Для цитирования: Шалаев М. А. Сборка компонентов программного обеспечения вычислительных комплексов семейства «Эльбрус» // Вопросы радиоэлектроники. 2017. № 3. С. 39–43.
УДК 004.4'2

М. А. Шалаев[1]

[1] АО «МЦСТ»

СБОРКА КОМПОНЕНТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ СЕМЕЙСТВА «ЭЛЬБРУС»

Описаны понятия и методы, свойственные процессу формирования программного обеспечения новой компьютерной платформы из уже используемых на практике программных пакетов, которые нашли применение и развитие при создании нескольких поколений вычислительных средств семейства «Эльбрус». Особо рассматриваются проблемы кросс-сборки, связанные с кофигурированием, сборкой и компиляцией пакетов программ.

Ключевые слова: *кросс-компиляция, сборка программных пакетов, аппаратные платформы.*

Введение

Одной из принципиальных задач создания компьютерной платформы является подготовка программного обеспечения (ПО), состоящего из уже опробованных на действующих платформах компонентов, которое может эффективно исполняться на вновь спроектированной аппаратуре, без затруднений осваиваться пользователями и быть основой промышленных решений. При этом большую часть ПО необходимо сформировать, когда аппаратура находится еще на стадии прототипа или даже программной модели. Быстрое появление работоспособных программ позволяет провести более полное тестирование процессоров на этих этапах, выявить возможные несовершенства или ошибки в аппаратуре и исправить их до начала производства.

В статье описываются проблемы и особенности сборки программного обеспечения, выявленные и решенные при формировании дистрибутива операционной системы для вычислительных комплексов (ВК) АО «МЦСТ», построенных на базе микропроцессоров с архитектурами SPARC и «Эльбрус». Наряду с этим рассматривается сборка программ, которые могут являться конечным решением, поставляемым вне рамок дистрибутива.

Дистрибутив операционной системы

Дистрибутив операционной системы – это комплекс из операционной системы (ОС) и конгломерата программ различного назначения, доступных на данной платформе. Дистрибутив выступает основным источником ПО, решающего задачи пользователей при переходе на новую платформу.

Операционная система эльбрусовских ВК (ОС «Эльбрус») строится на базе ОС Linux. В наиболее крупных дистрибутивах Linux налицо стремление охватить весь набор ПО с открытым исходным кодом, который существует в настоящий момент, а также минимально возможный набор критически важных программ с закрытыми исходными кодами в бинарном виде. Современные дистрибутивы UNIX-подобных ОС включают в себя тысячи различных программных пакетов, обеспечивающих удобство работы для множества пользователей.

Большинство популярных дистрибутивов предлагают сборки для нескольких аппаратных платформ, но их портирование под новые аппаратные платформы [1] в общем случае является нетривиальной задачей, решение которой может занять значительное время. Она осложняется особенностями сборочных сред и используемого инструментария. Это отдельная тема, которая не будет рассматриваться в рамках данной статьи. Однако анализ, проведенный программистами АО «МЦСТ» относительно базового состава большинства популярных дистрибутивов, позволил обозначить проблемы и связанные с ними трудности, которые возникают при сборке под новую платформу.

Портирование программных пакетов

Первым этапом, на котором возникают проблемы при портировании программы (программного пакета) на новую платформу, является конфигурирование, приведение в соответствие со свойствами конкретной целевой системы и ее ОС. Поскольку конкретные ОС отличаются друг от друга многими деталями [2] (в частности, функциональностью библиотек, их названиями и расположением), одной из таких проблем является проверка зависимостей. Суть этого понятия состоит в том, что программный пакет *pkgname1* для установки и (или) функционирования требует наличия в системе пакета *pkgname2*,

а последний, в свою очередь, может потребовать пакет *pkgname3* и т.д. В качестве зависимостей часто (хотя и не всегда) выступают системные библиотеки, о которых сказано выше. Зависимости бывают двух типов: зависимости, соблюдение которых необходимо только при сборке (*depends*), и зависимости этапа исполнения (*runtime depends*). В большинстве случаев они эквивалентны, однако это правило имеет многочисленные исключения. В частности, библиотеки, которые использует пакет, при статической сборке требуются только в момент компиляции – в дальнейшем необходимости в них не возникает.

