Linux Kernel HOWTO

Brian Ward, bri@blah.math.tu-graz.ac.at, перевод Alex Ott ott@phtd.tpu.edu.ru v0.80, 26 Мая 1997

Это детальное руководство по настройке ядра, его компиляции, обновлениям и разрешению проблем на системах построенных на базе ix86.

Содержание

Примечание переводчика: Шлите мне любый комментарии и замечания, даже небольшие.

1 Введение

Должны ли вы читать этот документ? Да, если у вас один из следующих симптомов:

- "Вах! Этот пакет wizzo-46.5.6 говорит, что ему нужно ядро 1.8.193, а я все еще пользуюсь версией 1.0.9!"
- В одном из более новых ядер существует драйвер необходимого вам устройства.
- Вы не знаете как компилировать ядро
- "В действительности ли в файле README находится все описание?"
- Вы старались, вы пытались, а это не работает
- Вам нужно что-то дать людям, которые просят вас установить ядра для них

1.1 Прочитайте это сначала! (Я это подразумеваю)

Некоторые из примеров в этом документе предполагают, что у вас есть GNU `tar`, `find`, и `xargs`. Эти программы довольно стандартны; это не должно вызвать проблем. Так же предполагается, что вы знаете структуру вашей файловой системы; если вы не знаете эьлгл, то вы должны сохранить копию вывода команды `mount` при обычных системных операциях (или содержимое файла `/etc/fstab`, если вы можете читать его). Эта информация является важной, и не изменяется до переразбивки вашего диска, добавления нового, перестановки вашей системы или чего-то подобного.
Последней "стабильной (production)" версией ядра во время написания этого документа была версия 2.0.30, это означает, что все ссылки и примеры относятся к этой версии. Даже хотя я пытался сделать этот документ как не зависящий от версии насколько это возможно, все равно ядра постоянно находятся в развитии, так что если вы получили новую версию, то она неизбежно будет иметь некоторые отличия. Вообще это не должно вызвать больших проблем, но может создать некоторые осложнения.
Существует две версии исходного кода ядра linux, "стабильная (production)" и "разрабатываемая (development)". Стабильные версии начались с 1.0.x и в настоящее время они идут с четными номерами; 1.0.x являются стабильными, 1.2.x являются стабильными, так же как и 2.0.x. Эти ядра считаются более стабильными и свободными от ошибок версиями во время их выпуска. Разрабатываемые ядра (1.1.x, 1.3.x, и т.п.) являются ядрами для тестирования, для людей желающих протестировать новые, возможно с ошибками, ядра. Я вас предупредил!

1.2 Одно слово о стиле

`Текст, который выглядит вот так` — это либо то, что появится на вашем экране, либо имя файла, либо то, что может быть прямо набрано, например команда, или опции команды (если вы читаете это как простой текст, то это различие не видно). Команды и другой ввод часто взяты в кавычки (с помощью ‘ ’), что вызывает классическую проблему пунктуации: Если такой пункт появляется в конце предложения в кавычках, то люди часто набирают ‘.’ вместе с командой, потому-что Американский стиль цитирования заставляет помещать точку внутри кавычек. Даже если здравый смысл (и к сожалению, это предполагает, что люди со "здравым смыслом" будут использовать американский стиль цитирования) говорит кому-то, что надо отбросить сначала знаки пунктуации, много людей просто этого не помнят, так что я буду в таких случаях помещать знаки пунктуации вне кавычек. Другими словами, когда надо показать, что вы должны набрать "`make config`", то я буду писать ‘`make config`’, а не ‘`make config.`’

2 Важные вопросы и ответы на них

2.1 Что вообще делает ядро?

Ядро Unix выступает как посредник между вашей программой и вашим оборудованием. Сначала оно делает (или подготавливается к) обслуживанию/распределению памяти компьютера для всех запущенных программ (процессов), и убеждается, что все они честно (или нечестно, если вы этого желаете) разделяют время процессора. В добавление к этому оно обеспечивает великолепный, довольно переносимый интерфейс для общения программ с оборудованием.
Конечно у ядро выполняет больше действий, чем мы здесь перечислили, но эти основные функции необходимо знать.

2.2 Почему я должен обновлять мое ядро?

Более новые ядра в общем поддерживают большее количество типов оборудования (они имеют больше драйверов устройств), они могут иметь улучшенное управление процессами, они могут выполняться быстрее, чем более старые версии, они могут быть более стабильными, чем старые

версии, и они исправляют глупые ошибки в более старых версиях. Большинство людей обновляют ядро, потому-что они хотят использовать новые драйвера устройств и исправить ошибки.

2.3 Какие типы оборудования поддерживают новые ядра?

Смотрите Hardware-HOWTO. В качестве альтернативы вы можете посмотреть файл '`config.in`' в исходных текстах ядра linux source, или просто найти нужное устройство запустив '`make config`'. Они показывают все оборудование поддерживаемое дистрибутивом ядра, но не все, которое поддерживает linux; много драйверов общих устройств (таких как драйвера PCMCIA и некоторые драйвера ленточных устройств) являются загружаемыми модулями сопровождаются и распространяются отдельно.

2.4 Какие версии gcc и libc мне нужны?

Рекомендации Linus по версии gcc находятся в файле `README`, включенном в исходные тексты linux. Если у вас нет этой версии, то документация по рекомендуемой версии gcc должна сообщить вам все сведения, если вам необходимо обновить вашу версию libc. Это не трудная процедура, но важно следовать инструкциям.

2.5 Что такое загружаемый модуль?

Это кусочки кода ядра, которые не включены прямо в ядро. Они компилируются отдельно и затем могут вставлять и удалять их в запущенное ядро почти в любое время. Вследствии их гибкости, сейчас это предпочтительный способ кодирования некоторых средств ядра. Много популярных драйверов устройств, таких как драйвера PCMCIA и драйвера ленточных устройств QIC-80/40, являются загружаемыми модулями.

2.6 Сколько места на диске мне надо?

Это зависит от конфигурации вашей системы. Первое, это сжатые исходные тексты ядра, примерно 6 мегабайт для версии 2.0.10. На многих машинах этот файл хранят даже после распаковки. В расжатом виде исходные тексты занимают до 24 мегабайт. Но это не конец — вам нужно больше для компиляции ядра. Требуемый размер зависит от того, как вы настроили ваше ядро. Например, на одной машине у меня настроена работа сети, драйвер 3Com 3C509, и три файловые системы, это все занимает примерно 30 мегабайт дискового пространства. Добавив сжатые исходные тексты, вам понадобиться около 26 Мб для такой конфигурации. На другой системе, без поддержки драйвера сетевой карты (но все равно с поддержкой сети) и звуковой картой, все занимает больше пространства. Также, более новые ядра имеют более большое дерево исходных текстов, так что в общем, если если у вас довольно много устройств, то убедитесь, что у вас достаточно большой жесткий диск (при сегодняшних ценах, я не могу помочь вам, но я рекомендую взять другой диск как ответ на вашу проблему отсутствия свободного пространства).

2.7 Как долго этот процесс идет?

Для большинства людей ответ будет такой: "довольно долго". Скорость вашей машины и количество имеющейся памяти определяют это время, но некоторая часть определяется, тем как вы включили в ядро. На машине 486DX4/100 с 16 МБ ОЗУ, на ядре версии 1.2 с пятью файловыми файловыми системами, поддержкой сети и драйвером звуковой карты, компиляция займет примерно 20 минут. На 386DX/40 (8 МБ ОЗУ) с примерно такой же конфигурацией, компиляция продолжается около 1.5 часов. В общем рекомендуем выпить кофе, посмотреть телевизор, повязать или поделать что-нибудь подобное пока ваша машина компилирует ядро.

