Встраиваемые системы на основе Linux

Крис Симмондс

PACKT PUBLISHING

ДМК издательство

УДК 004.453/.453: 004.451.9Linux
ББК 32.972.1
С37

Симмондс К.
С37 Встраиваемые системы на основе Linux / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2017. – 360 с.: ил.

ISBN 978-5-97060-483-0

В книге подробно рассказано о том, как сконструировать встраиваемую Linux-систему из свободных программ с открытым исходным кодом, получив в результате надежный и эффективный продукт. Рассмотрены наборы инструментов, начальные загрузчики, ядро Linux и конфигурирование корневой файловой системы. Показано, как работать с системами сборки Buildroot и Yocto Project. Описаны процессы, потоки и управление памятью. Не обделены вниманием вопросы отладки и оптимизации платформы, а также выполнение приложений реального времени.

Издание рассчитано на разработчиков программного обеспечения на платформе Linux и системных программистов, уже знакомых со встраиваемыми системами. Предполагаются знание основ языка С и опыт системного программирования.

УДК 004.453/.453: 004.451.9Linux
ББК 32.972.1

ISBN 978-1-78439-253-6 (анг.)
ISBN 978-5-97060-483-0 (рус.)

Содержание

Глава 2

О наборах инструментов

Набор инструментов – это первый элемент встраиваемой Linux-системы и отправная точка проекта. Решения, принятые на этом раннем этапе, оказывают решающее влияние на конечный результат. Набор инструментов должен эффективно использовать имеющееся оборудование: выбирать оптимальный набор команд процессора, задействовать блок вычислений с плавающей точкой, если он присутствует, и т. д. Он должен поддерживать необходимые вам языки программирования и достаточно полно реализовывать стандарт POSIX и другие системные интерфейсы. При этом должны быть доступны обновления в случае обнаружения ошибок или уязвимостей. Наконец, выбранный набор инструментов не должен меняться на протяжении проекта. Произвольная замена компиляторов или библиотек может стать причиной тонких ошибок.

Получить набор инструментов просто – нужно лишь скачать и установить пакет. Однако сам он весьма сложен, что я и продемонстрирую в этой главе.

Что такое набор инструментов?

Набор инструментов предназначен для компиляции исходного кода в исполняемый файл, который можно выполнить на целевом устройстве. Он включает компилятор, компоновщик и библиотеки времени выполнения. Прежде всего набор инструментов нужен, чтобы собрать остальные три элемента встраиваемой Linux-системы: начальный загрузчик, ядро и корневую файловую систему. Инструменты должны уметь компилировать код, написанный на языке ассемблера, C и C++, поскольку именно эти языки используются в основных пакетах с открытым исходным кодом.

Обычно наборы инструментов для Linux основаны на компонентах из проекта GNU (http://www.gnu.org), и на момент написания книги это остается справедливым в большинстве случаев. Однако в последние несколько лет компилятор Clang и связанный с ним проект LLVM (http://llvm.org) стали вполне зрелыми продуктами и могут считаться альтернативой инструментам от GNU. Одно из основных различий между LLVM и GNU заключается в политике лицензирования: инструменты LLVM распространяются на условиях лицензии BSD, тогда как проект GNU основан на лицензии GPL. У Clang есть и некоторые технические преимущества,

в частности он быстрее компилирует и выдает более качественную диагностику, зато GNU GCC совместим с существующей кодовой базой и поддерживает широчайший спектр архитектур и операционных систем. До сих пор существуют области, в которых Clang не может заменить компилятор GNU C, особенно в части компиляции стандартного ядра Linux. Но очень вероятно, что через год-два Clang будет способен откомпилировать все компоненты встраиваемых Linux-систем и, следовательно, составить полноценную конкуренцию GNU. На странице http://clang.llvm.org/docs/CrossCompilation.html подробно описано, как использовать Clang для кросс-компиляции. Если вы хотите включить его в состав системы сборки встраиваемой Linux-системы, то обратите внимание на проект EmbToolkit (https://www.embtoolkit.org), который поддерживает наборы инструментов GNU и LLVM/Clang. Кроме того, разные люди работают над включением Clang в проекты Buildroot и Yocto Project. Системы сборки рассматриваются в главе 6, а пока сосредоточимся на наборе инструментов GNU, поскольку в настоящее время это единственный вариант, содержащий все необходимое.

