УДК 004.057.5

К вопросу о переносимости кода и некоторых возможностях использования кроссплатформенного программного обеспечения

© А.В. Шикуть

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Статья посвящена вопросам переносимости кода в приложениях на языке C/C++. Рассмотрены основные проблемные вопросы, связанные с переносимостью кода на другую платформу, а также некоторые наиболее широко используемые способы и пути повышения переносимости, в том числе, путем использования кроссплатформенного программного обеспечения. Особое внимание уделено использованию в приложениях кроссплатформенного графического интерфейса. Рассмотрена возможность создания графического интерфейса с помощью кроссплатформенной библиотеки GTK+ в среде свободного программирования QT creator.

Ключевые слова: *переносимость кода, кроссплатформенное программное обеспечение, платформа, интерфейс, среда, приложение, среда программирования.*

Одной из наиболее важных проблем в процессе разработки приложений является проблема переносимости кода с одной платформы на другую. Задача сделать приложение переносимым на другие платформы является весьма актуальной и по сей день. Реально получаемый код зависит от особенностей реализации конкретного компилятора и типа использованного процессора. Программа, написанная на определенном языке программирования, требует определенного компилятора для преобразования в машинный код для данного процессора. При этом программа, которая поддается переносу на другой компьютер, оснащенный другой операционной системой и другим процессором, считается переносимой, мобильной, машинонезависимой. Одним из путей повышения переносимости кода является использование кроссплатформенного программного обеспечения. Программное обеспечение, работающее более чем на одной платформе, является кроссплатформенным.

Известно, что переносимость приложений зависит от многих факторов. В первую очередь, следует назвать аппаратную платформозависимость. Так как машинный код программы содержит команды вполне определенного процессора, то исполняемый файл не может быть запущен на другой аппаратной платформе, содержащей аппаратный и программный комплекс, который является основой только для определенной вычислительной системы. При этом следует учитывать особенности кода, написанного под разные архитектуры по разрядности машинного слова и адреса (с 8-, 16-, 32-, 64-разрядной адресацией).

Переносимость кода напрямую зависит от операционной системы (ОС) благодаря тому, что обычные исполняемые файлы содержат большие объемы данных, не являющиеся непосредственно компьютерной программой, в том числе, описание программного окружения, в котором программа может быть выполнена; данные для отладки программы; используемые константы; данные, которые могут понадобиться ОС для запуска процессора и др. Кроме того, исполняемые файлы также содержат вызовы системных библиотечных функций. И почти всегда набор системных библиотек и формат исполняемого файла являются уникальными для определенной операционной системы. Именно поэтому исполняемые файлы, разработанные для одной ОС, не могут быть запущены в другой ОС. Обычно код программы адаптируется для конкретной ОС, под управлением которой предполагается ее использование. Так, например, программа, предназначенная для использования в режиме потоковой многозадачности, не может быть использована для 16-разрядной ОС.

Таким образом, с одной стороны, привязка к особенностям определенной ОС необходима, чтобы получить хорошее быстродействие, а с другой стороны, зависимость от ОС усложняет процесс переноса программы.

Некоторые способы и пути повышения переносимости. Хотя и не существует жестких правил, позволяющих минимизировать зависимость разрабатываемых программ от ОС, однако можно сформулировать некоторые рекомендации, которых следует придерживаться. Например, следует отделять интерфейсную часть кода от реализации, чтобы отделить ту часть программы и те модули, которые взаимодействуют с ОС. В целях обеспечения переносимости кода между операционными системами и компиляторами следует избегать применения платформозависимых библиотек при разработке различных модулей.

Соответственно, созданный программный код требует соответствующего компилятора для преобразования исходного кода в машинный для определенного типа процессора.

Один из путей повышения переносимости программы заключается в том, чтобы сделать параметры независимыми от системы или процессора путем использования макроса *#define* [1]. Это позволит сделать параметры независимыми от операционной системы или от процессора. С помощью *#define* можно определять данные, которые могут изменяться при переносе программы, например, размер буфера, используемого при обращении к диску, или специальные команды при распределении памяти Define-определения упрощают выполнение всей работы, так как при этом значения таких данных необходимо изменить только один раз и в одном месте, а не во всей программе.

