BSD-систем. Ведь, как я уже говорил, файлы их образов самодостаточны для запуска и несут в себе всю необходимую информацию - и дело упирается только в то, что в обычной ситуации они расположены на носителе с файловой системой, поддержка которых начинается только после загрузки соответствующего ядра. Так что суть

универсальности предложенного метода и заключалась в том, чтобы загрузчик получил средства доступа к файловой системе.

Зримым же, действительно универсальным, воплощением Multiboot Specification и стал мультисистемный загрузчик GRUB. Собственно, единственным его ограничением для полноценного использования (ниже станет ясно, что “неполноценное” использование GRUB вообще ничем не ограничено) была совместимость с файловой системой загружаемой ОС. Ибо, в отличие от BSD Loader и Lilo, GRUB способен работать с очень многими файловыми системами напрямую, без всякого использования ресурсов соответствующей операционки - “монтировать”, читать с них данные, подвергать их редактированию и записывать изменения. Что превращает его в своеобразную мини-ОС, имеющую к тому же свой собственный, шелл-подобный, интерфейс пользователя.

Можно выделить две степени совместимости GRUB с файловыми системами - полную и частичную. Полностью совместимые файловые системы - это те, что способны нести GRUB сам по себе. И в их числе - абсолютно все нативные файловые системы Linux (включая JFS, имевшие некогда место проблемы с “монтированием” ReiserFS, насколько я знаю, в прошлом), Minix и FAT. То есть загрузчик GRUB может стартовать с любого носителя, отформатированного соответствующим образом - дискового раздела, дискеты Linux (на которых применяется файловая система Minix) или DOS (с файловой системой FAT). Хотя по причинам, которые скоро станут понятны, разработчики рекомендуют размещать GRUB на самостоятельном разделе с файловой системой Ext2fs.

Частично совместимые с GRUB файловые системы сами по себе нести его не могут. Но после своего старта он способен их идентифицировать, прочитать и загрузить с них ядро, если оно отвечает Multiboot Specification. И в эту группу попадают практически все файловые системы свободных операционок, внутренний формат которых общедоступен - кроме Linux, тут мы видим также все варианты FFS и UFS, используемых BSD-семейством. Если же какая-то из таких файловых систем (как правило, из числа только что созданных) кажется несовместимой с текущей версией GRUB - это исключительно временное явление. Потому что для любой открытой файловой системы нет никаких препятствий получить свою поддержку в GRUB.

Приведу пример: с появлением 5-й ветки FreeBSD она обрела новую нативную файловую систему, UFS2, несколько отличную от прототипа - UFS просто. В результате единовременная версия GRUB (0.93) оказалась неспособной работать с ней напрямую (ниже я покажу, что это все равно было обходимо). Однако уже для версии 0.94 появился патч поддержка UFS2, а с версии 0.95 GRUB поддерживает эту файловую систему, что называется, “из тарбалла”.

Можно видеть, что в списке частично совместимых с GRUB файловых систем нет тех, что созданы Самой Великой Софтверной компанией, в частности - всех вариантов NTFS. Что понятно - ведь их внутреннее устройство - тайна за семью печатями. Однако мир свободного софта в очередной раз подтверждает справедливость утверждения, что на самое хитрое ухо (вариант для дам) всегда найдется болт с левой резьбой. Существует он и здесь: операционки, файловые системы которых для GRUB недоступны, могут быть загружены “по цепочке” - передачей управления на загрузочный сектор соответствующего раздела. Тот же способ можно применить и для загрузки свободных ОС, файловые системы которых временно не поддерживаются на моей памяти таковыми бывали периодически некоторые версии OpenBSD и FreeBSD 5-й ветви. К слову сказать - даже “ухо с

закоулками” (несовместимость ядра с Multiboot Specification, имеющая место для DOS/Windows9X/ME) не оказывается помехой для винта от GRUB. И указанные операционки могут быть столь же успешно загружены “цепочечным” методом.

Таким образом, универсализм GRUB в рабочем (то есть установленном и настроенном) состоянии сомнений не вызывает. Однако с точки зрения установки и настройки он предстает столь же независимым от любой ОС. Ибо устанавливается он с самодостаточной загрузочной дискеты. А с недавнего времени, благодаря усилиям Владимира Попова, это можно проделать и с мультизагрузочного LiveCD (который можно скачать отсюда). То есть: наличия инсталляции какой-либо ОС (или дистрибутива Linux не требуется).