Стандартный набор инструментов GNU состоит из трех основных компонентов.

- **Binutils**: набор двоичных утилит, включающий ассемблер и компоновщик ld. Размещен по адресу http://www.gnu.org/software/binutils/.
- **Набор компиляторов GNU (GCC)**: включает компиляторы C и других языков: C++, Objective-C, Objective-C++, Java, Fortran, Ada и Go (в зависимости от версии). Все они пользуются общим кодогенератором, который порождает ассемблерный код, поступающий на вход ассемблера GNU. Размещен по адресу http://gcc.gnu.org/.
- **Библиотека C**: стандартизованный API на основе спецификации POSIX, описывающей интерфейс между ядром операционной системы и приложениями. Есть несколько заслуживающих внимания библиотек на C, они будут описаны в следующем разделе.

Кроме того, понадобятся заголовочные файлы ядра Linux, содержащие определения и константы, необходимые для прямого доступа к ядру. Прямо сейчас они потребуются для компиляции библиотеки C, а впоследствии – при разработке программ и компиляции библиотек, обращающихся к конкретным устройствам Linux, например для вывода графики с помощью драйвера буфера кадра. Но нельзя просто скопировать заголовки из каталога `include` в исходном коде ядра, поскольку они предназначены только для использования ядром, и содержащиеся в них определения приведут к конфликтам при попытке включить их в код обычного приложения.

Вместо этого нужно сгенерировать набор «подчищенных» заголовков ядра, как будет показано в главе 5.

Обычно не так важно, сгенерированы ли заголовки ядра по коду именно той версии Linux, с которой вы собираетесь работать, или какой-то другой. Поскольку интерфейсы ядра всегда сохраняют обратную совместимость, нужно лишь, чтобы заголовки были взяты из версии не младше той, что будет использоваться в целевом устройстве.

Многие считают отладчик GNU, GDB, частью набора инструментов, поэтому он обычно тоже собирается на этом этапе. Об отладчике GDB речь пойдет в главе 12.

Типы наборов инструментов – платформенные и перекрестные

Нас будут интересовать два типа наборов инструментов.

- **Платформенный.** Работает в системе того же типа (иногда просто той же самой), что и система, для которой генерируются программы. Это типично для настольных ПК и серверов, но приобретает популярность и для некоторых классов встраиваемых устройств. Например, в системе Raspberry Pi, работающей под управлением Debian для ARM, есть собственные платформенные компиляторы.
- **Перекрестный.** Работает в системе, отличной от целевой платформы, т. е. позволяет, например, вести разработку на быстром настольном ПК и загружать сгенерированный код во встраиваемое устройство для тестирования.

Почти всегда встраиваемые Linux-системы разрабатываются с помощью перекрестных наборов инструментов, отчасти потому, что встраиваемые устройства не обладают достаточными ресурсами: быстродействием, ОЗУ и внешней памятью, но также для того, чтобы разделять исходное и целевое окружения. Последнее особенно важно, когда исходная и целевая платформы имеют общую архитектуру, например X86_64. В таком случае возникает искушение откомпилировать код в исходной системе и просто скопировать двоичные файлы в целевую. До какого-то момента это будет работать, но весьма вероятно, что исходная система обновляется чаще, чем целевая, поэтому может случиться, что разные программисты собирают систему, используя слегка различающиеся версии библиотек, а значит, нарушен принцип постоянства набора инструментов на всем протяжении проекта. Чтобы этот подход работал корректно, окружение сборки в исходной и целевой системах необходимо изменять синхронно, но гораздо проще разделить обе системы, это как раз и позволяет сделать перекрестный набор инструментов.

Однако есть и контраргумент в пользу платформенной разработки. При перекрестной разработке приходится кросс-компилировать все библиотеки и инструменты, необходимые для целевой платформы. Ниже в этой главе мы увидим, что кросс-компиляция – не всегда простое дело, потому что большинство пакетов с открытым исходным кодом не рассчитано на такую сборку. Интегрированные инструменты сборки, в том числе Buildroot и Yocto Project, решают эту проблему, инкапсулируя правила кросс-компиляции ряда пакетов, требуемых в типичной встраиваемой системе, но если нужно откомпилировать много дополнительных пакетов, то лучше делать это на той же платформе, где они будут исполняться. Например, невозможно собрать дистрибутив Debian для Raspberry Pi и BeagleBone с помощью кросс-компилятора, приходится прибегать к платформенным инструментам. Создать платформенное окружение сборки с нуля – непростая задача, для ее решения нужно сначала собрать кросс-компилятор, с его помощью построить

