Сборка ядра Linux из исходного кода

О том, как я собираю свои ядра версий 2.6.
Почти полностью применимо к ядрам 3.x в Slackware 14 и последующих.

X и su

Я выполняю команды из терминала X, в соответствующий момент использую графическую версию конфигуратора ядра. Я вхожу в систему от своего имени, но собираю ядра от имени суперпользователя root. Чтобы позволить root использовать мой дисплей X, я делаю следующее в терминале X: получаю права root; объединяю свой (alien) файл Xauthority с одноимённым файлом пользователя root и устанавливаю переменную окружения DISPLAY. Это даёт возможность запускать приложения X из терминала «su».

```
echo $DISPLAY                    # это значение понадобится нам тремя строками ниже
sudo -i                          # или «su -» в более ранних версиях Slackware
xauth merge ~alien/.Xauthority # вместо «alien» используйте ваше имя пользователя
export DISPLAY=:0.0              # используйте значение DISPLAY, полученное на 3 строки выше
```

Вместо этого можно выполнить следующие 2 команды, которые дадут тот же результат:

```
sudo -s                          # сторонний эффект '-s' в разрешении для root запускать программы X
. /etc/profile                   # применение («source») настроек глобального профиля гарантирует
                # наличие у root каталога ''sbin'' в переменной окружения $PATH
```

Загрузка и конфигурирование

Теперь, когда сборочное окружение настроено, перейдём к получению исходного кода.

Загрузите архив исходного кода нового ядра, распакуйте его в `/usr/src` и создайте ссылку «linux», чтобы команды были более общего вида. Я возьму в качестве примера ядро версии «2.6.37.6». Если у вас версия отличается, вы будете знать, где поменять строку версии в оставшейся части истории. Если вы хотите узнать о том, как проверить целостность архива с исходным кодом при помощи ключа GPG, читайте последний раздел ниже.

```
wget http://www.us.kernel.org/pub/linux/kernel/v2.6/linux-2.6.37.6.tar.bz2
tar -C /usr/src -jxvf linux-2.6.37.6.tar.bz2
cd /usr/src
rm linux                       # удалим существующую символьную ссылку
ln -s linux-2.6.37.6 linux     # создадим ссылку на исходный код нового ядра
```

Менять символьную ссылку «`linux`» безопасно. Смена её цели на другое ядро вместо установленного Slackware, не приведёт к поломке приложений. Возможно, вы заметите другие

каталоги linux-* в /usr/src. Обычно ссылка «linux» указывает на ядро, с которым вы сейчас работаете. Однако, наличие этой символьной ссылки не является *обязательным*. Современное программное обеспечение, которому требуется знать расположение исходного кода установленного ядра, смотрит на символьную ссылку /lib/modules/<версия_ядра>/build.

Существует дискуссия о том, следует ли собирать ядра в дереве /usr/src или в совершенно ином месте. Причина в давнем сообщении Линуса Торвальдса (от июля 2000 года), где он советует пользователям проводить сборку в их домашнем каталоге. Я считаю, что этот совет не относится к Slackware с её настройками заголовков ядра и пакета glibc. Поэтому, **мой** совет – если хотите, можете игнорировать то сообщение Линуса и устанавливать исходный код ядра в /usr/src. Место сборки ядра вопрос исключительно личных предпочтений.

Теперь возьмите конфигурационный файл ядра Slackware как основу для вашей конфигурации. Конфигурационные файлы Патрика являются довольно общими. Возможно, когда вы читаете эти строки, доступна конфигурация для нового выпуска 2.6:

```
wget
http://slackware.mirrors.tds.net/pub/slackware/slackware-13.37/source/k/conf
ig-generic-smp-2.6.37.6-smp
cp config-generic-smp-2.6.37.6-smp /usr/src/linux/.config
```

Вместо этого можно воспользоваться конфигурацией текущего работающего ядра:

```
zcat /proc/config.gz > /usr/src/linux/.config
```

