LLVM: инфраструктура для разработки компиляторов

Бруно Кардос Лопес

Рафаэль Аулер

[PACKT] PUBLISHING

ДМК издательство

УДК 004.4'422LLVM
ББК 32.973.33
Л77

Л77 Бруно Кардос Лопес, Рафаэль Аулер

LLVM: инфраструктура для разработки компиляторов. / пер. с англ. Киселев А. Н. – М.: ДМК Пресс, 2015. – 342 с.: ил.

ISBN 978-5-97060-305-5

LLVM – новейший фреймворк для разработки компиляторов. Благодаря простоте расширения и организации в виде множества библиотек, LLVM легко поддается освоению даже начинающими программистами, вопреки устоявшемуся мнению о сложности разработки компиляторов.

Сначала эта книга покажет, как настроить, собрать и установить библиотеки, инструменты и внешние проекты LLVM. Затем познакомит с архитектурой LLVM и особенностями работы всех компонентов компилятора: анализатора исходных текстов, генератора кода промежуточного представления, генератора выполняемого кода, механизма JIT-компиляции, возможностями кросс-компиляции и интерфейсом расширений. На множестве наглядных примеров и фрагментов исходного кода книга поможет вам войти в мир разработки компиляторов на основе LLVM.

Издание предназначено энтузиастам, студентам, а также разработчикам компиляторов, интересующимся LLVM. Читатели должны знать язык программирования C++ и, желательно, иметь некоторые представления о теории компиляции.

ISBN 978-1-78216-692-4 (англ.)
ISBN 978-5-97060-305-5 (рус.)

ОГЛАВЛЕНИЕ

ГЛАВА 1. Сборка и установка LLVM

Фреймворк LLVM доступен для разных версий **Unix** (GNU/**Linux**, **FreeBSD**, **Mac OS X**) и **Windows**. В этой главе мы расскажем, какие шаги необходимо выполнить, чтобы получить действующий фреймворк LLVM во всех этих системах. Для некоторых систем доступны предварительно скомпилированные пакеты LLVM и Clang, но вы можете сами выполнить сборку из исходных текстов.

Пользователи, только начинающие изучать LLVM, должны знать, что компиляторы на основе LLVM включают в себя библиотеки и инструменты из обоих пакетов – LLVM и Clang. Поэтому все инструкции в этой главе описывают сборку и установку обоих пакетов. На протяжении всей книги мы будем подразумевать версию LLVM 3.4. Однако, имейте в виду, что LLVM – это молодой проект и продолжает активно развиваться, поэтому в последующих версиях некоторые аспекты могут измениться.

> **Совет.** *Когда мы работали над этой книгой, версия LLVM 3.5 еще не вышла. Но, несмотря на то, что в центре внимания этой книги находится версия LLVM 3.4, мы планируем обновить примеры до версии LLVM 3.5 к третьей неделе сентября 2014 года, чтобы дать вам возможность опробовать их с более новой версией LLVM. Обновленные примеры будут доступны по адресу:* `https://www.packtpub.com/sites/default/files/downloads/6924OS_Appendix.pdf`.

В этой главе рассматриваются следующие темы:

- порядок нумерации версий LLVM;
- установка предварительно скомпилированных файлов LLVM;
- установка LLVM с использованием диспетчера пакетов;
- сборка LLVM из исходных текстов для Linux;
- сборка LLVM из исходных текстов для Windows и Visual Studio;
- сборка LLVM из исходных текстов для Mac OS X и Xcode.

Порядок нумерации версий LLVM

Проект LLVM продолжает быстро развиваться, благодаря усилиям многих программистов. За 10 лет после выпуска первой версии и до момента появления версии 3.4, в репозиторий SVN (в качестве системы управления версиями используется Subversion) было передано более 200 000 исправлений и изменений. За один только 2013 год в репозиторий было передано 30 000 новых изменений. Как следствие, постоянно появляются новые возможности, а прежние – быстро устаревают. Как и в любом другом крупном проекте, разработчики должны точно следовать плотному графику выпуска стабильных версий, уверенно проходящих разнообразные тесты, чтобы дать пользователям возможность испытать новые возможности.

На протяжении всей своей истории проект LLVM следует стратегии выпуска двух стабильных версий в год. В каждой следующей версии увеличивается младший номер. Например, при переходе от версии 3.3 к версии 3.4 изменяется младший номер версии. Когда младший номер достигает значения 9, в следующей версии будет увеличен старший номер, например, после версии LLVM 2.9 была выпущена версия LLVM 3.0. Изменение старшего номера версии вовсе не означает наличие существенных изменений, в сравнении с предыдущей версией, а всего лишь служит границей очередного пятилетнего отрезка развития проекта.

