Роман Химов

Udev в разрезе

Структура устаревшей devfs, с точки зрения разработчиков, слишком громоздкая, запутанная и не очень удобная. Поэтому вполне логично, что на смену этой технологии пришла новая — udev. Как она работает и что дает конечному пользователю — об этом вы узнаете в этой статье.

Проблемы /dev и devfs

Концепция devfs, динамически создающей файлы виртуальных устройств, для своего времени выглядела очень привлекательно, однако практическая реализация была не очень удачной и в конечном итоге завела эту технологию в тупик. Если вы заглянете в статичную /dev, то увидите там около 18 000 файлов (с подкаталогами) — ну и где же там, к примеру, флеш-драйв? А еще, вам не кажется, что это довольно много, у вас ведь нет 18 000 физических/логических устройств? Или, наоборот, мало, если вы хотите построить систему с 10 000 SCSI-дисков! Наверное, было бы хорошо, если бы в /dev находилось только то, что реально присутствует в системе. Действительно, это очень здорово, и это есть, например, в FreeBSD — файлы устройств создаются динамически, /dev становится вполне читаемой как для пользователя, так и для программ. Аналогичная подсистема — devfs — есть и в Linux, казалось бы, какие еще проблемы?!

Однако они есть. Во-первых, devfs реально утяжеляет и усложняет ядро Linux, а это не любят ни разработчики, ни пользователи. А как быть с тем, что ее давно никто не хочет поддерживать (кстати, за все время существования в ядре она пребывала либо в состоянии EXPERIMENTAL — 2.4.x, либо OBSOLETE — 2.6.x? А это еще что такое: devfs, оказывается, не совместима с LSB, когда почти все (по крайней мере из тех, что на виду) дистрибутивы стараются обеспечить совместимость с этим стандартом! Идем дальше — вернемся к внешним USB-устройствам хранения данных: покуда у вас одно такое устройство, все очень просто, /dev/sda* дает нам все, что надо, но если у вас их два? Вас никогда не смущал тот факт, что приходится запоминать, в каком порядке вы подключали флеш-накопители и внешние винчестеры, чтобы знать, что есть /dev/sda, а что /dev/sdb, /dev/sdc и т. д.? А с принтерами путаться не приходилось? Плюс ко всему, чисто с точки зрения архитектуры, разве заниматься именованием устройств — это дело ядра? Его дело как раз в номерах major/minor, эти термины вообще оттуда и идут, имена ему совершенно безразличны. И, кстати, зачем нам, пользователям, знать номера major/minor?

Страшная картина вокруг, казалось бы, простой вещи — названий устройств, не так ли? Не совсем, конечно, так. Например, для серверов это, в общем-то, не играет никакой роли — конфигурация там одна и меняется крайне редко, там главное — чтобы действительно работало. Но мы-то с вами обычные пользователи настольного Linux и хотим простых и понятных удобств. Неужели все эти проблемы не разрешимы?

Решение для /dev

Давайте сначала посмотрим на еще одно новшество ядер 2.6.x — файловую систему sysfs. Туда экспортируется не просто масса, а прямо-таки тонны информации об устройствах — расположение на шинах, атрибуты вроде названий, классов устройств, серийных номеров и т. д., а также номера major и minor. То есть, оказывается, об устройствах мы легко все можем узнать и из пользовательского пространства! Теперь вспомним еще одну интересную вещь — механизм hotplug, которым ядра оснащаются еще начиная с ветки 2.4.x. В /proc/sys/kernel/hotplug можно посмотреть/изменить путь к одной маленькой, но очень важной программе, как минимум для настольной Linux-машины. Эта программа будет получать от ядра события, которые как раз касаются изменений в аппаратных конфигурациях. Тем самым мы и приходим к тому, что на самом деле вся информация, необходимая и даже избыточная для формирования /dev, уже давным-давно есть в пользовательском пространстве! Зачем тогда терзать ядро невнятными подсистемами вроде devfs?

Действительно незачем, знакомьтесь, наш герой — udev. Он органично сочетает возможности вышеописанных особенностей ядра Linux и дает нам то, что мы так хотели, — динамичный /dev! Кстати, а как он это делает?

