

Глава 17. Обновление ядра

17.1. Что такое ядро и когда его надо менять

Каждый, кто хоть немного интересовался тем, что такое Linux, обязательно встречал в различных руководствах термин "ядро", по-английски — *kernel*. Ядро — это важнейшая часть Linux, как и любой другой операционной системы, поскольку именно ядро обеспечивает взаимодействие с аппаратной частью компьютера, распределение ресурсов, управление процессами и многое другое. Когда вы загружаете какое-то приложение с жесткого диска в оперативную память, или переключаетесь между уже работающими приложениями, или когда какое-то приложение записывает информацию в файл на диске, операционная система или активное приложение должно запросить доступ к той части аппаратуры, которая ему необходима. Ядро обеспечивает исполнение таких запросов других частей операционной системы и приложений, а также распределяет память между запускаемыми приложениями. Ядро, таким образом, является посредником между аппаратным и программным обеспечением компьютера, обеспечивающим их взаимодействие.

Работа по совершенствованию ядра Linux ведется международным сообществом разработчиков постоянно, и регулярно появляются новые версии ядра. Естественно, что пользователи хотят иметь последнюю (или, по крайней мере, одну из последних) версий ядра ОС и рано или поздно вы приходите к выводу о том, что пора обновить ядро.

Можно задать вопрос: "В каких случаях это необходимо?". Действительно, если система неплохо работает со старым ядром, то стоит ли заниматься его обновлением? Основными причинами, приводящими к выводу о необходимости обновления ядра, являются:

- обновление аппаратуры компьютера, подключение новых устройств, которые не поддерживаются старым ядром;

- необходимость работы с новыми программами, которые рассчитаны на новую версию ядра и отказываются работать с версией, установленной у вас;

- обнаружение каких-то ошибок в старой версии ядра, в частности таких, которые представляют угрозы с точки зрения безопасности;

- желание повысить производительность системы, используя более совершенную версию ядра, либо оптимизировать ядро для работы с конкретным набором аппаратных средств, имеющихся на вашем компьютере;

- и, наконец, простое любопытство и желание работать с последней версией системы.

Обновить ядро можно двумя способами: установкой готового бинарного образа нового ядра из *rpm*-пакета и компиляцией ядра из исходных текстов. Первый способ проще, но надо иметь в виду, что скомпилированное где-то и кем-то ядро скорее всего не является оптимальным вариантом для вашей системы. Поэтому приходится применять второй способ — компиляцию ядра из исходных кодов. Для начинающих пользователей Linux компиляция ядра из исходных кодов кажется чем-то супер-сложным и недоступным. Однако я думаю, что, прочитав настоящую главу, вы убедитесь, что это не намного сложнее, чем установка ПО из *rpm*-пакета.

17.2. Нумерация версий ядра

Прежде, чем браться за обновление ядра вы должны четко представлять себе, что за версию вы собираетесь установить. В первую очередь необходимо иметь в виду, что разработчики ядра поддерживают две ветки ядра: стабильную и экспериментальную. Все новшества, вносимые в ядро, вначале появляются в экспериментальных версиях. И только после того, как сообщество разработчиков и добровольных тестировщиков опробует эти новшества, они переносятся в так называемую стабильную версию.

Версии ядра принято нумеровать тремя цифрами, разделенными точками, например, 2.4.8, при этом четная вторая цифра в номере ядра обозначает стабильные версии ядра, а нечетная — экспериментальные версии.

Так что принимая решение об установке новой версии ядра вы должны продумать ответ на вопрос, хотите ли вы участвовать в выявлении возможных ошибок в нестабильной версии или предпочитаете работать с уже оттестированным ядром.

Как заявил Линус Торвалдс в одном из своих интервью, он предпочитает как раз заниматься экспериментальной веткой, разрабатывать код, работающий с новыми устройствами. Основным координатором разработки стабильной ветки является в настоящее время Алан Кокс, регулярно выпускающий обновленные версии или "заплатки" к стабильным версиям ядра.¹

¹ Во время подготовки книги к изданию появилось сообщение, что Алан Кокс передал функции координации работ по стабильной ветке ядра Марчело Тосатти (Marcello Tosatti).

17.3. Установка нового ядра из *rpm*-пакета

Честно сказать, я довольно долгое время не решался браться за обновление ядра, поскольку первая из предпринятых мною попыток оказалась неудачной, причем до того неудачной, что мне пришлось полностью

переустановить систему. Я тогда пытался установить ядро из исходных текстов. Но однажды я наткнулся в новостях на сообщение о том, что выпущен грм-пакет с ядром 2.2.16-1. Поскольку мой опыт работы с грм-пакетами был вполне положительным, я решился попытаться еще раз, и попытка эта оказалась вполне успешной!

