Особенности сборки кросс-компилятора GCC и бинарных утилит

В.А. Галатенко[1], Г.Л. Левченкова[2], С.В. Самборский[3]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, galat@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, galka@niisi.ras.ru;

[3]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, sambor@niisi.ras.ru

Аннотация. Разработчики, использующие свободно распространяемый компилятор GCC, нередко сталкиваются с проблемой правильной сборки кросс-компилятора и необходимых бинарных утилит из исходных текстов для заданной целевой архитектуры. В статье приводится общая последовательность действий по самостоятельной сборке кросс-компилятора и бинарных утилит для архитектуры MIPS, подходящая для разных версий компилятора. Приводятся примеры проблем, которые могут при этом возникать, и способы их решения.

Ключевые слова: компилятор, бинарные утилиты, GCC, сборка, MIPS.

1. Введение

Свободно распространяемый компилятор GCC [1] широко используется разработчиками встраиваемых систем и систем реального времени, функционирующих под управлением широкого спектра аппаратных платформ.

В данной работе будет приведена апробированная авторами инструкция по правильной сборке кросс-компилятора и необходимых бинарных утилит для архитектуры MIPS. При необходимости эта инструкция может быть взята за основу при сборке компилятора для других аппаратных платформ.

2. Предварительные действия

Прежде всего надо убедиться, что доступно необходимое программное обеспечение. Для получения архивов исходных текстов может понадобится утилита wget, а для их распаковки - tar. Если планируется работа с git-репозиторием, то, естественно, нужен сам git.

Точный список необходимых программ и диапазон допустимых версий этих программ зависит от того, какие версии компилятора и бинарных утилит предполагается собирать. Обязательно потребуются следующие утилиты:

make
gcc (обязательно с поддержкой C++)
makeinfo (из пакета texinfo)

Также могут понадобиться утилиты bison, flex, m4.

Если сборка происходит под управлением операционной системы Linux, то bash и стандартный набор команд (cd, rm, mkdir, и т.п.) в системе уже есть. Библиотеки GMP, MPFR и MPC заранее устанавливать не нужно, так как их лучше собирать самим (см. п. 3).

Для удобства сборки компилятора требуется создать каталог, например, gcc-mke-build, и перейти в него (имя gcc-mke-build произвольно и дальше не будет использоваться). Затем в этом каталоге удобно создать подкаталог для исходных текстов, например, src.

```
mkdir gcc-mke-build
cd gcc-mke-build
mkdir src
```

Далее создадим каталог, в который будут установлено все собранное программное обеспечение:

```
sudo mkdir /opt/gcc-mke
```

Следует учесть, что для записи в /opt обычно требуются права администратора, поэтому использована команда sudo. Ниже все команды установки запускаются через sudo.

При необходимости, путь /opt/gcc-mke можно заменить на более удобный, используется он только в ключах команд configure. Например, если невозможно использовать sudo или нежелательно устанавливать компилятор в общедоступное место, то можно установить его в домашнем каталоге, но при этом рекомендуется использовать полный путь (начиная с ‘/’, а не с ‘~’). Для непродолжительных экспериментов можно установить компилятор в каталог временных файлов, например, в /var/tmp/gcc-mke.

3. Сборка арифметических библиотек

Для сборки компилятора (но не бинарных

утилит) требуются три библиотеки: GMP – функции для операций над целыми числами неограниченной длины, MPFR – функции для операций с произвольной точностью над числами с плавающей запятой и MPC – аналогичные функции для операций над комплексными числами с плавающей запятой. Библиотека MPFR использует функции библиотеки GMP, а библиотека MPC, в свою очередь, использует функции из MPFR, поэтому собирать их нужно именно в таком порядке.

3.1. Сборка GMP

Сперва необходимо скачать и распаковать актуальную версию библиотеки с официального сайта:

```
wget https://gmplib.org/download/gmp/gmp-6.2.1.tar.xz
cd src
tar xaf ../gmp-6.2.1.tar.xz
```

В команде распаковки (tar xaf ...) параметр ‘a’ означает автоматическое определение компрессора, которым сжат tar-архив. Это удобно, но может не сработать, если в системе не установлен достаточно современный tar, или нет соответствующего компрессора. В этом случае следует дополнительно установить необходимые пакеты или произвести распаковку на другом компьютере.

