АЛЕКСЕЙ СТАРОВОЙТОВ

bhv®

НАСТРОЙКА АППАРАТНЫХ СРЕДСТВ в Linux

ОБЩИЕ ПРИНЦИПЫ РАБОТЫ LINUX С УСТРОЙСТВАМИ

НАСТРОЙКА ОСНОВНЫХ УСТРОЙСТВ: ВИДЕОКАРТЫ, ЖЕСТКОГО ДИСКА, ОПТИЧЕСКИХ ПРИВОДОВ

НАСТРОЙКА ПЕРИФЕРИЙНЫХ УСТРОЙСТВ: ПРИНТЕРА, СКАНЕРА, МОДЕМА

ОСНОВЫ ПРОТОКОЛА TCP/IP И НАСТРОЙКА СЕТИ

МЕТОДЫ ОПТИМИЗАЦИИ LINUX

ОСОБЫЕ ПРИЕМЫ ПРИ РАБОТЕ С LINUX

АППАРАТНЫЕ СРЕДСТВА

УДК 681.3.06
ББК 32.973.26
С77

Старовойтов А. А.

С77 Настройка аппаратных средств в Linux. — СПб.: БХВ-Петербург, 2006. — 304 с.: ил.

ISBN 5-94157-839-3

Описывается установка и настройка аппаратных средств в операционной системе Linux: видеокарт, жестких дисков, оптических приводов и других основных устройств, а также принтеров, сканеров, модемов, flash-накопителей, сетевых карт и прочей периферии. При этом рассматриваются 2 варианта настройки — с использованием конфигурационных файлов и программ с графическим интерфейсом. Даются специальные приемы по работе с Linux: оптимизация операционной системы, восстановления после сбоя, диагностика устройств. Описание производится на примере дистрибутивов Mandrake и ASPLinux.

Для пользователей Linux

УДК 681.3.06
ББК 32.973.26

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Елена Кашлакова*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.05.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,51.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-839-3

Оглавление

Глава 1

Как Linux работает с устройствами

1.1. Ядро Linux

Ядро любой операционной системы представляет собой наиболее важную ее часть, относится это и к операционной системе Linux. В операционной системе ядро выполняет функции проводника между интерфейсом высокого уровня и аппаратными средствами персонального компьютера. То есть любое действие пользователя, будь то нажатие клавиши на клавиатуре или движение мышью, вначале обрабатывается ядром операционной системы, а уже затем поступает на более высокий уровень. На низком уровне ядро операционной системы выполняет такие функции:

- организация многозадачности, то есть распределение временных ресурсов процессора, и адресного пространства для различных задач;
- организация файловой системы;
- организация виртуальной памяти;
- низкоуровневое взаимодействие с устройствами.

Схему взаимодействия аппаратного обеспечения, ядра операционной системы и оболочки высокого уровня можно представить в следующем виде (рис. 1.1). Перед нами многослойный "колобок". В центре этого колобка располагается аппаратная часть. Доступ к любой аппаратной части возможен только через ядро операционной системы, это центральный слой колобка. Внешнее кольцо — это пользовательское пространство. В пользовательском пространстве работают все остальные процессы, запущенные в системе.

Ядро Linux, как и сама операционная система, находится в постоянном развитии. В ядро постоянно добавляются новые функции, изменяется алгоритм исполнения старых функций. Естественно, что такие изменения могут нести в себе ошибки и, как следствие, нестабильность в работе. По мере тестирования недостатки и недочеты, приводящие к нестабильности, устраняются,

и ядро становится стабильным. Естественно, что использование нестабильного ядра в определенных случаях не желательно. Для того чтобы отличить "стабильную" версию ядра от "нестабильной", была введена специальная система нумерации. Нумерация ядер состоит из трех групп цифр, разделенных точкой. Первая цифра номера ядра представляет собой старший номер ядра. На сегодняшний день это версия 2. Вторая цифра, или младший номер ядра, говорит о том, является ядро стабильным или нет. Если вторая цифра четная: 0, 2, 4, 6, 8 — то ядро является стабильным. Если вторая цифра нечетная: 1, 3, 5, 7, 9 — то это нестабильное ядро, то есть ядро, находящееся в стадии разработки. Третья цифра является номером редакции данного "стабильного" или "нестабильного" ядра (рис. 1.2).