Запустите *make oldconfig* в каталоге исходного кода ядра, чтобы применить умолчания из только что установленного вами файла .config. Скорее всего, ваш исходный код ядра свежее чем .config, поэтому вам для настройки будут предложены новые опции. Вам останется ответить только на них (нажимайте клавишу Ввод для предлагаемого по умолчанию ответа, который скорее всего сгодится, или M для сборки новых драйверов как модулей).

```
cd /usr/src/linux
make oldconfig
```

Сейчас у вас настроено довольно общее ядро (вероятно поэтому Патрик называет их «kernel-generic») и вы захотите изменить некоторые из умолчаний под свои нужды. Запустите графический конфигуратор (если вы вместо X используете текстовую консоль, запустите «make menuconfig» чтобы запустить основанную на curses диалоговую программу)

```
make xconfig
```

Пройдитесь по лесу настроек. Что обычно меняю я (в Slackware 13.37 и позже), это:

- Сборку в ядро ext3 и ext4 (также требует драйвер jbd) и драйверов файловых систем reiser/xfs/jfs вместо сборки модулями – это избавляет от необходимости в «initrd». (смотрите раздел «Filesystems» в конфигураторе).

- Поддержку 64ГБ ОЗУ.
  («Processor type and features» > «High Memory Support (64GB)»). Используйте для систем с 4ГБ ОЗУ или более.
- Поддержку отзывчивости («low-latency») ядра если компьютер настольный/переносной – мультимедийные приложения работают намного плавнее.
  («Processor type and features» > «Preemption model» > «Preemptible kernel»). Для настольной системы с большим количеством мультимедийных приложений это полезная опция, поскольку сохраняет отзывчивость системы даже при высокой нагрузке.
- Устанавливаю таймер в 1000 Гц («Processor type and features» > «Timer Frequency» > «1000 Hz»). Повышенная частота может быть полезной для мультимедийных 'настольных' систем.
- Или включаю тактонезависимый таймер («Processor type and features» > «Tickless System (Dynamic Ticks)»).
- Если вы (пере-)собираете ядро Slackware, нужно убедиться что установка нового ядра оставит оригинальные модули нетронутыми. Для этого укажите уникальную строку в поле *локальная версия* ядра («General setup» > «Local version - append to kernel release»). Этот параметр ядра соответствует **CONFIG_LOCALVERSION** в файле .config. В Slackware для SMP ядер этот параметр установлен в значение «-smp».
  Итоговый номер версии ядра (возвращаемый «`umake -r`») для ядра версии «`2.6.37.6`» с локальной версией «`-alien`» будет «`2.6.37.6-alien`».
- … и ещё что-то, о чём прямо сейчас не вспомню. Вы можете решить отключить большинство собираемых в конфигурации по умолчанию модулей для уменьшения времени компиляции, если такое оборудование отсутствует в вашем компьютере. Если у вас лэптоп, также можете обратить внимание на параметры *software suspend* и *CPU frequency scaling* (раздел «Processor type and features»).

И, наконец, сохраните конфигурацию, если она вас устраивает.

Сборка ядра

Теперь запустите сборку ядра, модулей и их установку на положенное место.

```
make bzImage modules             # компилирует ядро и модули
make modules_install             # устанавливает модули в
/lib/modules/<версия_ядра>
cp arch/x86/boot/bzImage /boot/vmlinuz-custom-2.6.37.6  # копирует файл нового
ядра
cp System.map /boot/System.map-custom-2.6.37.6          # копирует
System.map (опционально)
cp .config /boot/config-custom-2.6.37.6                 # резервная копия
конфигурации ядра
cd /boot
rm System.map                                           # удаляет старую
ссылку
ln -s System.map-custom-2.6.37.6 System.map             # создаёт новую ссылку
```

Для ядер 2.6.x должно быть достаточно выполнить «`make`» или «`make all`» вместо «`make`

bzImage modules». В этом случае будут собраны цели по умолчанию: *vmlinux* (несжатое ядро), *bzImage* (сжатое ядро, которое мы и будем использовать) и *modules* (все модули ядра). Поскольку несжатое ядро нам не нужно, я обычно использую команду «make bzImage modules».
Если захотите больше узнать о доступных целях *make*, можете выполнить «make help» и изучить вывод. Цели сборки по умолчанию отмечены звёздочкой (*).