Как это работает

Давайте проследим путь с самого начала — от ядра. Ядро видит появление нового устройства и запускает то, что мы прописали ему в /proc/sys/kernel/hotplug, передавая через переменные окружения все, что требуется hotplug для успешного выполнения своих функций. Это $ACTION, показывающий, что же случилось: добавили, удалили устройство или что-либо еще; $SUBSYSTEM, несущий в себе информацию о том, какая подсистема сгенерировала событие; $DEVPATH, содержащий очень важную вещь — путь к каталогу устройства в /sys; интересный параметр $SEQNUM — номер события, позволяющий упорядочить их и обрабатывать по очереди (события же могут доставляться непоследовательно), а также некоторые другие параметры, специфичные уже для конкретных подсистем.

Дело hotplug — загрузить нужные модули устройств, выполнить, возможно, их настройку и сообщить о событии udev. Сам по себе hotplug сегодня выполнен как солидный и весьма запутанный набор shell-скриптов. Однако Грег Кроа-Хартман, его автор, уже работает на hotplug-ng, выполненном на C, поскольку, хоть это и нормальный объем для shell-скриптов, он все же затормаживает загрузку системы, когда «симулянт событий» coldplug генерирует массу псевдособытий, но это к слову.

Вернемся к udev, застав его в момент получения события от hotplug. Стоп, какой такой udev получил событие от hotplug и как? Сам по себе udev тоже раскладывается на составные части в виде приложений udevsend и udev, а также демона udevd. Зачем потребовался демон? Вспомним, события могут приходить в hotplug не в том порядке, в каком они реально происходили, — это явно требует кого-то, кто бы следил за порядком событий, поскольку, если событие remove придет перед add, наверное, что-то здесь не так, но что с этим делать udev? Вот и получается, что udevd упорядочивает приходящие запросы так, чтобы они обрабатывались строго по мере появления.

Кто-то уже наверняка догадался, что udevsend здесь является связующим звеном, именно его вызывает hotplug (передавая параметры через переменные окружения), а udevsend уже знает, как доставить событие udevd. Зачем же тогда приложение udev? Все дело в том, что непосредственную работу по созданию файлов устройств выполняет именно оно, udevd не занимается этим напрямую — его дело следить за порядком. Вот так, через hotplug, udevsend, udevd и udev, мы приходим к самому натуральному динамическому /dev!

Ну и что, разве мы не видели этого в devfs, к тому же без лишних приложений/демонов? Да, но не в таком виде. Одна из самых впечатляющих возможностей данного подхода — реализация абсолютно любой схемы именования устройств, ведь теперь она никак не навязывается нам ядром, и мы знаем все о наших устройствах! Хотите назвать ваш флеш-драйв /dev/my-dear-flash — пожалуйста! Все это потому, что Грег Кроа-Хартман, который и реализовал тот udev, что мы знаем (а ведь возможны и другие реализации, например его можно тесно совместить с hotplug в какой-нибудь маленький бинарный файл на C — это подойдет для встраиваемых систем), делал его с прицелом на настольные применения и максимальную гибкость в конфигурации. А что у нас с конфигурацией? Давайте посмотрим.

Конфигурация

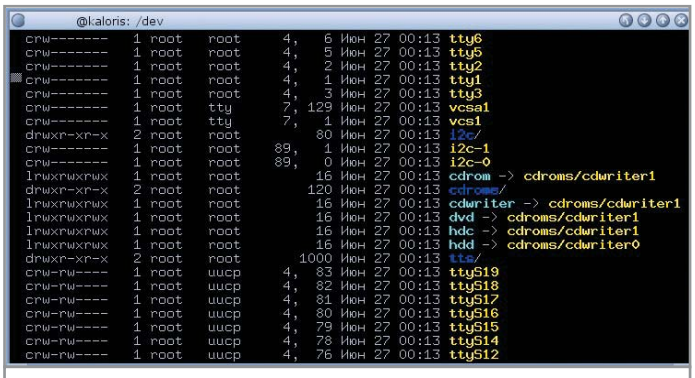
Udev имеет основной конфигурационный файл /etc/udev/udev.conf, в котором задается расположение каталога устройств udev_root (а ведь можно попробовать сделать так, чтобы это был не /dev!); местоположение базы данных udev (udev_db); каталог с правилами udev (udev_rules); каталог с файлами, описывающими права пользователей на устройства (udev_permissions), права по умолчанию (default_mode), а также маленький параметр, включающий/выключающий логирование событий udev (udev_log).