Итак, вначале рассмотрим установку нового ядра, откомпилированного кем-то и представленного в виде грм-пакета. Естественно, что первым делом надо скачать грм-пакет с новым ядром. Если вы не ставите своей целью тестирование новшеств в ядре, то ищите грм-пакет со стабильной версией, т. е. с четной второй цифрой в номере версии ядра (номер версии указывается в названии пакета). Я скачал ядро версии 2.2.16-1 с сервера <http://rufus.w3.org/linux/RPM/>.

Скачав ядро, запустите команду

```
[root]# rpm -i kernel-2.2.16-1.i386.rpm
```

По этой команде программа rpm установит в каталог /boot четыре файла: System.map-x.y.z-a, vmlinuz-x.y.z-a, vmlinuz-x.y.z-a и module-info-x.y.z-a (где x.y.z-a — это номер версии нового ядра), создаст каталог /lib/modules/x.y.z-a, в котором разместит модули нового ядра, а также установит скрипт /sbin/installkernel.

Если у вас есть такое желание, вы можете предварительно (или потом) выполнить команду

```
[root]# rpm -qpl kernel-2.2.16-1.i386.rpm,
```

которая выдаст полный список устанавливаемых из пакета файлов. Выполнение этих команд еще ничего не меняет в системе, только подготавливает файлы для запуска нового ядра.

Я после установки пакета с ядром решил сразу запустить скрипт /sbin/installkernel. Скрипт завис, его пришлось снимать комбинацией клавиш <Ctrl>+<C>. Тогда я стал разбираться с тем, что делает этот скрипт.

Вот его полный текст:

```
#----- начало скрипта -----
#! /bin/sh
# /sbin/installkernel - written by tyson@rwii.com
#
INSTALL_PATH=/boot
KERNEL_VERSION=$1
BOOTIMAGE=$2
MAPFILE=$3
if [ -f $INSTALL_PATH/vmlinuz-$KERNEL_VERSION ]; then
mv $INSTALL_PATH/vmlinuz-$KERNEL_VERSION \
$INSTALL_PATH/vmlinuz.old;
fi
if [ -f $INSTALL_PATH/System.map-$KERNEL_VERSION ]; then
mv $INSTALL_PATH/System.map-$KERNEL_VERSION \
$INSTALL_PATH/System.map.old;
fi
cat $BOOTIMAGE > $INSTALL_PATH/vmlinuz-$KERNEL_VERSION
cp $MAPFILE $INSTALL_PATH/System.map-$KERNEL_VERSION
ln -fs vmlinuz-$KERNEL_VERSION $INSTALL_PATH/vmlinuz
ln -fs System.map-$KERNEL_VERSION $INSTALL_PATH/System.map
if [ -x /sbin/lilo ]; then /sbin/lilo; else /etc/lilo/install; fi
#----- конец скрипта -----
```

Как видите, скрипту требуются некоторые параметры, а я запускал его без аргументов. Однако, поскольку программа rpm уже разместила в /boot файлы vmlinuz-2.2.16-1, vmlinuz-2.2.16-1 и System.map-2.2.16-1, а также создала ссылки с именами vmlinuz и System.map, я решил, что осталось только перезапустить lilo. Но прежде чем запускать lilo, я отредактировал файл /etc/lilo.conf, добавив туда секцию с указанием на ядро 2.2.16 (просто скопировал секцию с меткой linux, после чего подправил строки image и label). Причем новую секцию поставил первой. Вот что у меня получилось (добавленные мной строки выделены жирным шрифтом):

```
#----- начало файла /etc/lilo.conf -----
boot=/dev/hdb1
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
image=/boot/vmlinuz-2.2.16-1
label=linux-2.2.16
root=/dev/hdb1
read-only
image=/boot/vmlinuz-2.2.11-4bc
label=linux
root=/dev/hdb1
read-only
other=/dev/hda1
label=dos
table=/dev/hda
#----- конец файла /etc/lilo.conf -----
```

Обратите внимание на то, что в данном случае Linux грузится со второго диска, поскольку на первом диске стоит Windows NT, и в качестве загрузчика используется NT Loader.

После этого я запустил команду `/sbin/lilo`, причем вначале с параметрами `-t -v`, чтобы посмотреть, что она будет делать, а только потом уже — без параметров, для реального исполнения. Когда команда отработала, я перезагрузился, однако на этапе загрузки система зависла, выведя строку `LIL-`. Пришлось воспользоваться загрузочной дискетой (заготовленной еще при установке системы) и выполнить команду `[root]# dd if=/dev/hdb1 of=/mnt/hda1/bootsect.lnx bs=512 count=1`. Все правильно, команда `lilo` меняет загрузочный сектор, а я забыл "подсунуть" загрузчику его обновленный вариант.

Еще раз перезагружаюсь, и с радостью вижу сообщение о том, что загрузилось ядро 2.2.16-1.

Если у вас `LILO` является основным загрузчиком (установлен в `MBR` первого диска), то последнюю операцию (копирование загрузочного сектора в файл) выполнять, конечно, не нужно.

17.4. О компиляции нового ядра

17.4.1 Зачем вообще нужно компилировать ядро?

