Разработка среды программирования для мобильных устройств под управлением операционной системы Android

Р. К. Лебедев, И. А. Корякин

*Новосибирский государственный университет
Новосибирск, Россия*

Аннотация

Мобильные устройства часто используются для образовательных целей в связи с их высокой доступностью и портативностью. Однако образовательные возможности мобильных устройств ограничены из-за отсутствия качественных IDE. Большинство сред разработки языка C++ для Android не имеют автодополнения и анализа кода, а остальные предоставляют эти возможности с крайне низкой производительностью. В этой работе мы описываем технологию переноса программного обеспечения, примененную к компилятору Clang (и другим необходимым инструментам), и предлагаем метод ускорения компиляции, автодополнения и анализа кода в нашей IDE. Наряду с этим мы предлагаем методы оптимизации пользовательского интерфейса. В работе описан основанный на регулярных выражениях эвристический метод для генерации предкомпилированных заголовков, который имеет низкие требования к процессору и ОЗУ. Эффективность метода проверена на различных образовательных примерах.

Ключевые слова

портирование ПО, Android, оптимизация, мобильные устройства, образование, IDE

Для цитирования

Лебедев Р. К., Корякин И. А. Разработка среды программирования для мобильных устройств под управлением операционной системы Android // Вестник НГУ. Серия: Информационные технологии. 2019. Т. 17, № 2. С. 91–100. DOI 10.25205/1818-7900-2019-17-2-91-100

Development of IDE for Android Devices

R. K. Lebedev, I. A. Koryakin

*Novosibirsk State University
Novosibirsk, Russian Federation*

Abstract

Mobile devices are frequently used for educational purposes considering their high availability and portability. However, educational abilities of mobile devices are quite limited due to the lack of proper IDE applications. Most Android integrated development environments for C++ are lacking code completion and real-time code analysis, while others provide these features at unacceptable speeds.

Therefore, we decided to develop a next-generation mobile C++ IDE that will not only have these features, but will also provide them at real-time speeds. This will make mobile devices more suitable for learning C++ and will provide students with an opportunity to leave their laptops at home and to learn C++ in different situations, where usage of laptop is not convenient.

In this paper, we describe the software porting technology applied to Clang compiler and other required tools and propose the method to speed up compilation, code completion and code analysis in our IDE. We also propose UI optimization methods. We also describe a regular expression-based heuristic algorithm for precompiled header generation that has small processor time and RAM requirements. The efficiency of the algorithm is tested on different educational code examples.

Our experiments show that our IDE is the only one at the moment, that can keep code completion real-time in all usage scenarios. The performance gain is 300 % average, when STL is used, reaching 3350 % when Boost library is involved. There is a small performance loss when C is used, but it is not significant, as the performance is still acceptable for real-time usage.

Keywords

software porting, Android, optimization, mobile, education, IDE

For citation Lebedev R. K., Koryakin I. A. Development of IDE for Android Devices. *Vestnik NSU. Series: Information Technologies*, 2019, vol. 17, no. 2, p. 91–100. (in Russ.) DOI 10.25205/1818-7900-2019-17-2-91-100**Введение**

Развитие мобильных устройств приводит к значительному расширению числа областей, в которых они могут использоваться. Одной из таких новых областей является мобильное образование. В магазинах приложений популярных мобильных операционных систем появились приложения, обучающие языкам, точным и гуманитарным наукам [1]. Присутствуют в магазинах и приложения, обучающие различным языкам программирования.

Растущая производительность мобильных устройств позволила перенести на них и среды разработки языков программирования. В частности, существуют такие приложения, как RYDroid, интегрированная среда разработки языка Python, или C4droid, среда разработки языка C++. Данные приложения позволяют обучающимся избежать необходимости брать с собой на место учебы ноутбук или же и вовсе заменяют персональный компьютер, что наиболее важно в странах, где учащиеся зачастую могут позволить себе смартфон, но не могут приобрести ноутбук. Существуют и случаи замены персональных компьютеров в терминальных классах на планшеты под управлением операционной системы Android с установленной мобильной средой разработки¹.

