Ядро / традиционная компиляция

< Ядро

Эта статья представляет собой введение в построение пользовательских ядер из **kernel.org источники** . Этот метод компиляции ядер является традиционным методом, общим для всех дистрибутивов. Это может быть, в зависимости от вашего фона, более сложным, чем с помощью **ядра/арки системы сборки** . Рассмотрим **инструменты системы Arch Build** разработаны и поддерживаются, чтобы сделать повторяемые задачи компиляции эффективными и безопасными.

Содержание

Подготовка

Нет необходимости (или рекомендуется) использовать учетную запись root или привилегии root (т. е. через **Sudo**) для подготовки ядра.

Установите основные пакеты

Установите группу **пакетов base-devel (https://www.archlinux.org/groups/x86_64/base-devel/)**, содержащую необходимые пакеты, такие как **make (https://www.archlinux.org/packages/?name=make)** и **gcc (https://www.archlinux.org/packages/?name=gcc)**. Также рекомендуется установить следующие пакеты, как указано в стандартном Arch kernel **PKGBUILD (https://projects.archlinux.org/svntogit/packages.git/tree/trunk/PKGBUILD?h=packages/linux)**: **xmlto (https://www.archlinux.org/packages/?name=xmlto)**, **kmod (https://www.archlinux.org/packages/?name=kmod)**, **inetutils (https://www.archlinux.org/packages/?name=inetutils)**, **bc (https://www.archlinux.org/packages/?name=bc)**, **libelf (https://www.archlinux.org/packages/?name=libelf)**, **git (https://www.archlinux.org/packages/?name=git)**

Создание каталога компиляции ядра

Рекомендуется создать отдельный каталог сборки для вашего ядра(ядер). В этом примере каталог `kernelbuild` будет создан в самом `home` каталоге:

```
$ mkdir ~ / kernelbuild
```

Загрузите исходный код ядра

Предупреждение: **systemd** требует наличия ядра версии 3.12 по меньшей мере (4.2 или выше для поддержки иерархии унифицированных контрольных групп). Смотрите `/usr/share/doc/systemd/README` дополнительную информацию.

Загрузите исходный код ядра из **https://www.kernel.org** . это должен быть файл **tarball** (`tar.xz`) для выбранного ядра.

Он может быть загружен простым щелчком правой кнопкой мыши `tar.xz` по ссылке в вашем браузере и выбором `Save Link As...` или любым другим количеством способов с помощью альтернативных графических инструментов или инструментов командной строки, которые используют HTTP , **TFTP** , **Rsync** или **Git** .

Примечание: это хорошая идея, чтобы проверить подпись PGP любого загруженного tar-файла ядра. Это гарантирует, что он является законным и помогает построить паутину доверия. Смотрите **kernel.org/signature -да (https://kernel.org/signature.html#using-gnupg-to-verify-kernel-signatures)**.

В следующем примере командной строки **wget (https://www.archlinux.org/packages/?name=wget)** был установлен и используется внутри `~/kernelbuild` каталога для получения ядра 4.8.6:

```
$ cd ~ / kernelbuild
$ wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.8.6.tar.xz
```

Вы также должны проверить правильность загрузки, прежде чем доверять ему. Сначала возьмите подпись, а затем используйте ее, чтобы захватить отпечаток ключа для подписи, а затем используйте отпечаток для получения фактического ключа для подписи:

```
$ wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.8.6.tar.sign
$ gpg --list-packets linux-4.8.6.смола.знак
$ gpg --recv-keys <fingerprint-from-previous-step>
```

Примечание. подпись была сгенерирована для архива tar (т. е. расширения `.tar`), а не для сжатого `.tar.xz` файла, который вы загрузили. Вам нужно распаковать последний, не распутывая его. Убедитесь, что у вас установлен **`xz` (https://www.archlinux.org/packages/?name=xz)**, а затем вы можете продолжить так:

```
$ unxz linux-4.8.6.смола.xz
$ gpg --проверка linux-4.8.6.смола.подписать linux-4.8.6.смола
```

Не продолжайте, если это не приводит к выходу, который включает строку "хорошая подпись".

Если `wget` он не был использован внутри каталога сборки, то необходимо будет переместить в него tar-шар, например

```
$ mv / path/to / linux-4.8.6.смола.xz ~ / kernelbuild/
```

Распакуйте исходный код ядра

В пределах каталога сборки распакуйте архив ядра:

```
$ tar-xvJf linux-4.8.6.смола.xz
```

Чтобы завершить подготовку, убедитесь, что дерево ядра абсолютно чистое; не полагайтесь на то, что исходное дерево будет чистым после распаковки. Для этого сначала перейдите в новый исходный каталог ядра, созданный, а затем выполните `make mrproper` команду:

```
$ cd linux-4.8.6/
$ make mrproper
```

Примечание: цель `mrproper` Make зависит от `clean` цели, и таким образом, нет необходимости выполнять оба. Смотрите **[1] (https://unix.stackexchange.com/q/387640)**Для справки.