платформенное окружение сборки, в котором уже можно будет собирать пакеты. Для сборки потребуется целая ферма, состоящая из хорошо подготовленных целевых плат, хотя, если повезет, можно будет использовать QEMU для эмуляции целевой платформы. Если вас заинтересовала эта тема, можете познакомиться с проектом Scratchbox, который переживает свое второе рождение в облике Scratchbox2 (https://maemo.gitorious.org/scratchbox2). Проект был начат компанией Nokia для создания операционной системы Maemo Linux, а теперь используется в проектах Mer, Tizen и др.

Ну а в этой главе мы продолжим знакомство с более распространенным окружением для кросс-компиляции, которое сравнительно просто настроить и администрировать.

Архитектура процессора

Набор инструментов должен быть создан с учетом возможностей целевого процессора, в том числе:

- **Архитектура процессора**: arm, mips, x86_64 и т. д.
- **Порядок байтов в слове**: некоторые процессоры могут работать в обоих режимах (тупоконечном и остроконечном), но машинный код будет различаться.
- **Поддержка арифметики с плавающей точкой**: не во всех встраиваемых процессорах имеется аппаратный блок для операций с плавающей точкой, и в таком случае набор инструментов нужно сконфигурировать для вызова программной библиотеки.
- **Двоичный интерфейс приложений (ABI)**: соглашение о передаче параметров при вызове функций.

Для многих архитектур ABI один и тот же для всего семейства процессоров. Заметное исключение составляет архитектура ARM, для которой в конце 2000-х годов произошел переход на расширенный двоичный интерфейс приложений (**EABI**), а прежний ABI стал называться старым двоичным интерфейсом приложений (**OABI**). Хотя OABI теперь считается устаревшим, до сих пор можно встретить упоминания EABI. С той поры EABI разделился на два интерфейса в зависимости от метода передачи параметров с плавающей точкой. В оригинальном EABI используются регистры общего назначения (целочисленные), а в более позднем EABIHF – регистры с плавающей точкой. Интерфейс EABIHF показывает гораздо более высокое быстродействие на операциях с плавающей точкой, поскольку устраняет необходимость копирования между целыми и плавающими регистрами, однако он несовместим с процессорами, не оснащенными блоком операций с плавающей точкой. Следовательно, приходится выбирать между двумя несовместимыми ABI: использовать в одной программе оба нельзя, и решение нужно принимать на этом этапе.

В проекте GNU в названиях инструментов используется префикс, описывающий допустимые комбинации генерируемого кода. Префикс состоит из трех или четырех частей, разделенных дефисами.

- **Процессор**: архитектура процессора, например arm, mips или x86_64. Если процессор поддерживает оба порядка байтов, то для их различения можно добавить суффикс el (остроконечный – сначала младший байт) или eb (тупоконечный – сначала старший байт). Примерами могут служить остроконечный MIPS (mipsel) и тупоконечный ARM (armeb).
- **Поставщик**: производитель набора инструментов, например: buildroot, poky или просто unknown. Иногда эта часть вообще опускается.
- **Ядро**: для наших целей всегда 'linux'.
- **Операционная система**: имя компонента в пользовательском адресном пространстве: `gnu` или `uclibcgnu`. В конце может быть также указан ABI, так что наборы инструментов для ARM могут включать строки `gnueabi`, `gnueabihf`, `uclibcgnueabi` или `uclibcgnueabihf`.

Какая четверка была использована при сборке набора инструментов, можно узнать, выполнив команду с флагом `-dumpmachine`. Так, в исходной системе можно увидеть вот что:

```
$ gcc -dumpmachine
x86_64-linux-gnu
```

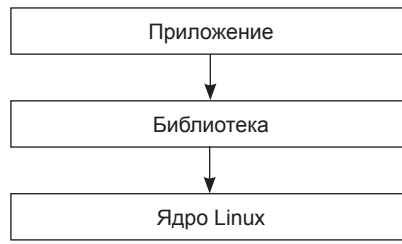
При установке на машину платформенного компилятора обычно создаются ссылки на все входящие в набор инструменты без префиксов, чтобы, к примеру, компилятор можно было запускать, набрав просто команду `gcc`.