Как правило, под интегрированной средой разработки понимают программный продукт, обладающий следующими возможностями:

- редактирование текста;
- компиляция или интерпретация исходного кода;
- подсветка синтаксиса;
- автоматическое дополнение кода;
- проверка ошибок во время редактирования;
- установка и подключение сторонних программных библиотек.

Некоторые из данных возможностей требовательны к ресурсам процессора. В случае языка C++ для автоматического дополнения кода и проверки ошибок в реальном времени требуется произвести большую часть шагов, обычно выполняемых при компиляции кода, за исключением связывания и трансляции программы в язык ассемблера. Подходы, используемые в средах программирования, предназначенных для рабочих станций, зачастую неприменимы на мобильных устройствах, поскольку в силу меньшей производительности последних не обеспечивают комфортной работы в режиме реального времени.

Существуют и особенности, связанные с размером экранов мобильных устройств: многим интегрированным средам разработки для обычных компьютеров свойственно наличие большого числа пунктов меню и кнопок, которые невозможно разместить в мобильном приложении [2].

Операционная система Android использует свою стандартную библиотеку языка Си, Bionic. По этой причине пользователи не могут работать со статическими или динамическими библиотеками, загруженными с их официальных сайтов: количество библиотек, собранных для Bionic, очень мало, большинство библиотек собраны для библиотек языка Си популярных дистрибутивов Linux или операционной системы Windows. Единственным оставшимся путем является самостоятельная сборка библиотек из исходного кода, что затруднительно для целевой аудитории программ такого рода: студентов и школьников.

Целью данной работы является разработка мобильной среды программирования языка C++, обладающей средствами автодополнения кода и проверки ошибок в режиме реального

¹ BVBCET (Индия) в 2012 г. лицензировал 900 копий приложения C4droid для использования на планшетных ПК.

времени, а также удобным для мобильного устройства интерфейсом и возможностью легкой установки популярных библиотек.

В работе описывается процесс переноса необходимых программных библиотек и программ на платформу Android, описан процесс сборки библиотек для программ конечного пользователя, а также рассмотрены методы оптимизации пользовательского интерфейса. Наряду с этим анализируются существующие подходы для ускорения автодополнения и анализа кода, а также описывается новый алгоритм, разработанный в рамках данной работы. Также приводится сравнение производительности разработанной среды программирования со всеми популярными конкурентами.

Выбор библиотек и программ для среды разработки языка C++

Типичным для программирования является следующий набор программных инструментов:

- командный интерпретатор;
- компилятор;
- набор утилит для работы с исполняемыми файлами;
- система автоматизации сборки программ из исходного кода.

В качестве командного интерпретатора был выбран проект BusyBox, поскольку он обладает небольшими требованиями к производительности, а также позволяет встроить базовый набор утилит командной строки, которые могут присутствовать не на всех версиях операционной системы Android, решая проблему совместимости.

В качестве компилятора был выбран проект Clang/LLVM, поскольку он предоставляет наиболее широкие возможности по анализу и автодополнению кода, а также обладает удобной лицензией.

Для работы с исполняемыми файлами был выбран проект GNU Binutils, поскольку он является самым популярным и стабильным проектом такого рода. В первую очередь данный проект обеспечивает связывание объектных файлов в исполняемый файл при помощи линкера.

Системой сборки, используемой для проектов, состоящих из нескольких файлов, была выбрана пара CMake и GNU Make. CMake легче в освоении для начинающих пользователей и позиционируется как более дружелюбный к кросс-компиляции, что обеспечивает возможность работы с одним и тем же проектом и на ПК, и на мобильном устройстве. Однако CMake не осуществляет непосредственную сборку, а только генерирует промежуточные файлы для других систем, поэтому возникла необходимость перенести и программу GNU Make. Последний, в свою очередь, может пригодиться и более опытным пользователям программы, предпочитающим эту систему сборки.

Перенос необходимых программ и библиотек на платформу Android

Кросс-компиляция – процесс сборки исходного кода на одной платформе для исполнения на другой. Для операционной системы Android официально рекомендуется использование программного инструментария Android NDK (*Android Native Development Kit*), позволяющего собирать код для мобильных устройств с использованием персонального компьютера [3]. На практике список поддерживаемых Android NDK по умолчанию систем сборки невелик: поддерживаются Soong, Android.mk и CMake. Первые две системы сборки эксклюзивны для операционной системы Android, поэтому мало полезны для переноса программного обеспечения, написанного для других платформ. CMake – более применимая в этом отношении система сборки, однако в случае использования ее с Android NDK возникают некоторые нюансы, например необходимость ручного редактирования файлов конфигурации.

