УДК

КОМПИЛЯТОР РАЗДЕЛЕННОГО КОДА НА ОСНОВЕ JIT КОМПИЛЯЦИИ

SPLIT CODE COMPILER BASED ON JIT COMPLATION

Зимин М.А. – студент, Институт информационных технологий и радиоэлектроники, кафедра ИСПИ, группа ПРИ-116, E-mail: zim.ma@yandex.ru

Салех Х.М. – научный руководитель, к.т.н., Институт информационных технологий и радиоэлектроники, доц., каф ИСПИ, E-mail: saleh@vlsu.ru.

Zimin – student, Vladimir state university, E-mail: zim.ma2010@yandex.ru

Saleh H.M. - candidate of technical sciences, Vladimir state university,

E-mail: saleh@vlsu.ru

Аннотация: Дано описание состояния проблемы Легковесного и защищенного от нелицензированного доступа запуска приложений с возможностью использования на нескольких OS. Описаны основные компоненты системы. Приведены результаты проектирования будущей системы.

Annotation: This article describes the state of the problem of Lightweight and protected from unlicensed access to launch applications that can be used on multiple OSS. The main components of the system are described. The results of designing the future system are presented.

Ключевые слова: JIT, компилятор, исполнение, WebAssembly, безопасность.

Keywords: JIT, compiler, execution, WebAssembly, security.

Сегодня существует несколько основных способов исполнения и сборки п программ. Самый первый и классический – это перевод

исходного кода на языке высокого уровня или более низкого, как, например, assembler или C++ в машинных код при помощи компилятор или трансляторов. Другой подход это JIT компиляция или компиляция на ходу, то есть исходный код преобразуется в машинные инструкции налету. Подходов JIT компиляции можно выделить несколько:

1. Byte code. Пример, C# или Java. Этот подход заключается в том, что инструкции языка высокого уровня переводятся в некое промежуточное представление, которое называется byte code, и уже CLR/JVM – некая среда исполнения, переводит этот промежуточный код в машинный и исполняет. Последнее время такие системы используют возможности прекомпиляции части кода, например, стартовых инструкций, чтобы увеличить скорость запуска и работы.
2. Прямая инерпретация. Пример – JS код. Более медленный подход. Чем-то похож на предыдущий, но только тут нету byte code. Исполнение кода происходит построчно, но также используется некий движок для исполнения. Например, движок для исполнения JS сейчас включен в большинство браузеров и OS, что позволяет писать и desktop приложения.

Главное достоинство такого подхода – это то, что код становиться мультиплатформенным, то есть индивидуально придется собирать только среду исполнения под OS, сам же byte code не меняется.

Недостаток логически выходит из достоинства – уменьшение скорости исполнения (работы), потому что система исполнения тратит время на перевод инструкций byte code в машинный + теряется возможность прямого перевод типов данных в процессорные данные, такие как DWORD, QWORD, WORD.

В сравнение этих подходов можно выделить достоинства и недостатки.

Первый:

- Время исполнения
- Безопасность исполнения (усложнение реверс-инжиниринга)

Второй:

- Мультиплатформенный
- Выше скорость разработки
- Защита на уровне обфускации кода

Из всего этого вытекает проблема легковесного и защищенного от нелицензированного доступа запуска приложений с возможностью использования на нескольких OS.

Разработку решения можно разделить на 2 отдельных проекта. Первый – компилятор, второй – среда, в которой будет исполняться программа, полученная в первом пункте. Ниже представлена схема будущей среды исполнения. На ней видно несколько основных сегментов:

- ExecutorAPI – сервис, который будет поставлять пакеты данных пользователям программ, написанных при помощи данного компилятора
- CLR – система исполнения программ
- AuthAPI - сервис авторизации пользователя

- Kubernetes Cluster – система автоматического балансирования и распределения нагрузки ExecutorAPI