Для проектов, зависящих от LLVM, типично использовать *стволовую* (*trunk*) версию, то есть, наиболее свежую, доступную в репозитории SVN, которая иногда может оказаться нестабильной. Недавно, начиная с версии 3.4, в сообществе LLVM было решено выпускать *промежуточные версии*, введя дополнительный номер версии – номер выпуска. Первым результатом такого решения стал выход версии LLVM 3.4.1. Цель выпуска промежуточных версий – включение исправлений из стволовой версии в последнюю помеченную (tagged) версию, не имеющую новых особенностей, чтобы обеспечить полную совместимость. Промежуточные версии должны выходить через три месяца после выхода последней основной версии. Так как такая нумерация версий еще достаточно нова, мы сосредоточимся на установке версии LLVM 3.4. Существует большое число предварительно скомпилированных пакетов LLVM 3.4, но, следуя нашим инструкциям, вы без особых усилий сможете самостоятельно собрать LLVM 3.4.1 или более новую версию.

Установка скомпилированных пакетов LLVM

Чтобы упростить задачу установки программного обеспечения и избавить вас от необходимости выполнять компиляцию самостоятельно, поставщики LLVM создают предварительно скомпилированные пакеты для определенных платформ. Компиляция программного обеспечения иногда может оказаться очень непростым делом; она может потребовать некоторого времени и должна выполняться, только если вы используете другую платформу или принимаете активное участие в разработке проекта. Поэтому, для желающих побыстрее приступить к исследованию LLVM, существуют предварительно скомпилированные пакеты. Однако, в этой книге мы советуем использовать исходные тексты LLVM. Вы должны быть готовы скомпилировать LLVM самостоятельно.

Существует два основных способа установки предварительно скомпилированных пакетов LLVM: на официальном веб-сайте или на сайтах проектов, выпускающих дистрибутивы для GNU/Linux и Windows.

Установка скомпилированных пакетов с официального сайта

На официальном сайте проекта LLVM можно загрузить следующие предварительно скомпилированные пакеты для версии 3.4:

Таблица 1.1. Официальные скомпилированные пакеты для версии LLVM 3.4

Архитектура	Версия
x86_64	Ubuntu (12.04, 13.10), Fedora 19, Fedora 20, FreeBSD 9.2, Mac OS X 10.9, Windows и openSUSE 13.1.
i386	openSUSE 13.1, FreeBSD 9.2, Fedora 19, Fedora 20 и openSUSE 13.1
ARMv7/ARMv7a	Linux

Более полный перечень вы найдете на сайте http://www.llvm.org/releases/download.html, в разделе **Pre-built Binaries** (предварительно скомпилированные пакеты) для версии LLVM, которую вы желали бы загрузить. Например, чтобы загрузить и установить LLVM в Ubuntu 13.10, можно выполнить следующие команды:

```
$ sudo mkdir -p /usr/local; cd /usr/local
$ sudo wget http://llvm.org/releases/3.4/clang+llvm-3.4-x86_64-linux-
gnu-ubuntu-13.10.tar.xz
$ sudo tar xvf clang+llvm-3.4-x86_64-linux-gnu-ubuntu-13.10.tar.xz
$ sudo mv clang+llvm-3.4-x86_64-linux-gnu-ubuntu-13.10 llvm-3.4
$ export PATH="$PATH:/usr/local/llvm-3.4/bin"
```

После этого LLVM и Clang готовы к использованию. Не забывайте о необходимости постоянно изменять системную переменную окружения `PATH`, потому что изменение, которое было выполнено в последней строке, действует только до завершения текущего сеанса. Чтобы проверить корректность установки, можно попробовать выполнить простую команду, которая выведет версию Clang:

```
$ clang -v
```

Если при попытке выполнить эту команду возникли какие-либо проблемы, попробуйте запустить выполняемый файл непосредственно из каталога, куда он был установлен, чтобы убедиться, что проблема не вызвана неправильной настройкой переменной окружения `PATH`. Если и в этом случае запустить Clang не удалось, возможно вы установили предварительно скомпилированную версию, несовместимую с вашей системой. Не забывайте, что во время компиляции, двоичные файлы компонуются с динамическими библиотеками определенных версий. Сообщение об отсутствии библиотек, появляющееся при попытке запустить приложение, служит явным признаком, что двоичные файлы были скомпилированы в системе, несовместимой с вашей.