Кроме того, если надо получить переносимый код, не следует ориентироваться на ожидаемые размеры данных. При этом стоит учитывать различия между 16- и 32-разрядной средой разработки. Известно, что размер слова в 16-разрядном процессоре равен 16 бит = = 2 байта, а в 32-разрядном — 32 бита = 4 байта. Размер слова часто совпадает с размером данных типа int. Код, созданный в предположении, что переменные типа int являются, например, 16-разрядными, не будет корректно работать после переноса его в 32-разрядную среду. Чтобы избежать жесткой привязки к размеру типа, следует вначале определить размер типа в байтах с помощью оператора *sizeof()*.

Например:

```
fread(&j, sizeof(int),1, stream);
```

Такой код является более предпочтительным, так как это выражение заносит значение типа int в дисковый файл и будет работать в любой среде программирования.

Для того чтобы создать программу под разные платформы, в общем случае требуется набор компиляторов (и компоновщиков) под разные платформы, каждый из которых способен компилировать (и компоновать) под платформу, на которую он установлен, а также набор библиотек под разные платформы. При этом для выбора компилируемого кода можно использовать директивы условной компиляции. Так, например, при использовании компилятора C++ из GCC можно использовать Common Predefined Macros _GNUG_, который, в свою очередь, можно применять для выбора компилируемого кода на этапе работы препроцессора:

```
#ifndef _GNUG_
…..код для компилятора GCC …
#else
….код для компилятора MSVS…
#endif
```

Использование кроссплатформенного программного обеспечения. Для повышения возможностей переносимости может быть использовано кроссплатформенное обеспечение, позволяющее работать более чем на одной платформе. Использование кроссплатформенных средств разработки (компиляторов, библиотек и фреймворков) является одним из путей повышения переносимости программного кода.

Процесс сборки приложения под платформу, отличную от платформы, на которой запущены компилятор и компоновщик, называется кросскомпиляцией.

Для получения объектного кода программы программистами используются трансляторы, которые, в свою очередь, могут быть ком-

пиляторами или интерпретаторами. Программа на интерпретируемом языке выполняется специальной программой, называемой интерпретатором. При этом программа остается на исходном языке и не может быть запущена без интерпретатора.

Компилятор GCC (GNU Compiler Collection) — это кроссплатформенный, свободно распространяемый набор компиляторов, в который входит в том числе и компилятор языка C++ — g++ [2]. GCC включает в себя возможности компоновщика и позволяет создавать программы под несколько десятков аппаратных платформ и под множество операционных систем, включая GNU/Linux, MS Windows, Mac OC и BSD.

Известны кроссплатформенные интерпретируемые языки, интерпретаторы которых существуют для многих платформ. Примерами интерпретируемых языков являются ActionScript, Perl, Python, Ruby, PHP, MathCad.

Некоторые языки, например, Java и C#, находятся между компилируемыми и интерпретируемыми [3]. В таком случае сначала программа компилируется не в машинный код, а в машинно-независимый код низкого уровня, байт-код; затем байт-код выполняется виртуальной машиной. Для выполнения байт-кода обычно используется интерпретация, хотя отдельные его части для ускорения работы программы могут быть транслированы в машинный код непосредственно во время выполнения программы по технологии компиляции «на лету» (Just-in-time compilation, JIT). Для Java байт-код исполняется виртуальной машиной Java (Java Virtual Machine, JVM), для C# — Common Language Runtime. Среды исполнения Java Virtual Machine и .NET также являются кроссплатформенными, а программы, написанные на Java и C#, могут быть запущены под разными ОС без предварительной перекомпиляции.

Виртуальные машины также можно запускать под разные платформы. В таком случае программы будут являться кроссплатформенными. Таковыми можно назвать большинство современных языков программирования. Например, C, C++ и Free Pascal — кроссплатформенные языки на уровне компиляции, т.е. для этих языков имеются компиляторы под различные платформы. Не менее важным для кроссплатформенности являются стандартизованные библиотеки времени выполнения. Так, например, стандартом является библиотека языка C. Из крупных кроссплатформенных библиотек следует назвать Qt, GTK+, STL, FLTK, Boost, SDL, OpenAl, OpenCl.

Использование кроссплатформенного пользовательского интерфейса. Особого внимания заслуживает кроссплатформенный пользовательский интерфейс. Дело в том, что на разных ОС стан-

дартные элементы интерфейса имеют разные размеры. Именно поэтому позиционирование элементов интерфейса требует особого подхода [4]. При этом существует несколько подходов.

1. Единый стиль, общий для всех ОС. Программы выглядят одинаково под всеми ОС. Так работают интерфейсные библиотеки Java. При этом можно жестко расставлять элементы управления, наподобие Delphi; имеет место оригинальный стиль. Но системе приходится иметь свои экранные шрифты, а стиль отличается от стиля ОС.