Далее надо сконфигурировать библиотеку:

```
cd gmp-6.2.1
./configure --disable-shared --enable-static \
--disable-assembly --host=`./configfsf.guess` \
--prefix=/opt/gcc-mke
```

Ключ --prefix=/opt/gcc-mke указывает конфигуратору каталог, куда будет установлена библиотека, ниже он будет использован для всех команд configure. Смысл этого и остальных ключей можно узнать, запустив ‘./configure –help’ или, при необходимости, посмотрев непосредственно сам файл configure. Использование ключей --disable-assembly и --host в данной команде обсуждается ниже в разделе 6.

После конфигурирования производим компиляцию и сборку библиотеки:

```
make -j5
```

Ключ -j используется для параллельной сборки, значение ключа следует устанавливать в зависимости от количества ядер процессора и размера оперативной памяти.

Если памяти достаточно, то рекомендуется указывать число N+1, где N - число ядер, обычно его можно узнать в /proc/cpuinfo. Если памяти совсем мало, то использование параллельной сборки не рекомендуется. Также использовать -j не рекомендуется для команд make check и make install.

Проверить успешность сборки можно следующей командой (все тесты должны проходить успешно):

```
make CFLAGS="-O0" check
```

Здесь используется отменяющий оптимизацию параметр CFLAGS="-O0". Связано это с тем, что есть отдельные версии GMP и отдельные версии GCC, такие что с оптимизацией один из тестов компилируется неправильно и дает ложную ошибку (это не означает наличие ошибки в GMP).

Установка собранной библиотеки GMP (файл libgmp.a) с заголовочными файлами и документацией производится командой:

```
sudo make install
cd ../..
```

3.2. Сборка MPFR

Надо скачать актуальную версию библиотеки с официального сайта и распаковывать ее:

```
wget https://www.mpfr.org/mpfr-current/mpfr-4.1.0.tar.xz
cd src
tar xaf ../mpfr-4.1.0.tar.xz
```

При конфигурировании необходимо указать путь к установленной ранее библиотеке GMP:

```
cd mpfr-4.1.0
./configure --disable-shared --enable-static \
--prefix=/opt/gcc-mke \
--with-gmp=/opt/gcc-mke
```

Оставшиеся действия аналогичны п. 3.1:

```
make -j5
make check
sudo make install
cd ../..
```

3.3. Сборка MPC

Получение и распаковка актуальной версии библиотеки:

```
wget --no-check-certificate \
https://www.multiprecision.org/dowloads/mpc-1.2.0.tar.gz
cd src
tar xaf ../mpc-1.2.0.tar.gz
```

Ключ --no-check-certificate потребовался, поскольку на данный момент на сайте

www.multiprecision.org просроченный сертификат.

При конфигурировании следует так же, как и в п. 3.2, указать расположение библиотек GMP и MPFR:

```
cd mpc-1.2.0
./configure --disable-shared --enable-static \
--prefix=/opt/gcc-mke \
--with-gmp=/opt/gcc-mke \
--with-mpfr=/opt/gcc-mke
```

Действия по сборке и установке библиотеки такие же, как и в предыдущих пунктах:

```
make -j5
make check
sudo make install
cd ../..
```

4. Сборка бинарных утилит

Бинарные утилиты и компилятор также можно собирать из архивов с исходными текстами определенной версии, как и библиотеки выше. Этот способ будет приведен позже. А сейчас продемонстрируем сборку из исходных текстов, полученных из git-репозитория.

Сперва следует клонировать репозиторий с бинарными утилитами и gdb (gdb - отладчик, его собирать не надо):

```
cd src
git clone \
git://sourceware.org/git/binutils-gdb.git
```

Далее надо извлечь из репозитория в рабочий каталог нужную нам версию:

```
cd binutils-gdb
git checkout binutils-2_37
cd ../..
```

Затем нужно создать отдельный каталог для сборки бинарных утилит (исходные тексты лежат отдельно) и перейти в него:

```
mkdir binutils-build
cd binutils-build
```

При конфигурировании бинарных утилит необходимо обратить внимание на маршрут к файлу configure и используемый ключ --srcdir):

```
../src/binutils-gdb/configure \
--srcdir=../src/binutils-gdb \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-gold --disable-gdb --disable-sim
```

Ключи --disable-gold, --disable-gdb, --disable-sim отменяют сборку программ, которые не понадобятся.