Что же касается конфигурирования GRUB, то, если следовать приведенным ранее рекомендациям разработчиков (установке GRUB на отдельный дисковый раздел с файловой системой Ext2fs - и резоны к тому сейчас прояснятся), то и тут никаких сложностей не предвидится ни в одной из свободных ОС. В Linux этот раздел монтируется как каталог /boot, куда, вместе с файлами собственно загрузчика (в подкаталоге /boot/grub) помещаются также файлы ядра (/boot/vmlinuz и подобные ему). Причем разработчики опять же рекомендуют монтировать /boot не автоматически при старте Linux - после ее загрузки никакой необходимости к этому каталогу в обычных условиях нет), а руками - по мере необходимости (установки нового ядра или реконфигурации GRUB).

За настройку загрузки GRUB отвечает специальный файл, доступный из Linux как /boot/grub/menu.lst. Это простой текстовый файл, который можно модифицировать в любом редакторе. Причем, что ценно, предварительно проверив работоспособность вносимых изменений в интерактивном режиме. Описывать его формат в деталях не буду, он очень прост и станет ясным и приводимого в заключении примера.

Так что для конфигурирования же GRUB из какой-либо иной ОС требуется только возможность чтения раздела с Ext2fs и записи в него. А, как уже говорилось, получить такую возможность из FreeBSD и DragonFlyBSD ныне никакого труда не составит (да и в случае Net- или OpenBSD в худшем случае потребуется только пересборка их ядер). Так что рекомендация разработчиков о выборе файловой системы загрузочного раздела становится понятной — прочие файловые системы Linux BSD-семейством не поддерживаются, а FAT… ну он FAT и есть, всерьез говорить о нем не стоит.

Таковы основные особенности (и возможности) GRUB. О деталях его установки, настройки и интерактивной работы в ходе загрузки можно было бы написать еще много. Однако это уже проделано Владимиром Поповым, и повторяться я не буду. Памятуя и о прекрасной штатной документации этой программы, правда, в нелюбимом мною формате (info grub, стандартный man grub содержит лишь краткие о ней сведения).

GRUB в Zenwalk’е

Об удобстве настройки и интерактивных возможностях grub много хороших слов написал Владимир Попов, и повторяться я не буду — заинтересованные могут для начала прочесть это и вот это. А я попробую рассказать о том, как обрести эти удобства и возможности в нашем Zenwalk’е. И тут, как всегда, открываются два пути.

Первый — тот, что описан в соответствующем How-to из Zenwalk Wiki. То есть — взять какой-либо LiveCD, в котором в качестве загрузчика используется grub

(например, RIP, также описанный Владимиром), и, следуя вышеуказанному документу, установить grub с него. Процедура несложная (хотя в How-to она описана несколько невнятно), но требует скачивания дополнительного iso'шника и его записи. Хотя сам по себе RIP в жизни и пригодится, это может оказаться обременительным.

Второй же путь — более прост: воспользоваться готовым пакетом. Правда, ни в официальном репозитории Zenwalk'а, ни в ZUR мы такого пакета не найдём. Но тут самое время вспомнить о старшей родственнице, Slackware, и отправиться в хранилище её пакетов, официальных или неофициальных, где искомый grub легко отыскивается поиском по всем версиям — последняя его версия там на данный момент grub-0.97-i486-2.tgz для Slackware аж 10.2, но в данном случае это значения не имеет.

Вероятно, если тщательно порыться по сети, то можно обнаружить пакет и посвежее, например, что-нибудь из пре-релизов 2-й версии, или собрать таковой самому. Однако не уверен, что это стоит делать: «пред-первые» grub'ы годами служили верой и правдой. А вот как себя поведут «пред-вторые» — пока не ясно. А загрузчик — не та штука, с которой хотелось бы экспериментировать.

Так что скачиваем ранее найденный пакет, несмотря на его преклонный возраст, с подходящего зеркала и устанавливаем его традиционным способом, по-патриковски:

```
$ installpkg grub-0.97-i486-2.tgz
```

Разумеется, это и последующие действия выполняются от лица суперпользователя. И вот еще о чем следует помнить: если, следуя рекомендациям разработчиков grub, мы вынесли каталог /boot на самостоятельный раздел (что подразумевается и в дальнейшем), не монтируемый автоматически при старте системы, то перед отдачей команды installpkg его следует подмонтировать вручную:

```
$ mount /dev/sda1 /boot
```

На этом, однако, дело не кончается, ибо всё, что мы после этого имеем, — это всего лишь несколько исполняемых файлов с префиксом grub* в каталоге /sbin и «проект» будущего конфига — /boot/grub/menu.lst.sample.

Установочного скрипта пакет grub для Slackware не имеет, и дальнейшие процедуры придётся выполнить собственноручно.