3 Как настраивать ядро

3.1 Получение исходных текстов

Вы можете получить исходные тексты с помощью анонимного ftp с ftp.funet.fi в директории /pub/Linux/PEOPLE/Linus, с его зеркала, или с другого сервера. Они обычно обозначены как linux-x.y.z.tar.gz, где x.y.z номер версии. Более новые (лучшие?) версии и заплатки (patches) обычно находятся в поддиректориях, таких как 'v1.1' и 'v1.2'. Самый большой номер имеет последняя версия и обычно является "тестовой версией", это значит, что если вы нелегко плохо чувствуете себя с альфа или бета версиями, то вы должны использовать стабильную версию. Я *настоятельно* рекомендую вам использовать сервера-зеркала вместо использования ftp.funet.fi. Здесь приведен короткий список серверов-зеркал и других серверов:

```
USA:          sunsite.unc.edu:/pub/Linux/kernel
USA:          tsx-11.mit.edu:/pub/linux/sources/system
UK:           sunsite.doc.ic.ac.uk:/pub/unix/Linux/sunsite.unc-mirror/kernel
Austria:      ftp.univie.ac.at:/systems/linux/sunsite/kernel
Germany:      ftp.Germany.EU.net:/pub/os/Linux/Local.EUnet/Kernel/Linus
Germany:      sunsite.informatik.rwth-aachen.de:/pub/Linux/PEOPLE/Linus
France:       ftp.ibp.fr:/pub/linux/sources/system/patches
Australia:    sunsite.anu.edu.au:/pub/linux/kernel
```

В общем зеркало сервера sunsite.unc.edu является хорошим местом, где можно взять исходные тексты ядра. Файл /pub/Linux/MIRRORS содержит список известных серверов-зеркал. Если у вас нет доступа к ftp, то список систем BBS, которые распространяют linux периодически посылается в группу comp.os.linux.announce; постарайтесь получить его.
Если вы ищете общую информацию о Linux и его дистрибутивах, то посмотрите на http://www.linux.org.

3.2 Распаковка исходных текстов

Войдите в систему как администратор или выполните команду su, и перейдите в директорию /usr/src.Если вы устанавливали исходные тексты ядра при установке linux (как делает большинство), то том у вас уже есть директория названная 'linux', которая содержит полное дерево устаревших исходных текстов. Если у вас есть свободное дисковое пространство, то вы можете сохранить эту директорию. Хорошая идея — определить какая версия ядра запущена и соответственно переименовать директорию. Команда 'uname -r' выдает номер текущей версии ядра. Поэтому, если команда 'uname -r' выдала '1.0.9', то вы должны переименовать (с помощью 'mv') 'linux' в 'linux-1.0.9'. Если вы не чувствуете, что поступаете опрометчиво, то просто сотрите всю директорию. В любом случае убедитесь, что никакой директории 'linux' в /usr/src до распаковки полного исходного кода ядра.
Теперь распакуйте в /usr/src исходные тексты, пользуясь командой 'tar zxpvf linux-x.y.z.tar.gz' (если вы получили просто файл .tar без расширения .gz на конце, то работает команда 'tar xpvf linux-x.y.z.tar'). Содержимое архива будет распаковано. После окончания процесса, будет существовать новая директория 'linux' в /usr/src. Перейдите linux и посмотрите файл README. Там будет раздел с заголовком 'INSTALLING the kernel (Установка ядра)'. Выполните соответствующие инструкции — символические ссылки должны быть на своем месте, удалите старые .o файлы, и т.п.

3.3 Настройка ядра

Замечание: Некоторое из этого являются повторенниями/пояснениями подобного раздела файла `README` поставляемого Linus.
Команда '`make config`' выполненная в `/usr/src/linux` запускает скрипт настройки, которая задает вам много вопросов. Она требует наличия bash, так что проверьте что bash находятся в `/bin/bash`, `/bin/sh`, или `$BASH`.
Существуют некоторые альтернативы команде '`make config`' и вы можете найти их более удобными и легкими для использования. Те, кто работает в X могут попробовать '`make xconfig`', если у вас установлен Tk ('click-o-rama' - Nat). '`make menuconfig`' — это для тех, у кого установлен (n)curses и предпочитает текстовые меню. Эти интерфейсы имеют одно явное преимущество: если вы сделали неправильный выбор в течении настройки, то очень легко вернуться и исправить ее.
Теперь вы готовы отвечать на вопросы, обычно ответы выглядят как '`y`' (да) или '`n`' (нет). Драйвера устройств обычно имеют опцию '`m`'. Это означает "module (модуль)", обозначая, что система будет компилировать этот драйвер, но не вставит его прямо в ядро, а сделает загружаемым модулем. Более комично эта опция описывается как "maybe (может быть)". Некоторые более ясные и некритичные опции здесь не описаны; смотрите раздел "Другие опции настройки" для их краткого описания.
В версиях 2.0.x и более поздних, существует опция '?', которая обеспечивает краткое описание параметра настройки. Эта информация скорее всего наиболее свежая.

3.3.1 Эмуляция математических функций ядром

Если у вас нет математического сопроцессора (у вас голый 386 или 486SX), то вы должны ответить '`y`' на этот вопрос. Если у вас есть сопроцессор и вы все равно ответили '`y`', то не беспокойтесь — сопроцессор все рано будет использоваться, а эмуляция будет проигнорирована. Единственное следствие этого в том, что ядро будет больше (расход ОЗУ). Я упоминал о том, что эмуляция очень медлена; хотя это не очень часто влияет, но все равно вспомните это, если столкнетесь с малой производительностью системы X-windos.

3.3.2 Поддержка обычных (MFM/RLL) дисков и дисков/cdrom IDE

Вам вероятно необходима эта поддержка; это означает, что ядро будет поддерживать стандартные жесткие диски PC, которые имеет большинство людей. Этот драйвер не включает поддержку SCSI дисков; их выбор идет далее в настройке.
Затем у вас спросят о драйверах "old disk-only (только старых дисков)" и "new IDE (новых IDE)". Вы захотите выбрать один из них; основное отличие в том, что старые диски поддерживают только два диска на одном интерфейсе, а новые поддерживают вторичный (secondary) интерфейс и накопители IDE/ATAPI cdrom. Новый драйвер на 4k больше старого и также предположительно "улучшен", убирая некоторое количество ошибок, он может улучшить производительность вашего диска, особенно если у вас новое оборудование (типа EIDE).

3.3.3 Поддержка сети

В принципе вы должны ответить '`y`', если ваша машина подключена к сети, такой как internet, или вы хотите использовать SLIP, PPP, term и т.п. для dial up доступа к internet. Однако много пакетов (таких как оконная система X) требует поддержку сети, даже если вы не подключены ни к какой сети, вы должны сказать '`y`'. Позже у вас спросят, хотите ли вы поддержку TCP/IP; далее скажите '`y`', если вы абсолютно уверены в своем выборе.

3.3.4 Ограничить память до менее 16MB

Существуют работающие с ошибками контролеры DMA на машинах с процессором 386, которые имеют проблемы с адресацией больше 16 Мб ОЗУ; вы можете ответить '`y`' в случае (редком) если

у вас такой контроллер.

3.3.5 System V IPC

Одно из лучших определений IPC (Interprocess Communication, Межпроцессного сообщения) данов в глоссарии книги по Perl. Не удивительно, что некоторые программисты на Perl используют этот механизм чтобы позволить процессу общаться с другими процессами, так же как и другие пакеты (самый заметный из них это DOOM), так что ответ `n` не является хорошей идеей, пока вы не будете точно уверены в том, что вы делаете.

3.3.6 Тип процессора (386, 486, Pentium, PPro)

(в старых ядрах: используйте флаг -m486 для оптимизации для процессора 486)
Традиционно, это делает некоторую оптимизацию для выбранного процессора; ядра работают быстрее, но ядро может быть несколько больше. В новых ядрах, это однако больше не является правдой, так что вы должны ввести процессор для которого вы компилируете ядро. Ядро для "386" будет работать на всех машинах.