Конфигурация

Это самый важный шаг в настройке ядра по умолчанию, чтобы отразить точные спецификации вашего компьютера. Конфигурация ядра задается в его `.config` файле, который включает в себя использование **модулей ядра** .

Примечание: На этом этапе нет необходимости использовать учетную запись root или привилегии root.

Правильно настроив параметры `.config` , ваше ядро и компьютер будут функционировать наиболее эффективно.

Конфигурация ядра

Вы можете выбрать один из двух вариантов настройки вашего ядра:

- O. используйте настройки Arch по умолчанию из официального ядра (рекомендуется)
- B. создайте файл конфигурации, который соответствует конфигурации текущего запущенного ядра. (полезно, если вы хотите дополнительно настроить параметры ядра)

Примечание: особенно если вы выбрали опцию * * B**, вам будет предложено вручную настроить ядро с помощью инструментов, описанных в `Advanced Configuration` разделе.

A. конфигурация свода по умолчанию

Этот метод создаст `.config` файл для пользовательского ядра, используя параметры ядра Arch по умолчанию. Если выполняется ядро со стандартной аркой, то в исходном каталоге пользовательского ядра можно использовать следующую команду:

```
$ zcat / proc / config.gz > .конфиг
```

В противном случае конфигурацию по умолчанию можно найти в интернете в **официальном пакете ядра Arch Linux (https://git.archlinux.org/svntogit/packages.git/tree/trunk/config?h=packages/linux)**.

Совет: при обновлении ядер некоторые параметры могут быть изменены или удалены. В этом случае при запуске `make` в разделе **#Compilation** вам будет предложено предоставить ответы на все параметры конфигурации , которые изменились между версиями. Чтобы принять значения по умолчанию без запроса, выполните `make olddefconfig` команду .

Предупреждение: если вы компилируете ядро с помощью вашего текущего `.config` файла, не забудьте переименовать вашу версию ядра "CONFIG_LOCALVERSION" в новой `.config` или в *общей установке* > *локальная версия - добавить к выпуску ядра*>, используя один из пользовательских интерфейсов, перечисленных в разделе **#Advanced configuration** . Если вы пропустите это, есть риск перезаписи одного из существующих ядер по ошибке.

B. сгенерированная конфигурация

Совет: подключите все устройства, которые предполагается использовать в системе при использовании этого метода.

Начиная с ядра 2.6.32, `localmodconfig` команда создаст `.config` файл для пользовательского ядра, отключив все параметры, *которые в данный момент не используются работающим ядром* . Другими словами, он будет включать только те параметры, которые используются в настоящее время.

Несмотря на то, что этот минималистский подход приведет к очень упрощенной и эффективной конфигурации, адаптированной специально для вашей системы, есть недостатки, такие как потенциальная неспособность ядра поддерживать более новое оборудование, периферийные устройства или другие функции.

Примечание. снова убедитесь, что все устройства, которые вы собираетесь использовать, были подключены к системе (и обнаружены ею) перед выполнением следующей команды

```
$ make localmodconfig
```

Дополнительная настройка

Совет: Если вы не хотите видеть много дополнительных сообщений при загрузке и завершении работы с пользовательским ядром, рекомендуется отключить соответствующие параметры отладки.

Существует несколько инструментов, доступных для точной настройки конфигурации ядра, которые предоставляют альтернативу тому, чтобы в противном случае тратить часы на ручную настройку всех и каждого из параметров, доступных во время компиляции.

Примечание: эти инструменты, перечисленные ниже, предоставят вам три варианта конфигурации для каждой функции ядра: `y` для включенного, `n` для отключенного и `m` для включенного как модуль ядра (загружается при необходимости).

Эти инструменты являются:

- `make menuconfig` : Интерфейс командной строки ncurses заменен на `nconfig`
- `make nconfig` : Новый интерфейс ncurses для командной строки
- `make xconfig` : Удобный графический интерфейс, который требует **установки packagekit-qt5 в (https://www.archlinux.org/packages/?name=packagekit-qt5)** качестве зависимости. Это рекомендуемый метод-особенно для менее опытных пользователей - так как он легче ориентируется, и информация о каждом параметре также отображается.
- `make gconfig` : Графическая конфигурация аналогична xconfig, но с использованием gtk.

Выбранный метод должен быть запущен в исходном каталоге ядра, и все они либо создадут новый `.config` файл, либо перезапишут существующий, где он присутствует. Все дополнительные конфигурации будут автоматически включены, хотя любые новые параметры конфигурации (например, с более старым ядром `.config`) могут быть выбраны не автоматически.