Для начала логика подсказывает нам, что к установке нашего загрузчика имеет отношение команда /sbin/grub-install, которую мы и запускаем:

```
$ /sbin/grub-install /dev/sda1
```

Аргументом команды выступает имя файла целевого устройства. А об опциях команды можно узнать таким образом:

```
$ /sbin/grub-install --help
```

Из доступных опций интерес могут представлять две:

```
--root-directory=DIR
```

с помощью которой можно сохранить копию grub'а в произвольном каталоге, указанном в качестве значения опции, и

```
--grub-shell=FILE
```

задающая шелл, который будет использоваться grub'ом в качестве оболочки при

интерактивной работе. Впрочем, по умолчанию в качестве таковой применяется нечто похожее на bash, чего вполне достаточно.

Однако вернёмся к установке. В результате наших действий каталог /boot/grub заполняется серией файлов «полуторной» стадии, *_stage1_5, обеспечивающих «понимание» grub'ом соответствующих файловых систем, как нативных для Linux (e2fs_stage1_5, reiserfs_stage1_5, xfs_stage1_5, jfs_stage1_5), так и иных (ufs2_stage1_5, ffs_stage1_5, minix_stage1_5, iso9660_stage1_5). И с тех, и с других, однако, grub может напрямую загружать ядра соответствующих ОС, не прибегая к «загрузке по цепочке», вынужденно используемой им при загрузке Windows.

Завершающий штрих установки grub'а — запуск его самого на предмет перезаписи загрузочного сектора:

```
$ /sbin/grub
```

После этого перед нами появляется приглашение командной строки его bash-подобного шелла:

```
grub>
```

где можно вводить нужные команды из числа встроенных (полный их список выдаётся по нажатию табулятора при пустой строке) с необходимыми агрументами — именами дисковых устройств и их разделов, причем в обоих случаях работает автодополнение и предложение альтернатив (посредством того же табулятора).

При задании аргументов команд необходимо помнить об особенностях номенклатуры устройств в grub'е и синтаксисе его шелла:

- все диски, вне зависимости от типа интерфейса, именуются hd (дискеты — fd, компакты — cd);
- нумерация дисков начинается с нуля — hd0 соответствует /dev/sda в номенклатуре Linux'а и так далее;
- с нуля же начинается и нумерация первичных разделов — то есть (hd0,0), (hd0,1), (hd0,1) и (hd0,3) соответствуют Linux'овым /dev/sda[1-4];
- как и в Linux'е, за логическими разделами в разделе расширенном закреплены номера, начиная за таковым последнего возможного первичного раздела, вне зависимости от того, все ли они размечены на самом деле, то есть (hd0,4) — это первый логический раздел (соответствует /dev/sda5) и так далее;
- номера дисков и разделов, как можно видеть из приведённых выше примеров, разделяются запятой (без пробела), а полное имя устройства заключается в круглые скобки, например: (hd0) — первый диск целиком, (hd0,0) — первый первичный раздел на нём и так далее.

Вся эта премудрость понадобится нам потом, при составлении конфигурационного файла grub'а и, в особенности, при интерактивной работе с ним, неизбежной при всякого рода экспериментах. Пока же нам достаточно указать имена корневого устройсва для grub и того устройства, в загрузочный сектор которого будет помещен инициирующий код. Первое делается командой

```
grub> root (hd0,0)
```

Аргумент в примере дан, исходя из предположения, что под каталог /boot отведен 1-й раздел 1-го диска. А команда

```
grub> setup (hd0)
```

запишет инициирующий код в MBR первого винчестера.

Всё, теперь командой

```
grub> quit
```

можно выйти из grub-окружения. И перезагрузиться? Нет, торопиться не надо, как говорил товарищ Саахов. Потому что установить-то grub мы установили, но еще не настроили его должным образом.

Это делается редактированием конфига — простого текстового файла /boot/grub/menu.lst. Ранее я уже упоминал, что прототип его в виде /boot/grub/menu.lst.sample у нас имеется. Так что берём его за основу

```
$ cp /boot/grub/menu.lst.sample /boot/grub/menu.lst
```

и результат копирования, то есть собственно конфиг menu.lst, загружаем в любимый текстовый редактор, например:

```
$ nano -w /boot/grub/menu.lst
```

Отступление: *в некоторых дистрибутивах (например, в Gentoo) конфигурационный файл grub'а так и называется — grub.conf. Могут различаться также детали внутреннего устройства, большая или меньшая полнота комментариев и приводимых примеров и тому подобные мелочи. Мы будем придерживаться того, как это подаётся в Slackware и модифицируется в Zenwalk'е, но большинство сказанного ниже имеет силу для любых систем, использующих grub.*Прототип нашего будущего конфига, в полном соответствие с заветами Великого Патрика, хорошо прокомментирован (комментарии предваряются привычным символом #), так что разобраться в нём будет нетрудно. Для тех, кто совсем не разумеет вражьей мовы, остановлюсь на самых существенных моментах.

