Лабораторная работа №1 «Сборка и компиляция ядра UNIX/Linux»

Ядро Linux

Подготовительные действия

Для пересборки ядра следует заполучить его исходники, если таковых в системе ещё не имеется (в некоторых случаях они могут быть установлены при первичной инсталляции). Часто это можно сделать штатными средствами дистрибутива, установив с помощью его пакетного менеджера пакет, носящий имя типа kernelsource-2.6.xx- и так далее.

Общий способ получения исходников — это просто скачать их с kernel.org или его многочисленных зеркал , среди которых есть и российские. Если вы не имеете пристрастий к зеркалам какой-либо определённой страны (например, у меня быстрее всего в общем случае работают норвежские зеркала чего бы то ни было

Какое бы зеркало вы ни выбрали, на нём следует отыскать каталог /pub/linux/kernel/v3.x/, а в нем — архивные файлы последней стабильной версии и ревизии, имеющие вид linux-3.z.xx.y.tar.gz или linux-3.z.xx.y.tar.bz2, где z.xx — номер текущей версии, а y — её ревизии. Если есть желание опробовать тестируемую версию (в некоторых случаях, о которых я скажу позже, это оправдано), её следует искать в каталоге /pub/linux/kernel/3.x/testing/ , где она будет иметь вид вроде 3.4.27-rc5 (опять же актуально для текущего момента).

Как явствует из предыдущего абзаца, дерево исходников ядра доступно в виде архивов двух видов — tar.gz и tar.bz2. Какой из них выбрать — решайте по возможностям: первый быстрее распаковывается, второй — существенно меньше (в текущих версиях 47-48 Мбайт против примерно 60), что экономит время и трафик.

Чем качать и куда? Не имеет значения. Для скачивания следует использовать любимый ftp-клиент или менеджер загрузок (например wget), а помещать архив удобно в какое-либо подходящее место домашнего каталога, например, ~/src или ~/data/src.

Если в системе уже имеется дерево исходников предыдущей версии ядра, что легко проверить командой типа

```
$ ls /usr/src
```

то не обязательно качать архив ядра целиком, а можно ограничиться патчем для преобразования их в следующую версию, однако этот случай мы здесь рассматривать не будем (желающим сильно сэкономить трафик предоставляю разобраться с ним самим).

Дальнейшие действия после скачивания архива исходников — переход в каталог исходников

```
$ cd /usr/src
```

и получение прав суперпользователя любым методом, например, через su:

```
$ su
[ввод пароля администратора]
```

В дистрибутивах семейства Ubuntu обычно нужно использовать команду sudo — целесообразно делать это в форме

```
$ sudo su
```

или

```
$ sudo -i
```

чтобы, введя после любой из них пароль пользователя, избежать его повторения перед каждым из последующих действий.

В литературе можно обнаружить указания на то, что конфигурирование ядра можно (или даже нужно) выполнять от лица обычного пользователя. В принципе — можно, но, как мы увидим дальше, не самыми удобными способами. И вообще, бояться ответственности, налагаемой правами root'a, когда они действительно необходимы, столь же неоправданно, как и выполнять от его лица обычную пользовательскую работу.

Итак, обретя привилегии суперпользователя и находясь в каталоге /usr/src, распаковываем архив исходников одной из команд

```
$ tar xzvf path2/linux-2.6.2x.y.tar.gz
```

или

```
$ tar xjvf path2/linux-2.6.2x.y.tar.bz2
```
в зависимости от типа архива.

В результате в текущем каталоге образуется подкаталог вида linux-2.6.2x.y/, в котором и содержится полное дерево исходников. В ряде источников обращается особое внимание на создание символической ссылки /usr/src/linux, указывающей на каталог версии ядра, предназначенной для сборки. Ни малейшей необходимости в этом нет — хотя, если такая ссылка существовала для предыдущей версии, её лучше переопределить на новую, от греха подальше.