> **Совет.** *В Linux, например, текст сообщения об отсутствии библиотеки содержит имя выполняемого файла и имя динамической библиотеки, которую не удалось загрузить. Обратите внимание на имя динамической библиотеки – это явный признак, что динамический компоновщик и загрузчик не смог загрузить данную библиотеку, потому что программа была собрана в несовместимой системе.*

Чтобы установить предварительно скомпилированные пакеты в других системах (кроме Windows), можно выполнить точно такую же последовательность шагов. Предварительно скомпилированные пакеты для Windows распространяются в виде простого в использовании инсталлятора, который сам распакует дерево каталогов пакета LLVM в папку `Program Files`. Инсталлятор предусматривает возможность автоматического изменения переменной окружения `PATH`, чтобы можно было запускать Clang из любого окна командной строки.

Установка с использованием диспетчера пакетов

Диспетчеры пакетов доступны в самых разных системах и также упрощают получение и установку предварительно скомпилированных пакетов LLVM/Clang. Это наиболее предпочтительный путь для большинства пользователей, поскольку диспетчер пакетов автоматически разрешает зависимости и проверяет совместимость вашей системы с устанавливаемыми двоичными файлами.

Например, в Ubuntu (10.04 и выше) установку можно выполнить следующей командой:

```
$ sudo apt-get install llvm clang
```

В Fedora 18 установка выполняется так же, но используется иной диспетчер пакетов:

```
$ sudo yum install llvm clang
```

Обновление с использованием промежуточных сборок

Пакеты могут собираться по ночам, из самых свежих исходных текстов, содержащих самые последние изменения. Такие пакеты могут оказаться полезными для разработчиков и пользователей LLVM, для кого представляют интерес самые свежие версии, или сторонним пользователям, стремящимися обеспечить актуальность своих локальных проектам.

Linux

Для Debian и Ubuntu Linux (i386 и amd64) доступны ежедневные сборки, полученные компиляцией исходных текстов из репозиториев LLVM. Подробности см. на сайте http://llvm.org/apt.

Например, установить такую сборку LLVM и Clang в Ubuntu 13.10 можно следующей последовательностью команд:

```
$ sudo echo "deb http://llvm.org/apt/raring/ llvm-toolchain-raring main"
>> /etc/apt/sources.list

$ wget -O - http://llvm.org/apt/llvm-snapshot.gpg.key | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install clang-3.5 llvm-3.5
```

Windows

Инсталляторы ежедневных сборок LLVM/Clang для Windows доступны для загрузки по адресу: http://llvm.org/builds/, в разделе **Windows snapshot builds** (ежедневные сборки для Windows). По умолчанию инструменты LLVM/Clang устанавливаются в папку `C:\Program Files\LLVM\bin` (это местоположение может измениться, в зависимости от версии). Обратите внимание, что существует отдельный драйвер Clang `clang-cl.exe`, имитирующий `cl.exe` из Visual C++. Если вы предполагаете использовать классический драйвер, совместимый с GCC, используйте `clang.exe`.

> **Совет.** *Имейте в виду, что ежедневные сборки не являются стабильными версиями и могут работать очень неустойчиво.*

Сборка из исходных текстов

В отсутствие предварительно скомпилированных пакетов, LLVM и Clang можно скомпилировать из исходных текстов. Этот путь поможет поближе познакомиться со структурой LLVM. Кроме того, вы сможете точнее настроить некоторые параметры сборки.

Системные требования

По адресу: http://llvm.org/docs/GettingStarted.html#hardware можно найти постоянно обновляемый список платформ, поддерживаемых проектом LLVM. Полный перечень предварительных требований, которые должны быть соблюдены перед компиляцией LLVM, можно найти по адресу: http://llvm.org/docs/GettingStarted.html#software. В системах Ubuntu, например, разрешить зависимости можно с помощью команды:

```
$ sudo apt-get install build-essential zlib1g-dev python
```

Если вы пользуетесь довольно старой версией дистрибутива Linux с устаревшими пакетами, найдите время, чтобы обновить свою систему. Исходные тексты LLVM написаны на языке C++ и очень требовательны к версии компилятора, с помощью которого они будут компилироваться, поэтому попытка скомпилировать их старой версией компилятора C++ с большой долей вероятности не увенчается успехом.