Все, что касается интересных моментов udev, вынесено в каталог правил. Это стандартный /etc/udev/rules.d, хотя встречаются вариации. А там чаще всего вы обнаружите один файл с интересным названием 50-udev.rules. Наверняка кто-то уже понял, что 50 — это неспроста, и будет прав: udev обрабатывает файлы правил (которые, кстати, должны обязательно оканчиваться на .rules) в лексикографическом порядке, то есть в том, который для нас вполне привычен и, как только находит подходящее правило для произошедшего события, останавливается. Соответственно, вряд ли стоит запускать руки в стандартный файл правил, можно просто создать какой-нибудь 10-vasya.rules и описывать свои правила там: меньше вероятность испортить стандартные правила.

Файл состоит из ключей, а ключи разделяются запятыми. С помощью идентификационных ключей мы выделяем устройства, обязанные подчиняться этому правилу, а с помощью назначающих указываем, что с ними надо делать в /dev; любое правило должно иметь как минимум один ключ каждого типа.

Ниже приводится список идентификационных ключей.

- BUS — шина, к которой подключено устройство, например PCI, USB или SCSI.
- KERNEL — имя устройства, данное ядром.
- ID — ID устройства на шине; например, для PCI это будет номер шины, а для USB — ID устройства.
- PLACE определяет место в топологии шины, например физический порт, в который включено USB-устройство.
- SYSFS_имя-файла, или SYSFS{имя-файла}, — проверка устройства по любому атрибуту из тех, что видны в /sys: это могут быть метки, вендоры, серийные номера, объемы и многое другое. Таких параметров в одном правиле можно задать до пяти штук.
- PROGRAM — еще одна мощная возможность — вызов внешней программы и проверка результата.



Часть структуры /dev

▸ RESULT: как раз через этот ключ можно сверять результат (возвращенную строку) вызова программы. Он должен следовать за вызовом PROGRAM.

Для успешного срабатывания правила необходимо, чтобы совпали все идентификационные ключи для рассматриваемого устройства.

Ключи целей:

▸ NAME: задает имя создаваемого файла устройства.

▸ SYMLINK: создает символическую ссылку на созданный файл устройства.

▸ OWNER, GROUP, MODE — стандартные владелец, группа и режим доступа к устройству соответственно.

В правилах можно применять подстановки в стиле printf:

▸ %n — номер, который дается устройству ядром; к примеру, для «sda3» это будет 3;

▸ %k — предлагаемое ядром имя устройства (в примере, указанном выше — «sda3»);

▸ %M — major-номер устройства;

▸ %m — minor-номер устройства;

▸ %b — ID устройства на шине;

▸ %c — результат выполнения программы, заданной через ключ PROGRAM; можно выбрать определенную часть этого результата, если они разделены пробелами, это делается через %c{N}, где N — номер части, а с помощью %c{N+} можно выбрать содержимое строки от начала N-го слова до конца строки;

▸ %s{имя-файла} — содержимое атрибута из /sys;

▸ %e — выдает минимальное целое N, если устройство с таким именем уже существует, после чего результирующее имя уже не будет совпадать с другим устройством; полезно при расстановке символических ссылок с целью обеспечения совместимости, а также нумерации устройств одного типа из разных подсистем ядра;

▸ %% — символ «%».

Кроме того, различные поля поддерживают шаблоны имен в стиле shell, то есть стандартные «*», «?» и «[]». Не многовато ли? Нормально, и сейчас вы в этом убедитесь сами.