Принципы конфигурирования

Теперь перед нами самый сложный этап всей процедуры — собственно конфигурирование ядра. Чтобы приступить к нему, для начала следует перейти в каталог с деревом исходников

```
$ cd /usr/src/linux-3.z.xx.y/
```

За конфигурацию ядра отвечает файл /usr/src/linux-3.z.xx.y/.config — в свежераспакованном дереве исходников его нет. Это не значит, что по умолчанию не подразумевается никаких настроек — они есть, и хранятся в файле /usr/src/linux-2.6.2x.y/arch/x86/configs/i386_defconfig, однако имеют настолько общий характер, что их использование нецелесообразно. И потому за основу дальнейших действий следует принять файл конфигурации ядра текущего — поскольку предполагается, система у нас грузится и функционирует нормально, её ядро заведомо работоспособно.

Возникает вопрос, где брать конфиг текущего ядра? Аккуратные юзеры предусматривают помещение его в каталог /boot — это файл вида /boot/config (именно так по умолчанию обстоит дело в Zenwalk), /boot/config-3.z.xx.y или им подобный. В этом случае просто берём и копируем его в каталог с нашим деревом исходников (здесь и далее предполагается, что он является текущим):

```
$ cp /boot/config .config
```

Если конфигурационного файла ядра в каталоге /boot по каким-либо причинам не обнаруживается, его можно поискать в дереве исходников ядра предыдущей, по сравнению с ныне собираемой, версии. Если его нет и там (или исходники предыдущих версий отсутствуют как класс), ситуация осложняется.

Иногда конфигурация текущего ядра может быть встроена в само ядро и существовать в виде отдельного файла в каталоге /proc — config.gz, в наличи (или отстутсвии) которого можно убедиться командой

```
$ ls /proc/config.gz
```

Если такой файл имеется — всё хорошо, его можно считать стандартным образом:

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

в том числе и с перенаправлением в нужное место:

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

Однако эта возможность определяется текущей конфигурацией ядра, которой мы как раз и не знаем. И если такого файла нет, то конфигурирование придётся таки начать с настроек, заданных в каноническом ядре по умолчанию. По крайней мере, нет иных возможностей определить конфигурацию текущего ядра — разве что покопаться на сайте наличного дистрибутива, и не факт, что успешно.

Будем исходить из предположения, что текущую конфигурацию ядра мы тем или иным способом получили и она наличествует в нашем дереве исходников в виде соответствующего файла.

И вот тут есть даже не два пути, а гораздо больше, любой из которых приведет нас к цели — созданию новой конфигурации ядра. Весь вопрос выбора между ними — только в удобстве применительно к текущим обстоятельствам.

Путь первый, о котором не любят говорить, — это банальная модификация конфига ядра вручную. Ибо это обычный текстовый файл, который можно просмотреть командой less или more, где он предстанет в примерно следующем виде:

```
#
# Automatically generated make config: don't edit
# Linux kernel version: 2.6.26
# Mon Jul 14 22:26:07 2008
#
```

```
# CONFIG_64BIT is not set
CONFIG_X86_32=y
# CONFIG_X86_64 is not set
CONFIG_X86=y
CONFIG_ARCH_DEFCONFIG="arch/x86/configs/i386_defconfig"
...
```

и так далее.

Не смотря на указание об автоматической генерации и надпись, запрещающую редактирование, его можно спокойно обрабатывать утилитами работы с текстом (типа ed или sed) или править в любимом текстовом редакторе. Правда, файл этот очень большой (без малого три тысячи строк) и, на первый взгляд, не очень понятно структурирован, так что для первого опыта это не самый простой путь. Однако при необходимости изменения одной-двух точно известных опций в конфигурации ядра сделать это вручную быстрее и проще, нежели любыми другими, описанными ранее, способами.

Для этого всего-то и нужно, что, используя функцию поиска текстового редактора, найти нужные строки и внести в них соответствующие изменения:

`CONFIG_ИМЯ_ОПЦИИ=y`

для жесткого встраивания её в ядро, или

`CONFIG_ИМЯ_ОПЦИИ=m`

для подключения в качестве модуля (если для данной опции это возможно), или просто закомментировать — для полного отключения; при автоматической генерации конфига отключённая строка приобретает следующий вид:

`# CONFIG_ИМЯ_ОПЦИИ is not set`

При этом следует помнить, что некоторые опции не могут быть отключены без вреда для здоровья системы, иные же могут быть только жестко встроены в ядро, но не поддерживаются в качестве модулей. В частности, и поэтому повторяю, что метод ручной правки можно рекомендовать только в том случае, когда пользователь точно знает, что и зачем он делает.