Правка lilo.conf

Отредактируйте /etc/lilo.conf и добавьте новый раздел для нового ядра. Помните, если где-то была допущена ошибка, ваше новое ядро может даже не загрузиться, поэтому существующие разделы текущих ядер лучше оставить как есть. В вашем /etc/lilo.conf ближе к концу есть раздел наподобие:

```
image = /boot/vmlinuz
root = /dev/sda1
label = linux
read-only # He-UMSDOS файловые системы должны монтироваться только для чтения для их проверки
```

Добавьте следом другой раздел (добавление его ниже гарантирует, что ваше текущее – рабочее – ядро останется для загрузки по умолчанию):

```
image = /boot/vmlinuz-custom-2.6.37.6
root = /dev/sda1
label = newkernel
read-only # He-UMSDOS файловые системы должны монтироваться только для чтения для их проверки
```

После добавления раздела с новым ядром в /etc/lilo.conf нужно сохранить файл и затем выполнить lilo для активации изменений:

```
lilo
```

Теперь пора перезагрузиться и проверить новое ядро! Когда появится загрузочный экран lilo, выберите «newkernel» вместо «linux» по умолчанию.

Если ваше новое ядро загружается как положено, можно сделать его загружаемым по умолчанию, добавив следующую строку в начало /etc/lilo.conf и перезапустив «lilo»:

```
default = newkernel
```

Пакет Slackware kernel-headers

Вы решили собрать и использовать новое ядро. У вас может появиться вопрос, что делать с пакетом Slackware *kernel-headers*.

Ответ: **не удаляйте этот пакет!**

Заголовочные файлы ядра можно обнаружить в двух местах; одно – внутри каталога исходного кода ядра (в нашем случае – каталог `/usr/src/linux-2.6.37.6`), другое – `/usr/include/linux`. Пакет *kernel-headers* обычно содержит заголовочные файлы, взятые из исходного кода ядра Slackware по умолчанию. Именно эти заголовочные файлы были использованы при сборке пакета *glibc*. Тот факт, что пакет `kernel-headers` устанавливает эти файлы в `/usr/include/linux` делает их независимыми от заголовочных файлов в каталоге исходного кода ядра.

Пока не обновлён пакет `glibc` вы не должны обновлять или удалять соответствующий ему пакет `kernel-headers`.

- Как связаны пакеты `kernel-headers` и `glibc`?

В какой-то момент времени вы пожелаете обновить (перекомпилировать!) части программного обеспечения вашей системы. Если это ПО связано (слинковано) с glibc (как большинство основного ПО), его успешная компиляция зависит от наличия соответствующих заголовочных файлов ядра в `/usr/include/linux`. Использование совершенно иного ядра вместо поставляемых по умолчанию в Slackware не имеет значения. Пакет `kernel-headers` отражает состояние системы во время сборки `glibc`. Если удалить пакет `kernel-headers`, на работе системы это никаким образом не отразится, но вы не сможете (пере-)компилировать большинство ПО.

- Нужен ли исходный код ядра для чего-либо после того, как ядро уже собрано?

В предыдущем абзаце я говорил, что для компиляции системного ПО используются заголовочные файлы, расположенные в `/usr/include/linux`. Однако, дерево исходного кода ядра потребуется для сборки сторонних модулей ядра (madwifi, linux-uvc, ndiswrapper, … этот список конца не имеет). Вы не ограничены в компиляции драйвера только для загруженного в текущий момент ядра. Вы можете собирать драйверы для любых ядер до тех пор, пока на месте их дерево модулей (ниже `/lib/modules`) и исходный код.
Допустим, вы собираетесь собрать модуль для ядра, версию которого указали в переменной окружения $KVER. Например, выполнив

```
export KVER=2.6.38.2
```

Для компиляции драйвера вам потребуются заголовочные файлы этой версии ядра в `/lib/modules/$KVER/build/include/linux`. Символьная ссылка `/lib/modules/$KVER/build` создаётся при установке нового ядра и модулей.
Если вы после сборки ядра удалите его исходный код, вы потеряете возможность собирать драйверы, не входящие в состав ядра (сторонние драйверы).