Следует различать зависимости жесткие и мягкие. Удовлетворение первых абсолютно необходимо при сборке. Так, практически любая программа использует (статически или динамически) главную системную библиотеку *glibc* (или *libc*), любое приложение для системы *X* – библиотеку *xlib*. В то же время мягкие зависимости не критичны для функционирования пакета – их удовлетворение лишь добавляет ему дополнительные функции, которые могут оказаться и лишними.

Можно сказать, что известны различные способы конфигурирования, для каждого из которых разработана своя система.

Autotools – это наиболее распространенная система, которая представляет собой набор утилит для генерации переносимых файлов сборки. Программы *autoconf* и *automake* создают из макроописаний файлы конфигурации, по результатам запуска которых для каждого пакета создается необходимый при сборке файл *Makefile* с правилами и флагами сборки. Целью работы скрипта *configure* является определение параметров системы, размеров используемых типов данных, наличие в системе установленных библиотек, их версий и необходимого функционала в них. Кроме того, может осуществляться проверка существования некоторых типов устройств, наличие проблем в используемой системной библиотеке или компиляторе. По результатам работы *configure* обычно создается файл *config.h*. В нем описан набор препроцессорных макросов, которые будут переданы компилятору в процессе сборки. Это дает авторам пакетов возможность реализовывать различное поведение программы в зависимости от конфигурации системы, на которой программа будет запускаться.

Cmake – это утилита для генерации файлов управления сборкой (*Makefile*) из файлов *CmakeLists.txt*. Система *CMake* обладает интеллектуальными средствами поиска инструментов сборки и библиотек на конкретной платформе (интроспекция) и автоматического конфигурирования. Благодаря этому она сама устанавливает многие параметры сборочных файлов, которые в других системах управления сборкой приходится устанавливать вручную. Результатом работы программы *Cmake* является файл, содержащий инструкции сборки приложения для конкретной платформы. Суть идеи заключается в том, что описание процесса сборки в файле *CMakeLists.txt* абстрагировано от конкретных особенностей как отдельных систем (расположение файлов, возможности компиляторов), так и целых платформ. Читая общее описание процесса сборки из файла *CMakeLists.txt*, программа *Cmake* создает файл инструкций сборки, учитывающий специфику конкретной системы.

Система *SCons* – это инструмент для автоматизации сборки программных проектов, разработанный как замена утилите *make*, при этом он обладает интегрированной функциональностью, аналогичной *autoconf/automake*. *SCons* имеет возможность автоматически анализировать зависимости между исходными файлами и требования к целевой операционной системе, а также генерирует конечные бинарные файлы для установки на целевую систему. Для конфигурации проектов используются файлы на языке *Python*.

Кросс-сборка программных пакетов

Кросс-сборка выполняется в системе, специальным образом подготовленной для сборки программных пакетов под другую систему. Применительно к вычислительным средствам семейства «Эльбрус» такая технология актуальна, ввиду того что разработка их ПО ведется уже тогда, когда базовый микропроцессор новой модели еще не изготовлен в кремнии и доступен только в виде прототипа, а первая партия выпущенных на его основе вычислительных комплексов еще не позволяет специально выделить ВК для построения дистрибутива.

Ниже приводятся основные категории проблем, связанных с кросс-сборкой.

Проблемы конфигурирования

Конфигурирование собираемого пакета – довольно сложный процесс, успешная реализация которого зависит от многих факторов, в т.ч. от окружения или от контекста, в котором этот процесс выполняется.

Для того чтобы сборка прошла успешно, конфигурирование в сборочной среде должно генерировать на выходе такие же данные, как если бы сборка осуществлялась на целевой платформе. Кроме этого, сборка пакета должна быть полностью воспроизводимой, т.е. контекст должен быть фиксируемым.

Как уже упоминалось, при конфигурировании и сборке пакетов используются сборочные зависимости. Они могут относиться не только к библиотекам, но и к исполняемым программам – например,

лексическим и синтаксическим анализаторам, различным генераторам документации и системных руководств. Помимо этого, в сборке могут использоваться интерпретаторы, такие как *Perl* и *Python*. Для того чтобы собираемые в кросс-режиме программы корректно работали на ВК с целевой архитектурой, окружение, в котором осуществляется сборка, должно быть согласовано с тем, в котором программы будут исполняться.