Второй путь — это использование диалогового метода конфигурирования, вызываемого посредством команды

```
$ make config
```

Она запускает сценарий конфигурирования, требующий ответов на многочисленные вопросы, варианты ответов для каждого — y, то есть включение опции, n, то есть выключение, и m — подключение в качестве модуля. Ответы эти далеко не всегда очевидны — правда, нажав клавишу ?, можно вызвать подсказку, иногда весьма подробную. Тем не менее, при любой ошибке вернуться назад нельзя — остается только прервать сценарий (например, через Control+C) и начать всё сначала, ранее данные ответы не сохраняются.

Метод make config считается устаревшим и к использованию не рекомендуется. Единственное его достоинство — то, что он позволяет выполнить конфигурирование без привилегий администратора, что, при возможности их получения (а я надеюсь, что на локальной пользовательской машине такая возможность есть) представляется сомнительным преимуществом.

Третий метод, наиболее часто употребляемый, — команда

```
$ make menuconfig
```

Она вызывает сценарий генерации меню, работающего в текстовом режиме с использованием псевдографики, основанной на библиотеке ncurses (рис. 1), где значения опций (включение, выключение или подключение как модуль) устанавливаются нажатием клавиши Enter или Spacebar. Значения опций по умолчанию берутся из нашего файла .config, по завершении работы сценария он сохраняется под именем .config.old, а имя .config получает вновь сгенерированный файл.