Получение исходных текстов

Исходные тексты LLVM распространяются на условиях BSD-подобной лицензии и могут быть загружены с официального сайта проекта или из репозиториев SVN. Чтобы получить исходные тексты для версии 3.4, можно перейти по адресу http://llvm.org/releases/download.html#3.4 или напрямую загрузить их, как показано ниже. Обратите внимание, что вам всегда нужны будут исходные тексты Clang и LLVM, а исходные тексты дополнительных инструментов Clang (clang-tools-extra) можно загружать и устанавливать по желанию. Однако эти инструменты потребуются тем, кто пожелает опробовать примеры из главы 10, «Инструменты Clang и фреймворк LibTooling». Информация по сборке дополнительных проектов приводится в следующей главе. Используйте следующие команды для загрузки и установки LLVM, Clang и дополнительных инструментов Clang:

```
$ wget http://llvm.org/releases/3.4/llvm-3.4.src.tar.gz
$ wget http://llvm.org/releases/3.4/clang-3.4.src.tar.gz
$ wget http://llvm.org/releases/3.4/clang-tools-extra-3.4.src.tar.gz
$ tar xzf llvm-3.4.src.tar.gz; tar xzf clang-3.4.src.tar.gz
$ tar xzf clang-tools-extra-3.4.src.tar.gz
$ mv llvm-3.4 llvm
$ mv clang-3.4 llvm/tools/clang
$ mv clang-tools-extra-3.4 llvm/tools/clang/tools/extra
```

> **Примечание.** *Загруженные исходные тексты в Windows можно распаковать с помощью* `gunzip`*, WinZip или любого другого архиватора.*

SVN

Чтобы получить исходные тексты непосредственно из репозитория SVN, убедитесь сначала, что в вашей системе установлен пакет subversion. На следующем шаге решите: хотите ли вы получить последнюю версию, хранящуюся в репозитории, или стабильную. Чтобы получить последнюю (стволовую) версию, выполните следующую последовательность команд (предполагается, что они запускаются из каталога, подготовленного для размещения исходных текстов):

```
$ svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
$ cd llvm/tools
$ svn co http://llvm.org/svn/llvm-project/cfe/trunk clang
$ cd ../projects
```

```
$ svn co http://llvm.org/svn/llvm-project/compiler-rt/trunk compiler-rt
$ cd ../tools/clang/tools
$ svn co http://llvm.org/svn/llvm-project/clang-tools-extra/trunk extra
```

Чтобы получить стабильную версию (например, версию 3.4), замените слово `trunk` на `tags/RELEASE_34/final` во всех командах. Вас может также заинтересовать возможность навигации по репозиторию LLVM SVN, с целью посмотреть историю включения в него изменений, журналы и структуру дерева каталогов с исходными текстами. для этого откройте страницу http://llvm.org/viewvc.

Git

Существует также возможность получить исходные тексты из Git-зеркала репозитория, синхронизированного с репозиторием SVN:

```
$ git clone http://llvm.org/git/llvm.git
$ cd llvm/tools
$ git clone http://llvm.org/git/clang.git
$ cd ../projects
$ git clone http://llvm.org/git/compiler-rt.git
$ cd ../tools/clang/tools
$ git clone http://llvm.org/git/clang-tools-extra.git
```

Сборка и установка LLVM

В этом разделе описываются разные способы сборки и установки LLVM.

С использованием сценария configure, сгенерированного с помощью autotools

Стандартный способ сборки LLVM заключается в том, чтобы создать платформо-зависимые файлы сборки с применением сценария `configure`, сгенерированного с помощью системы сборки GNU autotools. Эта система сборки пользуется большой популярностью и многие из вас наверняка знакомы с ней. Она поддерживает разные параметры настройки.

> **Примечание.** *Устанавливать систему сборки GNU autotools необходимо, только если вы намереваетесь подменить систему сборки LLVM и сгенерировать новый сценарий* `configure`. *Обычно в этом нет никакой необходимости.*

Потратьте некоторое время, чтобы познакомиться с имеющимися параметрами настройки, используя следующие команды:

```
$ cd llvm
$ ./configure --help
```

Ниже приводится описание некоторых из них:

- `--enable-optimized`: Этот параметр позволяет скомпилировать LLVM/Clang без отладочной информации и с оптимизациями. По умолчанию выключен. Включение отладочной информации и запрет оптимизаций рекомендуется, когда библиотеки LLVM используются для разработки, но перед выпуском готовой версии оптимизацию следует включить, потому что отсутствие оптимизации существенно замедляет работу LLVM.
- `--enable-assertions`: Этот параметр включает дополнительные проверки в коде и очень полезен для разработки основных библиотек LLVM. По умолчанию он включен.
- `--enable-shared`: Этот параметр позволяет скомпилировать библиотеки LLVM/Clang в виде разделяемых библиотек и скомпоновать инструменты LLVM с ними. Если вы планируете разрабатывать инструменты за пределами системы сборки LLVM и желаете динамически компоновать их с библиотеками LLVM, включите этот параметр. По умолчанию он выключен.
- `--enable-jit`: Этот параметр включает **динамическую компиляцию** (Just-In-Time Compilation) для всех целей, поддерживающих такую возможность. По умолчанию он включен.
- `--prefix`: Определяет путь к каталогу установки, куда будут сохраняться скомпилированные файлы инструментов и библиотек LLVM/Clang при установке; например, если определить параметр `--prefix=/usr/local/llvm`, выполняемые файлы будут устанавливаться в каталог `/usr/local/llvm/bin`, а библиотеки – в каталог `/usr/local/llvm/lib`.
- `--enable-targets`: Этот параметр позволяет выбрать множество целей, для компиляции. Стоит напомнить, что с помощью LLVM можно выполнять кросс-компиляцию, то есть, компилировать программы, которые должны работать на других платформах, таких как ARM, MIPS и т. д.. Данный параметр определяет, какие генераторы выполняемого кода (backends) должны быть включены в библиотеки, отвечающие за созда-

ние машинного кода. По умолчанию компилируются все цели, но вы можете сократить время компиляции, определив только те цели, которые вас интересуют.

> **Совет.** *Этого параметра недостаточно, чтобы получить автономный кросс-компилятор. За дополнительной информацией по этой теме обращайтесь к главе 8, «Кросс-платформенная компиляция».*

После запуска сценария `configure` с желаемыми параметрами, сборку можно завершить классической парой команд `make` и `make install`. А теперь перейдем к примеру.

Сборка и установка в Unix

В этом примере мы скомпилируем неоптимизированную версию LLVM/Clang (с отладочной информацией), выполнив последовательность команд, которые присутствуют в любой Unix-подобной системе или в оболочке Cygwin. Вместо каталога установки `/usr/local/llvm`, как было показано в предыдущих примерах, компиляция и установка будут выполняться в домашний каталог, что позволяет установить LLVM, не имея привилегий суперпользователя root. Такой подход широко используется разработчиками и позволяет установить сразу несколько версий. При желании вы можете изменить каталог установки на `/usr/local/llvm`, собрав общесистемный пакет. Только не забудьте использовать команду `sudo`, когда будете создавать каталог для установки и выполнять команду `make install`. Ниже приводится используемая для этого последовательность команд:

```
$ mkdir каталог-для-установки
$ mkdir каталог-где-будет-выполняться-сборка
$ cd каталог-где-будет-выполняться-сборка
```

В этом разделе мы создадим отдельный каталог, где будут храниться объектные файлы, то есть, промежуточные результаты компиляции. Не выполняйте сборку в том же каталоге, где хранятся исходные тексты. Используйте следующие команды с параметрами, описанными в предыдущем разделе:

```
$ /PATH_TO_SOURCE/configure --disable-optimized
--prefix=../каталог-для-установки
$ make && make install
```

> **Совет.** *Желающие могут использовать команду* `make -jN`*, чтобы позволить ей задействовать* `N` *экземпляров компилятора, которые будут работать параллельно, что повысит общую скорость процедуры сборки. Например, попробуйте команду* `make -j4` *(или укажите чуть большее число), если процессор вашего компьютера имеет четыре ядра.*

Подождите, пока компиляция и установка всех компонентов не завершится. Обратите внимание, что сценарии сборки также обрабатывают другие репозитории, загруженные и сохраненные в дерево каталогов с исходными текстами LLVM – нам не требуется отдельно настраивать Clang или дополнительные инструменты Clang.

Чтобы убедиться в успехе сборки, всегда полезно после ее окончания выполнить команду `echo $?`. Пара символов `$?` в командной оболочке – это имя переменной, хранящей код завершения последней команды, выполненной в текущем сеансе, а команда `echo` выводит значение этой переменной на экран. Поэтому очень важно выполнять данную команду сразу вслед за командой `make`. Если сборка прошла успешно, команда `make` вернет в переменной `$?` значение 0, как и любая другая программа, успешно завершившая работу:

```
$ echo $?
0
```

Настройте свою переменную окружения `PATH`, чтобы упростить доступ к только что установленным выполняемым файлам, и выполните первый тест, спросив у Clang номер версии:

```
$ export PATH="$PATH:каталог-для-установки/bin"
$ clang –v
clang version 3.4
```

С использованием CMake и Ninja

LLVM предлагает альтернативную, кросс-платформенную систему сборки на основе CMake, вместо традиционных сценариев `configure`. CMake может генерировать специализированные файлы `Makefile` для вашей платформы, чем-то напоминая сценарий `configure`, но CMake – более гибкая система и, способная также генерировать файлы сборки для других систем, таких как Ninja, Xcode и Visual Studio.