Как было сказано в начале данного раздела, основная функция ядра состоит в том, чтобы обеспечить взаимодействие с аппаратурой компьютера. Обслуживание некоторых составляющих аппаратного обеспечения (таких, как память, например) напрямую встроено в ядро. Для тех частей аппаратуры, которые могут быть нестандартными, имеются драйверы устройств, которые обеспечивают взаимодействие ОС с аппаратурой. Большинство пользователей компьютеров с ОС `Windows` знакомы с понятием драйвера хотя бы потому, что после установки нового оборудования они вынуждены устанавливать и программные драйверы для этого оборудования. Только после этого становится возможным использовать вновь установленную аппаратную составляющую. В терминологии, принятой в `Linux`, драйвера называются "модулями". Таким образом, поддержка аппаратных устройств может быть обеспечена двумя способами: либо путем встраивания такой поддержки в ядро, либо путем использования соответствующего модуля (драйвера).

Компании, которые выпускают дистрибутивы `Linux` (такие как `RedHat`, `Caldera`, `Debian` и т. д.) вынуждены встраивать в ядро поддержку как можно более широкого спектра устройств, потому что они не могут заранее знать, какие устройства (модели устройств) будут установлены на компьютере конкретного пользователя. Поддержка в ядре широкого спектра устройств облегчает установку и поддержку системы для покупателей, избавляя их от ненужных сложностей.

Как результат, ядро, поставляемое в составе дистрибутива, скорее всего содержит код для поддержки устройств, которых никогда не будет на Вашем конкретном компьютере. С другой стороны, если у вас есть устройство, которое не поддерживается стандартным ядром, у вас может появиться обоснованное желание построить поддержку этого устройства в ядро. Оптимизация ядра под конкретный набор аппаратных устройств ускоряет загрузку системы и экономит память.

Пользователи `Linux` имеют возможность или скомпилировать ядро с поддержкой всех устройств, имеющихся на конкретном компьютере, или скомпилировать ядро, поддерживающее минимальный набор оборудования, и загружать модули для поддержки остальных устройств. Если поддержка всего оборудования осуществляется в ядре, то такое ядро называется "монолитным". Ядро, скомпилированное таким образом, что поддержка части оборудования осуществляется с использованием модулей (драйверов), называется "модульным".

Какой тип ядра вам выбрать при компиляции? Однозначного ответа на это вопрос дать нельзя. Если вы не имеете привычки менять аппаратную конфигурацию компьютера, тогда вам лучше построить поддержку всех имеющихся компонентов в ядро. Необходимо только иметь в виду, что чем больше устройств поддерживаются непосредственно ядром, тем больше его объем. А поскольку ядро полностью загружается в оперативную память, повышаются требования к объему памяти. На медленных компьютерах из-за большого размера ядра может снизиться общая производительность. Если же вы часто меняете конфигурацию компьютера (например, у вас имеются съемные жесткие диски или другие временно подключаемые устройства), то, вероятно, имеет смысл использовать для управления ими подключаемые модули, которые загружаются в память только при необходимости (экономя тем самым системные ресурсы). Таким образом, в самом общем случае поддержка некоторой части устройств должна быть встроена в ядро, а остальные устройства должны поддерживаться за счет использования загружаемых модулей.

Кроме желания иметь ядро, оптимизированное для вашей системы, необходимость перекомпилировать ядро может быть вызвана обнаружением каких-то ошибок в старой версии ядра, в частности таких, которые представляют угрозы с точки зрения безопасности (когда еще появится `gpm`-пакет с исправленной версией ядра?).

Я был вынужден заниматься установкой ядра из исходных кодов потому, что система виртуальных машин `VMware` отказалась работать с установленным у меня ядром 2.2.16, сообщив, что эта версия ядра не поддерживает работу с `CDROM` из `VMware`, и предложив мне либо установить более позднюю версию ядра, либо вернуться к версии 2.2.15. Попытки установить новую версию ядра из `gpm`-пакетов тоже не решили проблему, потому что конфигурационный скрипт `VMware` сообщал, что ему не хватает `header`-файлов.

Установка пакетов `kernel-headers` (полностью соответствующих ядру) тоже не привела к успеху, вот и пришлось сделать попытку установить ядро из исходных текстов.

Надо сказать, что к тому времени мой опыт установки программного обеспечения для Linux из исходников был очень ограничен. Поэтому приступал я к этой процедуре только под давлением обстоятельств (очень хотелось запускать MS Office под Linux, не прибегая к перезагрузке компьютера). Приводимый ниже текст является как раз описанием того, что я тогда делал. Поскольку мой эксперимент оказался удачным, я могу со спокойной совестью утверждать: ничего такого, что оказалось бы не под силу начинающему пользователю, в компиляции ядра из исходных кодов нет.