Конфиг нашего загрузчика неявным образом разделяется на две секции. Первая из них описывает общие настройки grub'а, и потому мантайнер пакета, некто Acrux, резонно назвал её

```
### Global settings
```

В menu.lst других дистрибутивов она может не иметь никакого названия вообще.

Вторая же секция может распадаться на несколько субсекций, каждая из которых описывает свой вариант загрузки (не обязательно разных операционных систем).

«Глобальная» секция в нашем образцовом конфиге начинается с вещей совсем не обязательных — строки, описывающей загрузку фонового изображения

```
splashimage (hd0,0)/boot/grub/slack_nalug.xpm.gz
```

и закрытых комментариями строк, определяющих расцветку меню.

```
# foreground = FFFFFF
# background = AAAAAA
```

Очевидно, что без всего этого можно обойтись. Или — заменить своей картинкой и цветами. Картинка в формате xpm должна иметь размер, кажется, 640×480 и быть сжатой gzip'ом. Что же до цветов, то они не обязаны задаваться своими шестнадцатеричными кодами, можно использовать и соответствующие английские слова. Например, в конфигах загрузчиков многих дистрибутивов они определяются

так:

```
color cyan/blue white/blue
```

Столь же не обязательны и строки

```
## shaded text
#shade 1
```

Как это выглядит «в натуре», я проверить не удосужился.

А вот строка

```
timeout 5
```

в большинстве случаев очень желательна. Если её нет (или в качестве значения тайаута указан 0), загрузка происходит немедленно, без предоставления пользователю времени для выбора её варианта. Конечно, при единственной загружаемой системе с неизменными параметрами это оправданно, но всё-таки обычно grub используется как мультисистемный загрузчик, и потому время «на подумать» (в секундах) тут будет не лишним.

Наконец, завершающая «глобальную» секцию строка

```
default 0
```

указывает, какой из вариантов загрузки будет происходить по умолчанию, при отсутствии пользовательского выбора. По аналогии с именованием устройств легко догадаться, что значение 0 соответствует варианту, описываемому в первой субсекции второй секции — такое же умолчание используется и в том случае, если строка default вообще отсутствует.

Между первой и второй секциями Acrux поместил два блока весьма полезных комментариев. Во-первых, это таблица соответствия именования устройств в grub и в Linux:

```
## Linux         Grub
## ======================
## /dev/hda      (hd0)
## /dev/hda1     (hd0,0)
## /dev/hdb      (hd1)
## /dev/hdb1     (hd1,0)
## /dev/fd0      (fd0)
```

То есть примерно то, о чем говорилось выше.

Вторая же вставка — таблица кодов, соответствующих различным режимам фреймбуфера:

```
###  FRAMEBUFFER RESOLUTION SETTINGS
##     +----------------------------+
##           | 640x480    800x600    1024x768   1280x1024
##      ---+---------------------
##       256 | 0x301=769  0x303=771  0x305=773   0x307=775
##       32K | 0x310=784  0x313=787  0x316=790   0x319=793
##       64K | 0x311=785  0x314=788  0x317=791   0x31A=794
##       16M | 0x312=786  0x315=789  0x318=792   0x31B=795
##     +----------------------------+
```

Если мы желаем в ходе загрузки любоваться неким изображением (обычно это логотип дистрибутива, в нашем случае — дельфин), а в дальнейшем иметь режим графической консоли, то один из кодов (в любой форме), соответствующих определенному разрешению и глубине цвета, нужно будет указать в качестве значения соответствующего параметра ядра, о чём будет сказано чуть ниже.

Со строки

```
## Default menu entries
```

неявным образом начинается вторая секция. В нашем примере она имеет две субсекции, описывающие варианты типовой загрузки Slackware. Первая (в соответствии с определённым ранее параметром default) выглядит так:

```
title Slackware GNU/Linux
kernel (hd0,0)/boot/vmlinuz vga=0x315 root=/dev/hda1 ro
```

Первая строка (title) — это просто имя варианта загрузки, под которым он будет фигурировать в меню grub'a. Желательно, чтобы оно было осмысленным, как в нашем примере.

Вторая строка (kernel) описывает собственно загружаемое ядро и передаваемые ему параметры. Первое значение её — абсолютный путь к файлу образа ядра (vmlinuz — одно из традиционно принятых для него имен в Linux'е), отсчитываемый от имени устройства в нотации grub'a. Очевидно, что в данном примере каталог /boot не лежит на отдельном разделе, а входит в общее дерево корневого каталога.