Другие пакеты, содержащие модули ядра

Наверняка у вас будут установлены один или несколько пакетов, которые содержат модули, не входящие в состав используемого по умолчанию ядра. Например, Slackware устанавливает «svgalib-helper»; если вы установили драйверы беспроводной сети, то они, как правило, тоже

являются модулями ядра.

Имейте в виду, что после установки и загрузки нового ядра эти не входящие в состав ядра модули станут недоступны. Вам придётся перекомпилировать их исходный код для получения модулей, соответствующих версии нового ядра.

Получить список всех пакетов, содержащих модули для текущего ядра, можно при помощи команды (*её нужно выполнять при загруженном старом ядре*):

```
cd /var/log/packages
grep -l "lib/modules/$(uname -r)" *
```

Все перечисленные пакеты потребуют перекомпиляции, если вы хотите, чтобы их модули были пригодны и для нового ядра.

Если вы пересобрали пакет, содержащий модуль ядра, используйте для его установки не **upgradepkg**, а **installpkg**, который не станет удалять оригинальную версию модуля.

upgradepkg удалит модуль старого ядра, который ещё может вам понадобиться для перезагрузки со старым ядром. Этот трюк основан на предположении, что версия ядра является частью поля *ВЕРСИЯ* имени пакета, как здесь: svgalib_helper-1.9.25*_2.6.37.6*-i486-1.txz (Знаю, что пример ущербный, поскольку такого пакета больше нет).

Описанный выше метод **не имеет отношения** к модулям ядра, которые вы скомпилировали и установили вручную вместо создания пакетов для них. Иногда проприетарные графические драйверы, такие как от *Nvidia* или *Ati*, могут стать причиной для беспокойства, если забыть перекомпилировать их для нового ядра до запуска X Window... особенно если ваш компьютер по умолчанию загружается в графический *уровень исполнения (runlevel) 4*.

В этом случае перезагрузитесь в *уровень исполнения 3*, скачайте последнюю доступную версию графического драйвера и скомпилируйте/установите драйвер. Это позволит вам перезагрузиться в графический экран входа в систему. Для тех, кто забыл, загрузиться в другой уровень исполнения просто: когда отобразится экран LILO, наберите метку вашего ядра (в нашем примере выше это newkernel) и номер уровня исполнения: Newkernel Пробел 3 Ввод.

Создание initrd

Если в ядро не включены драйвер для вашей корневой файловой системы, или драйвер для шины SATA, или что-то ещё, собранное модулем, то ядро запаникует при загрузке, не имея возможности получить доступ к необходимым дискам, разделам и/или файлам. Обычно это

выглядит так

```
VFS: Cannot open root device "802" or unknown-block (8,2)
Please append a correct "root=" boot option
Kernel Panic-not syncing: VFS: unable to mount root fs on unknown block(8,2)
```

и означает, что вам необходимо собрать *initrd* (сокращение от «Initial Ram Disk») – диск в оперативной памяти для начальной инициализации, содержащий необходимые модули. Затем путь к initrd добавляется в соответствующий раздел `/etc/lilo.conf`, чтобы ядро при загрузке смогло его найти и загрузить драйверы для доступа к вашим дискам. Создать initrd довольно просто, ниже я покажу 2 варианта, один для случая, когда для корневого раздела используется файловая система Рейзера, другой – для ext3. Ниже я привожу команды для ядра версии 2.6.37.6, если версия вашего нового ядра отличается, измените номер версии в командах соответствующим образом.

- Перейдите в каталог `/boot`:

  ```
  cd /boot
  ```

- Выполните «`mkinitrd`» для создания файла `/boot/initrd.gz`, содержащего сжатую файловую систему с модулями, заданными в командной строке:

  ```
  mkinitrd -c -k 2.6.37.6 -m reiserfs
  ```

  для файловой системы Рейзера или

  ```
  mkinitrd -c -k 2.6.37.6 -m ext3
  ```

  если для корневого раздела вы используете файловую систему ext3.