3.3.7 Поддержка SCSI

Если у вас есть устройства SCSI, то ответьте '`y`'. У вас запросят дополнительную информацию, такую как поддержку CD-ROM, дисков, и какой тип адаптера SCSI у вас имеется. Смотрите SCSI-HOWTO для дополнительной информации.

3.3.8 Поддержка сетевых устройств

Если у вас есть сетевая карта, или вы хотите использовать SLIP, PPP, или адаптер параллельного порта для подключения к Internet, то ответьте '`y`'. Скрипт настройки запросит у вас тип карты и какие протоколы вы хотите использовать.

3.3.9 Файловые системы

Затем настроечный скрипт запросит у вас поддержку для каких файловых систем вы хотите иметь в своей системе:
Стандартная (minix) - Более новые дистрибутивы не создают файловые системы minix, и много людей не используют ее, но все равно хорошая идея настроить ее. Некоторые программы с "дисками для восстановления (rescue disk)" используют ее и все еще много гибких дисков могут использовать файловую систему minix, поскольку файловая система minix менее мучительная для использования на гибких дисках.
Extended fs - это была первая версия расширенной файловой системы, которая сейчас не является широко используемой. Если вы не знаете точно, что она вам нужна и сомневаетесь, то скорее всего она вам не нужна.
Second extended - эта файловая система широко используется в новых дистрибутивах. У вас скорее всего она есть и вам нужно ответить '`y`'.
файловая система xiafs - одно время она не была необычно, но во время написания этого документа я не знал никого использующего эту файловую систему.
msdos - если вы хотите использовать разделы вашего жесткого диска с MS-DOS, или монтировать гибкие диски, отформатированные под MS-DOS, то ответьте '`y`'.
umsdos - эта файловая система расширяет возможности файловой системы MS-DOS обычными Unix-подобными возможностями, такими как длинные имена. Это не является полезным для людей (таких как я), кто "не работает в DOS."
/proc - одна из величайших вещей со времен изобретения порошкового молока (я так предполагаю, идея была бесстыдно украдена у Bell Labs). Она не создает файловую систему proc на диске;

она является интерфейсом в виде файловой системы к ядру и процессам. Много программ, выдающих список процессов (таких как 'ps') используют ее. Как нибудь попробуйте выполнить 'cat /proc/meminfo' или 'cat /proc/devices'. Некоторые командные процессоры (в частности rc) используют /proc/self/fd (известный как /dev/fd в других системах) для ввода/вывода. Вы должны почти всегда ответить 'y' на этот вопрос; много важных утилит для linux зависят от этого выбора.
NFS - если ваша машина работает в сети и вы хотите использовать файловые системы находящиеся на других машинах с помощью NFS, то ответьте 'y'.
ISO9660 - имеется на большинстве CD-ROM. Если у вас есть привод CD-ROM и вы хотите использовать его в Linux, то ответьте 'y'.
OS/2 HPFS - во время написания работает как файловая система только для чтения для OS/2 HPFS.
System V и Coherent - для разделов машин с System V и Coherent (это другие варианты Unix для PC).

Но я не знаю какие файловые системы мне нужны! Хорошо, наберите команду 'mount'. Ее вывод будет выглядеть примерно так:

```
blah# mount
/dev/hda1 on / type ext2 (defaults)
/dev/hda3 on /usr type ext2 (defaults)
none on /proc type proc (defaults)
/dev/fd0 on /mnt type msdos (defaults)
```

Посмотрите на каждую строку; слово идущее за словом 'type (тип)' является типом файловой системы. В моем примере мои файловые системы / и /usr являются файловыми системами типа second extended, я использую /proc, и есть гибкий диск смонтированный используя файловую систему msdos.
Вы можете попробовать выполнить 'cat /proc/filesystems', если у вас в настоящее время разрешено использование /proc; эта команда перечислит файловые системы, поддержку которых имеет ваше ядро.
Настройка редко используемых, не критических файловых систем может вызвать раздувание вашего ядра; смотрите раздел о модулях чтобы избежать этого и раздел "Ловушки" о том, почему раздувшееся ядро является нежелательным.

3.3.10 Символьные устройства

В этом разделе вы выбираете драйвера для вашего принтера (параллельного принтера), шинной мыши, мыши для PS/2 (многие notebook используют протокол мыши PS/2 для своих встроенных трекболов), некоторые ленточные накопители и другие такие же "символьные" устройства. Ответьте 'y' где необходимо.
Замечание: Selection это программа, которая позволяет вам использовать мышь вне системы X window для вырезания и вставки между виртуальными консолями. Она работает довольно хорошо, если у вас мышь для последовательного порта, потому-что она хорошо работает с X, но вам необходимо выполнить некоторые действия, для того чтобы работали другие типы мышей. Поддержка Selection одно время была опцией настройки, но сейчас она является стандартом.
Замечание 2: Сейчас Selection считается устаревшей. Имя новой программы "gpm". Она может делать более фантастические вещи, такие как трансляцию протокола мыши, работать с несколькими мышами, ..

3.3.11 Звуковые карты

если вы чувствуете огромное желание слышать рявканье biff, то ответьте 'y', и далее другая программа настройки будет скомпилирована и будет задавать вам вопросы о вашей звуковой карте.

(Примечание о настройке звуковой карты: когда программа спросит у вас устанавливать ли полную версию драйвера, то ответьте 'n' и сохраните некоторое количество памяти в ядре выбором только необходимых возможностей драйвера). Я сильно рекомендую вам посмотреть в Sound-HOWTO для более детальной информации о поддержке звука, если у вас есть звуковая карта.

3.3.12 Другие опции настройки

Не все опции настройки перечислены здесь потому-что они слишком часто меняются или являются очевидными (например, поддержка 3Com 3C509 для компиляции драйвера для данной карты ethernet). Существует довольно полный список всех опций (плюс способ поместить их в скрипт `Configure`), который собран Axel Boldt (`axel@uni-paderborn.de`) по следующему адресу:

`http://math-www.uni-paderborn.de/~axel/config_help.html`

или через анонимный FTP по адресу:

`ftp://sunsite.unc.edu/pub/Linux/kernel/config/krnl_cnfg_hlp.x.yz.tgz`

где `x.yz` это номер версии.
Для последних ядер (2.0.x и более поздних), этот список был интегрирован в дерево исходных текстов.

3.3.13 Работа над ядром (Kernel hacking)

Из Linus README:

> действие опции настройки "kernel hacking" обычно проявляется в более большом или медленном ядре (или оба симптома), и может даже сделать ядро менее стабильным из-за настройки некоторых подпрограмм на попытку активно сломать плохой код, чтобы найти проблемы с ядром (kmalloc()). Таким образом вам скорее всего надо ответить 'n' на этот вопрос для "production" ядер.

3.4 Что теперь? (Makefile)

После того как `make config` выдаст сообщение о том, что ваше ядро было настроено, вы можете "проверить (настроить) основной `Makefile` для дополнительных настроек", и т.п.
Теперь вы можете посмотреть в `Makefile`. Вам вероятно не понадобится изменять его, но вы не повредите его если посмотрите. Вы можете также изменить опции в нем с помощью команды '`rdev`' после того как поместите ядро на его место.

4 Компиляция ядра

4.1 Очистка и создание зависимостей

Когда настроечный скрипт закончит свою работу, он также скажет вам, чтобы вы выполнили '`make dep`' и (вероятно) '`clean`'. Так что выполните '`make dep`'. Он обеспечит, чтобы все зависимости, такие как файлы заголовков, находятся на месте. Эта процедура не длиться долго, если у вас не медленный компьютер. Для более старых версий ядер, при окончании вы должны выполнить '`make clean`'. Эта процедура удаляет все объектные файлы и некоторые другие вещи оставшиеся от предыдущей компиляции. В любом случае, *не* забывайте выполнить этот шаг до начала перекомпиляции ядра.