Ниже приведен пример запуска кросс-компилятора:

```
$ mipsel-unknown-linux-gnu-gcc -dumpmachine
mipsel-unknown-linux-gnu
```

Выбор библиотеки C

Программный интерфейс обращения к операционной системе Unix определен на языке C и ныне закреплен в стандартах POSIX. Библиотека C представляет собой реализацию этого интерфейса – шлюз к ядру Linux из программ. Даже если программа написана на другом языке, скажем Java или Python, сопровождающие его библиотеки времени выполнения в конечном итоге должны будут обратиться к библиотеке C.



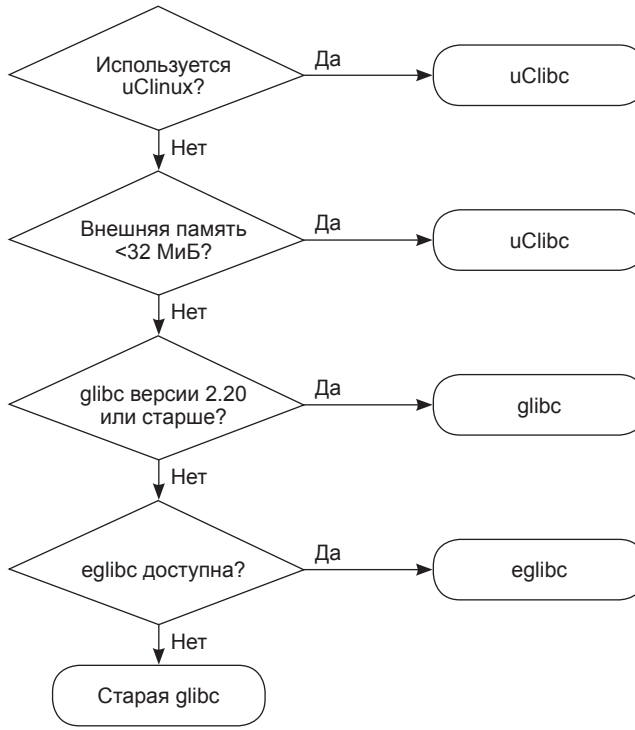
Библиотека C – шлюз к ядру из приложений

Когда библиотеке C необходимо получить доступ к сервису ядра, она производит системный вызов для перехода из пользовательского пространства в пространство ядра. Есть возможность вызвать ядро напрямую в обход библиотеки C, но это весьма трудоемко и почти никогда не нужно.

Существует несколько библиотек C на выбор. Перечислим основные.

- **glibc**. Размещена по адресу http://www.gnu.org/software/libc. Это стандартная библиотека GNU C. Она очень большая и до недавнего времени не допускала особой настройки, но содержит самую полную реализацию POSIX API.
- **eglibc**. Размещена по адресу http://www.eglibc.org/home. Это встраиваемая версия GLIBC. Раньше представляла собой набор заплат к glibc, которые добавляли возможности настройки и поддержку архитектур, не охваченных glibc (конкретно PowerPC e500). Разделение между eglibc и glibc всегда было несколько искусственным, но, к счастью, начиная с версии 2.20 код eglibc снова объединен с glibc, так что мы получили одну улучшенную библиотеку. Сопровождение eglibc прекращено.
- **uClibc**. Размещена по адресу http://www.uclibc.org. Буква 'u' на самом деле замещает греческую μ и означает, что речь идет о библиотеке C для микроконтроллеров. Изначально разрабатывалась для работы в uClinux (Linux для процессоров без блоков управления памятью), но затем была адаптирована для работы с полной версией Linux. Существует утилита конфигурирования, позволяющая точно настроить функциональность библиотеки для конкретных нужд. Даже в полной конфигурации эта библиотека меньше glibc, но уступает в полноте реализации стандартов POSIX.
- **musl libc**. Размещена по адресу http://www.musl-libc.org. Новая библиотека C, предназначенная для встраиваемых систем.

Так какую выбрать? Я рекомендую использовать uClibc, только если вы работаете с uClinux или очень ограничены в объеме ОЗУ либо внешней памяти, так что небольшой размер является преимуществом. В противном случае я предпочитаю современную версию glibc или eglibc. У меня нет опыта работы с musl libc, но если вы сочтете, что glibc (eglibc) не годится, можете устремить взоры в этом направлении. Процесс принятия решения показан на следующей блок-схеме.