- Добавьте строку «initrd = /boot/initrd.gz» в раздел newkernel файла `/etc/lilo.conf`, сохраните изменения и запустите `lilo`; я буду использовать ранее приведённый пример раздела `lilo.conf`:

  ```
  image = /boot/vmlinuz-custom-2.6.37.6
    root = /dev/sda1
    initrd = /boot/initrd.gz
    label = newkernel
    read-only # Не-UMSDOS файловые системы должны монтироваться только для
  чтения для их проверки
  ```

  затем запустите

  ```
  lilo
  ```

  При следующей загрузке ваше новое ядро паниковать перестанет.

- Если вы уже используете образ initrd с вашим текущим ядром, у вас есть два варианта:
  - Создать другой образ initrd при помощи показанной выше команды, указав имя создаваемого файла initrd (которое должно отличаться от используемого по

умолчанию, чтобы уже существующий файл не был перезаписан):

```
mkinitrd -c -k 2.6.37.6 -m ext3 -o /boot/initrd-custom-2.6.37.6.gz
```

и затем изменить раздел в `lilo.conf` подобным образом:

```
image = /boot/vmlinuz-custom-2.6.37.6
  root = /dev/sda1
  initrd = /boot/initrd-custom-2.6.37.6.gz
  label = newkernel
  read-only # Non-UMSDOS filesystems should be mounted read-only
for checking
```

- Добавить модули для вашего нового ядра в существующий файл initrd. Таким образом у вас получится один образ initrd, содержащий модули для нескольких ядер. Всё, что вам для этого потребуется сделать – это убрать параметр «`-c`», который предназначен для предварительной очистки каталога `/boot/initrd-tree`:

```
mkinitrd -k 2.6.37.6 -m ext3
```

Я написал сценарий оболочки (mkinitrd_command_generator.sh).
Скрипт ничего *не делает* с системой. Он *проанализирует* вашу работающую систему Slackware и покажет пример команды `mkinitrd`. Если вы выполните эту команду, будет создан образ initrd, который будет содержать все модули ядра и библиотеки, необходимые для загрузки системы с *generic* ядром Slackware.
Вот пример запуска команды и её вывода:

```
/usr/share/mkinitrd/mkinitrd_command_generator.sh /boot/vmlinuz-
generic-2.6.37.6
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command
to use
# in case you require an initrd image to boot a kernel that does
not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:

mkinitrd -c -k 2.6.37.6 -f ext3 -r cryptslack -m
mbcache:jbd:ext3 -C /dev/sda8 -u -o /boot/initrd.gz
# An entry in 'etc/lilo.conf' for kernel '/boot/vmlinuz-
generic-2.6.37.6' would look like this:
# Linux bootable partition config begins
# initrd created with 'mkinitrd -c -k 2.6.37.6 -f ext3 -r
cryptslack -m mbcache:jbd:ext3 -C /dev/sda8 -u -o
/boot/initrd.gz'
```

```
image = /boot/vmlinuz-generic-2.6.37.6
  initrd = /boot/initrd.gz
  root = /dev/mapper/cryptslack
  label = 2.6.37.6
  read-only
# Linux bootable partition config ends
```

Вы можете отметить, что мой зашифрованный по LUKS корневой раздел был опознан.
Этот сценарий включён в пакет *mkinitrd* начиная с выпуска Slackware 12.2.

Подгрузка модулей при загрузке

До Slackware 11.0 модули для вашего ядра загружались либо подсистемой горячего подключения (hotplug), либо командами modprobe в файле /etc/rc.d/rc.modules. Наличие общего файла rc.modules для ядер версий 2.4.x и 2.6.x не было оптимальным решением.

В Slackware 12.0 и последующих более недоступны ядра 2.4. Загрузка модулей ядра обеспечивается udev и командами `modprobe`: модули, не загружаемые udev, по прежнему могут быть прописаны в файле `rc.modules`. Только теперь этих файлов может быть больше одного. Slackware проверяет существование нижеследующих исполняемых файлов в таком порядке:

- Если существует `/etc/rc.d/rc.modules.local`, он будет запущен
- Иначе, если существует `/etc/rc.d/rc.modules-$(uname -r)`, он будет запущен
- Иначе, если существует `/etc/rc.d/rc.modules`, он будет запущен

$(uname -r) – это версия текущего ядра. Если версия вашего ядра `2.6.37.6-smp`, то Slackware будет проверять наличие файла `/etc/rc.d/rc.modules-2.6.37.6-smp`. Таким образом, возможно наличие специальных файлов *rc* для разных ядер, что позволяет настраивать вашу систему оптимальным образом.