Я экспериментировал на версии 2.2.16-22 из свежееустановленного дистрибутива ASPLinux Release Candidate 3 и устанавливал ядро версии 2.4.2. Поэтому все примеры в данной главе приводятся для случая, когда система уже работает на ядре версии 2.2.x и вы пришли к решению установить ядро версии 2.4.x.

17.4.2 Что надо знать до начала компиляции

Пожалуй, самое первое, к чему нужно быть готовым, приступая к компиляции ядра, — это то, что процедура эта длительная. Так что не рассчитывайте скомпилировать ядро "между делом", в свободную минутку. Заранее планируйте, что потратите на это несколько часов, иначе вы будете вынуждены прервать процедуру посередине.

Во-вторых, до начала компиляции ядра вам необходимо хотя бы в общих чертах представлять, какую именно конфигурацию аппаратного обеспечения будет обслуживать новое ядро, и решить, какие устройства будут обслуживаться самим ядром, а какие — модулями.

В-третьих, ядро версии 2.4.2 даже в архивированном (программой `bzip2`) виде занимает более 20 Мбайт, а после разархивации объем исходных текстов превышает 108 Мбайт. Еще столько же может потребоваться для промежуточного `tar`-архива, так что надо иметь не менее 250 Мбайт свободного места. Поэтому прежде чем приступить к компиляции, позаботьтесь о том, чтобы места на диске было достаточно.

В-четвертых, приготовьтесь отвечать на те вопросы, которые программа задаст вам в процессе конфигурирования ядра (а их около 500). Перевод всех этих вопросов, относящийся к версии 2.0.x ядра, можно найти в файле `Configure.help`, который находится по адресу

<http://nevod.perm.su/service/linux/doc/kernel/Configure.help>.

Этот файл можно использовать и для более поздних версий, поскольку в основном вопросы, задаваемые при конфигурации, одинаковы, только появляются некоторые дополнительные в связи с появлением в ядре новых возможностей. Лучше всего скачать этот файл, распечатать его и использовать этот перевод в процессе конфигурирования ядра (см. ниже шаг 3). Если вы не смогли скачать этот файл, то после развертывания исходников ядра (см. ниже) вы найдете файл `Configure.help` в подкаталоге `linux/Documentation`. Имеет смысл просмотреть его (а лучше — распечатать) до начала конфигурирования, даже если вы не очень хорошо владеете английским.

На этом предварительные предупреждения сделаны, и я перехожу к изложению пошаговых инструкций по компиляции и установке нового ядра. В дальнейшем предполагается, что все описываемые действия выполняются с правами суперпользователя `root`, и что домашним каталогом является каталог `/root`.

17.5. Семь шагов к новому ядру

17.5.1. Получение и разархивация ядра

Исходные тексты новой версии ядра можно скачать с сайта **ftp.kernel.org**. Как уже было сказано, `bzip2`-архив исходных кодов ядра версии 2.4.2 имеет объем более 20 Мбайт, так что скачать его — тоже еще проблема. Но перекачивать исходные тексты необходимо только в том случае, если вы желаете установить новую версию ядра. Если же вы просто хотите перекомпилировать существующее ядро (скажем, из-за того, что необходимо обеспечить поддержку какого-то протокола или нового оборудования), то можно обойтись и без перекачки пакета из Интернета, поскольку с дистрибутивами обычно поставляются и исходные коды (на втором диске или просто в подкаталоге `SRPM`). Кроме того, даже если вы захотели скомпилировать ядро новой версии, существует возможность сократить объем информации, которую необходимо скачивать из Интернета, но об этом мы поговорим в последнем разделе данной главы. А пока будем предполагать, что вы тем или иным способом получили полный пакет исходных кодов ядра.

Поместите архив в каталог, в котором вы имеете достаточные права, например, в ваш домашний каталог. Не используйте каталог `/usr/src/linux` для разархивации исходников! Этот каталог содержит (обычно неполный) набор заголовочных файлов (`kernel headers`), которые используются заголовочными файлами библиотек (`the library header files`). Они должны соответствовать установленным в системе библиотекам, поэтому не стоит заранее вносить путаницу в эти файлы.

Распакуйте архив командой:

```
[root]# bzip2 -d linux-2.4.XX.tar.gz | tar xvf -
```

(где "XX" надо заменить на номер версии ядра, у меня была просто 2). Если вы скачали архив, сжатый программой `gzip`, то, естественно, команда будет иметь вид:

```
[root]# gunzip linux-2.4.XX.tar.gz | tar xvf -
```

Можно также воспользоваться следующим вариантом команды:

```
[root]# tar xvzf linux-2.4.2.tar.gz
```

В результате в текущем каталоге появится новая директория linux. Сделайте ее текущей с помощью команды `cd`.

17.5.2. Обновление программного обеспечения

Если вы владеете английским, то неплохо просмотреть файл README в каталоге linux и файлы Changes и Configure.help в подкаталоге linux/Documentation. Впрочем, тем кто не владеет английским, тоже необходимо заглянуть по крайней мере в файл linux/Documentation/Changes. Дело в том, что в этом файле приведен состав программного обеспечения, необходимого для компиляции нового ядра.