Далее — параметр vga, определяющий видеорежим при загрузке и, в дальнейшем, в консоли. В примере это режим фреймбуфера с разрешением 800×600 и глубиной цвета 16 млн (т.н. 24-битный цвет). Однако можно выбрать любое подходящее значение из таблицы (так, для LCD-дисплеев можно определить разрешение, соответствующее физическому разрешению матрицы), опустить этот параметр (тогда загрузка будет происходить в текстовом режиме) или задать значение ask (запрос режима перед загрузкой). Для первого раза я рекомендовал бы опустить параметр vga, ибо без него меньше шансов получить ошибку при старте системы.

Параметр root определяет устройство, несущее корневую систему для уже загруженного ядра, и потому даётся в нотации Linux: grub своё дело сделал, и его правила больше не действуют. В примере это первый раздел первого IDE-диска, то есть тот же самый (hd0,0), который выступал в качестве корневого для grub при указании пути к загружаемому ядру.

Наконец, параметр ro означает read only: в процессе загрузки корневая файловая система монтируется обычно в режиме «только для чтения»; доступной для записи она становится в ходе инициализации системы, когда происходит монтирование всех файловых систем, описанных в файле /etc/fstab с теми опциями, которые там приведены. Считается, что монтирование в режиме read only может сберечь корневую файловую систему от повреждений. Так это или нет — науке не известно, но вреда от параметра ro нет, хотя он, разумеется, не обязателен.

Вторая субсекция определена на случай пересборки ядра. Если нет уверенности, что новое ядро загрузится и будет работать нормально (а полной уверенности в этом быть не может до тех пор, пока сам не убедишься на практике), то резонно было бы сохранить возможность загрузки ядра старого, проверенного. Что и достигается нижеследующими строками:

```
title Slackware GNU/Linux (old)
kernel (hd0,0)/boot/vmlinuz.old root=/dev/hda1
```

Можно видеть, что они ничем не отличаются от аналогичных строк первой субсекции, кроме имени (дабы не забыть, что именно здесь загружается) и имени файла образа ядра, столь же традиционного, как указанное выше.

Следующие несколько субсекций, озаглавленных все вместе как

```
### Special cases
```

именно что и описывают разные специальные случаи. Впрочем, все они закомментированы, являя собой просто образец для подражания или «болванку» для модификации. Для для нас интерес будет представлять только один из этих примеров, который мы рассмотрим под занавес. А пока вернемся к первой, основной субсекции.

Как нетрудно догадаться, основываясь на параметре title, в примере рассмотрен случай с дистрибутивом Slackware. А Zenwalk, хотя и приходится ей близким родственником, отличается рядом особенностей, в частности, именно условиями загрузки.

Главное отличие заключается в том, что Zenwalk при загрузке использует так называемый initrd — виртуальный диск в оперативной памяти (Initial RAM-диск, подобнее об этом говорится в Интермедии 2.2), содержащий ядро, в котором поддерживается минимальное количество оборудования, но обеспечивается загрузка всех остальных компонентов системы. И это, разумеется, должно быть отражено в конфиге grub’a.

Кроме того, форма описания параметров загрузки, принятая Acrux’ом, далеко не самая распространённая. Практически это не важно, ибо результат достигается точно тот же, но выглядит достаточно непривычно.

Наконец, номенклатура файлов дисковых устройств во многих современных дистрибутивах (и Zenwalk тут не исключение), отличается от использованной в примере нашего конфигурационного файла: жесткие диски, вне зависимости от принадлежности к PATA- или SATA-семейству, именуются одинаково — /dev/sd?.

В соответствие с этим мы и отредактируем прототип. Точнее, я приведу просто фрагмент своего конфига для дистрибутива Zenwalk и прокомментирую его.

Итак, секция, первая,

```
### Global settings
```

Это название удачно, и я его сохранил. Хотя и сводится она у меня к двум строкам. Первая:

```
timeout 10
```

этого времени мне как раз хватает на размышления, если в планах стоит загрузить систему, отличную от умолчальной:

```
default 0
```

Если не догадались, какая же умолчальная — можно посмотреть на название книжки, или перейти ко второй секции, каковая с неё и начинается:

```
title   Zenwalk
```

Значение title, думаю, в комментариях не нуждается.

```
root    (hd0,0)
```

Этой строкой определяется устройство, несущее корневой каталог для grub — загрузочный раздел у меня выделен в самом начале диска как первичный (/dev/sda1). Как мы увидим позднее, такое самостоятельное определение корневого устройства, отдельно от пути к образу ядра в строке kernel, в ряде случаев удобнее. Хотя в иных ситуациях может оказаться предпочтительным задание полного пути, как в примере из «образцового» конфига.