Рис. 1.1. Взаимодействие аппаратной части, ядра и процессов в Linux

Рис. 1.2. Нумерация ядра

В мире аппаратного обеспечения существует большое количество периферийных устройств, иногда попадаются и экзотические. Большим преимуществом Linux является открытый код ядра, что позволяет пользователям, обладающим определенной подготовкой, писать свои драйверы для этих устройств. Конечно, сегодня копаться в исходном коде ядра и писать драйверы среднестатистическому пользователю практически не приходится, но настраивать ядро и добавлять уже существующие модули должен уметь каждый уважающий себя администратор.

Зачем конфигурировать ядро? — спросите вы. Дело в том, что изначально при установке операционной системы создается конфигурация ядра, рассчитанная на выполнение широкого круга задач, в различной аппаратной среде. В созданную при установке конфигурацию попадают различные драйверы устройств, которые вы, возможно, никогда и не будете использовать, значения многих параметров поставлены в "среднее" положение. В то же время, некоторые необходимые драйверы могут отсутствовать. Чтобы добавить поддержку необходимого драйвера, недостаточно просто скопировать файлы, необходимо интегрировать код драйвера и ядро. В конце концов, не оптимально построенное ядро сказывается на производительности системы.

1.2. Драйверы устройств

Драйвер — это программа, которая организует взаимодействие "высшей" программной части операционной системы с аппаратной частью компьютера. Являясь частью ядра, драйвер выполняет функции переводчика между аппаратной частью и остальной системой. Необходимость наличия драйверов продиктована большим разнообразием аппаратной части. С помощью драйверов ядро делается как бы независимым от различного вида аппаратного обеспечения.

Поэтому, когда вы, например, купили новый принтер, вам не надо искать новую версию операционной системы, достаточно подключить к ядру новый драйвер, который идет в комплекте с вашим устройством. Драйверы должны быть для конкретной версии ядра. Недопустимо использование драйверов для других операционных систем.

Драйверы для устройств могут поставляться в двух формах:

- загружаемый модуль;
- заплатка к ядру определенной версии.

1.2.1. Загружаемый модуль

Загружаемый модуль является очень удобным способом подключения или отключения драйверов. (Помимо драйверов при помощи модульной структуры можно подключать и отключать и другие компоненты ядра.) При использовании подключаемых модулей нет необходимости менять код ядра ОС. Благодаря использованию подключаемых модулей уменьшается размер ядра. Подключать и отключать драйвер можно "на лету", то есть без перезагрузки операционной системы.

Как правило, модули хранятся в отдельном каталоге /lib/modules/x.x.x, где x.x.x представляет собой версию ядра, для которой скомпилированы данные модули. В каждом каталоге модули систематизированы по группам, о содержании каталогов можно судить по их наименованию.

О том, какие модули загружены в данный момент, можно узнать, выполнив команду `lsmod` (рис. 1.3).

```
[root@localhost ~]# lsmod
Module                  Size  Used by
md5                     4161  1
ipv6                  237569  10
parport_pc             24577  1
lp                     11565  0
parport                42249  2 parport_pc,lp
autofs4                24261  0
sunrpc                161701  1
ds                     17221  0
yenta_socket           18753  0
pcmcia_core            60937  2 ds,yenta_socket
dm_mod                 55253  0
button                  6609  0
battery                 8645  0
ac                      4933  0
uhci_hcd               31577  0
snd_ens1371            27685  2
```

Рис. 1.3. Выполнение команды **lsmod**

Поскольку подключаемый модуль представляет собой часть ядра, подключение любого нового непроверенного модуля может вызвать нарушение нормальной работы системы, поэтому вначале рекомендуется проверить работу модуля в режиме ручной загрузки, затем уже проверенный модуль перевести в режим автоматической загрузки.

Ручная загрузка

Как правило, она используется при отладке модулей и загрузке редко используемых модулей. Для ручной загрузки и выгрузке модулей используется несколько команд. Одну из них, `lsmod`, позволяющую посмотреть загруженные модули, мы уже рассмотрели. Рассмотрим другие команды.

Команда `insmod` позволяет загрузить модуль в работающее ядро. При загрузке модуля не учитываются существующие зависимости. Команда использует следующий синтаксис:

- `#insmod /lib/modeles/2.4.19/samples.o` — загрузка модуля samples.o без передачи ему параметров;
- `#insmod /lib/modeles/2.4.19/samples.o i=12 o=20` — загрузка модуля samples.o с передачей ему параметров.