В приводимой ниже табл. 17.1 показан его состав для случая ядра 2.4.2.

Таблица 17.1. ПО, необходимое для компиляции нового ядра

Программа	Версия	Как определить версию
Gnu C	2.91.66	<code>gcc --version</code>
Gnu make	3.77	<code>make --version</code>
Binutils	2.9.1.0.25	<code>ld -v</code>
util-linux	2.10o	<code>fdformat --version</code>
Modutils	2.4.2	<code>/sbin/insmod -V</code>
e2fsprogs	1.19	<code>/sbin/tune2fs</code>
Reiserfsprogs	3.x.0b	<code>reiserfsck 2>&1 grep reiserfsprogs</code>
pcmcia-cs	3.1.21	<code>cardmgr -V</code>
PPP	2.4.0	<code>pppd --version</code>
isdn4k-utils	3.1pre1	<code>isdnctrl 2>&1 grep version</code>

Приведенные в правой колонке команды позволяют произвести проверку того, что необходимый пакет имеется, и имеет соответствующую версию (более новые версии не возбраняются). Не все перечисленные в этой таблице пакеты безусловно необходимы для компиляции ядра: если в Вашей системе нет PCMCIA-карт (PC Card), например, то вам не нужен и пакет `pcmcia-cs`. Я посчитал ненужными последние 4 пакета (`reiserfs` и `pcmcia` у меня нет, а удаленный дозвон и соединение по `isdn` я не использую), а для остальных пакетов нашел на rpmfind.net последние версии и установил их. Все указанные пакеты установились из `rpm`-файлов без проблем.

Примечание

Совет для "чайников" вроде меня: используйте команду `rpm -Uhv paket_name`, а не `rpm -i paket_name`.

17.5.3. Конфигурирование будущего ядра

Следующий этап заключается в конфигурировании будущего ядра. Если вы при установке обновленных версий ПО покинули каталог linux, то вернитесь в него (например, с помощью команды `cd ~/linux`).

Примечание

В некоторых случаях перед началом конфигурирования ядра необходимо выполнить команду `make mrproper`. Необходимо это бывает тогда, когда этот каталог уже использовался для компиляции ядра и надо удалить следы былых, может быть ошибочных, действий, в частности, ранее созданные файлы с расширением `.o`.

Собственно конфигурирование выполняется командой `make config`. Конфигурация в этом случае будет производиться в текстовом режиме. Процедура конфигурации будет заключаться в том, что вы должны будете последовательно ответить на серию вопросов о том, какое значение присвоить определенному параметру. На каждый вопрос предлагается обычно несколько вариантов ответа. Допустимые варианты ответа предлагаются в виде символов, заключенных в квадратные скобки. Я думаю, что смысл символов "y" и "n" пояснять не требуется. Пояснения требуют символы "?" и "m".

Символ "?" присутствует среди возможных вариантов ответа на любой вопрос и позволяет получить подсказку (конечно, по-английски). Эти подсказки и составляют содержание файла `Configure.help`, который упоминался выше, и перевод которого (даже не в последней версии) я очень рекомендую вам иметь под рукой.

Если вы выбираете вариант "m", это означает, что драйвер соответствующего устройства будет сконфигурирован (и впоследствии скомпилирован) в виде отдельного подключаемого модуля.

Один из вариантов ответа на каждый вопрос представлен большой буквой, что означает, что он выбирается по умолчанию (когда вы не задаете явно свой вариант выбора, а просто нажимаете клавишу <Enter>).

В процессе конфигурации ядра следует иметь в виду следующее:

включение в ядро драйвера любого устройства делает его больше и, более того, добавление ненужных драйверов в ядро может привести к проблемам, когда ядро будет пытаться обратиться к несуществующему устройству;

если задать тип процессора ("Processor type") выше, чем 386, то ядро не будет работать на 386-х процессорах. Ядро обнаружит это при загрузке и откажется работать;

ядро, скомпилированное с опцией эмуляции математического сопроцессора, все равно будет использовать сопроцессор, если он имеется. Функция эмуляции никогда не будет использоваться в этом случае. Правда,

ядро в этом случае будет чуть больше, но зато будет работать на любых компьютерах, независимо от того, имеется ли на них сопроцессор; некоторые вопросы помечены указаниями "development", "experimental" или "debugging", которые указывают на то, что соответствующая функция или драйвер включены в ядро в виде эксперимента. Включение этих функций в ядро может сделать ядро не только больше, но и нанести ущерб стабильности его работы. Так что на такие вопросы лучше ответить "н", если, конечно, вы не ставите своей целью провести тестирование этих новых доработок в ядре;

если вы компилируете ядро для использования на своем персональном компьютере, то поддержка мультипроцессорной обработки (Symmetric multi-processing support) вам, скорее всего, не нужна.