4.2 Время компиляции

После выполнения `dep` и `clean`, вы можете выполнять '`make zImage`' или '`make zdisk`' (эта часть процесса занимает длительное время). '`make zImage`' скомпилирует ядро и оставит в директории `arch/i386/boot` файл названный '`zImage`' (среди других вещей). Это новое сжатое ядро. '`make zdisk`' делает тоже самое, но также помещает новый файл `zImage` на гибкий диск, который вы должны вставить в устройство "A:". '`zdisk`' является довольно удобным для тестирования новых ядер; если оно не загружается (или просто работает неправильно), то просто вытащите дискету из дисковода и загрузитесь со старым ядром. Это может быть также удобным в том случае, если вы случайно удалили ядро (или сделали что-нибудь подобное по своему разрушительному действию). Вы также можете использовать его для установки новых систем, в том случае когда вы просто делаете дамп с одного диска на другой("это все и больше! Теперь, сколько вы могли бы купить?"). Все, даже сравнительно недавние ядра являются сжатыми, поэтому они имеют букву '`z`' в начале имени. Сжатое ядро автоматически разжимается при выполнении.

4.3 Другие вещи, можно сделать с помощью "make"

'`make mrproper`' выполнит более интенсивную очистку дерева исходных текстов. Иногда она является необходимой; вы можете выполнять эту команду после каждого наложения заплаток. '`make mrproper`' также удалит ваши файлы конфигурации, так что вы можете захотеть сохранить резервную копию вашего файла (`.config`), если вы считаете его ценным.
'`make oldconfig`' попытается настроить ваше ядро используя старый файл настроек; он проделает путь по процессу конфигурации '`make config`' вместо вас. Если у вас нет скомпилированного ядра или у вас нет старого файла настроек, то вам скорее всего не надо делать этой операции, поскольку вы вероятно захотите изменить настройки по умолчанию.
Смотрите раздел о модулях для описания операции '`make modules`'.

4.4 Установка ядра

После того как вы установили, что новое ядро работает так как вам надо, наступает время его установки. Большинство людей для этого использует LILO (Загрузчик Linux). Команда '`make zlilo`' установит новое ядро, запустит для него LILO, и все будет готово к перезагрузке, НО ТОЛЬКО если lilo настроено правильно в вашей системе: ядро располагается в файле `/vmlinuz`, lilo находится в директории `/sbin`, и ваш конфигурационный файл lilo (`/etc/lilo.conf`) отражает эти условия.
Иначе вам придется использовать LILO непосредственно. Это довольно легкий в установке и в работе пакет, но он имеет тенденцию вводить в замешательство людей своим конфигурационным файлом. Посмотрите конфигурационный файл (либо `/etc/lilo/config` для старых версий либо `/etc/lilo.conf` для более новых версий), и посмотрите текущие настройки. Конфигурационный файл выглядит примерно так:

```
image = /vmlinuz
    label = Linux
    root = /dev/hda1
    ...
```

'`image =`' указывает на установленное в настоящее время ядро. Большинство людей используют `/vmlinuz`. '`label`' используется для определения какое ядро или операционная система будет загружаться, и '`root`' это корневой раздел отдельной операционной системы. Сделайте резервную копию вашего ядра и скопируйте только что сделанное ядро на его место (вы должны выполнить команду '`cp zImage /vmlinuz`' если вы используете '`/vmlinuz`'). Затем перезапустите lilo — на более новых системах вы можете просто запустить '`lilo`', но на старых вы должны выполнить `/etc/lilo/install` или даже `/etc/lilo/lilo -C /etc/lilo/config`.

Если вы хотите знать больше о настройке LILO или у вас его нет, то возьмите самую новую его версию с вашего любимого сервера ftp и следуйте инструкциям.
Для загрузки одного из ваших старых ядер на жестком диске (еще один способ обезопасить себя при использовании нового ядра), скопируйте нижеприведенные строки (и включите) 'image = xxx' в файл конфигурации LILO в конце файла, и измените 'image = xxx' на 'image = yyy', где 'yyy' это полный путь вашего старого ядра. Затем измените 'label = zzz' на 'label = linux-backup' и перезапустите lilo. Вам может быть будет необходимо поместить строку в конфигурационный файл, которая выглядит так 'delay=x', где x это количество десятых долей секунды, на которое LILO задержится до загрузки, так что вы можете прервать его выполнение (например при помощи клавиши shift), и набрать метку имя сохраненного образа загрузки (в случае, если произойдут разные неприятные вещи).

5 Исправление ядра с помощью заплаток

5.1 Наложение заплаток

Накладывающиеся обновления ядра распространяются в виде заплаток. Например, если у вас версия 1.1.45, и вас оповестили, что выпущен 'patch46.gz' для него, это означает, что вы можете обновить ядро до версии 1.1.46 приложив эту заплатку. Вы можете захотеть сначала сделать резервную копию дерева исходных текстов ядра (сначала 'make clean' и затем 'cd /usr/src; tar zcvf old-tree.tar.gz linux' создаст для сжатый архивный файл с резервной копией).
Так, продолжая пример, приведенный выше, предположим, что у вас файл 'patch46.gz' расположен в директории /usr/src. Выполним cd в /usr/src и выполним 'zcat patch46.gz | patch -p0' (или 'patch -p0 < patch46' если эта заплатка не является сжатой). Вы увидите некоторые вещи мелькающие на экране, которые сообщают вам, что программа пытается приложить куски заплатки на нужные файлы и информацию о том, удачна данная операция или нет. Обычно этот процесс идет слишком быстро, чтобы вы могли прочитать и вы можете не быть уверенными, удачна эта операция или нет, в этом случае вы можете использовать опцию -s для программы patch, которая заставляет patch выдавать только сообщение об ошибках (вы не получите сообщения "эй, мой компьютер действительно что-то делает!", но если вы это предпочитаете..). Для того, чтобы взглянуть на то какие разделы не прошли гладко, перейдите в директорию /usr/src/linux и посмотрите файлы с расширением .rej. Некоторые версии программы patch (старые версии, которые могли быть скомпилированы на более худших файловых системах) оставляют отклоненные файлы с расширением #. вы можете использовать команду 'find' для того чтобы найти эти файлы:

```
find .  -name '*.rej' -print
```

эта команда печатает список всех файлов, которые находятся в текущей директории и ее поддиректориях и имеют расширение .rej на стандартный вывод.
если все прошло правильно, то выполните команды 'make clean', 'config', и 'dep' как описано в разделах 3 и 4.
Существует еще несколько опций для команды patch. Как было отмечено выше, patch -s запретит вывод всех сообщений за исключением сообщений об ошибках. Если вы храните ваше ядро где-то в другом месте, отличном от /usr/src/linux, то выполнение patch -p1 (в этой директории) правильно выставит заплатку. Другие опции команды patch хорошо описаны в ее справочной странице.

5.2 Если что-то неправильно

(Замечание: этот раздел относится к большинству старых ядер).