Команда `rmmod` используется для удаления загруженных модулей. Команда будет выполнена только в том случае, если число ссылок на модуль в данном случае равно нулю. На рис. 1.4 можно видеть, что выгрузить модуль `raid0` невозможно, так как он используется одним процессом, это можно видеть по фрагменту команды `lsmod`. А выгрузить модуль `lp` оказалось возможным, так как на него нет ссылок.

```
raid0                   7745  1
BusLogic               72157  0
sd_mod                 16449  0
scsi_mod              119825  2 BusLogic,sd_mod
[root@localhost ~]# rmmod raid0
ERROR: Module raid0 is in use
[root@localhost ~]# rmmod lp
[root@localhost ~]#
```

Рис. 1.4. Использование команды **rmmod**

Команда `modprobe` осуществляет проверку зависимостей модулей, просматривая файл /lib/modules/x.x.x/modules.dep, затем с использованием команды `insmod` осуществляет загрузку необходимых модулей.

Файл modules.dep имеет следующий формат (листинг 1.1).

Листинг 1.1. Файл modules.dep

```
/lib/modules/2.6.9-1.667asp/kernel/sound/soundcore.ko:
/lib/modules/2.6.9-1.667asp/kernel/sound/usb/snd-usb-lib.ko:
/lib/modules/2.6.9-1.667asp/kernel/sound/core/snd-rawmidi.ko
```

```
/lib/modules/2.6.9-1.667asp/kernel/sound/core/seq/snd-seq-device.ko
/lib/modules/2.6.9-1.667asp/kernel/sound/core/snd.ko /lib/modules/2.6.9-
1.667asp/kernel/sound/soundcore.ko
```

В приведенном примере модуль

```
/lib/modules/2.6.9-1.667asp/kernel/sound/soundcore.ko:
```

не зависит от других модулей. А модуль

```
/lib/modules/2.6.9-1.667asp/kernel/sound/usb/snd-usb-lib.ko
```

зависит от четырех модулей. Реально в файле каждая строка относится к отдельному модулю. В нашем случае строка получилась длинной, поэтому имеет место перенос строк. `modprobe` имеет следующий синтаксис:

```
modprobe [-adnqv] [-C config] module [symbol=value ...]
modprobe [-adnqv] [-C config] [-t type] pattern
modprobe -l [-C config] [-t type] pattern
modprobe -c [-C config]
modprobe -r [-dnv] [-C config] [module ...]
modprobe -Vh
```

Ключи команды могут быть следующими:

- `-C` — позволяет использовать альтернативный файл конфигурации вместо /etc/modules.conf;
- `-v` — позволяет получить подробный отчет о процессах, происходящих во время загрузки модулей;
- `-c` — просмотр текущей конфигурации модулей.

Автоматическая загрузка

Для проверенных модулей, используемых в работе системы, было бы нецелесообразным использовать ручную загрузку. Поэтому основным методом загрузки модулей остается автоматическая загрузка модулей. В большинстве дистрибутивов для загрузки модулей используется файл /etc/modules.conf. Однако в некоторых версиях может использоваться файл /etc/conf.modules, в некоторых клонах Red Hat может использоваться файл /etc/rc.d/rc.modules. Несмотря на такое разнообразие в файлах, в конечном итоге все дистрибутивы должны перейти на использование файла /etc/modules.conf. Сгенерировать файл /etc/modules.conf можно, выполнив команду

```
modprobe -c.
```

При таком синтаксисе сгенерированный файл выводится на экран, а чтобы его сохранить, можно ввести команду

```
modprobe —c >> /etc/ver1.conf,
```

в этом случае результаты выполнения команды перенаправятся в файл /etc/ver1.conf. Попробуем выполнить данную операцию для ASPLinux, в частности, этот файл может выглядеть примерно так, как показано на листинге 1.2.