Впрочем, я начал уже давать пояснения по тому, как задавать значения отдельных параметров. Но их, во-первых, очень много, так что эта глава грозит распухнуть до неприличия, а, во-вторых, самое приемлемое решение определяется конкретной конфигурацией вашего компьютера и вашими личными потребностями и пожеланиями, так что обратитесь лучше к упоминавшемуся выше файлу `Configure.help`.

В заключение данного раздела замечу, что вместо `make config` можно использовать два альтернативных варианта команды конфигурации: `make menuconfig` и `make xconfig`. Эти команды отличаются от `make config` тем, что предоставляют пользователю возможность вместо ответа на вопросы производить выбор вариантов из предлагаемых меню. Команда `make menuconfig` работает в текстовом режиме, а `make xconfig` — в графическом. На рис. 17.1 представлен вид окна, которое появляется при запуске команды `make xconfig`.

Кнопки, представленные в этом окне, отображают основные группы параметров конфигурации (кроме 4 кнопок в правом нижнем углу окна, которые служат для сохранения конфигурации или загрузки ранее сохраненных вариантов конфигурации). Щелчок по любой кнопке, соответствующей группе параметров, приводит к появлению нового окна, в котором уже можно задавать значения для конкретных параметров. Например, щелчок по кнопке **General setup** приводит к появлению окна, изображенного на рис. 17.2.

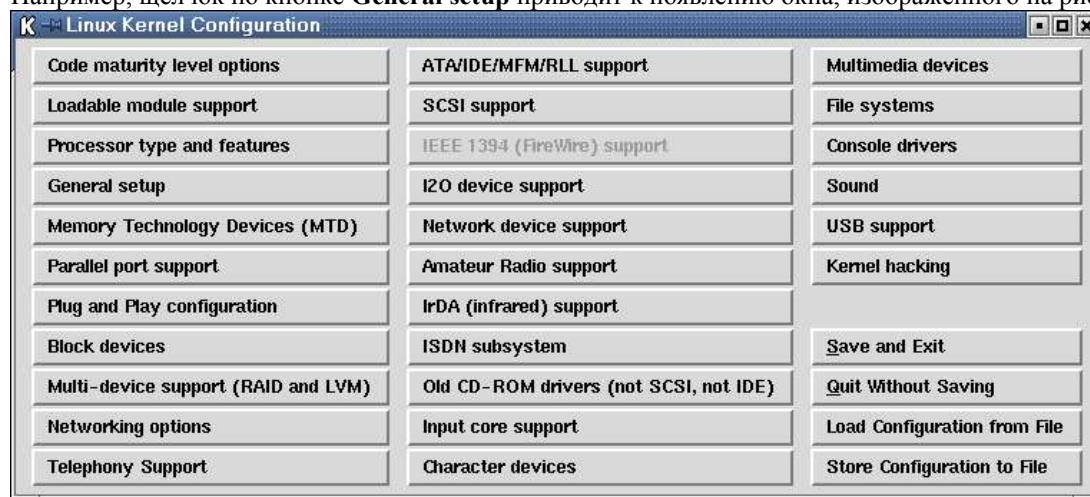


Рис. 17.1. Основное меню программы `make xconfig`

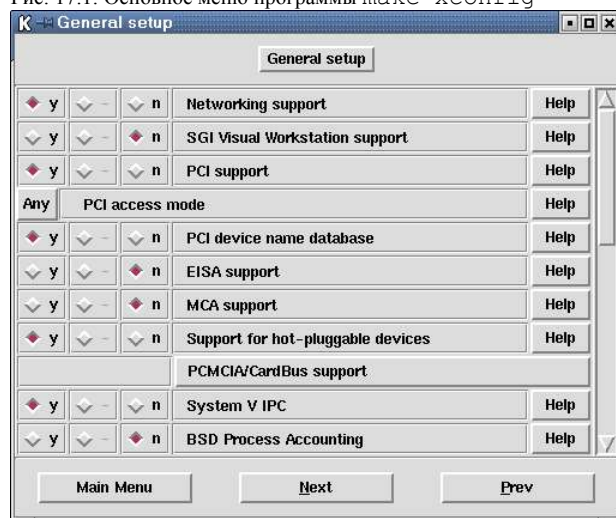


Рис. 17.2. Меню **General setup**

Как видите, здесь можно просто выбрать один из трех вариантов "y", "n" и "m" (если какой-то из вариантов недоступен, то и возможности его выбрать нет) или получить подсказку по отдельному параметру (кнопка **Help**). Поскольку число параметров в этой группе велико, справа имеется полоса прокрутки.