Итак, в следующей строке

```
kernel  /vmlinuz vga=791 noapic nolapic root=/dev/sda6 ro quiet splash
```

определяются образ загружаемого ядра и передаваемые ему параметры. Поскольку раздел под каталог /boot мы только что определили как корневой для grub'а, абсолютный путь к файлу образа от корня же и указывается.

Обратим внимание, что vmlinuz — это символическая ссылка на реальное имя файла образа ядра:

```
$ ls -l /boot/vmlinuz
lrwxrwxrwx 1 root root      14 2008-08-07 00:07 /boot/vmlinuz -> vmlinuz-2.6.26
```

То есть при апгрейде версии ядра никаких изменений в menu.lst вносить не нужно.

Значение параметра vga задаёт разрешение 1024×768 при 16-битном цвете при загрузке. В дальнейшем этот же режим будет использоваться в консоли. Впрочем, на современных широкоформатных мониторах любые стандартные VESA-режимы передаются с сильными искажениями, а задание режимов нестандартных — это совсем отдельная история, которой здесь мы касаться не будем.

Параметры noapic и nolapic запрещают использование улучшенного контроллера прерываний (APIC) — как ввода/вывода (первый), так и локального. Без этих параметров большинство дистрибутивов на моей машине (и многих других современных) просто не грузится. Нужны ли эти параметры в общем случае — обычно определяется методом ползучего эмпиризма.

Параметр root определяет устройство корневой файловой системой для уже загруженного ядра — в нашем случае она отличается от корневого устройства загрузчика и предписывает монтирование корня в режиме read only, о чём уже говорилось ранее. Параметр quiet подавляет вывод сообщений о ходе загрузки, подменяемых, благодаря параметру splash, заготовленной splash-картинкой. Разумеется, оба эти параметра не обязательны, у меня они унаследованы от какого-то старого конфига, развлечения же со сплэшами — также тема специально для любителей.

В строке

```
initrd  /initrd.splash
```

определяется имя того самого инициирующего диска, о котором говорилось ранее. Он находится в каталоге /boot, вместе с ядром, и в Zemwalk'е по умолчанию носит имя, указанное в вышеприведенной строке. Файлам initrd, собранным для

специальных целей (например, для поддержки какого-либо экзотического оборудования), можно давать иные имена.

Отступление: *вообще, имена файлов для образа ядра и инициирующего диска в Linux'е не есть что-то предопределенное свыше; они могут быть любыми — нужно только не забывать в соответствующих местах указывать их явным образом. Не обязаны они также находиться в каталоге /boot: довольно часто и ядро, и initrd выносят непосредственно в корневой каталог, хотя обычно в виде символических ссылок. Только опять-таки пути к этим файлам должны быть указаны в абсолютной форме (то есть от корня загрузчика или, если нужно, файловой системы).*

Последняя строка нашей субсекции

```
quiet
```

предписывает «молчаливый» режим по завершении загрузки и также досталась мне по наследству — острой необходимости в ней нет.

Итак, мы описали основной вариант загрузки нашей системы. По образу и подобию его можно создать сколько угодно субсекций для дополнительных вариантов — с различными сборками образов ядра и (или) initrd. Нужно только заботиться об уникальности имен файлов и тех, и других, а также их соответствии в реальности и в описании конфига. Уникальными желательно быть и значениям строки title.

А вот строка root в дополнительных субсекциях в этом случае не нужна — корневое устройство для загрузчика мы определили в первой субсекции, и определение это будет сквозным, до его переопределения явным образом. Так что каждая из дополнительных субсекций сведется к обязательным строкам примерно такого вида:

```
title   Zenwalk Old
kernel  /vmlinuz.old noapic nolapic root=/dev/sda6
initrd  /initrd.old
```

Однако grub позволяет загружать не только разные варианты одного дистрибутива, но и разные дистрибутивы, каждый из которых лежит, разумеется, на собственном разделе (разделах), что требует переопределения корневого устройства для загрузчика.

Как это делается — проще всего рассмотреть на примере. Одной из систем, установленных у меня на машине, является Xubuntu, которая занимает третий логический раздел (/dev/sda7). Вот как выглядит субсекция, обеспечивающая её загрузку:

```
title   Xubuntu
root    (hd0,6)
kernel  /boot/vmlinuz-2.6.24-19-generic vga=791 noapic nolapic root=/dev/sda7 ro
quiet
initrd  /boot/initrd.img-2.6.24-19-generic
quiet
```

То есть:

- для начала мы идентифицируем систему, дабы знать, что загружает соответствующий пункт меню grub'а,
- затем переопределяем корневой раздел для загрузчика — (hd0,6) == /dev/sda7,
- потом, исходя из этого, указываем путь к соответствующему образу ядра —

каталог /boot теперь является составной частью будущей корневой файловой системы Xubuntu,
- задаём ядру необходимые параметры, в том числе, и новый корень файловой системы, совпадающий в этом случае с корнеым устройством загрузчика — /dev/sda7,
- и, наконец, прописываем имя файла инициирующего диска initrd.