В Slackware 13.37 пакет `/slackware/a/kernel-modules-smp-2.6.37.6_smp-i686-1.txz` устанавливает файл `/etc/rc.d/rc.modules-2.6.37.6-smp`. Вы можете использовать его в качестве примера на случай, если захотите собрать своё ядро, требующее загрузки отдельных модулей посредством команд `modprobe`.

Если вы решите собрать своё ядро 2.6 из исходного кода, вас может обеспокоить отсутствие файла `/etc/rc.d/rc.modules-$(uname -r)` – вы должны создать его самостоятельно. Обычно `rc.modules` – это символьная ссылка на `rc.modules-2.6.37.6-smp`. Типичный результат отсутствия файла `rc.modules` для вашего ядра – неработающая мышь. Считайте такое поведение намёком на необходимость создания файла `rc.modules`! Вы можете взять полную копию любого существующего файла `rc.modules-2.6.xx`. Если в вашей системе нет файла rc для ядер версии 2.6, вы можете позаимствовать его на диске Slackware, например: /source/k/kernel-modules-smp/rc.modules.new. Допустим, вы решили собрать новое ядро с версией 2.6.38.2.alien, и у вас

установлено ядро Slackware 2.6.37.6-smp:

```
cp        -a        /etc/rc.d/rc.modules-2.6.37.6-smp
/etc/rc.d/rc.modules-2.6.38.2.alien
```

Файл /etc/rc.d/rc.modules-2.6.38.2.alien будет использован при загрузке вашего нового ядра 2.6.38.2.alien.

Подпись GPG

Архивы исходного кода ядра подписаны ключом OpenPGP «Linux Kernel Archives Verification Key» (проверочный ключ архивов ядра Линукс). Это позволяет удостовериться, что загруженный вами исходный код является оригинальным архивом и не был подделан. В этой главе описан способ такой проверки.

- Сначала импортируйте ключ OpenPGP в GnuPG либо скопировав ключ со страницы подписи или импортировав его с сервера ключей. Идентификатор ключа ядра 0x517D0F0E. Пример выглядит следующим образом:

```
gpg --keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
```

Вывод будет наподобие такого:

```
gpg: key 517D0F0E: public key "Linux Kernel Archives Verification Key
<ftpadmin@kernel.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

- Затем загрузите файл подписи для загруженного архива ядра:

```
wget
http://www.us.kernel.org/pub/linux/kernel/v2.6/linux-2.6.37.6.tar.bz2.s
ign
```

и убедитесь, что он находится в том же каталоге, что и архив ядра.

- На последнем шаге запустите gpg на этом файле подписи и проверьте, что он выведет:

```
gpg --verify linux-2.6.37.6.tar.bz2.sign linux-2.6.37.6.tar.bz2
```

Вывод будет как здесь:

```
gpg: Signature made Mon Mar 28 01:08:08 2011 CEST using DSA key ID
517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key
<ftpadmin@kernel.org>"
```

```
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the
owner.
Primary key fingerprint: C75D C40A 11D7 AF88 9981  ED5B C86B A06A 517D
0F0E
```

Если бы вы указали gnupg доверять этому ключу, завершающая часть выглядела бы иначе. По мне, добавление этого ключа в список доверенных не имеет практического смысла, если только вы не встретились с одним из разработчиков ядра, у которого при себе был ключ и который мог предъявить заслуживающие доверия полномочия.

Тем не менее, архив с исходным кодом действительно подписан ключом, который вы только что импортировали. А это хорошая новость.

Переводы

- Настоящий перевод на русский: Serg Bormant
- Перевод этой статьи на испанский от Jorge Courbis: http://coredump.cl/wiki/doku.php?id=compilaci%C3%B3n_de_kernel

Источники

- Оригинал: http://alien.slackbook.org/dokuwiki/doku.php?id=linux:kernelbuilding
- Автор: Eric Hameleers

howtos, software, kernel, author alienbob, translator bormant