В процессе конфигурирования проводятся проверки не только непосредственно зависимостей пакетов, но и доступных свойств системы, в т.ч. размеров переменных различных типов, доступности необходимых символов в системных библиотеках или зависимостях, в некоторых случаях – наличия устройств и системных шин. В ряде ситуаций осуществить эти проверки в кросс-режиме невозможно из-за того, что они реализуются с помощью компиляции и исполнения тестовых программ, в зависимости от результатов выполнения которых выставляются значения флагов или макросов, используемых при дальнейшей сборке программного пакета. Несколько упрощает дело то, что применительно к большинству пакетов система конфигурации унифицирована и для различных пакетов требуется установка одинаковых переменных и макросов. В систему конфигурации *Autotools* заложена возможность использования значений, предварительно размещенных в кэш-памяти. Таким образом, решить возникающую проблему и получить большинство требуемых значений можно, единожды произведя конфигурацию на целевой платформе.

К сожалению, не все разработчики программных пакетов предусматривают возможность использования кэшированных данных в своих проверках. Некоторые тесты выставляют конфигурационные значения только в том случае, если исполнение бинарного файла завершилось успешно, а часть завершает процесс конфигурации ошибкой, если ведется сборка в кросс-режиме. Большинство проблем такого рода исходит из того, что разработчики программного пакета не предусматривают возможность сборки в кросс-режиме при написании базового сценария *Autotools*. Многие из представляемых в них макросов дают возможность задать специальное поведение в случае кросс-компиляции. Для того чтобы осуществить сборку таких пакетов в кросс-режиме, требуется проводить доработку конфигурационных файлов.

До сих пор не все необходимые для дистрибутивов пакеты имеют конфигурационные скрипты. Некоторые распространяются только с *Makefile*, и определить все параметры для таких пакетов становится гораздо сложнее, поскольку их поведение может закладываться на конкретную реализацию некоторого функционала. Наиболее часто можно столкнуться со следующими ситуациями:

- Наличие в среде пользователя переменных, которые используются в процессе исполнения *Makefile*. Если в окружении присутствуют переменные, которые могут негативно влиять на процесс сборки, их необходимо обнулить.
- Задание параметров системы через исполнение системных команд, например, определение разрядности системы через исполнение утилиты *uname*. Решить проблемы такого рода можно либо доработкой программных пакетов, с тем чтобы требуемые значения задавались с помощью опций или переменных среды, либо заменить в сборочной среде вызываемые утилиты на возвращающие значения, соответствующие целевой платформе.
- Зависимость проведения сборки от конкретной реализации программы в окружении. Такая ситуация может возникать, например, когда директория ее установки зависит от выдачи опции, которая присутствует только в специальных сборках компилятора.

С целью минимизации затрат ресурсов при портировании программ для большинства описанных проблем этапа конфигурации можно найти системные решения, как то: использование сгенерированных кэш-данных или фиксация согласованного окружения.

Проблемы сборки программ

Основной проблемой этого этапа являются архитектурно-зависимые участки кода. В программных модулях ряда программ присутствует код (как правило, под макросами), поведение которого меняется в зависимости от конкретной архитектуры. Возможно также, что реализация некоторых функциональных участков связана с программными соглашениями, принятыми для конкретной архитектуры.

В частности, в компилируемом коде могут использоваться ассемблерные вставки с реализацией той или иной функциональности. В тех случаях, когда это сделано для ускорения (оптимизации) критических функций, разработчики, как правило, оставляют так называемую generic-реализацию, основанную на использовании стандартной системной библиотеки. При первичной сборке в большинстве случаев вообще не возникает проблем, и архитектурно-зависимые части можно будет доработать в будущем для ускорения работы программы. В тех случаях, когда разработчики не предусмотрели generic-реализации, требуется проводить анализ кода для имеющихся платформ и разрабатывать по аналогии реализацию для проектируемой

аппаратной платформы. Это в общем случае нетривиальная задача, которая может занять значительное время разработчиков.