Листинг 1.2. Динамически созданный файл /etc/modules.conf

```
# Generated by modprobe -c (2.4.18)
path[boot]=/lib/modules/boot
path[toplevel]=/lib/modules/2.4.18-19.7se
path[toplevel]=/lib/modules/2.4
path[kernel]=/lib/modules/kernel
path[fs]=/lib/modules/fs
path[net]=/lib/modules/net
path[scsi]=/lib/modules/scsi
path[block]=/lib/modules/block
path[cdrom]=/lib/modules/cdrom
path[ipv4]=/lib/modules/ipv4
. . . . .
# Prune
prune modules.dep
prune modules.generic_string
prune modules.pcimap
prune modules.isapnpmap
prune modules.usbmap
prune modules.parportmap
prune modules.ieee1394map
. . . . .
# Below
below r128 agpgart
below radeon agpgart
below mga agpgart
below i810 agpgart
# Aliases
. . . . .
alias block-major-1 rd
alias block-major-2 floppy
alias block-major-3 ide-probe-mod
alias block-major-7 loop
alias block-major-8 sd_mod
```

```
alias block-major-9 md
alias block-major-11 sr_mod
alias block-major-13 xd
. . . .
# Options
options dummy0 -o dummy0
options dummy1 -o dummy1
options sb io=0x220 irq=7 dma=1 dma16=5 mpu_io=0x330
# Commands
post-install binfmt_misc /bin/mount -t binfmt_misc none
/proc/sys/fs/binfmt_misc > /dev/null 2>&1 || :
pre-remove binfmt_misc /bin/umount /proc/sys/fs/binfmt_misc > /dev/null
2>&1 || :
post-install sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -L
pre-remove sound-slot-0 /bin/aumix-minimal -f /etc/.aumixrc -S
# Miscellaneous file and directory names
generic_stringfile=/lib/modules/2.4.18-19.7se/modules.generic_string
pcimapfile=/lib/modules/2.4.18-19.7se/modules.pcimap
isapnpmapfile=/lib/modules/2.4.18-19.7se/modules.isapnpmap
usbmapfile=/lib/modules/2.4.18-19.7se/modules.usbmap
parportmapfile=/lib/modules/2.4.18-19.7se/modules.parportmap
ieee1394mapfile=/lib/modules/2.4.18-19.7se/modules.ieee1394map
pnpbiosmapfile=/lib/modules/2.4.18-19.7se/modules.pnpbiosmap
depfile=/lib/modules/2.4.18-19.7se/modules.dep
persistdir=/var/lib/modules/persist
```

Файл разбит на несколько секций, наиболее интересны секции `path`, `alias`, `options`.

В данном случае секция `path` указывает то, где находится конкретный модуль. При необходимости данную запись можно откорректировать, например, в случае если вы используете нестандартный модуль.

Инструкция `alias` обеспечивает привязку номеров устройств к соответствующим модулям.

Инструкция `options` позволяет передать модулю параметры при загрузке. В нашем случае строка:

```
options sb io=0x220 irq=7 dma=1 dma16=5 mpu_io=0x330
```

передает адреса и прерывания звуковой карте.

Кроме этого в секции `#Commands` могут указываться также команды `pre-install`, `post-install`, `pre-remove`, `post-remove`, `install`, `remove`.

При помощи этих строк указываются команды, выполняемые в тот момент, когда модуль подключается к ядру. Момент выполнения команды определяется соответственно в моменты:

- `pre-install` — перед подключением;
- `post-install` — после подключения;
- `pre-remove` — перед удалением;
- `post-remove` — после удаления;
- `install` — во время подключения;
- `remove` — во время удаления.

1.2.2. Заплатка к ядру

Заплатка может быть поставлена в качестве инсталляционного сценария либо в качестве файла. Заплатка устанавливается только на ядро конкретной версии, поэтому, чтобы избежать ошибок, надо быть внимательным к номеру ядра.

Переходим в каталог, где хранятся исходные файлы ядра:

```
cd /usr/src/linux/
```

Затем для того, чтобы проверить корректность заплатки, необходимо воспользоваться следующей командой:

```
patch –p1 –dry-run < имя_файла_заплатки
```

Если эта команда прошла без сбоев и не появлялось строк типа `FILED`, значит, заплатку можно устанавливать:

```
patch –p1 < имя_файла_заплатки
```

1.3. Файлы устройств

С точки зрения пользователя (процесса), все устройства в Linux представляются файлами: хотите получить какую-то информацию из устройства — считываете файл, хотите вывести — наоборот, записываете в файл. Представление устройства в виде файлов облегчает распределение прав доступа к тем или иным аппаратным ресурсам.