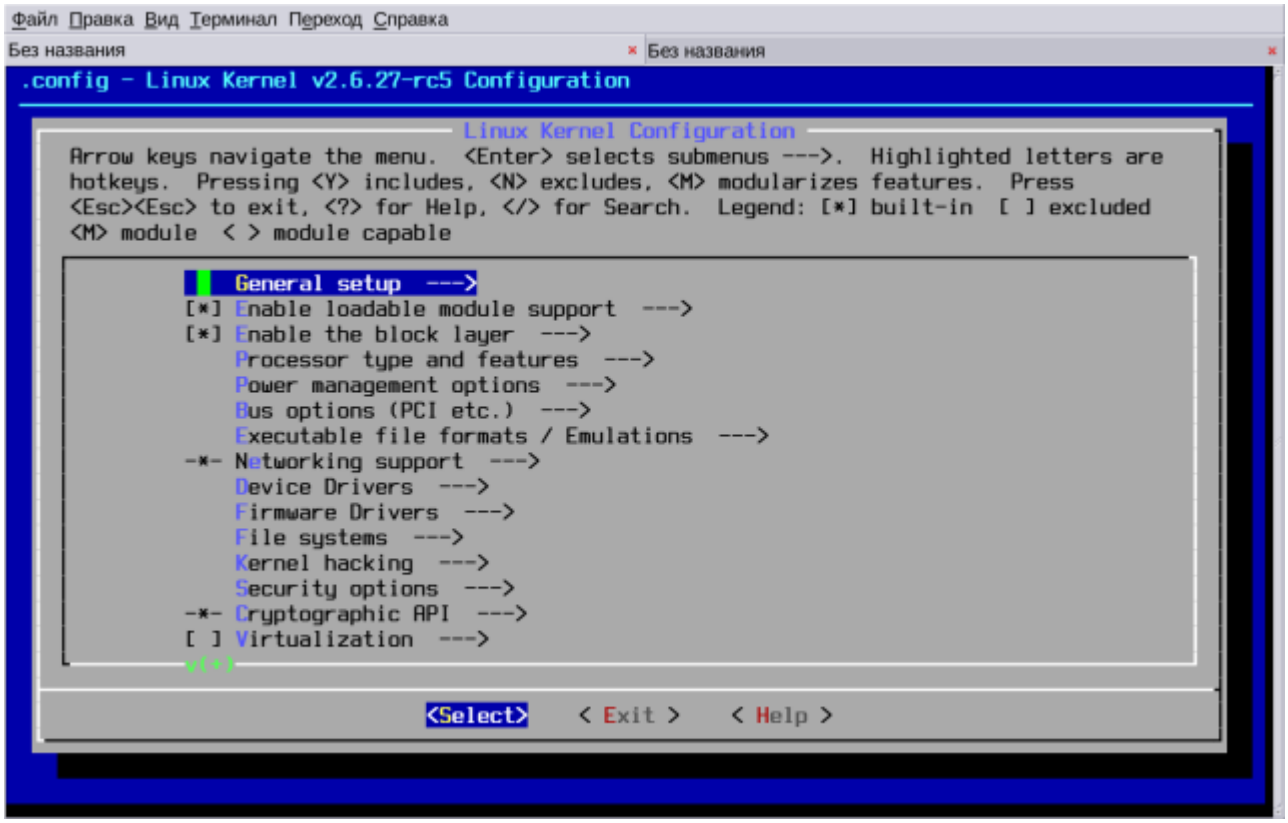

Рис. 1. Начальное меню при методе make menuconfig

Существует также два метода конфигурирования ядра, работающие в графическом режиме — make xconfig и make gconfig. Они полностью аналогичны методу make menuconfig, отличаясь лишь интерфейсом: в первом случае он основан на библиотеке Qt, во втором — Gtk (рис. 2). Соответственно, использование того или другого возможно только при наличии в системе соответствующих библиотек, причем для метода make xconfig требуется версия Qt для разработчиков).

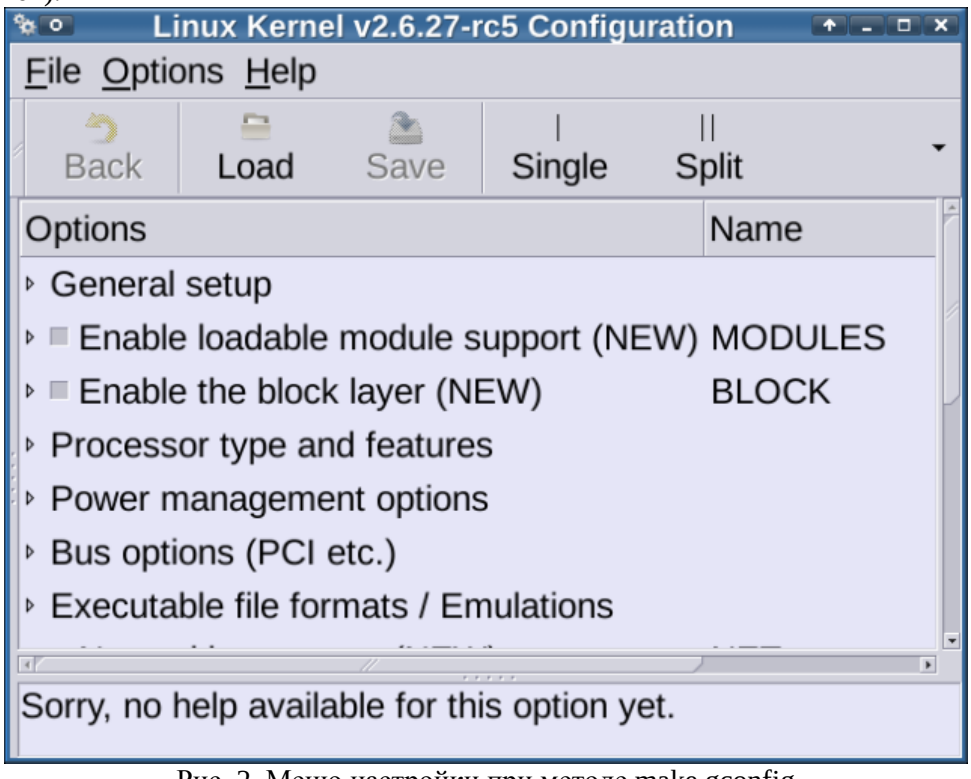

Рис. 2. Меню настройки при методе make gconfig

Ниже будет говориться только о конфигурировании ядра с помощью метода make menuconfig. Параллельно будуn даватьcz фрагменты файла .config, соответствующие рассматриваемым опциям.

Рассмотрение конкретных примеров конфигурирования отдельных частей ядра и составят предмет остальных заметок настоящего цикла. Пока же предположим, что нужные нам настройки выполнены и остановимся на вопросе, что же следует делать

После конфигурирования

Очевидно, что после конфигурирования ядра его следует скомпилировать и установить в должное место файловой иерархии. В различных источниках можно найти описания и того, и другого процесса, чуть различающиеся в деталях.