Итак, посмотрим на простейшее правило udev:

```
KERNEL="fd[0-9]*", NAME="floppy/%n", SYMLINK="%k"
```

Это тривиальное правило работает для флоппи-дисков, которым ядро как раз и предлагает имя fd{номер}, оно создает реальный файл устройства в каталоге (/dev/)floppy/ с простым числовым именем, а также символическую ссылку на него в виде привычного fd{номер}. Скучно? Конечно же, но давайте посмотрим дальше:

```
KERNEL="[hs]d[a-z]", PROGRAM="name_cdrom.pl %M %m",
NAME="%c{1}", SYMLINK="cdrom"
```

Этот пример приводил сам Грег Кроа-Хартман, самое большое его достоинство в скрипте name_cdrom.pl, который сперва определяет, является ли то устройство, которое ему дали, CD-ROM, а потом запрашивает по сети базу данных CDDB в поисках соответствующих аудиодиску исполнителя и названия альбома, после чего возвращает все это, а мы создаем файл устройства с таким именем (исполнитель-название) и ссылку cdrom на него! Впечатляет, не так ли? Этот маленький скрипт входит в стандартную поставку udev — можете посмотреть сами. А вот еще один пример, уже более близкий к жизни:

```
SYSFS{vendor}="TOSHIBA", SYSFS{model}="MK6021GAS",
NAME="toshiba-drive"
```

Подобным образом представляется в /sys маленький внешний USB-винчестер 2,5", этим правилом мы его отлавливаем и даем имя «toshiba-drive». Что интересно — хотя у нас не было возможности проверить, но это же правило должно, по идее, отловить этот винчестер даже тогда, когда мы его подключим по FireWire — совершенно другой шине! После этого, вполне естественно, можно прописать где-нибудь в fstab простое и удобное для себя место монтирования этого конкретного диска — он уже не потеряется среди множества других.

Маленькое замечание, прежде чем вы начнете наводить порядок в своем /dev: для того чтобы изменения в правилах вступили в силу, необходимо запустить программу udevstart, которая заполняет /dev с нуля теми устройствами, которые найдет в /sys. Еще одна интересная утилита-компаньон /dev, о которой хотелось бы упомянуть, — udevinfo. Она позволяет получить различную информацию из базы данных udev и sysfs. А протестировать правила udev вы сможете с помощью утилиты udevtest.

Не только динамический /dev

Ну что ж, мы получили то, что хотели? Несомненно. Но udev приносит еще одно изменение — /etc/dev.d/. Там хранятся ссылки на те программы, которым udev будет передавать управление уже после того, как создаст/удалит файл устройства. То есть, цепочка продолжает развиваться, и мы можем аккуратно заскриптовать что-нибудь, например, насчет конфигурации сети, после того как у нас появится или будет удалено какое-нибудь беспроводное устройство, или монтируется диск, для которого мы уже так заботливо определили однозначное имя в /dev. Именно через помощника в dev.d мы можем известить демона HAL, который дальше уже сможет рассказать в деталях о событии другим приложениям вашей свободной настольной среды. Имеет dev.d также и свою структуру, но за этим лучше обратиться к оригинальной документации udev на английском языке, которая входит в состав пакета udev, расположенного на сайте www.kernel.org/pub/linux/utils/kernel/hotplug/. Смело качайте свежую версию (а заодно обновите hotplug) и заглядывайте в каталог docs. Впрочем, возможно, вы предпочитаете инструменты своего дистрибутива.

Теперь, наверное, ни у кого не вызывает удивления то, что udev был очень хорошо принят разработчиками дистрибутивов — переход на него состоялся быстро даже среди тех, кто пытался использовать devfs, поскольку количество успешно преодолеваемых им проблем и качество их решения просто великолепны. Кстати, что касается devfs, то все тот же Грег Кроа-Хартман уже смастерил патчи, удаляющие ее из ядра. К тому моменту, когда вы будете читать эти строки, devfs, скорее всего, уже не будет являться частью ядра. Кстати, как уже говорилось, нам не нужны major/minor, одним из предложений Грега для Linux 2.7 (если эта ветка будет открыта) является последовательное или даже случайное выделение этих номеров для устройств — это проще и эффективнее, ведь теперь у нас есть инструмент, который уже готов с этим работать: свободный десктоп становится все лучше и удобнее.