С другой стороны, Ninja – это небольшая и быстрая система сборки, замещающая систему GNU Make и связанные с ней файлы `Makefile`. Если вам интересно узнать причины появления системы Ninja и историю ее развития, посетите страницу http://aosabook.

org/en/posa/ninja.html. Систему CMake можно настроить на создание файлов сборки для Ninja вместо файлов `Makefile`, что позволяет использовать либо связку CMake и GNU Make, либо CMake и Ninja.

Использование последней связки дает преимущество более быстрой пересборки после внесения изменений в исходные тексты LLVM. Это особенно удобно для тех, кто занимается разработкой инструментов и расширений в дереве исходных текстов LLVM и использует систему сборки LLVM для компиляции своих проектов.

Проверьте, установлены ли у вас CMake и Ninja. Например, в системе Ubuntu выполните следующую команду:

```
$ sudo apt-get install cmake ninja-build
```

Кроме того, сборка LLVM с применением CMake дает возможность определять множество параметров настройки. Полный их список можно найти по адресу: http://llvm.org/docs/CMake.html. Ниже приводится перечень параметров, соответствующих параметрам, перечисленным выше. Эти параметры имеют те же значения по умолчанию, что и предыдущие параметры сценария `configure`:

- `CMAKE_BUILD_TYPE`: Строка, значение которой определяет тип сборки `Release` (выпуск) или `Debug` (отладочная). Тип сборки `Release` эквивалентен параметру `--enable-optimized` сценария `configure`, а тип `Debug` – параметру `--disable-optimized`.
- `CMAKE_ENABLE_ASSERTIONS`: Булево значение, соответствующее параметру `--enable-assertions` сценария `configure`.
- `BUILD_SHARED_LIBS`: Булево значение, соответствующее параметру `--enable-shared` сценария `configure`. Определяет, как должны компилироваться библиотеки – как разделяемые (`true`) или как статические (`false`). Разделяемые библиотеки не поддерживаются на платформе Windows.
- `CMAKE_INSTALL_PREFIX`: Строка, соответствующая параметру `--prefix-configure`. Определяет путь к каталогу установки.
- `LLVM_TARGETS_TO_BUILD`: Список целей для сборки, разделенных точками с запятой, примерно соответствует списку целей, разделенных запятыми, в параметре `--enable-targets` сценария `configure`.

Для определения значений параметров в команде `cmake` используется синтаксис `-DPARAMETER=value`.

Сборка в Unix с использованием CMake и Ninja

Воспроизведем тот же пример, что был показан выше, с применением сценария `configure`, но на этот раз выполним сборку с использованием CMake и Ninja:

Сначала создадим каталог и для сборки и установки:

```
$ mkdir каталог-где-будет-выполняться-сборка
$ mkdir каталог-для-установки
$ cd каталог-где-будет-выполняться-сборка
```

Напоминаю, что для сборки следует использовать каталог, отличный от каталога с исходными файлами LLVM. Далее запустим CMake с набором выбранных параметров:

```
$ cmake /каталог-с-исходными-текстами -G Ninja -DCMAKE_BUILD_TYPE="Debug"
-DCMAKE_INSTALL_PREFIX="../каталог-для-установки"
```

Вместо `/каталог-с-исходными-текстами` следует подставить абсолютный путь к каталогу с исходными текстами LLVM. Если вы пожелаете использовать традиционный путь на основе GNU Makefiles, параметр `-G Ninja` можно опустить. Теперь можно завершить сборку командой `ninja` или `make`, в зависимости от выбранного инструмента сборки. Если вы решили использовать `ninja`, выполните следующую команду:

```
$ ninja && ninja install
```

Если вы решили использовать `make`:

```
$ make && make install
```

Так же как в более раннем примере, проверить успешность компиляции можно с помощью простой команды. Помните, что она должна запускаться непосредственно после команды сборки, то есть, никакие другие команды не должны выполняться между ними, потому что они тоже сохраняют код завершения в переменной `$?`:

```
$ echo $?
0
```

Если предыдущая команда выведет ноль, значит сборка была выполнена успешно. В заключение настройте свою переменную окружения `PATH` и воспользуйтесь новым компилятором:

```
$ export PATH=$PATH:where-you-want-to-install/bin
$ clang -v
```

Устранение ошибок сборки

Если команда сборки вернула ненулевое значение, это означает что возникла ошибка. В этом случае Make или Ninja выведет на экран со-