Поскольку необходимое для проекта программное обеспечение использует различные системы сборки, не поддерживаемые Android NDK, было решено собрать более традиционный набор инструментов для кросс-компиляции (*cross-compiler toolchain*), совместимый с системами сборки GNU Make и GNU Autotools [4]. Для этого был использован сценарий

командной строки `make-standalone-toolchain.sh`, доступный в комплекте поставки Android NDK.

При сборке инструментов были выбраны библиотека языка C++ LLVM `libc++` и компилятор Clang. Данный выбор параметров обусловлен тем, что позволяет переиспользовать собранные на рабочей станции библиотеки непосредственно в мобильной среде разработки. Поскольку библиотека языка Си, библиотека языка C++ и компилятор совпадают, была обеспечена полная двоичная совместимость между библиотеками и заголовками кросс-компилятора и целевой системы. Таким образом, при помощи этой же системы сборки можно собирать и образовательные библиотеки, предназначенные для использования непосредственно в пользовательских программах. Полный список необходимых для воспроизведения результатов параметров различных систем сборки приведен в табл. 1, значением переменной окружения `PREFIX` является полное имя целевой платформы (например, `arm-linux-androideabi` для устройства с ARM процессором и ОС Android).

Таблица 1

Параметры, использованные для различных систем сборки

Table 1

Parameters Used for Different Build Systems

Система сборки	Имя переменной / параметра	Значение
GNU Make	CC	\$PREFIX-clang
	CXX	\$PREFIX-clang++
GNU Autotools	--host	\$PREFIX
	--target	\$PREFIX
CMake	CMAKE_SYSTEM_NAME	Android
	CMAKE_C_COMPILER	\$PREFIX-clang
	CMAKE_CXX_COMPILER	\$PREFIX-clang++

Слой совместимости с другими средами разработки

К сожалению, на практике стандарты языков программирования не всегда соответствуют тому, что ожидают увидеть пользователи. В разных средах программирования на различных операционных системах могут присутствовать некоторые уникальные функции, не описанные в стандартах. Хорошими примерами таких функций являются Windows API, доступный на операционной системе Windows, или `iostream.h` – заголовочный файл Borland C++, использовавшийся во времена, когда язык программирования C++ еще не был стандартизован.

Был проведен анализ отрицательных отзывов у приложений-конкурентов, доступных в Google Play, в результате чего был выделен следующий список возможностей, критичных для достаточно большого числа пользователей.

- Команды `system("cls")` и `system("pause")`. Данные команды операционной системы Windows используются для того, чтобы очистить экран или ожидать нажатия клавиши. Несмотря на то, что существуют более удачные способы осуществления данных операций, было решено написать свою реализацию этих команд, исполняемые файлы `cls` и `pause` были собраны при помощи кросс-компилятора, пути к ним добавлены в переменную окружения `PATH`.
- Заголовочный файл `conio.h`. Данный заголовочный файл использовался в операционной системе MS-DOS для создания текстового интерфейса пользователя. Было решено реализовать наиболее популярные функции этого заголовочного файла с использованием управляющих последовательностей ANSI – традиционного в Linux способа организации текстового интерфейса.

- Команда `fflush(stdin)`. Данная команда, по мнению пользователей, должна очищать буфер стандартного потока ввода, однако стандарт не гарантирует данное поведение: функция `fflush` определена только для потоков вывода. Изучение исходного кода библиотеки языка Си Android показало, что на этой операционной системе `fflush` потоков ввода ничего не делает. Более ожидаемое поведение было реализовано при помощи вызова `fpurge`, реализация была добавлена в статическую библиотеку `libandroid_support`, включаемую в каждую программу пользователя. Функция названа `fflush_compat`, а в заголовочный файл `stdio.h` добавлено переопределение всех вызовов `fflush` на `fflush_compat`. Это решение позволяет сохранить полную совместимость с библиотеками, которые ожидали вызов оригинальной функции `fflush`.