Наиболее частая проблема возникает когда заплатка модифицирует файл, называемый 'config.in' и он не выглядит достаточно правильно, потому-что вы изменили его опции для вашей машины. Это было исправлено, но эта проблема может возникнуть со старыми выпусками ядра. Для исправления этой проблемы посмотрите в файл config.in.rej, и посмотрите что осталось сделать оригинальной заплатке. Изменения обычно обозначаются символами '+' и '-' в начале строки. Посмотрите строки, которые окружают эти символы и запомните где установлено 'y' или 'n'. Теперь отредактируйте файл config.in, и измените 'y' на 'n' и 'n' на 'y' где это нужно. Выполните команду

```
patch -p0 < config.in.rej
```

и если она выдала, что операция проведена удачно (без ошибок), то вы можете продолжать работу выполняя настройку ядра и его компиляцию. Файл config.in.rej все равно останется, но вы можете удалить его.
если у вас все равно существуют проблемы, то это значит, что вы могли установить заплатку не с тем номером. Если программа patch сообщает 'previously applied patch detected: Assume -R? (обнаружена предыдущая заплатка: использовать опцию -R?', то вы скорее всего пытаетесь приложить заплатку с номером меньшим, чем номер версии вашего ядра; если вы ответите 'y', то программа попытается вернуть ваши исходные тексты к предыдущей версии, и скорее всего это вызовет ошибку; поэтому вам понадобится установить заново все дерево исходных текстов (что может быть не такой уж плохой идеей).
Для того чтобы убрать изменения внесенные заплаткой, используйте команду 'patch -R' с оригинальной заплаткой.
Лучше всего в случае, когда заплатки делают что-то неправильно, начать применять ее с новым деревом исходных текстов (например извлеченным из одного из файлов linux-x.y.z.tar.gz), и запустит процесс заново.

5.3 Избавляемся от файлов .orig

После всего нескольких заплаток у вас накопится куча файлов с расширением .orig. Например одно ядро 1.1.51, которое было последний раз почищено при версии 1.1.48. Удаление .orig файлов сохранило примерно половину мегабайта дисковой памяти.

```
find .  -name '*.orig' -exec rm -f {} ';'
```

эта команда позаботится о вас. Версии программы patch, которые используют знак # для отклоненных файлов используют знак тильды вместо .orig.
Существует лучший способ избавится от .orig файлов, который зависит от GNU программы xargs:

```
find .  -name '*.orig' | xargs rm
```

или "довольно безопасный, но несколько более многословный" метод:

```
find . -name '*.orig' -print0 | xargs --null rm --
```

5.4 Другие заплатки

Также существуют другие заплатки (я буду назвать их "нестандартными"), кроме поставляемых Linus. Если вы накладываете такие заплатки, то заплатки от Linus могут работать неправильно и вы должны будете либо убрать их, изменить исходные тексты или заплатку, либо установить новое дерево исходных текстов, или выполнить комбинацию описанных действий. Это может быть очень расстраивающим, так что если вы не хотите изменять исходные тексты (с возможно плохим результатом), то удалите нестандартные заплатки до приложения заплаток полученных от Linus,

или просто установите новое дерево исходных текстов. Затем вы можете посмотреть работают ли нестандартные заплатки. Если они не работают, то вы либо задержались с использованием старого ядра, поиграйтесь с исходными текстами или измените заплатку, либо просто дождитесь выхода новой версии нестандартной заплатки.
Какие существуют заплатки не входящие в стандартный дистрибутив? Вы вероятно слышали о них. Я использую такую заплатку для того, чтобы курсор на моей консоли не мигал, я ненавижу мигающие курсоры (Эта заплатка часто обновляется (или по крайней мере обновлялась) по мере выпуска новых версий ядра. Для большинства новых устройств драйвера разрабатываются как загружаемые модули и частота использования нестандартных заплаток значительно уменьшается.

6 Дополнительные пакеты

Ваше ядро имеет много возможностей, которые не объясняются в исходных текстах ядра; эти возможности обычно используются через использование внешних пакетов. Некоторые из наиболее общих пакетов перечислены здесь.

6.1 kbd

Консоль linux вероятно имеет больше возможностей, чем она заслуживает. Среди них возможность переключения шрифтов, изменения раскладки клавиатуры, переключение видеорежимов (в более новых ядрах) и т.п. Пакет kbd имеет программы, которые позволяют сделать все это, и в дополнение много шрифтов и раскладок клавиатуры для большинства клавиатур и он доступен с тех же самых серверов, которые распространяют исходные тексты ядра.

6.2 util-linux

Rik Faith (`faith@cs.unc.edu`) собрал вместе большой набор утилит для linux, который по странному совпадению называется util-linux. В настоящее время этот набор сопровождается Nicolai Langfeldt (`util-linux@math.uio.no`). Он доступен по анонимному ftp с sunsite.unc.edu в директории `/pub/Linux/system/misc`, он содержит такие программы как `setterm`, `rdev`, и `ctrlaltdel`, которые имеют отношение к ядру. Как сказал Rik, *не устанавливайте их без раздумий;* вам не нужно устанавливать весь пакет, и у вас могут возникнуть серьезные проблемы, если вы сделаете это.

6.3 hdparm

Как и много других пакетов, раньше это был пакет из заплатки на ядро и программ поддержки. Сейчас эти заплатки включены в официальное дерево исходных текстов ядра и программы для оптимизации и настройки вашего жесткого диска поставляются отдельно.

6.4 gpm

gpm это обозначение для мыши общего назначения (general purpose mouse). Эта программа позволяет вам вырезать и вставлять текст между виртуальными консолями, а также делать другие действия с большим количеством мышей разных типов.

7 Некоторые ловушки

7.1 make clean

Если новое ядро делает какие-то странные вещи после текущего его обновления, то есть большая вероятность, что вы забыли выполнить `make clean` до компиляции нового ядра. Симптомы могут

быть любыми от полного краха вашей системы, странных проблем с вводом/выводом до малой производительности. Убедитесь также, что вы сделали `make dep`.

7.2 Большие или медленные ядра

Если ваше ядро поглощает достаточное количество памяти, слишком большое и/или просто долго компилирует, даже когда вы заставили ваш новый 786DX6/440 работать с ним, то вы вероятно получили набор ненужных вам вещей (драйверов устройств, файловых систем и т.п.). Если вы не используете их, то не настраивайте их, потому, что это занимает память машины. Наиболее очевидный симптом раздутия ядра, это интенсивное свапирование памяти на диск и с диска; если ваш диск создает шум и он не один из старых винчестеров Fujitsu Eagles, чей звук напоминал звук выключаемого двигателя реактивного самолета, то посмотрите в конфигурацию ядра.
Вы можете узнать сколько оперативной памяти занимает ядро взяв общее количество памяти на машине и вычтя из него количество "общей памяти" в файле `/proc/meminfo` или вывод команды '`free`'. Вы можете также определить это выполнив команду '`dmesg`' (или посмотрев в файл протокола ядра, если он есть в вашей системе). Там будет строка, которая выглядит примерно так:
`Memory: 15124k/16384k available (552k kernel code, 384k reserved, 324k data)`
Моя машина с процессором 386 (которая была настроена с меньшим количество опций) выдает следующее:
`Memory: 7000k/8192k available (496k kernel code, 384k reserved, 312k data)`
Если у вас просто получается большое ядро, но система не позволяет вам это, то вы можете попытаться выполнить '`make bzimage`'. Вам также может понадобиться установить новую версию LILO чтобы сделать это.