Выбор библиотеки C

Получение набора инструментов

Есть три варианта получить набор инструментов: найти уже собранный набор, отвечающий вашим потребностям; взять тот, что сгенерирован инструментом сборки встраиваемой системы (см. главу 6), или собрать самостоятельно, как описано ниже в этой главе.

Готовый набор инструментов привлекателен тем, что вам остается только скачать и установить его, но при этом вы ограничены заданной кем-то конфигурацией и зависите от поставщика. Скорее всего, в роли поставщика будет выступать:

- Производитель SoC-системы или платы. Большинство производителей предлагает набор инструментов для Linux.
- Консорциум, организованный для системной поддержки определенной архитектуры. Например, некоммерческая организация Linaro (https://www.linaro.org) распространяет готовые наборы инструментов для архитектуры ARM.
- Сторонние поставщики инструментов для Linux, например: Mentor Graphics, TimeSys или MontaVista.
- Пакеты перекрестных инструментов, входящие в дистрибутив Linux для настольных ПК. Например, в дистрибутивах, производных от Debian, есть

пакеты кросс-компиляции для платформ ARM, MIPS и PowerPC.

- Двоичный SDK, сгенерированный одним из интегрированных инструментов сборки встраиваемых систем. В проекте Yocto Project есть примеры на странице http://autobuilder.yoctoproject.org/pub/releases/CURRENT/toolchain, а по адресу ftp://ftp.denx.de/pub/eldk/ выложен комплект средств разработки Denx Embedded Linux Development Kit.
- Опубликованная на каком-то форуме ссылка, которую вы больше не можете найти.

В любом случае предстоит решить, отвечает ли готовый набор инструментов вашим требованиям. Используется ли в нем предпочтительная для вас библиотека C? Публикует ли поставщик обновления для исправления ошибок и устранения уязвимостей (учтите при этом замечания по поводу поддержки и обновления, приведенные в главе 1). Если ответ хотя бы на один вопрос отрицательный, то стоит подумать о самостоятельном построении набора инструментов.

Увы, собрать набор инструментов не так-то просто. Если вы твердо решили пройти этот путь до конца, зайдите на сайт *Cross Linux From Scratch* (http://trac.clfs.org). Там приведены пошаговые инструкции по созданию каждого компонента.

Есть и более простой способ – воспользоваться средством crosstool-NG, которое инкапсулирует весь процесс в набор скриптов, управляемый меню. Но для работы с ним все равно нужно много знать.

Проще прибегнуть к системе сборки типа Buildroot или Yocto Project, поскольку генерация набора инструментов в этом случае является составной частью процесса сборки. Лично я предпочитаю именно такое решение и опишу его в главе 6.

Сборка набора инструментов с помощью crosstool-NG

Я начну с использования crosstool-NG, потому что это позволит нам увидеть весь процесс создания набора инструментов и создать несколько таких наборов разного типа.

Несколько лет назад Дэн Кегел (Dan Kegel) написал ряд скриптов и makefайлов для генерации перекрестных наборов инструментов и назвал их crosstool (kegel.com/crosstool). В 2007 году Ианн Э. Морин (Yann E. Morin) взял их за основу для создания следующего поколения crosstool, получившего названия crosstool-NG (crosstool-ng.org). В настоящий момент это самый удобный способ сборки автономного перекрестного набора инструментов из исходного кода.

Установка crosstool-NG

Прежде всего понадобится работоспособный платформенный набор инструментов и средства сборки на вашем исходном ПК. Для работы с crosstool-NG в системе Ubuntu нужно установить следующие пакеты:

```
$ sudo apt-get install automake bison chrpath flex g++ git gperf gawk
libexpat1-dev libncurses5-dev libsdl1.2-dev libtool
python2.7-dev texinfo
```

Затем скачайте текущую версию croostool-NG со страницы http://crosstool-ng.org/download/crosstool-ng. В примерах ниже используется версия 1.20.0. Распакуйте архив и создайте интерфейсную программу на основе ct-ng:

```
$ tar xf crosstool-ng-1.20.0.tar.bz2
$ cd crosstool-ng-1.20.0
$ ./configure --enable-local
$ make
$ make install
```

Флаг `--enable-local` означает, что программа будет установлена в текущий каталог. Это позволяет обойтись без привилегий root, которые понадобились бы для установки в каталог по умолчанию `/usr/local/bin`. Для запуска меню наберите `./ct-ng`, находясь в текущем каталоге.