После внесения изменений сохраните `.config` файл. Это хорошая идея, чтобы сделать резервную копию вне исходного каталога. Вам может потребоваться сделать это несколько раз, прежде чем вы получите все варианты правильно.

Если вы не уверены, измените только несколько параметров между компиляциями. Если вы не можете загрузить свое новое ядро, смотрите список необходимых элементов конфигурации **здесь (https://www.archlinux.org/news/users-of-unofficial-kernels-must-enable-devtmpfs-support/)**.

Запуск `$ lspci -k #` из liveCD перечисляет имена используемых модулей ядра. Самое главное, вы должны поддерживать поддержку контрольных групп. Это необходимо для **systemd** .

Сборник

Совет: если вы хотите, чтобы **gcc (https://www.archlinux.org/packages/?name=gcc)** оптимизировался для наборов инструкций вашего процессора, отредактируйте `arch/x86/Makefile` (как для 32, так и для 64 бит, см. **[2] (https://lkml.org/lkml/2007/7/20/447)**) в исходном каталоге ядра:

- Ищите `CONFIG_MK8,CONFIG_MPSC,CONFIG_MCORE2,CONFIG_MATOM,CONFIG_GENERIC_CPU` то, что вы выбрали в `Processor type and features > Processor Family`
- Измените флаг CC-options вызова с `-march=native` на тот, который вы выбрали в семействе процессоров, например `cflags-$(CONFIG_MK8) += $(call cc-option,-march=native)` . Это, вероятно, лучший способ компиляции с `-march=native` тем, как это работает.
- Примечание: Для 32-битных ядер, вам нужно отредактировать `arch/x86/Makefile_32.cpu` вместо этого и установить `-march=native` для вашего процессора.

Время компиляции будет варьироваться от всего лишь пятнадцати минут до более чем часа, в зависимости от конфигурации ядра и возможностей процессора. После `.config` того, как файл был настроен для пользовательского ядра, в исходном каталоге выполните следующую команду для компиляции:

```
$ делать
```

Совет: чтобы скомпилировать быстрее, *make* можно запустить с `-jX` аргументом, где `X` находится целое число параллельных заданий. Наилучшие результаты часто достигаются при использовании количества ядер процессора в машине; например ,при запуске 2- `make -j2` х ядрового процессора. **Дополнительную информацию смотрите в разделе Makepkg#улучшение времени компиляции**.

Установка

Предупреждение: начиная с этого шага, команды должны быть выполнены либо от имени пользователя root, либо с привилегиями пользователя root. Если нет, то они потерпят неудачу.

Установить модуль

После того, как ядро было скомпилировано, модули для него должны следовать. От имени пользователя root или с привилегиями root выполните следующую команду:

```
# make modules_install
```

Это позволит скопировать скомпилированные модули `/lib/modules/<kernel version>-<config local version>` В. Например, для версии ядра 4.8, установленной выше, они будут скопированы `/lib/modules/4.8.6-ARCH` . При этом модули для отдельных используемых ядер остаются разделенными.

Совет: Если вашей системе требуются модули, которые не распространяются с обычным ядром Linux, вам нужно скомпилировать их для вашего пользовательского ядра, когда он будет завершен. Такие модули обычно являются теми, которые вы явно установили отдельно для своей работающей системы. Смотрите **пример NVIDIA#Custom kernel**.

Скопируйте ядро в каталог / boot

Примечание: убедитесь, что файл `bzImage` ядра был скопирован из соответствующего каталога для вашей системной архитектуры. Смотреть ниже.

Процесс компиляции ядра будет генерировать сжатый `bzImage` (большой zImage) этого ядра, который должен быть скопирован в `/boot` каталог и переименован в процессе. При условии, что имя имеет префикс `vmlinuz-` , вы можете назвать ядро так, как вы хотите. В приведенных ниже примерах установленное и скомпилированное ядро версии 4.8 было скопировано и переименовано в `vmlinuz-linux48` :

- 32-bit (i686) kernel:

```
# cp -v arch/x86/boot/bzImage /boot/vmlinuz-linux48
```

- 64-битное ядро (x86_64) :

```
# CP-v arch/x86_64 / boot/bzImage / boot/vmlinuz-linux48
```

Сделать начальный RAM диск

Примечание: Вы можете назвать файл изображения initramfs все, что вы хотите при его создании. Однако рекомендуется использовать `linux<major revision><minor revision>` конвенцию. Например, имя 'linux48' было дано как '4' является основной редакцией, а '8' является второстепенной редакцией ядра 4.8. Эта конвенция позволит упростить обслуживание нескольких ядер, регулярное использование mkinitcpio и сборку сторонних модулей.