7.3 Ядро не компилируется

Если ядро не компилируется, то скорее всего произошел сбой при накладывании заплатки или ваши исходные тексты были повреждены каким-либо образом. У вас также может быть неправильная версия gcc или также может быть повреждена (например включаемые файлы могут быть с ошибками). Убедитесь, что символические ссылки, которые описывает Linus в файле `README` установлены правильно. В общем, если стандартное ядро не компилируется, то у час что-то серьезное с системой и вероятно необходима переустановка некоторых утилит.
или возможно вы компилируете ядро 1.2.x при помощи ELF компилятора (gcc 2.6.3 и выше). Если вы получили набор ошибок типа `so-and-so undefined` в течении компиляции, то скорее всего у вас такая проблема. Исправление в большинстве случаев очень просто. Добавьте эти строки в начало файла `arch/i386/Makefile`:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Затем заново выполните `make dep` и `zImage`.
В редких случаях gcc может не работать из-за аппаратных проблем. Сообщение об ошибке будет примерно такое "xxx exited with signal 15" и это в общем будет выглядеть очень загадочно. Я вероятно не должен был здесь это упоминать, за исключением того что это со мной однажды случилось — у меня была испорченная кэш-память и компилятор время от времени не работал. Попробуйте сначала переставить gcc, если у вас есть такая проблема. ВЫ должны стать подозрительным только если ваше ядро нормально компилируется с отключенным внешним кэшем, с уменьшенным количество оперативной памяти и т.п.
Это имеет склонность беспокоить людей, когда они предполагают, что их оборудование не в порядке. Хорошо, я не буду делать это. Об этом существует FAQ — он находится на `http://www.bitwizard.nl/sig11/`.

7.4 Не видно чтобы новая версия ядра грузилась

Вы не запустили LILO, или он не настроен правильно. Одна вещь которая случилось однажды со мной это была проблема в файле конфигурации; там говорилось 'boot=/dev/hda1' вместо 'boot=/dev/hda' (Это может быть раздражающим в начале, но когда вы сделаете рабочий файл конфигурации, то вам не нужно будет его больше изменять).

7.5 Вы забыли запустить LILO, или система просто не грузится

Оххх! Лучшая вещь, которую вы можете сделать в этом случае это загрузиться с дискеты подготовить другой загрузочный диск (такой какой должна сделать команда 'make zdisk'). Вам необходимо знать где находится ваша корневая файловая система (/) и какой тип она имеет (например second extended, minix). В нижеприведенном примере, вам также необходимо знать на какой файловой системе находится дерево исходных текстов /usr/src/linux, ее тип и где она обычно монтируется.
В следующем примере, / находится на /dev/hda1, а файловая система, которая содержит /usr/src/linux находится на /dev/hda3, обычно смонтированной как /usr. Обе относятся к типу second extended файловых систем. Рабочее ядро находится в директории /usr/src/linux/arch/i386/boot и называется zImage.
Идея заключается в том, что если есть работающее ядро, то можно использовать его для создания нового загрузочного гибкого диска. Другой вариант, который может работать лучше (а может и не работать, это зависит от конкретного метода которым вы сломали свою систему) обсуждается дальше после примера.
С начала загрузимся с комбинации загрузочного/корневого дисков или спасательного (rescue) диска, и смонтируем файловую систему, которая содержит работающее ядро:

```
mkdir /mnt
mount -t ext2 /dev/hda3 /mnt
```

Если mkdir сообщает вам, что директория уже существует, то просто проигнорируйте это сообщение. Затем перейдите в ту директорию, где находится работающее ядро. Заметим, что

```
/mnt + /usr/src/linux/arch/i386/boot - /usr = /mnt/src/linux/arch/i386/boot
```

Поместите отформатированную дискету в привод "A:" (только не загрузочную дискету и не дискету с корневой файловой системой!), и перебросьте ядро на дискету и настройте его на вашу корневую файловую систему:

```
cd /mnt/src/linux/arch/i386/boot
dd if=zImage of=/dev/fd0
rdev /dev/fd0 /dev/hda1
```

перейдите в / и отмонтируйте обычную файловую систему /usr:

```
cd /
umount /mnt
```

Теперь вы должны иметь возможность перезагрузить ваш компьютер как обычно с созданной дискеты. Не забудьте перезапустить lilo (или выполнить то, что вы сделали не правильно) после перезагрузки!
Как было упомянуто выше, существует другая общая альтернатива. Если у вас к счастью имеется рабочее ядро находящееся на разделе / (например /vmlinuz), то вы можете использовать его для загрузочной дискеты. Предполагая все вышеприведенные условия, и что наше ядро находится в /vmlinuz, то просто сделайте изменения в вышеприведенном примере: измените /dev/hda3 на

/dev/hda1 (корневая файловая система), /mnt/src/linux на /mnt, и if=zImage на if=vmlinuz. Замечание о том как получить доступ к /mnt/src/linux может быть проигнорировано.
Используя LILO с большими дисками (больше чем 1024 цилиндра) может вызвать проблемы. Смотрите LILO mini-HOWTO или документацию для помощи в этом случае.

7.6 Ядро сообщает 'warning: bdflush not running (предупреждение bdflush не запущен)'

Это может быть серьезной проблемой. Начиная с ядер после 1.0 (примерно 20 апреля 1994), программа названная 'update', которая периодически сохраняла буфера файловой системы была изменена/заменена. Возьмите исходные тексты программы 'bdflush' (вы должны найти их там где вы брали исходные тексты ядра), и установите эту программу (вы вероятно захотите запустить старое ядро пока вы делаете это). Эта программа сама установится как 'update' и после перезагрузки, новое ядро не будет больше выражать недовольство ее отсутствием.

7.7 Выводятся сообщения о неопределенных символах и не компилируется

У вас вероятнее всего ELF компилятор (gcc 2.6.3 и выше) и исходные тексты ядра 1.2.x (или более раннего). Обычное исправление заключается в добавлении этих трех строк в начало файла arch/i386/Makefile:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Это заставит выполнять компиляцию ядра 1.2.x с библиотеками a.out.

7.8 Я не могу заставить работать мой привод IDE/ATAPI CD-ROM

Достаточно странно, но много людей не могут заставить работать свои устройства ATAPI, потому что существуют некоторые вещи, который могут быть сделаны неправильно.
Если ваш CD-ROM единственное устройство на отдельном интерфейсе IDE, то оно должно быть выставлено как "master" или "single". Предположительно это наиболее общая ошибка.
Creative Labs (для некоторых) поместил интерфейс IDE на свои звуковые карты. Однако это приводит к интересной проблеме, заключающейся в том, что некоторые люди имеют только один интерфейс, много имеют два IDE интерфейса, встроенных в материнские платы (обычно на IRQ15), так что общая практика в том, чтобы сделать интерфейс на soundblaster третим IDE портом (IRQ11).
Это вызывает проблему с linux в том, что в версиях 1.2.x не поддерживается третий IDE интерфейс (эта поддержка началась где-то в серии 1.3.x, но это было для разработчиков, помните об этом, и не был автоматической пробы). Для того чтобы заставить это работать у вас есть несколько возможностей.
Если вы уже имеете второй IDE порт, то существует вероятность, что вы не используете его или у вас не два устройства на нем. Уберите привод ATAPI со звуковой карты и поместите его на второй интерфейс. Затем вы можете запретить интерфейс на звуковой карте, что сохранит вам IRQ.
Если у вас нет второго интерфейса, то переключите интерфейс на звуковой карте (только не часть работающую со звуком) на использование IRQ15, как второй интерфейс. Это должно работать.
Если по некоторым причинам ваше устройство должно быть на так называемом "третьем" интерфейсе, или в случае других проблем возьмите ядро версии 1.3.x (например ядро 1.3.57 имеет такую поддержку), и прочитайте файл drivers/block/README.ide. Там существует гораздо больше информации.

7.9 Ядро сообщает странные вещи об устаревших запросах маршрутизации

Возьмите новую версию программы `route` и любую другую программу, которая выполняет манипуляцию маршрутизацией. `/usr/include/linux/route.h` (который является файлом в `/usr/src/linux`) был изменен.

7.10 Firewalling не работает в 1.2.0

Обновите ядро по крайней мере до версии 1.2.1.

7.11 "Not a compressed kernel Image file (Не является файлом сжатого образа ядра)"

Не используйте файл `vmlinux`, созданный в `/usr/src/linux` как образ загрузки; Правильным образом загрузки является `[..]/arch/i386/boot/zImage`.