Разработка и оптимизация интерфейса для мобильных устройств

В ходе разработки было уделено отдельное внимание удобству интерфейса. Под удобством понималось соответствие ожидаемого поведения программы действительному, а также общая скорость выполнения различных операций пользователем.

Исследование проводилось в два этапа. На первом этапе была выбрана небольшая группа пользователей для качественного исследования наиболее часто используемых функций путем опроса, согласно [5].

Выделен следующий список наиболее часто используемых функций:

- ввод особых символов, специфичных для программирования, однако недоступных без дополнительных действий на большинстве программных клавиатур общего назначения;
- автоматическая компиляция и запуск программы по нажатию на одну кнопку;
- автодополнение кода;
- проверка кода в реальном времени;
- установка библиотек;
- форматирование кода.

Далее был спроектирован интерфейс программы: наиболее часто используемые функции были размещены в зоне быстрого доступа в соответствии с исследованием [6], а функции, к которым обращаются реже, перенесены в зоны с меньшей доступностью. Прототип был представлен крупной группе потенциальных пользователей программы для количественной оценки качества спроектированного прототипа. В ходе оценки качества выявлено, что часть функций не пользуется ожидаемой популярностью, поэтому эти функции были перенесены в зону меньшей доступности, а на их место перенесены функции, оказавшиеся более популярными. Кроме того, было подтверждено соответствие функций элементов интерфейса их иконкам: по результатам измерений, пользователи без труда находили запрашиваемое действие среди предложенных иконок.

Метод дальнейшей оптимизации интерфейса

После выпуска первой версии продукта для принятия решений стал использоваться способ А/Б тестирования. Части случайно выбранных пользователей (в количестве 30 %), установивших приложение из Google Play Store, предоставлялся доступ к А-версии продукта (версии с некоторыми изменениями, которые нужно было проверить), а другой части – доступ к Б-версии (первоначальной версии, выступающей в качестве эталона).

По прошествии двух недель измерялись ключевые показатели качества продукта:

- вовлеченность пользователей – частота совершения определенных действий в приложении;
- среднее время нахождения в приложении;
- количество пользователей, запустивших приложение хотя бы раз на второй и седьмой дни эксперимента.

При помощи данного тестирования удалось определить оптимальный порядок и состав пунктов меню приложения: в частности, расположение кнопки запуска в строке заголовка приводило к снижению вовлеченности пользователей, в то время как расположение кнопки

ближе к правому нижнему углу оказалось предпочтительным, хотя на первый взгляд приводит к уменьшению рабочего пространства редактора.

Снимки экрана двух вариантов, использованных в эксперименте с расположением кнопки запуска, представлены на рис. 1.

Рис. 1. А и Б варианты продукта с различным расположением кнопки запуска
Fig. 1. A and B Variants of Product with Different Locations of Launch Button

Автодополнение и проверка ошибок в реальном времени

Традиционный процесс автодополнения и проверки кода требует полной обработки исходного текста с самого начала каждый раз, когда вызывается автодополнение или проверка. Так работают некоторые настольные среды разработки, поскольку даже средний персональный компьютер имеет достаточную производительность, для того чтобы выполнять эти задачи в режиме реального времени.

Большинство программ на языке C++ используют различные заголовочные файлы, которые включаются в программу с использованием специальной директивы компилятора, `#include`. Эти заголовочные файлы часто достаточно велики, чтобы занимать большую часть времени компиляции, более того, существуют библиотеки, состоящие только из заголовочных файлов, которые полностью компилируются каждый раз, когда пользователь компилирует основную программу. Этот процесс может быть оптимизирован с использованием предкомпилированных заголовков, специальных файлов, которые содержат заэкшированное состояние компилятора после обработки некоторого списка заголовочных файлов [7]. Большинство сред разработки, однако, требует ручного создания данных файлов (значимым примером является файл `stdafx.h`, используемый Microsoft Visual Studio), что может быть затруднительно для пользователя мобильного устройства, поскольку ему придется редактировать несколько файлов и переключаться между ними каждый раз, когда он захочет добавить в программу очередной заголовочный файл. Поэтому возникла необходимость реализации алгоритма, который сочетал бы в себе удобство традиционного медленного подхода и скорость подхода, использующего предкомпилированные заголовки.