После конфигурирования производится непосредственно сборка и установка:

```
make -j5
sudo make install
cd ..
```

Проверка (make check) для бинарных утилит не запускается, так как тестов очень много, и не вся функциональность бинарных утилит имеет смысл для целевой машины без операционной системы. Поэтому не все тесты (по крайней мере для архитектуры MIPS) будут выполнены успешно. Соответственно, интерпретация результатов проверки требует содержательного анализа. То же самое, еще в большей степени, касается проверки компилятора.

Как было упомянуто выше, можно получить исходные тексты бинарных утилит (той же версии 2.37) скачав их с официального сайта, не клонируя весь репозиторий:

```
wget http://ftp.gnu.org/gnu/binutils/binutils-2.37.tar.bz2
```

или

```
wget --no-check-certificate \
https://ftp.gnu.org/gnu/binutils/binutils-2.37.tar.bz2
```

(опять просроченный сертификат).

Затем распаковываем архив в каталог src:

```
cd src
tar xaf ../binutils-2.37.tar.bz2
cd ..
mkdir binutils-build
cd binutils-build
```

Надо обратить внимание, что распакованный каталог с исходным текстами имеет другое имя: не binutils-gdb, а binutils-2.37. Поэтому потребуется немного модифицировать команду конфигурации, запускаемую в binutils-build:

```
../src/binutils-2.37/configure \
--srcdir=../src/binutils-2.37 \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-gold --disable-gdb --disable-sim
```

(или переименовать binutils-2.37 в binutils-gdb в каталоге src).

Далее производится сборка и установка бинарных утилит:

```
make -j5
sudo make install
cd ..
```

5. Сборка компилятора

Также, как и для бинарных утилит, следует начать с клонирования репозитория (вариант с распаковкой архива определенной версии будет приведен ниже):

```
cd src
git clone git://gcc.gnu.org/git/gcc.git
```

Далее извлекается в рабочий каталог требуемая версия:

```
cd gcc
git checkout releases/gcc-11.3.0
cd ../..
```

Затем надо создать каталог для сборки компилятора (исходные тексты лежат отдельно) и перейти в него:

```
mkdir gcc-build
cd gcc-build
```

Конфигурируем компилятор:

```
../src/gcc/configure --srcdir=../src/gcc \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-libssp --disable-gcov --disable-lto \
--enable-languages=c --enable-multilib \
--with-gmp=/opt/gcc-mke \
--with-mpfr=/opt/gcc-mke \
--with-mpc=/opt/gcc-mke
```

Если кроме компилятора C надо собрать также компилятор C++, то следует заменить ключ конфигурации

```
--enable-languages=c
```

на

```
--enable-languages=c,c++
```

и добавить ключ

```
--disable-libstdcxx
```

поскольку невозможно собрать библиотеку libstdc++ (библиотека времени выполнения GNU C++) для целевого процессора без операционной системы.

Затем следует сборка и установка компилятора:

```
make -j5
sudo make install
cd ..
```

Как и в случае с бинарными утилитами можно получить исходные тексты определенной версии компилятора gcc, скачав их с официального сайта без клонирования всего репозитория:

```
wget ftp://ftp.gnu.org/gnu/gcc/gcc-11.3.0/gcc-11.3.0.tar.gz
```

Заметим, что в последней команде использован доступ по протоколу FTP, а не HTTP или HTTPS, как ранее. Какой протокол использовать, зависит от настройки сети и политик безопасности системы, в которой происходит сборка. Кроме того, тот или иной протокол может просто временно (или постоянно) не поддерживаться сайтом, с которого скачиваются исходные тексты.