7.12 Проблемы с консолью после обновления до 1.3.x

Измените слово `dumb` на `linux` в записи для консоли в файле `/etc/termcap`. Вам также может понадобиться создать запись в terminfo.

7.13 Не могу скомпилировать некоторые вещи после обновления ядра

Исходные тексты ядра linux включают некоторое количество заголовочных файлов (файлов, чьи имена заканчиваются на `.h`), на которые ссылаются стандартные заголовочные файлы в `/usr/include`. На них обычно ссылаются примерно так (где `xyzzy.h` должен быть чем-то в `/usr/include/linux`):

```
#include <linux/xyzzy.h>
```

Обычно существует ссылка, названная `linux` в `/usr/include` на директорию `include/linux` в исходных текстах вашего ядра (`/usr/src/linux/include/linux` в обычной системе). Если эта ссылка находится не там, или указывает на неправильное место, то некоторые вещи вообще не будут компилироваться. Если вы посчитали, что исходные тексты ядра занимают слишком много места на диске и удалили их, то это скорее всего вызовет проблему. Другая проблема может возникнуть при неправильных правах доступа на файлы; если ваш администратор установил umask в такое значение, которое не позволяет другим пользователям видеть его файлы по умолчанию, и вы разархивировали исходные тексты без опции `p` (сохранение прав доступа), то эти пользователи не смогут пользоваться компилятором C. Хотя вы могли бы воспользоваться командой `chmod` для исправления этого, но вероятно более легко заново разархивировать заголовочные файлы. Вы можете сделать это также, как и со всеми исходными текстами, но только с дополнительным аргументом:

```
blah# tar zxvpf linux.x.y.z.tar.gz linux/include
```

Замечание: "`make config`" заново создаст ссылки в `/usr/src/linux`, если они отсутствуют.

7.14 Увеличение предельных значений

Следующие несколько показательных команд могут быть полезны для тех кто не знает как увеличить некоторые программные предельные значения The following few *example* commands may be helpful to those wondering how to increase certain soft limits imposed by the kernel:

```
echo 4096 > /proc/sys/kernel/file-max
echo 12288 > /proc/sys/kernel/inode-max
echo 300 400 500 > /proc/sys/vm/freepages
```

8 Замечание для обновления до версии 2.0.x

Ядра версии 2.0.x внесли довольно много изменений в установке ядра. Файл `Documentation/Changes` в дереве исходных текстов ядра 2.0.x содержит информацию, которую вы должны знать когда обновляете до версии 2.0.x. вам скорее всего надо обновить несколько ключевых пакетов, таких как gcc, libc, и SysVInit, и возможно изменить некоторые системные файлы, так что ожидайте этого. Хотя не паникуйте.

9 Модули

Загружаемые модули ядра могут сохранить память и упростить настройку. Область применения модулей включает файловые системы, драйвера карт ethernet, драйверы ленточных накопителей и т.п.

9.1 Установка утилит для работы с модулями

Утилиты для работы с модулями доступны от туда же откуда вы получили свое ядро, они называются `modules-x.y.z.tar.gz`; выберите самый большой номер `x.y.z`, который равен или ниже чем номер вашего текущего ядра. Распакуйте их с помощью команды '`tar zxvf modules-x.y.z.tar.gz`', перейдите в директорию, которую эта команда создала (`modules-x.y.z`), посмотрите файл `README`, и выполните приведенные в нем инструкции по установке (которые обычно являются очень простыми, такими как `make install`). Вы должны теперь получить программы `insmod`, `rmmod`, `ksyms`, `lsmod`, `genksyms`, `modprobe`, и `depmod` в директории `/sbin`. Если вы хотите, протестируйте полученные программы с помощью демонстрационного драйвера "hw" в `insmod`; для более детальной информации смотрите файл `INSTALL`, который находится в директории с исходными текстами.
Команда `insmod` вставляет модуль в работающее ядро. Модули обычно имеют расширение `.o`; пример драйвера, упомянутый выше называется `drv_hello.o`, так для того чтобы вставить его, вы должны выполнить '`insmod drv_hello.o`'. Для того чтобы увидеть список загруженных модулей используйте команду `lsmod`. Ее вывод выглядит примерно так:

```
blah# lsmod
Module:         #pages:  Used by:
drv_hello           1
```

'`drv_hello`' это имя модуля, он использует 1 страницу оперативной памяти (4k), и ни какие модули ядра не зависят от него на текущий момент. Для удаления этого модуля используйте команду '`rmmod drv_hello`'. Заметим, что `rmmod` требует *имя модуля,* а не имя файла; вы можете получить его из списка выдаваемого `lsmod`. Назначение других утилит для работы с модулями описано в их справочных страницах.

9.2 Модули распространяемые с ядром

В версии 2.0.30, почти все доступно как загружаемые модули. Для их использования сначала сначала убедитесь, что вы не настроили их вкомпилированными в ядро; то есть вы не ответили `y` в процессе выполнения '`make config`'. Скомпилируйте новое ядро и загрузитесь с ним. Затем снова перейдите в `/usr/src/linux`, и выполните '`make modules`'. это скомпилирует все модули, которые вы не указали при настройке ядра, и поместит ссылки на них в `/usr/src/linux/modules`. ВЫ можете использовать их прямо из этой директории, или выполните команду '`make modules_install`', которая установит модули в директорию `/lib/modules/x.y.z`, где `x.y.z` это версия ядра.

Это может быть особенно полезным в использовании файловых систем. Вы можете нечасто использовать файловые системы minix или msdos. Например, если я сталкиваюсь с гибким диском с msdos, я должен сделать `insmod /usr/src/linux/modules/msdos.o`, и затем `rmmod msdos`, когда все закончено. Эта процедура сохраняет примерно 50k ОЗУ в ядре в течении нормальной работы. Маленькое замечание для использования файловой системы minix: вы должны *всегда* настроить его прямо в ядро для использования в "восстановительных (rescue)" дисках.

10 Другие опции настройки

Этот раздел содержит описания избранных опций настройки ядра (в `make config`), которые не перечислены в разделе конфигурации. Большинство драйверов устройств не перечислены.

10.1 Общая настройка

`Normal floppy disk support (Поддержка обычных гибких дисков)` - что и написано. Вы можете прочитать файл `drivers/block/README.fd`; это особенно важно для пользователей IBM Thinkpad.
`XT harddisk support (поддержка жестких дисков XT)` - если вы хотите использовать 8-битные контроллеры XT, пылящиеся в углу.
`PCI bios support поддержка PCI bios)` - если у вас имеются PCI устройства, то вы можете попробовать использовать эту опцию; будьте осторожны, поскольку некоторые старые материнские платы с поддержкой PCI могут не работать с этой опцией. Более детальную информацию о использовании шины PCI под linux вы можете найти в PCI-HOWTO.
`Kernel support for ELF binaries (поддержка ядром исполняемых файлов в формате ELF)` - ELF это попытка позволить исполняемым файлам охватывать разные архитектуры и операционные системы; linux по видимому идет в этом направлении, так что вы вероятно захотите использовать эту опцию.
`Set version information on all symbols for modules (установка информации о версии на все символы для модулей)` - в прошлом, модули ядра перекомпилировались с каждым новым ядром. Если вы ответите `y`, то станет возможным использование модулей скомпилированных для других версий ядра. Прочитайте файл `README.modules` для более подробной информации.

10.2 Сетевые опции

Сетевые опции хорошо описаны в NET-3-HOWTO (или NET-какой-то-номер-HOWTO).