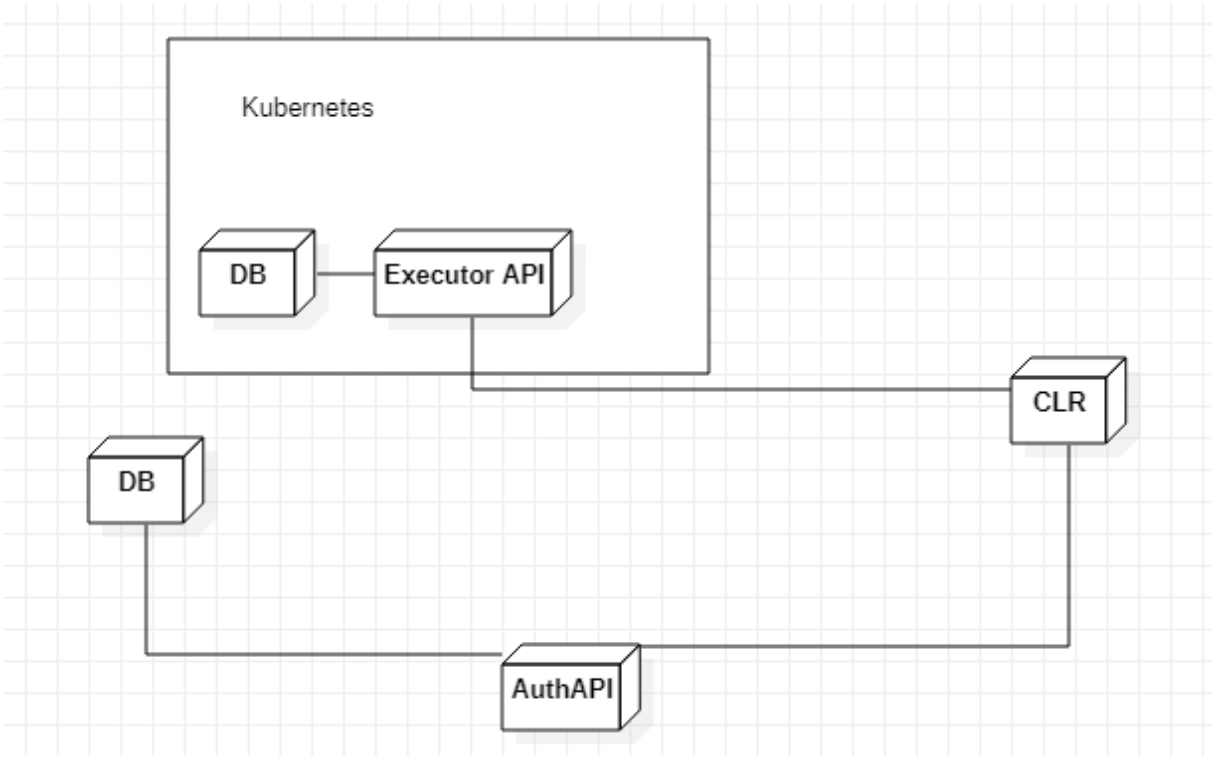

Рисунок 1. Схема среды исполнения

Сегодня существует несколько похожих технологий. Можно выделить несколько аналогов схожих по типу компиляции. Их можно подразделить на 2 типа:

- JIT исполнение на основе byte code. Например, Java, C# или WebAssembly
- Интерпретация. Например, JavaScript

Сравнение аналогов представлено в сводной таблице

Таблица 1. Сравнение аналогов

	C#	Java	WebAssembly	JavaScript	Seac
Мультиплатформенность	+	+	+	+	+

JIT компиляция	+	+	+	+-	+
Интерпретация	-	-	-	+	-
Защита Byte code	+-	+-	+-	-	+-
Распределенное исполнение	-	-	+	-	+
Полная поставка кода при распределенном исполнении	-	-	+	-	-
Зависимость от платформы исполнения	Только CLR	Только JVM	Браузер	Браузер	Только CLR
Год выпуска	2001	1995	2019	1995	-

Из всех представленных в списке аналогов только WebAssembly может составлять конкуренцию, но пока он функционирует только на базе браузеров и не может исполняться вне их среды. Возможно в дальнейшем данная технология будет поддерживаться Electron, Blazor или подобными решениями.

Это приложение создается с целью перевода языка высокого уровня в byte code, который будет представлять ЯВУ в более простом и удобном для CLR виде, что скажется на повышении производительности и позволит писать код, не опираясь на CLR зависимости.

Алгоритм преобразования ЯВУ в byte code:

1. Конкатенация включенных файлов (#include)
2. Создание первичного дерева и деревьев математики

3. Повторный обход деревьев для разрешения проблем линковки
4. Конвертация кода в byte code
5. В случае remote compile – дополнительное разбиение кода на пакеты

Написание на ЯВУ компилятора представляет из себя написание классического кода, как на C++, используя только функциональный подход.

Отдельно хотелось бы выделить возможность использовать директивы #start_package и #end_package, которые будут обозначать компилятору о разделении кода на определенный пакеты. На данном этапе развития компилятора данные директивы следует прописывать руками, в последствие можно дополнить процесс конвертации в byte code системой автоматического разделения на пакеты на основе аналитики кода.

CLR (среда исполнения) отвечает за исполнение byte code, созданного компилятором. В его основные задачи входит хранение виртуальной таблицы памяти для хранения состояний (переменных), исполнение команд byte code, менеджер памяти, который отвечает за выделение новых участков и их очистку.

Также дополнительно CLR отвечает за полный цикл исполнения программы на пользовательской машине, в который входит:

1. Создание подключения к серверу источнику приложения
2. Получение информации о исполняемом приложении
3. Создание и обеспечение работоспособности цикла получения, расшифровки и обработки пакетов кода
4. Очистка памяти от устаревших данных

Если говорить точнее, то второе решение будет включать в себя также и систему, которая похожа на магазины электронной коммерции, например, Steam. Она будет состоять из:

1. Сервер. Приложение – источник данных для запуска программ в режиме Remote, либо поставщик ключей для дешифровки, например, standalone application
2. WPF Application. Пользовательское приложение, через которое пользователь может авторизоваться и получить доступ к установленным приложениям, либо запустить remote application
3. CLR (система исполнения)

Из выше написанного можно сказать, что данное решение будет интересно, как рынку, так и потребителям в лице глав крупных компаний для создания корпоративных защищенных приложений или для разработки приложений с повышенной безопасностью.

Список используемой литературы:

1. Compilers: Principles, Techniques, and Tools, Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman, ISBN 0-201-10088-6
2. Стивен Прата "Язык программирования C++. Лекции и упражнения" (6-е изд.), ISBN 978-5-8459-1778-2
3. The C++ Programming Language: Special Edition, Bjarne Stroustrup, Addison-Wesley ISBN 978-0321563842. May 2013.