Итак, первая команда после завершения конфигурирования --

```
$ make bzImage
```

которая соберёт само ядро со всеми опциями, жёстко в него встроенными при настройке. То есть — с теми, значения которых в файле .config обозначены как "y". А bzImage — это одна из целей (target) команды make и одновременно — традиционное имя файла образа ядра Linux, которое в результате компиляции появляется в каталоге arch/i386/boot/.

Почему bzImage? Чтобы ответить на этот вопрос, надо чуть-чуть углубиться в историю. Традиционно, испокон веков, файлы образов ядер Unix-систем именовались — кто бы мог подумать? — unix. Когда эти системы обрели поддержку виртуальной памяти, то их ядра, для отличия от ядер прежнего поколения, стали называть vmunix.

Linux унаследовал эту традицию от своей тётки-мачехи, и первые ядра этой ОС носили имя linux. Со временем ядра Linux'a всё распухали в объёме, но их по прежнему требовалось подчас запихать на одну трёхдюймовую (а то и пятидюймовую) дискету. По этому поводу был придуман механизм компрессии образа ядра, а соответствующие файлы, чтобы отличить их от образов несжатых, стали называть vmlinuz (если конечное z вызовет у вас ассоциацию с zip-файлами, вы будете правы), или, иначе говоря, zImage.

Однако время шло, Linux рос и развивался — и с ним росло и его ядро. И даже в компрессированном виде объем его перевалил за размер дискеты. И потому был создан механизм сборки больших компрессированных ядер — за ними и закрепилось имя bzImage, то есть Большой Компрессированный Образ (Big zImage).

Всё сказанное выше призвано объяснить, что имена файлов образа ядра — не какие-то мистические заклинания, в которых нельзя изменить ни полбуквы: это просто общепринятые соглашения, запечатлённые как цели команды make для сборки ядра в файле Makefile. Полный список таких целей лучше всего в нём и смотреть — они имеют обыкновение меняться, хотя и не часто, и не принципиально. Тем не менее, только make-файл из дерева исходников текущей версии гарантирует полную актуальность всех возможных target'ов.

Вернёмся к командам сборки. Скомпилировав ядро, не следует забывать и о его модулях — они также нуждаются в компиляции. Так что следующая на очереди — команда

```
$ make modules
```

которая соберет все объектные модули для всех опций, соответствующим образом отмеченных в файле конфигурации ядра.

Обе приведенные выше команды можно скомбинировать в единую конструкцию:

```
$ make bzImage && make modules
```

```
или дать их одной строкой
```

```
$ make bzImage modules
```

А можно поступить ещё проще, дав команду

```
$ make all
```

которая соберет и образ ядра (всё тот же bzImage), и все отмеченные при конфигурации модули.

Более того есть способ ещё меньше напрягать подушечки пальцев, напечатав в командной строке совсем простую директиву:

```
$ make
```

действие которой эквивалентно команде make all.

И еще один немаловажный момент. Как известно, нынче подавляющее большинство процессоров, которые можно найти в продаже, имеют два, а то и четыре ядра. Правда, это мало где используется. Но как раз компиляция программ из исходников (в том числе и ядра) — как раз та

задача, на которой выигрыш от многопроцессорности вполне реален — и пренебрегать им не следует, так как сборка ядра и, особенно, модулей, если их много, занимает немало времени даже на гиперсовременных машинах.

Однако само по себе распараллеливание процессов, протекающих при компиляции, волшебным образом не возникнет — не смотря на то, что ядра "по-умолчанию" практически всех современных дистрибутивов собраны с поддержкой SMP: возможность распараллеивания задач команде make нужно задать явным образом, посредством опции -j#, где # рекомендуется задать равным удвоенному числу процессоров.

Таким образом, итоговая команда для сборки ядра вместе с его модулями для обычных нынче машин с двухядерным процессором будет выглядеть так:

```
$ make -j4
```

Для чего было приведено столько вариантов, вместо того, чтобы сразу привести самый простой и понятный вариант сборки? Во-первых, чтобы осветить все варианты этого процесса, которые могут встретиться в источниках (причем подчас без указания на существование альтернатив). Во-вторых же, и главных, — потому что сборку ядра и модулей иногда действительно целесообразно разделить: гарантию безошибочной сборки может дать, как известно, только страховой полис, а сообщения об ошибках, буде таковые появятся, лучше изучать для каждой составляющей порознь.