2. Самоадаптирующийся интерфейс, подстраивающий сетку под реальные размеры элементов управления. Так работают интерфейсные библиотеки wxWidgets. При этом используется стандартный очень быстрый стиль ОС под Windows XP, Vista. Но для того, чтобы собрать самоадаптирующуюся сетку, требуется квалифицированный программист, а плотная компоновка затруднена.

3. Гибридный подход реализации, наподобие GTK+. При этом шрифты можно брать из системы, а не свои. Но в этом случае имеют место все недостатки первых двух подходов. Стиль отличается от стиля ОС, а плотная компоновка затруднена.

В любом случае под другими ОС требуется хотя бы минимальное тестирование, так как возможны ошибки компоновки.

Большое количество прикладных программ также являются кроссплатформенными, особенно те программы, которые изначально разрабатывались для Unix-подобных операционных систем. Важным условием их переносимости на другие платформы является использование компилятора GCC для платформы, на которую осуществляется перенос. GNU Compiler Collection представляет собой набор компиляторов для различных языков программирования [2]. При использовании GCC для компиляции кода под разные платформы применяется один и тот же синтаксический анализатор.

Современные операционные системы также зачастую являются кроссплатформенными (в аппаратном смысле этого слова). Например, операционные системы с открытым исходным кодом, такие как NetBSD, GNU/Linux, FreeBSD, могут работать на большом диапазоне различных платформ. Операционная система NetBSD является самой переносимой, она может быть установлена на большинство существующих платформ. Microsoft Windows может работать как на платформе Intel x86, так и на Intel Itanium.

Так или иначе, основными препятствиями на пути обеспечения полной кроссплатформенности остаются средства работы с динамическими библиотеками, связанные с .dll.

Использование графического интерфейса. Рассмотрим некоторые особенности создания графического интерфейса с помощью кроссплатформенной среды GTK+ [5].

Кроссплатформенная среда GTK+ (сокращение от GIMP ToolKit) представляет собой библиотеку функций графического интерфейса, так называемых виджетов (Widget). GTK+ написана на языке C и предназначена для работы в операционной системе GNU/Linux, Mac OS X, других Unix-подобных системах, а также в операционной среде Windows. Наряду с Qt, GTK+ является одной из наиболее популярных на сегодняшний день библиотек для X Window System. Изначально среда GTK+ являлась частью графического редактора GIMP, впоследствии развилась в отдельный самостоятельный проект и приобрела значительную популярность. GTK+ представляет собой свободное программное обеспечение и позволяет создавать как свободное, так и проприетарное программное обеспечение. Кроме того, GTK+ является официальной библиотекой для создания графического интерфейса проекта GNU. Несмотря на то, что GTK+ написана на языке C, она является объектно-ориентированной средой и позволяет легко создавать интерфейсы для других языков программирования таких как Ada, C, C++, C# и других языков программирования платформы .NET, а также Fortran, FreeBASIC, Free Pascal, Java, JavaScript, PureBasic, Python, Smalltalk.

Рассмотрим некоторые особенности создания приложений на языке C++ с использованием графического интерфейса кроссплатформенной библиотеки GTK+ в среде разработки Qt Creator и компилятора GCC. Для решения задачи преобразования матриц организуем обработку событий взаимосвязанных объектов с помощью системы управления событиями. При этом для реализации событий используются сигналы (*signal*) и слоты (*slot*) [5].

Сигнал генерируется в ответ на некоторое событие и несет в себе информацию об этом событии

Слот — это функция, которая вызывается в ответ на определенный сигнал. Следует иметь в виду, что если раньше требовалось отмечать слоты ключевым словом *slots*, то в последнее время, так как реализация слотов стала основаться на шаблонах (*templates*), необходимость в этом ключевом слове отпала. В целях поддержки старого исходного кода эта возможность языка перешла в состояние *deprecated.*

К сигналам и слотам предъявляются следующие требования:

1) сигналы ничего не должны знать о слотах, в которые они отправляются;

2) сигналы несут в себе некоторые обобщенные данные;

3) слот должен знать, какие данные он хочет получить от сигнала;

4) слот не должен знать, кто отправил сигнал;

5) допускаются множественные соединения (сигнал может быть присоединен к разным слотам, и к одному слоту может быть подсоединено несколько сигналов);

6) сигналы и слоты должны иметь свои имена.