По принятой договоренности файлы устройств хранятся в каталоге /dev. В этом каталоге при необходимости они могут группироваться по типам.

С каждым из файлов устройств связаны номера старший и младший. Старший номер определяет тип устройства, младший указывает номер устройства в группе типов. По старшему номеру определяется тип драйвера, который используется для этого устройства.

Большинство устройств являются блочными (хранение данных — дисковод флоппи-дисков, жесткий диск) или символьными (передача и прием данных — модем, мыши, порты).

Некоторые распространенные типы устройств приведены в табл. 1.1.

Таблица 1.1. *Некоторые распространенные символьные и блочные устройства*

Файл	Описание устройства
/dev/fd0	Дисковод флоппи-дисков
/dev/hda	Жесткий диск (IDE) — master на первичном канале
/dev/hdb	Жесткий диск (IDE) — slave на первичном канале
/dev/hdc	Жесткий диск (IDE) — master на втором канале
/dev/hdd	Жесткий диск (IDE) — slave на втором канале
/dev/sda	Первый диск SCSI
/dev/sdb	Второй диск SCSI
/dev/ttys0	Первый последовательный порт (com1)
/dev/ttys1	Второй последовательный порт (com2)
/dev/md0	RAID-массив номер 0
/dev/eth0	Сетевой интерфейс номер 0

Как правило, после установки системы все устройства уже созданы, однако устройство можно создать принудительно, воспользовавшись командой `mknod`. Синтаксис команды таков:

```
mknod [ПАРАМЕТР]... ИМЯ ТИП_устройства [Старший Младший]
```

В качестве параметров могут быть следующие:

- `-m`, `--mode=MODE` — устанавливает атрибуты доступа (как для `chmod`), а не стандартные `=rw`;
- `--help` — выдает эту информацию и заканчивает работу;
- `--version` — выдает информацию о версии и заканчивает работу.

Тип устройства может принимать значение:

- `b` — создать блоковый (буферизованный) файл;
- `c`, `u` — создать символьный (небуферизованный) файл;
- `p` — создать файл FIFO.

Старшие и младшие номера устройства узнать можно, введя команду:

```
ls –l ИМЯ_устройства
```

Например, для дисковода флоппи-дисков это будет выглядеть так:

```
[root@localhost ~]# ls -l /dev/fd0
brw-rw----  1 root floppy 2, 0 Авг 22  2005 /dev/fd0
```

Рис. 1.5. Старший и младший номера дисковода флоппи-дисков

Как видно из рис. 1.5, старший байт равен 2, младший равен 0, соответственно, создание такого устройства будет выполнено командой:

```
# mknod /dev/fd0 b 2 0
```

Вообще надо сказать, что, как правило, создавать устройства вручную приходится довольно редко.

1.4. Создание ядра и его компиляция

Конфигурирование ядра сводится к конфигурированию файла .config в каталоге с исходными кодами ядра. Как правило, это каталог /usr/scr/linux, но для этого у вас должны быть установлены исходные файлы ядра.

Редактировать файл .config можно тремя способами:

- `make xconfig` — этот метод предпочтителен, так как использует графический интерфейс. Соответственно, этот метод может быть использован только в том случае, если в системе инсталлирована оболочка X Window (рис. 1.6);
- `make menuconfig` — тоже достаточно наглядная среда редактирования файла .config, однако использовать ее лучше только в том случае, если нет графического интерфейса (рис. 1.7);
- в самую последнюю очередь стоит использовать утилиту config. При редактировании файла .config она наименее наглядна и удобна. В процессе своей работы эта утилита выдает запрос на изменение каждого параметра и не позволяет перемещаться по диалогу вверх и вниз (рис. 1.8).

Для того чтобы ваша команда была выполнена, необходимо перейти в каталог, где располагаются исходные коды ядра. Это, как правило, /usr/src/linux. Для того чтобы перейти в этот каталог, необходимо выполнить команду

```
cd /usr/src/linux,
```

а затем выполнить одну из трех описанных уже команд.