После автоматического анализа исходного кода многих популярных C++ проектов с открытым исходным кодом и ручной проверки выводов на простых учебных программах мы пришли к следующим выводам:

- программы могут иметь информацию о лицензии и авторе, расположенную в комментарии в самом начале исходного текста;
- директивы `#define` используются для констант и макросов (реже), они также часто появляются в начале программы;
- директивы `#include` обычно располагаются до объявленных пользователем функций, неподалеку от макросов и констант (но не всегда).

На основании анализа истории изменений проектов с открытым исходным кодом было установлено, что упомянутые части меняются достаточно редко в сравнении с остальным кодом. Из этого можно сделать вывод, что данные части программы являются хорошими кандидатами на включение в кэш компилятора.

Алгоритм ускорения автодополнения и проверки ошибок

Было решено встроить технологию оптимизации в процесс проверки исходного кода: мы можем безопасно генерировать предкомпилированные заголовки, только если текущая программа является корректным исходным кодом C++. Если проверка исходного кода не находит ошибок, мы создаем новый текст предкомпилированного заголовка с использованием регулярных выражений, и тогда, только если заэкшированного заголовка с таким текстом не существует, мы собираем его в фоновом потоке.

Если сгенерированный текст предкомпилированного заголовка совпадает с одним из существующих файлов кэша, автодополнение кода и проверка ошибок выполняются в ускоренном режиме с заменой текста на включение единственного заэкшированного заголовочного файла.

Схема алгоритма представлена на рис. 2.

Рис. 2. Схема алгоритма генерации предкомпилированных заголовков

Fig. 2. Diagram of Precompiled Header Generation Algorithm

Для генерации кэшируемого текста было решено использовать наиболее простое возможное регулярное выражение, поскольку это часто выполняемая операция [8]. В результате в кэшируемый текст были включены следующие части:

- однострочный и многострочный комментарий;
- любая директива компилятора, начинающаяся с символа `#`;
- пустые строки.

Существует, однако, важное замечание: алгоритм также проверяет, что количество включенных в заголовков директив `#if` и `#ifdef` равно количеству `#endif`. Хотя данные конструкции сравнительно нечасто используются в коде образовательных программ (особенно в опасной для алгоритма форме, по разные стороны от пользовательских объявлений переменных и функций), их игнорирование приводит к ошибке компиляции, что недопустимо для надежного алгоритма.

Оценка производительности полученного алгоритма

Чтобы оценить производительность алгоритма, было решено сравнить производительность автодополнения кода с наиболее популярными приложениями-конкурентами, доступными в магазине приложений Google Play: Cppdroid и C4droid. Производительность проверки кода не замерялась, поскольку реализована только в приложении Cppdroid, в котором, в свою очередь, был дополнительно доступен статический анализатор, что могло повлиять на достоверность результатов.

Для замеров использован набор исходных текстов образовательных программ, разбитый на несколько категорий:

- 1) примеры языка Си;
- 2) примеры языка C++, базовое использование стандартной библиотеки шаблонов (STL), разрешен только заголовок `iostream`;
- 3) примеры языка C++, обычное использование STL, разрешены заголовок `iostream` и контейнеры (например, `vector` и `map`);
- 4) примеры языка C++, расширенное использование STL, разрешено всё, включая последние стандарты;
- 5) примеры языка C++ с использованием библиотеки Boost.

Для замера времени выполнения операции автодополнения был выбран метод записи экрана и последующего перевода числа кадров во время, поскольку этот метод позволяет непосредственно оценить отзывчивость программы, а также избегает необходимости в обратной разработке программ, что может быть запрещено их лицензионным соглашением.

Устройством для произведения замеров был выбран планшетный компьютер Google Nexus 7 (2013), поскольку его невысокая производительность позволяет оценить снизу скорость работы программы на реальных устройствах, а также понижает относительную погрешность измерений за счет больших временных интервалов.

В результате проведения серий экспериментов, были получены результаты, представленные в табл. 2.