Ничего неожиданного, не так ли? Разве что имена файлов образа ядра и инициирующего диска заданы полностью, с указанием версии и субверсии: в дистрибутивах семейства Ubuntu в пределах жизненного цикла одного релиза смена версии ядра, как правило, не происходит. Однако, поскольку корневое устройство загрузчика в этой субсекции уже иное, ничто не мешает нам создать символические ссылки вроде

```
/boot/vmlinuz -> /boot/vmlinuz-2.6.24-19-generic
/boot/initrd -> /boot/initrd.img-2.6.24-19
```

дабы избавиться от лишней писанины.

И, наконец, вспомним, что grub яыляется истинно мультисистемным загрузчиком и способен грузить не только Linux, но и иные системы. Делать он это может двояким образом.

Для операционок, файловую систему которых он понимает (к их числу принадлежат, в частности, BSD их FFS и UFS2), grub обеспечивает старт примерно тем же методом, что и для любого Linux'а — то есть непосредственной загрузкой образа ядра или некоей программы, способной эту загрузку выполнить. Например, для FreeBSD (и DragonFlyBSD) это делается примерно так:

```
title  FreeBSD
root   (hd0,2)
kernel (hd0,2)/boot/loader
```

С титулом здесь всё понятно. Корневое устройство для загрузчика — это BSD-слайс (первичный раздел в терминологии DOS/Windows и Linux, в данном случае третий), разбитый на логические разделы в соответствии с правилами BSD-разметки, а /boot/loader — один из комопнентов загрузчика BSD Loader, который и выполнит всю остальную работу (подробности обо всех этих материях можно прочитать здесь).

Всё это, конечно, очень благородно, скажите вы мне вслед за доном Тамэо, но как там на счёт баб? То есть, пардон, загрузки операционок, файловую систему которых grub не понимает и понимать в принципе не способен из-за их закрытости? А ведь именно загрузка одной из таких операционок представляет наибольший интерес для начинающего пользователя — вы, конечно, уже догадались, о чём идет речь…

На такой случай разработчики grub'а припасли второй метода — загрузку «по цепочке», то есть просто передачу управления на загрузочный сектор раздела, несущего соответствующую операционную систему. Каковую можно и назвать — мы ведь не боимся поминать чёрта к ночи, а Windows — по утру.

Чтобы понять, как это делается, вернёмся к рассмотрению нашего «образцово-показательного» примера — файлу /boot/grub/menu.lst.sample. Ближе к его концу можно обнаружить такую субсекцию:

```
#title Windows
#root (hd0,5)
#makeactive
```

```
#chainloader +1
```

Снимаем знаки комментария со всех этих строк, вместо устройства (hd0,5), при необходимости, указываем то, которое в вашей системе несёт Windows (простите — на этот раз именно вашей, у меня таковой не имеется), и — вуаля! — загрузка Windows вам обеспечена.

Правда, задать корневое устройство для загрузчика более корректно следует так:

```
rootnoverify (hd?,#)
```

где ? и # — номера диска и раздела соответственно. А параметр rootnoverify не просто назначает загрузочное устройство, но и запрещает проверку его файловой системы, резонно полагая, что проверять заведомо непонимаемое смысла не имеет.

Далее строка

```
makeactive
```

делает загрузочное устройство активным, а

```
chainloader +1
```

«по цепочке» передаёт на него (точнее, на его загрузочный сектор) управление. Дальнейшее — забота Партии, то есть штатного загрузчика Windows, о котором вы наверняка знаете больше меня.

Остаётся только добавить, что загрузка по «цепочке» применима не только к чуждым, но и к вполне соплеменным системам, в отношении которых к ней подчас приходится прибегать. В частности, такая ситуация сложилась, когда во FreeBSD как умолчальная была принята файловая система UFS2, а grub еще не научился её понимать. Как это делалось — ныне представляет только исторический интерес, однако общий принцип «цепочечной» загрузки, думаю, ясен — запомним его на предмет возможного повторения аналогичных ситуаций в будущем:

- определение корневого устройства загрузчика без проверки его файловой системы,
- активизация оного, и
- передача управления на его загрузочный сектор.