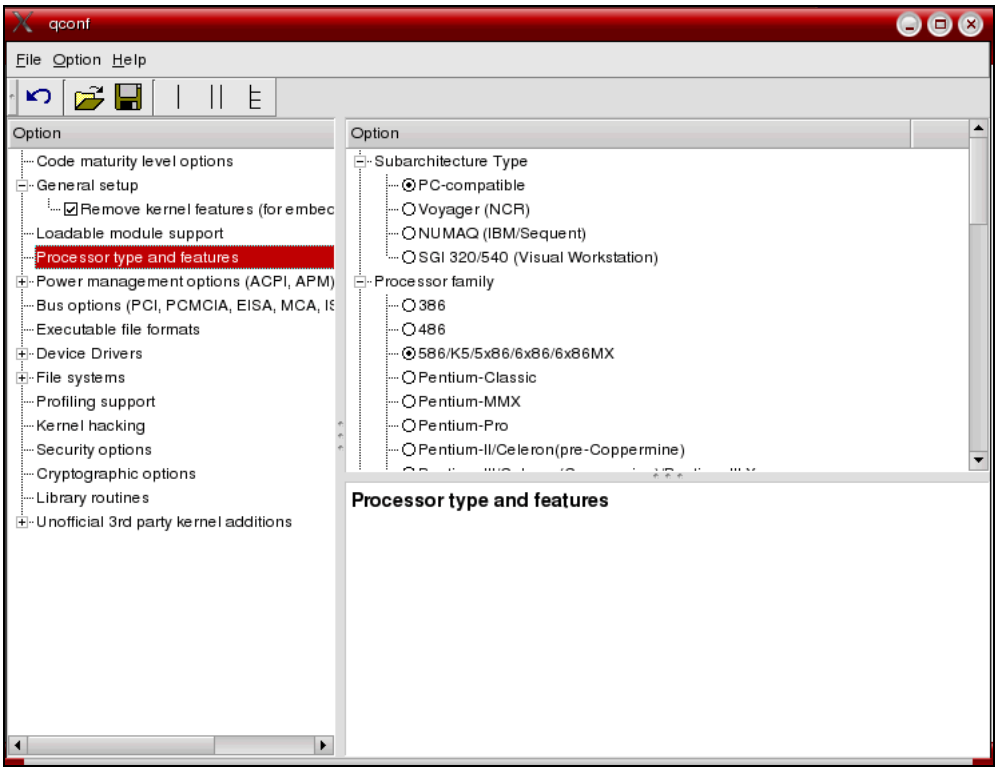

Рис. 1.6. Утилита xconfig

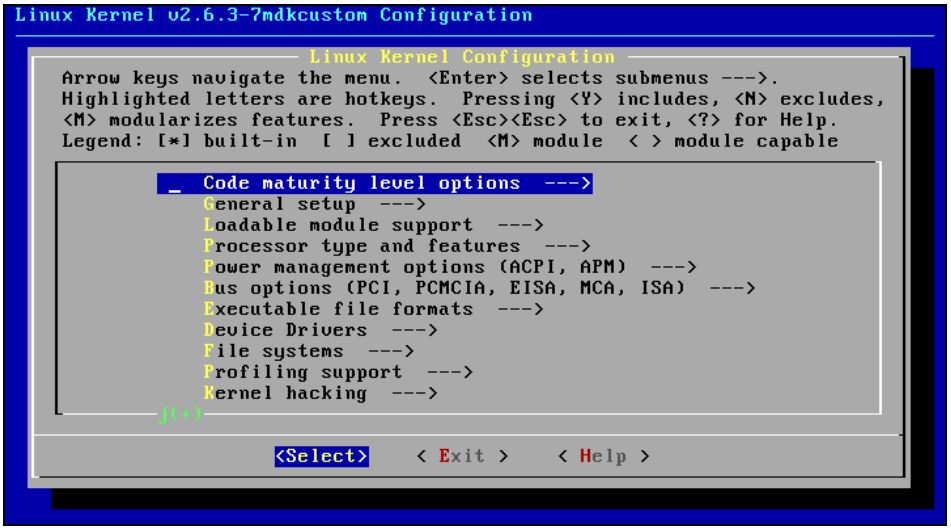

Рис. 1.7. Утилита menuconfig

```
#
# using defaults found in .config
#
*
* Linux Kernel Configuration
*
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [Y/n/?] y
  Select only drivers expected to compile cleanly (CLEAN_COMPILE) [Y/n/?] _
```

Рис. 1.8. Работа утилиты config

Во время конфигурирования ядра пользователь выбирает опции, которые в дальнейшем намерен использовать в ядре.