Затем распаковываем архив в каталог src:

```
cd src
tar xaf ../gcc-11.3.0.tar.gz
cd ..
mkdir gcc-build
cd gcc-build
```

Конфигурируем компилятор, не забыв сменить имя gcc на gcc-11.3.0:

```
../src/gcc-11.3.0/configure \
--srcdir=../src/gcc-11.3.0 \
--prefix=/opt/gcc-mke --target=mips-mke-elf \
--disable-libssp --disable-gcov --disable-lto \
--enable-languages=c --enable-multilib \
--with-gmp=/opt/gcc-mke \
--with-mpfr=/opt/gcc-mke \
--with-mpc=/opt/gcc-mke
```

Дальше так же, как ранее:

```
make -j5
sudo make install
cd ..
```

6. Рекомендации и проблемы

Следует помнить, что собранный компилятор (в отличии от бинарных утилит) нельзя просто копировать из каталога в каталог, так как он запускает свои проходы (cc1, as, и т.д.) по абсолютным маршрутам. Поэтому, если все же по какой-

то причине (например, при оптимизации дискового пространства) требуется переместить компилятор, то необходимо на старом месте поместить символьную ссылку на новый каталог компилятора, чтобы все старые маршруты остались корректными.

Если планируется использовать собранный компилятор разными пользователями на разных компьютерах, то рекомендуется выполнять все процедуры сборки от специального пользователя, созданного для этой цели. Иначе возможны проблемы, например, с тем что у реального пользователя, исполняющего вышеописанные процедуры могут быть нестандартные значения переменных окружения (PATH, LD_LIBRARY_PATH и т.п.).

Также, если планируется использовать компилятор и бинарные утилиты на разных компьютерах следует быть осторожным с разделяемыми библиотеками. В частности, можно заметить, что библиотеки GMP, MPFR и MPC собирались выше только в неразделяемом варианте (с расширением ‘.a’), для того, чтобы они включались в компилятор на стадии компоновки. Тем самым исключается путаница с версиями этих библиотек, которые могут быть уже установлены на компьютере.

Последняя рекомендация не является бесспорной: возможны ситуации, когда именно неразделяемая библиотека ограничивает переносимость собранной с ней программы. При сборке библиотеки GMP в п. 3.1 были использованы ключи конфигуратора --disable-assembly и --host=`./configfsf.guess`. Они нужны для того, чтобы помешать конфигуратору оптимизировать GMP с использованием возможностей конкретного процессора компьютера, на котором происходит сборка (следует заметить, что конфигуратор по умолчанию полагается на информацию от файла сценария config.guess).

```
./configure --disable-shared --enable-static \
--disable-assembly --host=`./configfsf.guess` \
--prefix=/opt/gcc-mke
```

Если не использовать ключ --host возможна ситуация, когда, например, конфигуратор опознает, что сборка библиотеки GMP происходит на компьютере с процессором Skylake и прописывает в файлах сборки (Makefile) использование инструкций из набора BMI2 (Bit Manipulation Instruction set 2). После того как полученная библиотека GMP была использована для сборки компилятора, при дальнейшей попытке запустить этот компилятор на компьютере с процессором Sandy Bridge произошло аварийное завершение компилятора, поскольку Sandy Bridge не поддерживает инструкции из BMI2.

Проблема с оптимизацией под конкретную модель процессора возникает только для библиотеки GMP, при конфигурировании остальных библиотек, а также бинарных утилит и самого компилятора, нет необходимости использовать ключ --host.

Кроме приведенных выше общих рекомендаций, для конкретных версий компилятора и бинарных утилит встречаются отдельные проблемы при их сборке. Например, если вместо использованной выше версии компилятора gcc-11.3.0 собирать более старую, но популярную версию gcc-7.5.0 (последнюю в седьмой ветке), то можно столкнуться со следующей ошибкой. При запуске команды ‘make -j5’ на 64-битном компьютере процесс сборки прерывается при попытке сконфигурировать каталог zlib (содержащий библиотеку сжатия данных). При этом конфигуратор библиотеки zlib выводит следующее сообщение:

```
error: Link tests are not allowed after GCC_NO_EXECUTABLES.
```

Это означает, что конфигуратор прекращает работу, потому что не знает, как собирать исполняемые файлы. Эта ошибка специфична именно для библиотеки zlib.

Причина в том, что файлы сборки содержат указание на компиляцию библиотеки zlib в двух вариантах: 64-битном и 32-битном, при том что 32-битный вариант, как правило, бесполезен, поскольку производится сборка 64-битного компилятора. Проще всего обойти эту ошибку, установив на 64-битную систему, где производится сборка, пакеты, требуемые для сборки 32-битных программ.