11 Советы и приемы

11.1 Перенаправление вывода команд make или patch

Если вы хотите протоколировать действия команд '`make`' или '`patch`', то вы можете перенаправить вывод в файл. Сначала определите какой интерпретатор команд вы используете выполнив команду: '`grep root /etc/passwd`' и ищите строку, которая выглядит примерно так '`/bin/csh`'.
Если вы используете sh или bash, то команда

```
(command) 2>&1 | tee (output file)
```

поместит копию вывода команды `(command)` в файл '`(output file)`'.
Для csh или tcsh, используйте следующую последовательность

```
(command) |& tee (output file)
```

Для rc (Замечание: вы скорее всего не используете rc) это выглядит так:

```
(command) >[2=1] | tee (output file)
```

11.2 Условная установка ядра

Вместо использования гибкого диска существует другой метод тестирования нового ядра без удаления старого. В отличии от многих других Unix-систем, LILO имеет возможность загружать ядро с любого места на диске (если у вас большой диск (500 MB или больше), то пожалуйста прочитайте документацию на LILO о том как это может вызвать проблемы). Итак, если вы добавите что-то похожее на следующие строки

```
image = /usr/src/linux/arch/i386/boot/zImage
    label = new_kernel
```

в конец вашего файла настроек LILO, то вы сможете выбрать запуск свежескомпилированного ядра без удаления старого `/vmlinuz` (конечно после предварительного запуска `lilo`). Самый легкий способ заставить LILO загрузить новое ядро - это нажать клавишу shift во время загрузки (когда на экран выводится сообщение `LILO`), это заставит программу загрузки выдать приглашение. В этом месте вы можете ввести название '`new_kernel`' для загрузки нового ядра.
Если вы хотите хранить несколько исходных тексты разных ядер на своем компьютере одновременно (это займет достаточно много места на диске, будьте осторожны) то наиболее удобный способ называть их `/usr/src/linux-x.y.z`, где дерево исходных `x.y.z` это номер версии ядра. Вы можете "выбирать" дерево исходных текстов с помощью символической ссылки. например, команда '`ln -sf linux-1.2.2 /usr/src/linux`' должна сделать текущим дерево исходных текстов ядра версии 1.2.2. До создания символической ссылки убедитесь, что последний аргумент команды `ln` не является настоящей директорией (старая символическая ссылка это нормально); результат будут не такой какой вы ожидали.

11.3 Обновления ядра

Russell Nelson (`nelson@crynwr.com`) подводит итоги сделанных изменений в новых выпусках ядер. Они являются короткими и вы можете захотеть взглянуть на них до начала обновления ядра. Эти данные доступны через анонимный ftp с `ftp.emlist.com` в директории `pub/kchanges` или со следующего URL

```
http://www.crynwr.com/kchanges
```

12 Другие HOWTO, которые могут быть полезными

- Sound-HOWTO: звуковые карты и утилиты
- SCSI-HOWTO: все о контроллерах и устройствах SCSI
- NET-2-HOWTO: работа с сетями
- PPP-HOWTO: в отдельности работа с PPP
- PCMCIA-HOWTO: о драйверах для вашего переносного компьютера (notebook)
- ELF-HOWTO: что такое ELF, преобразование..
- Hardware-HOWTO: обзор поддерживаемого оборудования
- Module-HOWTO: более детально о модулях ядра

- Kerneld mini-HOWTO: о kerneld
- BogoMips mini-HOWTO: в случае если вас что-то удивляет

13 Разное

13.1 Автор

Автором и сопроводителем Linux Kernel-HOWTO является Brian Ward (bri@blah.math.tu-graz.ac.at). Пожалуйста посылайте мне любые комментарии, добавления, исправления (в частности исправления наиболее важны для меня).
Вы можете взглянуть на мою домашнюю страницу по одному из этих URLs:

```
http://www.math.psu.edu/ward/
http://blah.math.tu-graz.ac.at/~bri/
```

Даже хотя я стараюсь быть внимательным с почтой, пожалуйста помните, что я получаю *достаточно много* сообщений каждый день, так что ответ на ваше письмо может занять некоторое время. Особенно если вы задаете мне вопрос, пожалуйста постарайтесь изложить его ясно и детально в вашем сообщении. Если вы пишите о неработающем оборудовании (или о чем-нибудь подобном), то мне необходимо знать как оно настроено. Если вы сообщаете об ошибке, не сообщайте просто "я пробую, а она выдает мне ошибку", мне необходимо знать какая ошибка произошла. Я также буду рад знать какую версию ядра, gcc, и libc вы используете. Если вы просто скажете, что вы используете тот или иной дистрибутив, то это не скажет мне многого. Я не беспокоюсь, если вы зададите мне простые вопросы; помните, если вы не будете спрашивать, то вы никогда не получите ответ! Я хочу поблагодарить всех, кто связывался со мной.
Если вы написали мне и не получили ответ за какое-то вполне достаточное количество времени (три недели или месяц), то вероятно, что я случайно удалил ваше сообщение (извините) Пожалуйста попробуйте еще раз.
Я получаю довольно много писем о вещах, которые в действительности являются аппаратными проблемами или их последствиями. Это нормально, но пожалуйста постарайтесь запомнить, что я не знаю все аппаратное обеспечение мира и я не знаю как вам помочь; я сам использую машины с IDE и SCSI дисками, SCSI CD-ROM, сетевыми картами 3Com и WD, последовательными мышами, материнскими платами с шиной PCI, контроллерами SCSI NCR 810, процессорами AMD 386DX40 w/Cyrix, AMD 5x86, AMD 486DX4, и Intel 486DX4 (Это обзор того что я использую и с чем я хорошо знаком, это ни в коем случае не рекомендация, но если вы хотите этого, то пожалуйста спрашивайте :-)).
Версия -0.1 была написана 3 октября 1994 года. Этот документ доступен в форматах SGML, PostScript, TeX, roff и простого текста.

13.2 Что сделать

Раздел "Советы и приемы" очень маленький. Я надеюсь расширить его с помощью ваших пожеланий.
То же самой для раздела "Дополнительные пакеты."
Требуется больше информации об отладке/восстановлении.

13.3 Сотрудничество

Включена небольшая часть файла README написанного Linus (kernel hacking options). (Спасибо, Linus!)
uc@brian.lunetix.de (Ulrich Callmeier): patch -s и xargs.
quinlan@yggdrasil.com (Daniel Quinlan): исправления и дополнения во многих разделах.

nat@nat@nataa.fr.eu.org (Nat Makarevitch): mrproper, tar -p, много других вещей.
boldt@math.ucsb.edu (Axel Boldt): собранные в сети описания опций настройки ядра;
lembark@wrkhors.psyber.com (Steve Lembark): дополнение про множественную загрузку
kbriggs@earwax.pd.uwa.edu.au (Keith Briggs): некоторые исправления и пожелания
rmcguire@freenet.columbus.oh.us (Ryan McGuire): дополнения к информации о возможных вариантах команды make.
dumas@excalibur.ibp.fr (Eric Dumas): Французский перевод
simazaki@ab11.yamanashi.ac.jp (Yasutada Shimazaki): Японский перевод
jjamor@lml.ls.fi.upm.es (Juan Jose Amor Iglesias): Испанский перевод
mva@sbbs.se (Martin Wahlen): Шведский перевод
jzp1218@stud.u-szeged.hu (Zoltan Vamosi): Венгерский перевод
bart@mat.uni.torun.pl (Bartosz Maruszewski): Польский перевод
donahue@tiber.nist.gov (Michael J Donahue): печатные ошибки, победитель "sliced bread competition"
rms@gnu.ai.mit.edu (Richard Stallman): уведомление о концепции/распространении "свободной" документации
dak@Pool.Informatik.RWTH-Aachen.DE (David Kastrup): часть относящаяся к NFS
esr@snark.thyrsus.com (Eric Raymond): различные пикантные новости
Люди, кто посылал мне письма с вопросами и проблемами, которые были очень полезны.

13.4 Уведомление об авторских правах, Лицензия и все такие вещи