Если какой-то из параметров вам не знаком, вы можете использовать установки по умолчанию.

Итак, закончив работу с одной из описанных ранее программ, мы создали файл .config, но для компиляции ядра этого недостаточно, нужно выполнить еще несколько операций:

- `make dep` — при этом создаются файлы зависимостей;
- `make clean` — удаление промежуточных файлов от предыдущих операций;
- `make bzImage` — собственно создание ядра, при этом вновь созданное ядро будет располагаться в каталоге /usr/src/linux/arch/i386/boot в файле под именем bzImage;
- `make modules` — эта команда выполняет сборку модулей, соответствующих ядру, созданному командой раньше;
- `make modules_install` — перемещает созданные модули из исходного дерева ядра в каталог /lib/modules.

Теперь необходимо переместить два файла из каталога /usr/src/linux/arch/i386/boot:

- скопировать файл /usr/src/linux/arch/i386/boot/bzImage в файл /boot/bzImage_new;
- скопировать файл /usr/src/linux/arch/i386/boot/System.map в файл /boot/System.map.

Необходимо также отредактировать загрузчик, чтобы он знал, где брать новое ядро.

Поскольку загрузчик LILO, пожалуй, является основным загрузчиком для клонов Red Hat, то начнем с него. Конфигурация загрузчика LILO находится

в файле /etc/lilo.conf, в нашем случае он может выглядеть примерно как в листинге 1.3.

Листинг 1.3 Файл конфигурации загрузчика /etc/lilo.conf

```
boot=/dev/sda
map=/boot/map
default="linux-smp"
keytable=/boot/ru4.klt
prompt
nowarn
timeout=100
message=/boot/message
menu-scheme=wb:bw:wb:bw
image=/boot/vmlinuz
        label="linux"
        root=/dev/sda1
        initrd=/boot/initrd.img
        append="devfs=mount acpi=ht resume=/dev/sda5 splash=silent"
        vga=788
        read-only
image=/boot/vmlinuz
        label="linux-nonfb"
        root=/dev/sda1
        initrd=/boot/initrd.img
        append="devfs=mount acpi=ht resume=/dev/sda5"
        read-only
image=/boot/vmlinuz-2.6.3-7mdk
        label="263-7"
        root=/dev/sda1
        initrd=/boot/initrd-2.6.3-7mdk.img
        append="devfs=mount acpi=ht resume=/dev/sda5 splash=silent"
        read-only
image=/boot/vmlinuz-smp
        label="linux-smp"
        root=/dev/sda1
        initrd=/boot/initrd-smp.img
        append="devfs=mount acpi=ht resume=/dev/sda5 splash=silent"
        read-only
```

```
image=/boot/vmlinuz
        label="failsafe"
        root=/dev/sda1
        initrd=/boot/initrd.img
        append="failsafe acpi=ht resume=/dev/sda5 devfs=nomount"
        read-only
other=/dev/fd0
        label="floppy"
        unsafe
```

Для того чтобы добавить в список выбираемых при загрузке наше ядро, необходимо добавить в конец этого файла следующий фрагмент (листинг 1.4).

Листинг 1.4. Дополнительный фрагмент

```
image=/boot/bzImage_new
    # Ссылка на ядро
        label="linux-new"
    # Метка
        root=/dev/sda1
    # Расположение корневой файловой системы
        append="devfs=mount acpi=ht resume=/dev/sda5 splash=silent"
    # Через append осуществляется передача дополнительных параметров
        read-only
    # Режим монтирования корневой файловой системы
```

Теперь выполните команду

`/sbin/lilo.`

После этих манипуляций ваше новое ядро появится в конце списка выбираемых вариантов загрузки.

В ASPLinux для использования рекомендован ASPLoader. Изменить конфигурацию вариантов загрузки у ASPLoader можно, отредактировав файл /etc/aspldr.conf. По умолчанию этот файл выглядит как в листинге 1.5.

Листинг 1.5. Файл /etc/aspldr.conf

```
[asplinux1@ASPLinux(2.6.9-1.667asp)]
icon linux
kernel /boot/vmlinuz-2.6.9-1.667asp root=/dev/md0 ro rhgb
```