Впрочем, мне не кажется, что конфигурировать ядро, используя графический интерфейс, проще, чем с помощью команды `make config`. Так или иначе, надо продумать и дать ответ на вопрос о каждом параметре конфигурации. Так что, может быть, лучше уж не торопиться и последовательно отвечать на вопросы, задаваемые `make config`? А иначе (в графическом режиме, я имею в виду) возникает шанс сбиться с пути и пропустить какую-то из кнопок. С другой стороны, преимуществом графического режима является возможность вернуться к какому-то из уже пройденных этапов и изменить ранее заданные значения тех или иных параметров. Кроме того, в каждом из окон имеется кнопка **Next**, пользуясь которой можно последовательно пройти все этапы конфигурации.

В общем, представление о вариантах осуществления шага конфигурации вы получили, а решать Вам.

Отмечу только, что воспользоваться командой `make menuconfig` вы сможете только при условии, что у вас в системе установлена библиотека `ncurses` и пакет `ncurses-devel`.

Я думаю, что сказанного достаточно для того, чтобы вы смогли сконфигурировать ядро, поэтому переходим к дальнейшим шагам.

17.5.4. Проверки

Документация, поставляемая вместе с ядром, советует после завершения конфигурации выполнить два действия:

Заглянуть в файл `Makefile`, чтобы вручную поправить некоторые значения.

Дать команду `make dep` для установки зависимостей.

Конечно, для ручной правки файла `Makefile` надо понимать, для чего и как этот файл используется. Если вы этого не знаете, то можно этого и не делать, положившись на опыт и интуицию разработчиков. Ну, а вторую рекомендацию я советую выполнить. По экрану побегут непонятные сообщения, причем будет их очень много. Но если в конце не появляется сообщения об ошибке, значит, все завершилось успешно.

17.5.5. Компиляция ядра

Вот мы и добрались до основного этапа — собственно компиляции ядра. На этом этапе Ваше участие сводится только к запуску команды `make bzImage`, которая служит для создания сжатого образа ядра. Снова по экрану бегут непонятные сообщения (которые просто не успеваешь воспринять). О том, что процесс компиляции завершился без ошибок, вы можете судить по появлению сообщений примерно такого вида (естественно, я привожу те сообщения, которые были выданы в процессе компиляции ядра на моем компьютере):

```
Root device is (3,5)
Boot sector 512 bytes.
Setup is 2360 bytes.
System is 887 kB
make[1]: Выход из каталог '/opt/kernel/2.4.2/linux/arch/i386/boot'
```

И в текущем каталоге должны появиться файлы `System.map` и `vmlinux`. Кроме того, в подкаталогах каталога `linux` тоже появляется масса новых файлов (в том числе и `.o`-файлы, которые упоминались выше, как мешающие повторному проведению компиляции ядра в том же каталоге).

На этом собственно компиляция ядра и заканчивается. В дополнение приведу еще три замечания из документации к ядру.

Если в качестве загрузчика вы используете LILO, то после компиляции можно выполнить команду `make install`, однако рекомендуется вначале проверить (и подкорректировать) конфигурационный файл LILO. Хотя такое событие и очень маловероятно, но если ваша система не может загружать ядро в формате `bzImage`, вы можете скомпилировать его как `zImage`. Однако, надо иметь в виду, что поддержка `zImage` будет в ближайшем будущем отменена, поэтому разработчики предлагают тем, у кого возникнут проблемы с загрузкой ядра в формате `bzImage` сообщить об этом (приложив детальное описание конфигурации) в список рассылки `linux-kernel` и Питеру Анвину (H.Peter Anvin, hpa+linux@zytor.com).

И, наконец, если вы вдруг захотите создать загрузочный диск (без корневой файловой системы или LILO), вставьте дискету в дисковод A: и дайте команду `make bzdisk`.

17.5.6. Компиляция модулей

Если вы сконфигурировали какие-то драйверы как отдельные модули (выбирали при конфигурации вариант "m" при ответе на некоторые вопросы), то вы теперь должны еще выполнить команду `make modules`, а затем еще команду `make modules_install`. В файле `Documentation/modules.txt` можно найти дополнительную информацию по этому поводу, а также объяснение того, как использовать модули.

В завершение этапа компиляции стоит выполнить еще команду `make clean`, чтобы удалить образующиеся в процессе компиляции промежуточные объектные файлы (файлы с расширением `.o`).

17.5.7. Установка ядра

После этого остается сделать последний шаг — установить ядро и перезагрузиться. Для установки ядра вы должны иметь права суперпользователя. (Хотя в начале главы и было сказано, что для компиляции ядра надо иметь права суперпользователя, однако все предыдущие шаги можно было выполнить и от имени обычного пользователя. Не стоит работать от имени суперпользователя без необходимости!)

Разработчики рекомендуют вначале сохранить где-нибудь копию старого ядра на случай, если что-то пойдет не так, как задумано. Эта рекомендация особенно актуальна для случая, если вы ставите ядро из нестабильной ветки, поскольку такие версии могут содержать неотлаженный код. Кроме самого ядра надо сделать backup-копии модулей, соответствующих ядру. Если вы устанавливаете новое ядро с тем же самым номером версии, что и у работающего в Вашей системе ядра, сделайте резервную копию всего каталога с модулями перед выполнением команды `make modules_install`.