Например, в случае системы Linux Fedora Core достаточно при помощи команды dnf установить 32-битные варианты библиотеки glibc.i686 и glibc-devel.i686 (при этом будут установлены еще несколько 32-битных пакетов, необходимых для glibc):

```
sudo dnf install glibc.i686 glibc-devel.i686
```

Осталось заметить, что возможность сборки 32-битных программ на 64-битном компьютере легко проверить, выполнив команду:

```
gcc -m32 hello.c -o hello
```

где hello.c - любая программа на C, например, самая короткая:

```
main(){}
```

Некоторые версии бинарных утилит также

содержат разные ошибки в сценариях сборки. Например, для использованных выше бинарных утилит версии 2.37 в каталоге /opt/gcc-mke/share/man/man1 устанавливаются пустые man-страницы.

Это ошибка именно версии 2.37, исправленная в версии 2.38. Ее можно исправить, применив к исходным текстам бинарных утилит версии 2.37 заплатку (patch), доступную по ссылке [2]. Сделать это можно вручную (просто добавить две строки в texi2pod.pl) или при помощи команды

```
patch -p1 < файл_с_заплаткой
```

Как можно видеть на этих примерах, при сборке различных версий бинарных утилит и компилятора могут возникать разные проблемы, требующие внимательного изучения. В следствии этого, полная автоматизация процесса сборки возможна только для конкретных версий, после «ручной» сборки и проверки корректности и переносимости полученных инструментов.

7. Заключение

В каталог /opt/gcc-mke установлены бинарные утилиты и компилятор для целевой архитектуры mips-mke-elf. Исполняемые файлы компилятора и бинарных утилит (ассемблер, компоновщик, objdump и др.) находятся в каталоге /opt/gcc-mke/bin.

Кроме исполняемых файлов, включаемых файлов и библиотек в /opt/gcc-mke установлена документация в формате info, для ее просмотра необходимо правильно задать переменную окружения INFOPATH:

```
INFOPATH=/opt/gcc-mke/share/info info
```

Можно указать в качестве аргумента конкретную программу, например, компилятор:

```
INFOPATH=/opt/gcc-mke/share/info info gcc
```

или компоновщик:

```
INFOPATH=/opt/gcc-mke/share/info info ld
```

Также в /opt/gcc-mke/share/man устанавливаются man-страницы, для их просмотра необходимо указывать имя команды, под которым она помещена в каталог bin, например, mips-mke-elf-gcc для компилятора:

```
MANPATH=/opt/gcc-mke/share/man man mips-mke-elf-gcc
```

Описанный выше подход позволяет собирать разные версии бинарных утилит и компилятора. Для этого можно просто выполнить команду ‘git checkout' для другой версии. В статье описана сборка актуальных на 2021 год версий бинарных утилит и компилятора, но если планируется собирать компилятор и бинарные утилиты более старых версий, то могут потребоваться другие версии библиотек GMP, MPFR, MPC. Рекомендуемый согласованный набор старых версий: gmp-4.3.2, mpfr-2.4.2, mpc-0.8.1.

Разумеется, нельзя в одном тексте описать все, что касается сборки компилятора и бинарных утилит. Например, сознательно опущены вопросы, связанные с подписями скачанных архивов и их проверкой. Принципиально описывалась только «ручная» процедура, без попыток автоматизировать этот процесс сценариями на языке shell, общим файлом сборки make или с использованием утилиты rpmbuild.

«Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Building the GCC Compiler and Binary Utilities as Cross Tools

Vladimir Galatenko, Galina Levchenkova, Sergej Samborskij

Abstract. Developers using the freely distributed GCC compiler often face the problem of correctly building the compiler and the necessary binary utilities from source texts, as well as their configuration to the desired target architecture. The article provides a general sequence of actions for building of a cross-compiler and binary utilities for the MIPS architecture, suitable for different versions of the compiler. Examples of problems that may arise in this case and ways to solve them are given.

Keywords: compiler, binary utilities, GCC, building, MIPS

Литература

1. GCC: The GNU Compiler Collection, https:// https://gcc.gnu.org/.
2. Binutils commit, https://sourceware.org/git/?p=binutils-gdb.git;a=commit;h=2dad02b6