Таблица 2

Сравнительная таблица производительности автодополнения кода

Table 2

Comparison of Code Completion Performance

Программный продукт	Среднее время автодополнения кода в данной категории примеров, с				
	1	2	3	4	5
C4droid	0,35	2,62	3,05	3,86	27,29
Cppdroid	0,23	3,08	3,79	Н/Д ²	Н/Д ³
Cxxdroid (данная работа)	0,49	0,63	0,97	0,67	0,79
Отношение производительности работы к усредненному конкуренту	0,59	4,52	3,53	5,76	34,54

² Используемая в Cppdroid стандартная библиотека шаблонов устарела и не поддерживает стандарт C++11, требуемый примерами данной категории.

³ Cppdroid не поддерживает библиотеку Boost, и ее не удалось установить.

Заключение

В результате выполнения работы на платформу Android были успешно перенесены инструменты Clang, GNU Binutils, BusyBox, их зависимости, а также набор популярных библиотек, таких как Boost, GMP, libcurl. Был разработан оптимизированный для мобильных устройств интерфейс. Возможности по анализу и автодополнению кода были реализованы при помощи Clang. Создан алгоритм ускорения упомянутых возможностей, проверена его производительность и корректность. Все данные компоненты объединены в мобильную среду разработки языка C++.

Эксперименты показали, что разработанная среда программирования является единственной, обеспечивающей работу автодополнения и анализа кода в режиме реального времени в любых сценариях использования. Прирост производительности в среднем составил 300 %, когда используется стандартная библиотека шаблонов языка C++, достигая 3350 %, когда используется библиотека Boost. Присутствует небольшая потеря производительности при использовании языка Си, однако она незначительна в абсолютной величине и не влияет на возможность использования продукта в реальном времени.

Список литературы / References

1. **Prextořá T.** Mobile technology in education. In: International Conference of Computational Methods In Sciences and Engineering 2018 (ICCMSE 2018), 2018. DOI 10.1063/1.5079081.
2. **Brewster S.** Overcoming the Lack of Screen Space on Mobile Computers. *Personal and Ubiquitous Computing*, May 2002, vol. 6, no. 3, p. 188–205. DOI 10.1007/s007790200019.
3. **Son K.-C., Lee J.-Y.** The method of android application speed up by using NDK. In: 3rd International Conference on Awareness Science and Technology (iCAST), 2011. DOI 10.1109/icawst.2011.6163104
4. **Schiele R.** Building and Using a Cross Development Tool Chain. In: GCC Developers Summit, May 2003, p. 213.
5. **Vermeeren A. P. O. S., Law E. L.-C., Roto V., Obrist M., Hoonhout J., Väänänen-Vainio-Mattila K.** User experience evaluation methods. In: Proceedings of the 6th Nordic Conference on Human-Computer Interaction Extending Boundaries – NordiCHI'10, 2010. DOI 10.1145/1868914.1868973
6. **Yoo H., Yoon J., Ji H.** Index Finger Zone. In: Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct – MobileHCI'15, 2015. DOI 10.1145/2786567.2793704
7. **Krishnaswamy T.** Automatic precompiled headers: Speeding up c++ application build times. In: Proc. of the 1st Conf. on Industrial Experiences with Systems Softw. (WIESS), 2000. Berkeley, CA, USENIX Association, vol. 1, p. 57–66.
8. **Cody-Kenny B., Fenton M., Ronayne A., Considine E., McGuire T., O'Neill M.** A search for improved performance in regular expressions. In: Proceedings of the Genetic and Evolutionary Computation Conference on – GECCO'17, 2017. DOI 10.1145/3071178.3071196

Материал поступил в редколлегию

*Received
05.04.2019*

Сведения об авторах / Information about the Authors

Лебедев Роман Константинович, магистрант факультета информационных технологий, Новосибирский государственный университет (ул. Пирогова, 1, Новосибирск, 630090, Россия)

Roman K. Lebedev, Master's Student, Novosibirsk State University (1 Pirogov Str., Novosibirsk, 630090, Russian Federation)

n0n3m4@gmail.com

Корякин Илья Алексеевич, магистрант факультета информационных технологий, Новосибирский государственный университет (ул. Пирогова, 1, Новосибирск, 630090, Россия)

Ilya A. Koryakin, Master's Student, Novosibirsk State University (1 Pirogov Str., Novosibirsk, 630090, Russian Federation)

ed4140@gmail.com