Выбор набора инструментов

Программа Crosstool-NG умеет строить различные комбинации наборов инструментов. Для упрощения начальной настройки в ее состав входит ряд примеров, покрывающих многие типичные случаи. Чтобы получить их список, выполните команду `./ct-ng list-samples`.

Предположим, что целевой платформой является плата BeagleBone Black с процессорным ядром ARM Cortex A8 и блоком операций с плавающей точкой VFPv3 и что мы хотим использовать текущую версию glibc. Ближе всего к этой комбинации подходит пример `arm-cortex_a8-linux-gnueabi`. Чтобы узнать его текущую конфигурацию, добавим в начало имени префикс `show-`:

```
$ ./ct-ng show-arm-cortex_a8-linux-gnueabi
[L..] arm-cortex_a8-linux-gnueabi
OS              : linux-3.15.4
Companion libs  : gmp-5.1.3 mpfr-3.1.2 cloog-ppl-0.18.1 mpc-1.0.2 libelf-0.8.13
binutils        : binutils-2.22
C compiler      : gcc-4.9.1 (C,C++)
C library       : glibc-2.19 (threads: nptl)
Tools           : dmalloc-5.5.2 duma-2_5_15 gdb-7.8 ltrace- 0.7.3 strace-4.8
```

Чтобы выбрать именно эту целевую конфигурацию, выполните команду:

```
$ ./ct-ng arm-cortex_a8-linux-gnueabi
```

В этот момент можно просмотреть конфигурацию и внести необходимые изменения, воспользовавшись системой меню:

```
$ ./ct-ng menuconfig
```

Система меню основана на программе конфигурирования ядра Linux `menuconfig`, поэтому навигация по интерфейсу покажется знакомой любому, кто когда-нибудь конфигурировал ядро. Если вы не из их числа, обратитесь к главе 4, где описана программа `menuconfig`.

На этой стадии я рекомендую внести несколько изменений.

- В разделе **Paths and misc options** (Пути и разные параметры) выключите флажок **Render the toolchain read-only** (Генерировать доступный только для чтения набор инструментов) (`CT_INSTALL_DIR_RO`).
- В разделе **Target options → Floating point** (Параметры целевой платформы → Плавающая точка) выберите **hardware (FPU)** (аппаратная) (`CT_ARCH_FLOAT_HW`).
- В разделе **C-library → extra config** добавьте флаг **--enable-obsolete-rpc** (`CT_LIBC_GLIBC_EXTRA_CONFIG_ARRAY`).

Первое необходимо, если вы захотите добавить библиотеки в состав набора инструментов после его установкеи, о чем я расскажу ниже в этой главе. Второе служит для выбора оптимальной реализации операции с плавающей точкой в случае, когда процессор оснащен соответствующим аппаратным блоком. И последнее заставляет программу сгенерировать набор инструментов с устаревшим заголовком `rpc.h`, который все еще используется в ряде пакетов (отметим, что эта проблема существует, только если вы выбрали библиотеку glibc). Имена в скобках соответствуют меткам в конфигурационном файле. Внеся изменения, сохраните их и выйдите из `menuconfig`.

Конфигурационные данные сохраняются в файле `.config`. В первой его строчке написано *Automatically generated make config: don't edit* (Автоматически сгенерированный файл. Не редактировать). Это хороший совет, но сейчас мы им пренебрежем. Помните, обсуждая ABI набора инструментов для архитектуры ARM, мы упомянули о двух вариантах: передача параметров с плавающей точкой в целочисленных регистрах и использование только регистров с плавающей точкой? Выбранная конфигурация относится ко второму типу, поэтому часть, относящаяся к ABI, должна иметь вид `eabihf`. Существует конфигурационный параметр, который делает ровно то, что нам надо, но по умолчанию он выключен, а в меню отсутствует, по крайней мере в этой версии crosstool. Поэтому придется отредактировать файл `.config`, добавив в него строки, выделенные полужирным шрифтом:

```
[...]
#
# arm other options
#
CT_ARCH_ARM_MODE="arm"
CT_ARCH_ARM_MODE_ARM=y
# CT_ARCH_ARM_MODE_THUMB is not set
# CT_ARCH_ARM_INTERWORKING is not set
CT_ARCH_ARM_EABI_FORCE=y
CT_ARCH_ARM_EABI=y
CT_ARCH_ARM_TUPLE_USE_EABIHF=y
[...]
```

Теперь можно использовать crosstool-NG, чтобы получить, сконфигурировать и собрать компоненты в соответствии с заданной спецификацией:

```
$ ./ct-ng build
```

Сборка занимает примерно полчаса, построенный набор инструментов помещается в каталог ~/x-tools/arm-cortex_a8-linux-gnueabihf/.