Проблемы, связанные с компилятором

Многие существующие front-end модули компиляторов не имеют полной совместимости с компилятором GCC. При этом сборка большинства свободно распространяемых программных продуктов проверяется разработчиками исключительно компилятором GCC, а в коде используются GNU-расширения, не поддерживаемые стандартом. Кроме того, при сборке могут использоваться флаги компиляции, которые поддержаны только в компиляторе GCC и не поддержаны в других либо имеют в них иную семантику. Решать эти проблемы можно, либо добавляя поддержку разрабатываемой архитектуры в дерево исходных кодов компилятора GCC, либо обеспечивая совместимость с GCC собственного компилятора. Остается также возможность проводить доработку сборочных средств конкретного пакета, с тем чтобы ввести в них проверки на сборку под новую архитектуру и использовать для нее собственный набор опций.

Заключение

При реализации новых аппаратных решений необходимо уже на ранних этапах формировать ПО, которым смогут воспользоваться конечные пользователи. Для таких целей наиболее подходит кросс-компиляция программных пакетов, которая позволяет подготовить окружение для тестирования прототипов и готовых устройств, затрачивая на сборку минимальные ресурсы. При использовании этого метода приходится сталкиваться с рядом трудностей, большинство из которых решается настройкой сборочного окружения.

Проблемы при портировании программных пакетов под новую платформу можно связать с двумя этапами – конфигурированием и сборкой. Проблемы первого этапа решаются формированием сборочной среды, в которой используются согласованные версии пакетов и набор параметров для целевой архитектуры. Проблемы второго этапа напрямую связаны с применяемыми компилятором и целевой аппаратурой. При использовании компилятора, отличного от GCC, нужно быть готовым к тому, что могут потребоваться доработки в сборочных скриптах, вызванные различным поведением front-end модулей.

Наибольшие сложности при портировании возникают из-за вставок, вызванных особенностями конкретной архитектуры, либо вставок машинного кода целевой архитектуры. Для решения таких проблем необходим анализ исходного кода, требующий знания особенностей, системы команд и программных соглашений целевой архитектуры.

На сегодняшний день программистами АО «МЦСТ» разработана система кросс-сборки программных пакетов для дистрибутива ОС «Эльбрус». В данной системе реализован функционал, позволяющий в автоматическом режиме разрешать большинство проблем этапов сборки и конфигурирования, а также производить сборки дистрибутива под все разрабатываемые аппаратные платформы. В разработке находятся утилиты для автоматизированного преобразования ассемблерных вставок, присутствующих в кодах программ, в коды семейства «Эльбрус».

СПИСОК ЛИТЕРАТУРЫ

1. Walker A., Frischknecht S. Guide to port Linux on x86 applications to Linux on Power [Электронный ресурс]. 2014. 24 с. URL: http://www.ibm.com/developerworks/library
2. Гриневич А., Марковцев Д., Рубанов В. Проблемы совместимости Linux-систем [Электронный ресурс]. URL: http://www.osp.ru

ИНФОРМАЦИЯ ОБ АВТОРЕ

Шалаев Михаил Андреевич, начальник сектора, АО «МЦСТ», 119334, Москва, ул. Вавилова, д. 24, тел.: 8 (495) 363-96-65, e-mail: mikhail.a.shalaev@mcst.ru.

For citation: Shalaev M.A. Building software for «Elbrus» microprocessor family. Voprosy radioelektroniki, 2017, no. 3, pp. 39–43.

M. A. Shalaev

BUILDING SOFTWARE FOR «ELBRUS» MICROPROCESSOR FAMILY

The article reviewed the best practices and methods used to port software for a net computer platform. Usually new software comes in a form of packages. Given the nature of The «Elbrus» system these packages need to be builded for a specific system version. The article describes build process and configuration as well as most common issues.

Keywords: cross compiling, building program packages, computer platforms.

REFERENCES

1. Walker A., Frischknecht S. Guide to port Linux on x86 applications to Linux on Power. 2014. 24 p. Available at: http://www.ibm.com/developerworks/library.
2. Grinevich A., Markovtsev D., Rubanov V. Problemy covmestimosti Linux-sistem [Compatibility Problems Linux-systems] (In Russ.). 2007. Available at: http://www.osp.ru.

AUTHOR

Shalaev Mikhail, head of sector, JSC «MCST», 24, Vavilova st., Moscow, 119334, Russian Federation, tel.: +7 (495) 363-96-65, e-mail: mikhail.a.shalaev@mcst.ru.