общение. Найдите самую первую ошибку в выводе команды. Ошибки сборки стабильной версии LLVM обычно возникают, когда система не соответствует требованиям к версиям программных компонентов. Наиболее часто ошибки возникают из-за использования устаревшего компилятора. Например, попытка собрать LLVM 3.4 с помощью GNU g++ Version 4.4.3 закончится следующей ошибкой, которая появится уже после того, как большая половина исходных файлов LLVM будет скомпилирована:

```
[1385/2218] Building CXX object projects/compiler-rt/lib/
interception/
CMakeFiles/RTInterception.i386.dir/interception_type_test.cc.o

FAILED: /usr/bin/c++ (...)_test.cc.o -c /local/llvm-3.3/llvm/projects/
compiler-rt/lib/interception/interception_type_test.cc

test.cc:28: error: reference to 'OFF64_T' is ambiguous

interception.h:31: error: candidates are: typedef __sanitizer::OFF64_T
OFF64_T

sanitizer_internal_defs.h:80: error: typedef __sanitizer::u64
__sanitizer::OFF64_T
```

Чтобы исправить эту проблему, можно было бы внести необходимые исправления в исходный код LLVM (как это сделать, можно узнать после недолгих поисков в Интернете или заглянув в исходные тексты), но едва ли вам хотелось бы исправлять каждую версию LLVM, которую понадобится скомпилировать. Намного проще и правильнее обновить компилятор.

Вообще, когда при сборке стабильной версии возникают ошибки, в первую очередь следует подумать о том, чем параметры вашей системы отличаются от рекомендуемых. Помните, что стабильные версии тщательно тестируются на нескольких платформах. С другой стороны, если вы пытаетесь выполнить сборку исходных текстов нестабильной версии из репозитория SVN, вполне возможно, что ошибки были внесены в результате одного из последних изменений, и тогда можно попробовать откатиться к более ранней версии из SVN.

Использование других подходов в Unix

В некоторых системах Unix имеются диспетчеры пакетов, которые автоматически собирают и устанавливают приложения из исходных текстов. Они поддерживают собственные способы и приемы компиляции, хорошо отлаженные и обеспечивающие автоматическое раз-

решение зависимостей. Далее мы попробуем оценить такие платформы в контексте сборки и установки LLVM и Clang:

- В Mac OS X имеется диспетчер пакетов *MacPorts*, который можно задействовать следующей командой:

```
$ port install llvm-3.4 clang-3.4
```

- В Mac OS X имеется также диспетчер пакетов *Homebrew*, который можно вызвать так:

```
$ brew install llvm -with-clang
```

- В FreeBSD 9.1 поддерживается система *портов* (ports), которой можно воспользоваться, как показано ниже (имейте в виду, что начиная с версии FreeBSD 10, Clang является компилятором по умолчанию, то есть, он уже установлен):

```
$ cd /usr/ports/devel/llvm34
$ make install
$ cd /usr/ports/lang/clang34
$ make install
```

- Ниже показано, как произвести сборку и установку в Gentoo Linux:

```
$ emerge sys-devel/llvm-3.4 sys-devel/clang-3.4
```

Windows и Microsoft Visual Studio

Порядок компиляции LLVM и Clang в Microsoft Windows мы покажем на примере использования Microsoft Visual Studio 2012 в Windows 8. Выполните следующие шаги:

1. Установите Microsoft Visual Studio 2012.
2. Установите официальный дистрибутив инструментов CMake, который можно получить на сайте http://www.cmake.org. В процессе установки не забудьте отметить флажок **Add CMake to the system PATH for all users** (Добавить CMake в системную переменную PATH).
3. CMake может сгенерировать файлы проекта, необходимые Visual Studio для настройки и сборки LLVM. Для этого сначала запустите инструмент `cmake-gui`. Затем щелкните на кнопке **Browse Source...** (Выбрать исходный каталог) и выберите каталог с исходным кодом LLVM. Затем щелкните на кнопке **Browse Build** (Выбрать каталог сборки) и выберите каталог,

куда должны сохраняться файлы, сгенерированные компилятором CMake, как показано на рис. 1.1:

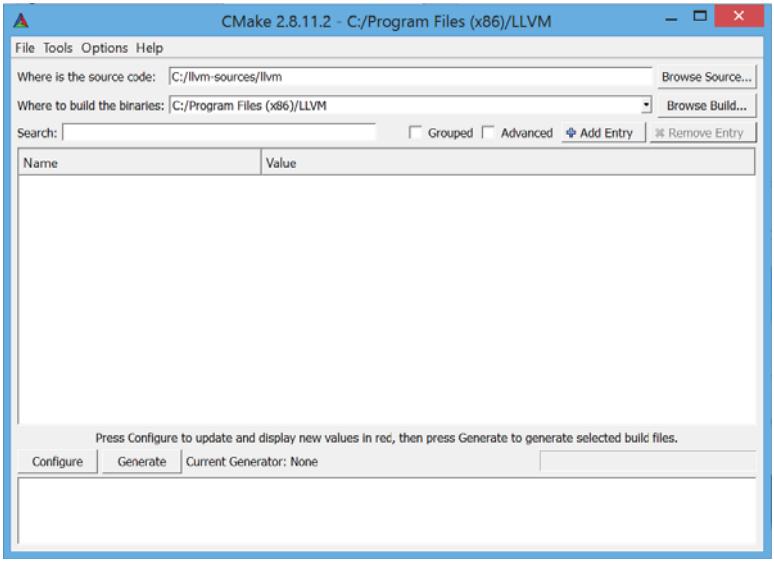

Рис. 1.1. Подготовка пакета LLVM к компиляции

4. Щелкните на кнопке **Add Entry** (Добавить элемент) и определите параметр `CMAKE_INSTALL_PREFIX`, содержащий путь установки инструментов LLVM, как показано на рис. 1.2:

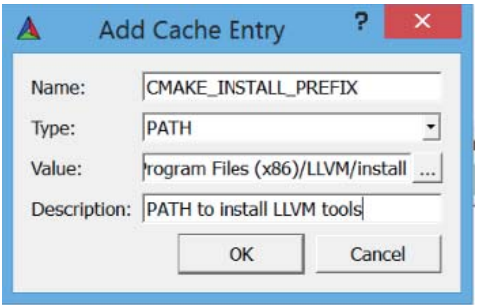

Рис. 1.2. Определение параметра CMAKE_INSTALL_PREFIX

5. Дополнительно можно определить поддерживаемые цели в виде параметра `LLVM_TARGETS_TO_BUILD`, как показано на рис. 1.3. При необходимости можно добавить любые другие параметры для CMake, описанные выше.

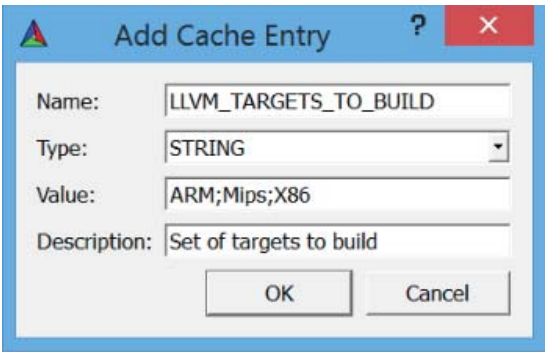

Рис. 1.3. Определение параметра LLVM_TARGETS_TO_BUILD

6. Щелкните на кнопке **Configure** (Настроить). На экране появится диалог, предлагающий выбрать *генератор* для этого проекта и компилятор; выберите **Use default native compilers** (Использовать компиляторы по умолчанию) и для Visual Studio 2012, выберите в раскрывающемся списке пункт **Visual Studio 11**. Щелкните на кнопке **Finish** (Завершить), как показано на рис. 1.4:

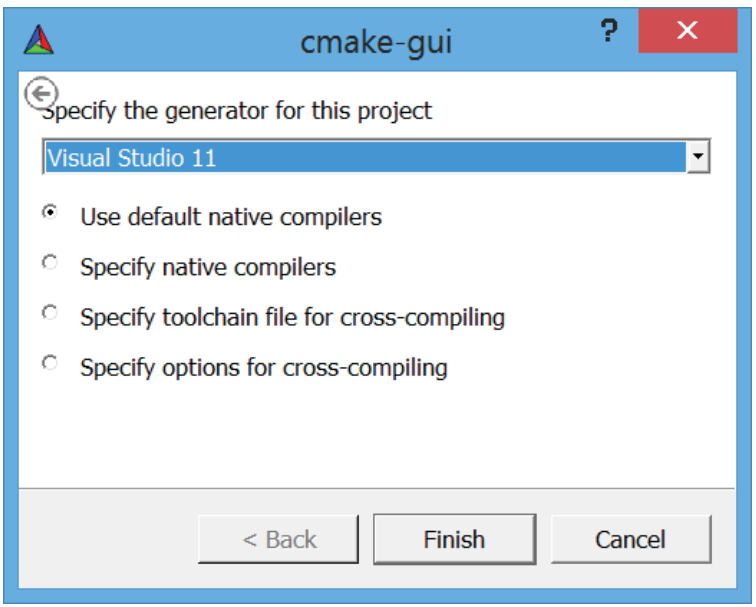

Рис. 1.4. Выбор генератора и компилятора