Совет: Если вы используете загрузчик LILO, и он не может взаимодействовать с драйвером устройства ядра-mapper, вы должны сначала запустить `modprobe dm-mod` .

Если вы не знаете, что такое создание начального RAM-диска, см. **Initramfs в Википедии** и **mkinitcpio** .

Автоматизированный заранее поставленный метод

Существующий **набор настроек mkinitcpio** может быть скопирован и изменен таким образом, что пользовательские образы ядра initramfs могут быть созданы таким же образом, как и для официального ядра. Это полезно, когда вы собираетесь перекомпилировать ядро (например, при обновлении). В приведенном ниже примере предустановленный файл для ядра stock Arch будет скопирован и изменен для ядра 4.8, установленного выше.

Во-первых, скопируйте существующий предустановленный файл, переименовав его в соответствии с именем пользовательского ядра, указанного в качестве суффикса к `/boot/vmlinuz-` При копировании `bzImage` (в этом случае, `linux48`):

```
# cp / etc / mkinitcpio.d / linux.preset /etc / mkinitcpio.d / linux48.предустановка
```

Во-вторых, отредактируйте файл и внесите изменения в пользовательское ядро. Обратите внимание (еще раз), что этот `ALL_kver=` параметр также соответствует имени пользовательского ядра, указанного при копировании `bzImage` :

```
/ etc / mkinitcpio.d / linux48.предустановка
```

```
...
ALL_kver= " / boot / vmlinuz-linux48"
...
default_image=" / boot/initramfs-linux48.HBФ"
...
fallback_image=" / boot/initramfs-linux48-fallback.HBФ"
```

Наконец, создайте образы initramfs для пользовательского ядра таким же образом, как и для официального ядра:

```
# mkinitcpio-P linux48
```

Ручной метод

Вместо того, чтобы использовать предустановленный файл, mkinitcpio также может быть использован для создания файла initramfs вручную. Синтаксис команды таков:

```
# mkinitcpio-k <kernelversion> - g/boot / initramfs-<kernelversion><имя файла>.HBФ
```

- `-k` (--kernel <kernelversion>): указывает модули, используемые при создании образа initramfs. Это `<kernelversion>` имя будет совпадать с именем исходного каталога пользовательского ядра (и каталога модулей для него, расположенного в `/usr/lib/modules/`).
- `-g` (--generate <filename>): указывает имя файла initramfs для создания в каталоге <filename> `/boot` . Опять же, рекомендуется использовать соглашение об именовании, упомянутое выше.

Например, команда для пользовательского ядра 4.8, установленного выше, была бы:

```
# mkinitcpio-k linux-4.8.6-g / boot / initramfs-linux48.HBФ
```

Система Копирования.Карта

Этот `System.map` файл не требуется для загрузки Linux. Это своего рода" телефонный справочник " список функций в конкретной сборке ядра. Он `System.map` содержит список символов ядра (т. е. имена функций, имена переменных и т. д.) и их соответствующие адреса. Этот "символ-имя для сопоставления адресов" используется:

- Некоторые процессы, такие как klogd, ksymoops и т. д
- По обработчику OOPS, когда информация должна быть сброшена на экран во время сбоя ядра (т. е. информация, например, в какой функции он разбился).

Совет: разделы UEFI форматируются с использованием FAT32, который не поддерживает символьные ссылки.

Если вы `/boot` используете файловую систему, поддерживающую символьные ссылки (т. е. не FAT32), скопируйте `System.map` ее в `/boot` , добавив имя вашего ядра в файл назначения. Затем создайте символическую `/boot/System.map` ссылку от to до `/boot/System.map-YourKernelName` :

```
# система ср.карта / загрузка / система.карта-YourKernelName
# ln-sf / boot / System.map-YourKernelName / boot / System.Карта
```

После выполнения всех описанных выше действий, вы должны иметь следующие 3 файла и 1 программную символическую ссылку в вашем `/boot` каталоге вместе с любыми другими ранее существующими файлами:

- Ядро: `vmlinuz-YourKernelName`
- Initramfs: `Initramfs-YourKernelName.img`
- системная карта: `System.map-YourKernelName`
- Системная карта ядро symlink

Конфигурация загрузчика

Добавьте запись для вашего нового ядра в конфигурационный файл загрузчика. Смотрите **Arch boot process#Feature** comparation для возможных загрузчиков, их вики-статьи и другую информацию.

Совет: исходные файлы ядра включают скрипт для автоматизации процесса для **LILO** : `$ arch/x86/boot/install.sh` . Не забудьте ввести `lilo` как root в приглашении обновить его.

Смотреть также

- **https://cateee.net/lkddb/web-lkddb/** содержит полный список строк конфигурации ядра и сопроводительный текст, отсортированных в алфавитном порядке.