Для того, чтобы иметь возможность загружать новое ядро, копию образа ядра (которая после компиляции создана в виде файла `.../linux/arch/i386/boot/bzImage`) необходимо поместить туда, где у вас расположены загружаемые ядра (обычно это каталог `/boot`).

Скопируйте в каталог `/boot` три файла: файлы `System.map` и `vmlinux`, появившиеся в каталоге `...linux`, и файл `.../linux/arch/i386/boot/bzImage`. Я при копировании добавил к их именам номер версии ядра, превратив их, соответственно, в `System.map-2.4.2`, `vmlinux-2.4.2` и `vmlinux-2.4.2`, чтобы не путать с теми ядрами, которые уже были в системе ранее. Переименовывать `bzImage` в `vmlinux` в принципе не обязательно, потому что образ ядра может иметь как то, так и другое имя, и обычно располагается либо в корневом каталоге (`/`), либо в каталоге `/boot`.

Далее нам осталось только обеспечить загрузку нового ядра с помощью `lilo`. Для этого надо вначале подкорректировать файл `/etc/lilo.conf`. Ваш файл `/etc/lilo.conf` может иметь примерно такой вид:

```
boot = /dev/hda2
compact
delay = 50
root = current
image = /boot/vmlinuz-2.2.11-4bc
label = linux
read-only
other = /dev/hda1
table = /dev/hda
label = dos
```

Начиная со строки `image`, идут секции конфигурационного файла, соответствующие разным операционным системам, которые должны загружаться по выбору пользователя. В каждой такой секции имеется строка `label`. В этой строке записывается имя, которое вводится в ответ на приглашение LILO и служит для выбора пользователем загружаемой ОС.

Скопируйте секцию `image` и замените в новой секции название образа ядра и метку. Файл `/etc/lilo.conf` примет следующий вид:

```
boot = /dev/hda2
compact
delay = 50
root = current
image = /boot/vmlinuz-2.2.11-4bc
label = linux
read-only
image = /boot/vmlinuz-2.4.2
label = linux-2.4.2
read-only
other = /dev/hda1
table = /dev/hda
label = dos
```

После того, как вы откорректировали файл `/etc/lilo.conf`, необходимо выполнить команду `/sbin/lilo`, чтобы изменения вступили в силу. Эта команда (которая в руководстве называется `map-installer`) обновляет карту загрузки системы. Прежде, чем запускать `/sbin/lilo` для модификации загрузочных процедур, выполните эту команду с параметром `-t`. При этом будет выполнена вся процедура инсталляции загрузчика, кроме изменения `map`-файла и записи модифицированного загрузочного сектора, т. е. выполнен тест нового варианта. Если добавить еще опцию `-v`, это позволит убедиться в том, что сделанные Вами изменения разумны.

Не стоит полностью убирать возможность загрузки предыдущей, работающей версии ядра, тем более, что это никак не мешает загрузке нового ядра, ведь LILO обеспечивает многовариантную загрузку.

Все! После переустановки LILO командами

```
[root]# /sbin/lilo -t -v
[root]# /sbin/lilo
```

Вы можете перезагрузить компьютер и выбрать при загрузке новое ядро.

17.6. Заключение

В заключение этой главы необходимо привести несколько коротких замечаний.

Если вы уже компилировали ядро из исходных кодов, то перекачивать полный текст исходников нет необходимости, можно провести обновления ядра через так называемые патчи (patch). Но, поскольку сам я таких операций не производил, рассказать об этом способе не берусь.

В документации к ядру говорится, что команду `make config` пропускать ни в коем случае нельзя, даже если вы провели самые незначительные обновления в исходных кодах.

Если вы хотите провести перекомпиляцию установленного ядра с минимальными трудозатратами, то вместо `make config` можно воспользоваться командой `make oldconfig`, которая задаст вам только вопросы, касающиеся тех параметров, которые вы хотите изменить. Но, насколько я понимаю, ее можно использовать только в том случае, если вы уже проводили установку ядра из исходников в своей системе, поскольку эта команда использует существующий файл `./config`. В этом файле сохраняются все ответы на вопросы, которые были вам заданы при выполнении команды `make config`. При проведении повторной процедуры конфигурирования ядра командой `make oldconfig` используется тот ответ на каждый вопрос, который сохранен в файле `./config`. Если же в этом файле нет готового ответа (не задано значение соответствующего параметра), программа конфигурирования будет ждать ответа пользователя. А готового ответа не оказывается в этом файле в двух случаях:

- если устанавливается обновленная версия ядра, в которой появился новый параметр;
- если вы закомментировали или удалили строку, соответствующую данному параметру, из файла `./config`.

Поэтому, если вы хотите просто перекомпилировать ядро, чтобы, например, вынести какие-то драйверы в отдельные модули, удалите соответствующие строки из файла `./config` и воспользуйтесь командой `make oldconfig` вместо `make config`.