Таким образом, grub не мытьём, так катаньем способен загрузить практически любую операционную систему — существующую, былую и грядущую. Правда, при одном необходимом и достаточном условии — что его конфигурационный файл составлен безошибочно.

Однако, во-первых, все мы люди, все мы человеки — и никто из нас не застрахован, скажем, от банальных «ачипяток» (а уж автор этих строк — меньше всех). А во-вторых, как в случае с файловой системой UFS2, ошибки могут быть вызваны и (почти) объективными причинами. И в том, и в другом случае результатом будет не загрузка выбранной системы, а сообщение об ошибке. Что делать?

И тут на помощь придёт режим редактирования — после неудачной загрузки и возврата в главное меню выбранный его пункт можно на лету видоизменить. Для этого нажимаем на клавишу с символом **e** (от edit) и видим содержимое субсекции, соответствующей данному пункту, а также подсказку по доступным командам, которым соответствуют односимвольные горячие клавиши:

- **e** — редактирование текущей линии,
- **o** — добавление строки после текущей,
- **O** — добавление строки перед текущей,
- **d** — удаление текущей строки,
- **D** — удаление предыдущей строки,
- **c** — переход в режим командной строки, о котором речь пойдёт позже.

Перемещаясь стрелками управления курсором, повторным нажатием **e** вызываем для редактирования подозрительную строку, вносим те изменения, которые кажутся нам правильными, нажатием клавиши Enter фиксируем эти изменения, после чего нажатием клаивши **b** (от boot) пытаемся загрузить систему.

Если загрузка проходит успешно — всё хорошо: немедленно, пока не забыли, вносим соответствующие коррективы в /boot/grub/menu.lst и спокойно живём дальше. Если же опять следует сообщение об ошибке, возвращаемся в режим редактирования — и так до победного конца.

В процессе редактирования можно пользоваться автодополнением и выводом альтернатив, для чего служит клавиша табулятора, действующая точно так же, как и в большинстве командных оболочек. То есть, находясь в пустрой строке

```
grub> root (
```

по нажатию табулятора мы получим

```
grub> root (hd
 Possible disks are:  hd0 hd1
```

Дополнив строку номером раздела, получим

```
grub> root (hd0,
 Possible partitions are:
   Partition num: 0,  Filesystem type is ext2fs, partition type 0x83
   Partition num: 3,  Filesystem type is ext2fs, partition type 0x83
   Partition num: 4,  Filesystem type unknown, partition type 0x82
   Partition num: 5,  Filesystem type is ext2fs, partition type 0x83
   Partition num: 6,  Filesystem type is ext2fs, partition type 0x83
```

И так далее. Это же применимо и к именам файлов образа ядра и инциирующего диска. Только напоминаю, что сам grub никаких изменений в собственный конфиг не вносит: это предстоит сделать руками после успешной загрузки.

Посредством grub’а можно загрузить систему, которой не соответствует никакая подсекция в menu.lst, и даже если этот файл пуст или вообще не существует. Для этого нажатием клавиши **c** надо перейти в режим командной строки и последовательно ввести все необходимые для загрузки команды и их аргументы:

- root с указанием корневого раздела,
- kernel с именем файла образа и необходимыми аргументами,
- initrd, если он используется в данной системе.

Или же, если с определением аргументов указанных команд существуют неясности, можно попробовать интерактивно воспроизвести «цепочечную» загрузку, то есть набрать руками содержимое соответствующей субсекции (например, по образу и подобию субсекции, загружающей Windows).

Подводя итог, можно со всей ответственностью заявить: с помощью grub’а любую

операционную систему можно загрузить всегда. Для этого потребуется лишь терпение и толика знаний. Если таковых недостанет после прочтения настоящей интермедии, обратитесь к документации. Правда, на этот раз я пошлю заинтересованного читателя не к man-странице. Ибо всё, на что способен man grub — это, после перечисления опций вызова команды, послать ещё дальше. К счастью, не на страничку Лео Каганова, а всего только к документации в формате info:

```
The  full documentation for grub is maintained as a Texinfo manual.
```

Правда, с чтением info-страниц могут возникнуть затруднения. Некоторые особо аскетические дистрибутивы, такие как CRUX, просто не имеют их в своём составе. В Zenwalk’е info-страницы есть, и для grub таковая появляется после установки соответствующего пакета. Однако саму команду info искать бесполезно — по умолчанию она отсутствует. И для её появления следует установить пакет texinfo:

```
$ netpkg texinfo
```

после чего можно будет вволю насладиться всеми достижениями GNU’той мысли. Если же читателю этих строк, подобно их автору, слабо пробираться сквозь дебри info-страниц, то Google в помощь: документации по grub в Сети вдоволь, в том числе, как уже было сказано, и на русском языке.