Однако, продолжая оставаться оптимистами, решим, что сборка и ядра, и модулей, вместе ли, раздельно ли, но завершилась без ошибок. Теперь остаётся поместить их куда следует. Как? Для модулей ответ однозначен: команда

```
$ make modules_install
```

скопирует их в новообразованный каталог /lib/modules/3.z.xx.y, где им и надлежит пребывать. И откуда впредь они будут подгружаться либо автоматически, по мере необходимости, либо вручную, командой `modprobe`.

А вот с установкой ядра возможны варианты. Часто рекомендуют автоматизировать этот процесс, дав команду

```
$ make install
```

которая и проделает всю остальную работу.

На первых порах от этого стоит воздержался по ряду причин, главная из которых — отсутствие уверенности в том, что ядро собралось безошибочно: отсутствие сообщений об ошибках в ходе компиляции ядра не гарантирует его безошибочной загрузки и работы.

Поэтому по первости стоит проделать работу по установке нового ядра вручную. Тем более, что работа эта не сложна: она всего-то и требует, что скопировать в каталог /boot три файла:

```
$ cp arch/i386/boot/bzImage /boot/vmlinuz-3.z.xx.y
$ cp System.map /boot/System.map-3.z.xx.y
$ cp .config /boot/config-3.z.xx.y
```

Указание номера версии и релиза желательно, чтобы легко отличить файлы, имеющие отношение к новому ядру, от одноименных — от ядра старого. Файл System.map требуется для работы загрузчика Lilo — GRUB его не использует, но порядку для скопируем его, даже если в нашей системе применяется последний. Ну и конфиг нового ядра также копируется для порядка, дабы в случае неудачи не начинать всё сначала.

Теперь остаётся только обеспечить загрузку нового ядра. Если в роли загрузчика системы выступает GRUB, это делается так: к имеющейся секции, загружающей старое ядро и выглядещей примерно так

```
title   Zenwalk
root    (hd0,0)
kernel /vmlinuz vga=791 root=/dev/sda6 ro quiet splash
initrd  /initrd.splash
quiet
```

добавляем ещё одну, примерно такого вида:

```
title   Zenwalk-2.6.27-rc5
kernel /vmlinuz-2.6.27-rc4 root=/dev/sda6 ro
```

Параметр quiet из неё убран для получения более подробных сведений о ходе загрузки (мы ведь ещё только проверяем новое ядро, правда?), параметры vga и splash — как архитектурные излишества, которым не место на данном этапе.

Теперь перезагружаем машину, при появлении меню GRUB выбираем пункт, соответствующий новому ядру и надеемся, что оно загрузится успешно. Если да — всё хорошо, работаем в системе с новым ядром или продолжаем совершенствовать его, асимптотически приближаясь к идеалу.

Если же нет — внимательно читаем сообщения об ошибках, потом перезагружаемся со старым ядром и ищем причины неудачи. Таковые составят предмет отдельной заметки. Пока же, забегая вперёд нужно знать, что наиболее вероятным сообщением об ошибках будет — kernel panic в результате невозможности смонтировать корневую файловую систему. А причиной такого явления — то, что поддержка интерфейса носителя корневой файловой системы и её типа не встроены в ядро жёстко, а подключены как модули.

2. Unix

Что касается Unix-подобных дистрибутивов, таких как FreeBSD или OpenBSD, то компиляция их ядер практически не отличается друг от друга, и в общем варианте выглядит примерно так:

```
# cd /usr/src/sys/arch/i386/conf
# config GENERIC
# cd ../compile/GENERIC
# make clean && make depend && make
    [...lots of output...
# make install
```

Однако, для точной настройки необходимо воспользоваться официальными руководствами :

FreeBSD - http://www.freebsd.org/doc/ru/books/handbook/

OpenBSD - http://obsd.ru/8/?q=node/6

Задание

На выбранном дистрибутиве произвести пересборку ядра с минимальным набором поддерживаемых устройств для ускорения загрузки.