Анатомия набора инструментов

Чтобы вы могли составить представление о типичном наборе инструментов, я рассмотрю только что созданный набор.

Набор инструментов находится в каталоге ~/x-tools/arm-cortex_a8-linux-gnueabihf/bin. Там вы найдете кросс-компилятор arm-cortex_a8-linux-gnueabihf-gcc. Чтобы им воспользоваться, нужно будет добавить каталог в переменную окружения PATH:

```
$ PATH=~/x-tools/arm-cortex_a8-linux-gnueabihf/bin:$PATH
```

Теперь напишем простейшую программу hello world:

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
printf ("Hello, world!\n");
return 0;
}
```

И откомпилируем ее:

```
$ arm-cortex_a8-linux-gnueabihf-gcc helloworld.c -o helloworld
```

Выполнив команду file, которая печатает тип файла, убеждаемся, что она действительно кросс-компилирована:

```
$ file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 3.15.4, not stripped
```

Получение информации о кросс-компиляторе

Допустим, вы только что получили набор инструментов и хотели бы узнать, как он был сконфигурирован. Многое может сообщить сам gcc. Например, версию можно получить, задав флаг --version:

```
$ arm-cortex_a8-linux-gnueabi-gcc --version
arm-cortex_a8-linux-gnueabi-gcc (crosstool-NG 1.20.0) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Чтобы узнать, как был сконфигурирован кросс-компилятор, зададим флаг -v:

```
$ arm-cortex_a8-linux-gnueabi-gcc -v
Using built-in specs.
COLLECT_GCC=arm-cortex_a8-linux-gnueabihf-gcc
```

```
COLLECT_LTO_WRAPPER=/home/chris/x-tools/arm-cortex_a8-linux-gnueabihf/libexec/gcc/arm-cor-
tex_a8-linux-gnueabihf/4.9.1/lto-wrapper
Target: arm-cortex_a8-linux-gnueabihf
Configured with: /home/chris/hd/home/chris/build/MELP/build/crosstool-ng- 1.20.0/.build/
src/gcc-4.9.1/configure --build=x86_64-build_unknown-linux-gnu --host=x86_64-build_unknown-
linux-gnu --target=arm-cortex_a8-linux-gnueabihf --prefix=/home/chris/x-tools/arm-cortex_
a8- linux-gnueabihf --with-sysroot=/home/chris/x-tools/arm-cortex_a8- linux-gnueabihf/
arm-cortex_a8-linux-gnueabihf/sysroot --enable-languages=c,c++ --with-arch=armv7-a --with-
cpu=cortex-a8 --with-tune=cortex-a8 --with-float=hard --with-pkgversion='crosstool-
NG 1.20.0' --enable-__cxa_atexit --disable-libmudflap --disable-libgomp --disable-
libssp --disable-libquadmath --disable-libquadmath-support --disable-libsanitiz-
er --with-gmp=/home/chris/hd/home/chris/build/MELP/build/crosstool-ng- 1.20.0/.build/
arm-cortex_a8-linux-gnueabihf/buildtools --with-mpfr=/home/chris/hd/home/chris/build/MELP/
build/crosstool-ng- 1.20.0/.build/arm-cortex_a8-linux-gnueabihf/buildtools --with-mpc=/
home/chris/hd/home/chris/build/MELP/build/crosstool-ng- 1.20.0/.build/arm-cortex_a8-linux-
gnueabihf/buildtools --with-isl=/home/chris/hd/home/chris/build/MELP/build/crosstool-
ng- 1.20.0/.build/arm-cortex_a8-linux-gnueabihf/buildtools --with-cloog=/home/chris/hd/
home/chris/build/MELP/build/crosstool-ng- 1.20.0/.build/arm-cortex_a8-linux-gnueabihf/
buildtools --with-libelf=/home/chris/hd/home/chris/build/MELP/build/crosstool-ng- 1.20.0/.
build/arm-cortex_a8-linux-gnueabihf/buildtools --with-host-libstdcxx='-static-libgcc -Wl,-
Bstatic,-lstdc++,-Bdynamic -lm' -- enable-threads=posix --enable-target-optspace --en-
able-plugin -- enable-gold --disable-nls --disable-multilib --with-local-prefix=/home/
chris/x-tools/arm-cortex_a8-linux-gnueabihf/arm-cortex_a8-linux-gnueabihf/sysroot --enable-
c99 --enable-long-long
Thread model: posix
gcc version 4.9.1 (crosstool-NG 1.20.0)
```