С точки зрения разработчика, система сигнал/слот состоит из трех частей:

1) сигналы;

2) слоты;

3) менеджер соединений.

Сигнал — это некий именованный объект, который может хранить в себе данные. Слот — это в общем случае указатель на некоторую функцию, которой будет передан объект типа Сигнал. Менеджер соединений — набор функций, который обеспечивает соединение сигналов и слотов, а также доставку сигналов по их назначению.

Самая простая часть в реализации — сигналы. Учитывая предоставляемые требования, реализация **сигнала** для рассматриваемого случая может иметь вид

```
class Signal
{
….
}
Слоты:
void on_ spnMatrix1NRow_valueChanged(int nrow);
void on_ spnMatrix1NCol_valueChanged(int ncol);
void on_ spnMatrix2NRow_valueChanged(int nrow);
void on_ spnMatrix2NCol_valueChanged(int ncol);
void setStretch(QTableWidget *table);
void setCentered(QTableWidget *table);
void fillZero(QTableWidget *table);
void makeGood(QTableWidget *table);
void on_btnTranspose_clicked();
qreal value(QTableWidget *table, int I, int j);
```

Менеджер сигналов должен реализовывать набор базовой функциональности [5].

1. Регистрация слота необходима для того, чтобы менеджер узнал, что такой слот вообще существует; при этом множество сигналов связываются со множеством слотов.

2. Отмена регистрации слота — соответственно, нужна, чтобы забыть о слоте. При этом здесь необходимо также отсоединить все сигналы от данного слота, что позволяет по имени слота найти соответствующий объект.

3. Соединение и рассоединение сигнала и слота (connect/ disconnect) необходимы, чтобы задать соответствие слота и сигнала; при этом соединение слота сводится к одной вставке в SignalMap. А его отсоединение — задача более сложная, требует удалить из SignalMap все записи для указанного сигнала;

4. Посылка сигнала — такой метод, который должен обеспечивать поиск слота и отправку ему сигнала. Также важно иметь воз-

можность по возврату слота и определению, стоит ли дальше посылать этот сигнал остальным слотам. При этом функция посылки сигнала в такой схеме хранения также становится достаточно простой: необходимо найти первое и последнее вхождения имени сигнала в SignalMap и для каждого вхождения в этом диапазоне вызвать соответствующий слот через маппинг SlotMap;

5. Определенный метод посылки сигнала по указанному адресу. При этом получить данный сигнал должен только указанный слот.

В результате такой несложной реализации приложения с помощью графического интерфейса получается весьма мощная система управления событиями, позволяющая создавать достаточно переносимый код на другие платформы.

Заключение. Таким образом, рассмотрены основные проблемные вопросы, связанные с переносимостью кода на другую платформу, а также некоторые наиболее широко используемые способы и пути повышения переносимости, в том числе путем использования кроссплатформенного программного обеспечения. Показано, что с помощью кроссплатформенной библиотеки GTK+ с большим успехом может быть создан графический интерфейс приложения на языке C++. На практике при использовании ее в лабораторных работах библиотека показала себя быстрой в работе и легко переносимой на разные платформы. Делать интерфейсы пользователя достаточно легко, особенно при использовании дизайнера формы.

ЛИТЕРАТУРА

[1] Шилдт Герберт. *Полный справочник по C*. Москва, Издательский дом «Вильямс», 2005, 704 с.

[2] Bertrand Meyer. Approaches to portability. *JOOP (Journal of Object-Oriented Programming)*, vol. 11, N 6, July-August 1998, pp. 93–95.

[3] Tanenbaum A.S. *Structured computer organization 5th Print*. Amazon Prentice Hall, 2005, 800 p.

[4] Hook B. *Write portable code: an introduction to developing software for multiple platforms*. No Starch Press, 2005, 248 p.

[5] *Переносимость.* alice.pnzgu.ru/~dvn/uproc/books/site_tarasov/c15_port.html

Статья поступила в редакцию 10.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Шикуть А.В. К вопросу о переносимости кода и некоторых возможностях использования кроссплатформенного программного обеспечения. *Инженерный журнал: наука и инновации*, 2013, вып. 6. URL: http://engjournal.ru/catalog/it/hidden/817.html

Шикуть Алла Васильевна — канд. техн. наук, доцент кафедры «Программное обеспечение ЭВМ и информационные технологии» МГТУ им. Н.Э. Баумана. Автор около 10 научных работ. Область научных интересов: информатика, программирование, переносимость кода. e-mail: alla.shikut@rambler.ru