Напечатано много, но отметим самые интересные строки:

- `--with-sysroot=/home/chris/x-tools/arm-cortex_a8-linux-gnueabihf/arm-cortex_a8-linux-gnueabihf/sysroot`: это путь к каталогу sysroot по умолчанию, пояснения приведены ниже;
- `--enable-languages=c,c++`: это означает, что включена поддержка языков C и C++;
- `--with-arch=armv7-a`: кодогенератор использует набор команд ARM v7a;
- `--with-cpu=cortex-a8` and `--with-tune=cortex-a8`: код оптимизирован для процессорного ядра Cortex A8;
- `--with-float=hard`: генерируются коды команд блока операций с плавающей точкой, параметры передаются в регистрах с плавающей точкой (VFP);
- `--enable-threads=posix`: включена поддержка потоков POSIX.

Это все параметры по умолчанию, используемые компилятором. Большинство из них можно переопределить в командной строке gcc. Например, чтобы откомпилировать программу для другого процессора, нужно переопределить сконфигурированный параметр `--with-cpu`, добавив в командную строку флаг `mcpu`:

```
$ arm-cortex_a8-linux-gnueabihf-gcc -mcpu=cortex-a5 helloworld.c -o helloworld
```

Для вывода списка архитектурно-зависимых параметров зададим флаг `target-help`:

```
$ arm-cortex_a8-linux-gnueabihf-gcc --target-help
```

Возникает вопрос: так ли важно задавать точную конфигурацию в момент генерации набора инструментов, если впоследствии ее можно изменить. Ответ зависит от того, как вы планируете использовать инструменты. Если вы собираетесь создавать новый набор инструментов для каждой целевой платформы, то имеет смысл задать все с самого начала, чтобы уменьшить риски ошибки впоследствии. Забегая вперед, скажу, что я называю такой подход «философией Buildroot». Если же вы хотите собрать обобщенный набор инструментов и готовы подставлять правильные параметры при сборке для конкретной целевой платформы, то требования к точности конфигурации не так важны. Такой точки зрения придерживается проект Yocto Project. В примерах выше мы следовали философии Buildroot.

sysroot, библиотека и файлы-заголовки

В терминологии набора инструментов sysroot – это каталог, содержащий подкаталоги для библиотек, файлов-заголовков и других конфигурационных файлов. Его можно задать при конфигурировании набора с помощью флага `with-sysroot=` или в командной строке с помощью флага `sysroot=`. Чтобы узнать, куда указывает sysroot по умолчанию, запустите компилятор с флагом `-print-sysroot`:

```
$ arm-cortex_a8-linux-gnueabi-gcc -print-sysroot
/home/chris/x-tools/arm-cortex_a8-linux-gnueabihf/arm-cortex_a8-
linux-gnueabihf/sysroot
```

В каталоге sysroot находятся такие подкаталоги:

- `lib`: содержит разделяемые объекты библиотеки C и динамического компоновщика/загрузчика `ld-linux`;
- `usr/lib`: статическая библиотека C и другие библиотеки, установленные позже;
- `usr/include`: заголовки для всех библиотек;
- `usr/bin`: утилиты, исполняемые на целевой платформе, например `ldd`;
- `/usr/share`: файлы, необходимые для локализации и интернационализации;
- `sbin`: утилита `ldconfig` для оптимизации загрузки динамических библиотек.

Одни файлы нужны на машине разработки для компиляции программ, другие, например разделяемые библиотеки и `ld-linux`, – используются на целевой платформе во время выполнения.

Другие элементы набора инструментов

В таблице ниже перечислены другие команды, входящие в набор инструментов:

Команда	Описание
`addr2line`	Преобразует адреса программных объектов в имена файлов и номера строк, читая таблицы отладочных символов в исполняемом файле. Очень полезно для интерпретации адресов, встречающихся в отчете о крахе системы