Это позволяет перед передачей содержимого принять решение о том, разрешен ли данному пользователю просмотр содержимого запрашиваемой страницы. Такой способ позволяет избежать проблемы неограниченного доступа к содержимому, однако не позволит нам возвращать разные HTML-страницы для разных ролей пользователя при одном и том же URL запросе.

4. *Кэширование фрагментов*. Как было сказано нами ранее, при создании динамических веб-приложений кэширование страницы целиком оказывается возможным только в очень редких случаях, поскольку в современных приложениях каждая HTML-страница представляет собой не единый блок данных, а набор различных компонент, выполняющих свои роли. Кэширование отдельных фрагментов страницы способно разрешить эту проблему. Очевидно, что если наша страница представляет из себя набор компонент, мы можем кэшировать не саму страницу, а её составные части. При этом отсутствует ограничение на количество кэшируемых фрагментов на странице. Поскольку каждая страница может состоять из почти неограниченного числа компонент (как в случае бесконечно прокручиваемых страниц) при использовании этого метода резко увеличивается объем хранимой информации для каждой страницы, перед нами встает вопрос об эффективном средстве для хранения кэша. Ruby on Rails предоставляет несколько стратегий хранения: в памяти, в отдельном каталоге, на отдельном сервере и др. Наиболее рациональным средством для кэширование фрагментов на сегодняшний день является memcached. Memcached – высокопроизводительная распределенная система кэширования. По сути, данная технология представляет из себя структуру данных на подобии ассоциативного массива, где каждом ключу (идентификатору фрагмента) ставится в соответствие готовый код компоненты HTML-страницы.

Инструмент кэширования фрагментов достаточно гибок и лёгок в использовании, что позволяет значительно ускорять работу приложения, сохраняя часто отображаемые фрагменты веб-страниц. При этом область применения кэширования фрагментов практически не ограничена.

Список использованных источников:

1. Edd Dumbill Learning Rails Rails from the Outside In // ch.18 November 2008 O'Reilly Media
2. Jose Valim Crafting Rails 4 Applications // p.137-161 November 24, 2013 Pragmatic Bookshelf
3. Brad Ediger Advanced Rails // ch.6 December 2007 O'Reilly Media

РЕАЛИЗАЦИЯ ВИРТУАЛЬНОЙ МАШИНЫ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Сернацкий В. И.

Жвакина А. В. – канд. техн. наук, доцент

Со стремительным развитием информационных технологий мощности компьютеров достигли такого уровня, когда одна физическая машина может поддерживать несколько изолированных сред в виртуальных машинах. На данный момент разрабатывается и существует множество вариантов их применения. Виртуальная машина по сравнению с физической обладает большей гибкостью в контексте переносимости на другие платформы. В работе рассматривается спецификация мнимой архитектуры набора команд и реализация виртуальной машины, исполняющей код этой архитектуры.

Виртуальная машина – программная или аппаратная система, эмулирующая некоторую платформу и исполняющая код для этой платформы в изолированной среде на другой или такой же платформе.

Виртуальные машины применяются для решения различных задач: эмуляция различных архитектур, моделирование информационных компьютерных систем на одной машине, исследование поведения и производительности ПО, оптимизация использования виртуальными средами физических ресурсов, защита от вредоносного кода, требующего доступа к физическим ресурсам, спецификация некоторой вычислительной среды.

Архитектура набора команд – часть архитектуры компьютера, определяющая программируемую часть ядра микропроцессора: модели памяти, взаимодействие с внешними устройствами ввода-вывода, режимы адресации, машинные инструкции, типы внутренних данных, обработчики прерываний и исключений.

Разработка велась на языке программирования C++ с соблюдением современных общепринятых стандартов, таких как ISO/IEC 14882:2014, и без внедрения платформозависимого кода. Это гарантирует способность разработанной виртуальной машины работать на более чем одной аппаратной платформе – кроссплатформенность. Получившаяся архитектура достаточно компактная, чтобы уместиться и работать на микроконтроллерах и однокристальных системах, таких, как Arduino и RaspberryPi. Это распространяет спектр применения на сферу интернета вещей.

От таких известных аналогов, как OracleVMVirtualBox, MicrosoftHyper-V, VMwareWorkstation, созданная виртуальная машина отличается тем, что её архитектура набора команд разработана специально для данной виртуальной машины и не имеет реализации в существующих аппаратных платформах. В этом данный программный продукт схож с виртуальными машинами языков программирования: Java Virtual Machine, Common Language Runtime, Forth.

Для упрощения рассматриваемая архитектура оперирует единственной моделью памяти –

последовательностью адресованных ячеек фиксированной длины, составляющей 4 байта. В данной памяти хранятся инструкции, данные и системные структуры, определяющие состояние машины. Также данное устройство памяти поддерживает виртуальную адресацию: для кода, выполняющегося в сегменте с началом по некоторому адресу и некоторой длины, адреса отличаются от адресов вне сегмента. Это позволяет размещать программу в любом месте в физической памяти. При попытке кода в сегменте обратиться к ячейкам вне сегмента, возникает исключительная ситуация, которая обрабатывается на уровне, где определен сегмент. Это позволяет обеспечить защищенность и изоляцию данных каждой программы в своем сегменте.

Инструкции, принимающие ненулевое число операндов, содержат в старших битах информацию о типах операндов. Инструкции могут работать с тремя типами операндов (операнды, инструкции и адреса занимают одну ячейку памяти, то есть 4 байта): абсолютное значение, значение по адресу, значение по адресу, находящемуся по адресу.

Рассматриваемая архитектура имеет следующий набор команд:

Тип	Код	Мнемоника	Кол-во операндов	Пояснение
Управляющие	0x0	nop	0	Ничего не делать
	0x1	hlt	0	Завершить выполнение
	0x2	itr	1	Вызвать прерывание или сервис
Переход	0x3	jz	2	Перейти, если указанная ячейка равна нулю
Работа с данными	0x4	mov	2	Переместить данные из первого операнда во второй
Арифметические	0x5	add	2	Данные инструкции помещают результат во второй операнд
	0x6	div	2	
Логические	0x7	or	2	
	0x8	and	2	

Несмотря на малое количество инструкций, разработанная архитектура является тьюринг-полной, что позволяет решать задачи самой разной сложности.

Обработка прерываний, в том числе взаимодействие с внешними устройствами ввода-вывода реализуется выполнимыми на стороне хоста Сервисами, которые выполняются при возникновении исключительной ситуации или при выполнении инструкции itr. Количество и характер сервисов могут меняться в зависимости от задачи или во время выполнения виртуальной машины. Таким образом обеспечивается масштабируемость платформы.

В начале памяти хранятся системные данные: указатель на выполняемую инструкцию, указатель на список определенных сегментов и зарезервированные для расширения спецификации ячейки. Для упрощения понятие стека вызовов, соглашение о вызове подпрограмм и др. не входят в спецификацию, тем не менее реализуемы в виде соглашений.

Множество упрощений в процессе разработке привели к так называемой «трясине Тьюринга»: созданная архитектура тьюринг-полна, но обладает крайне примитивным синтаксисом и семантикой. Простейшие программы выглядят невероятно большими, что может помешать работе на платформах с малым количеством памяти. Возможными решениями этой проблемы является расширение набора команд или сокращение кода с помощью применения технологии, подобной Thumb в архитектурах ARM.

Исследованиеподдержанопроектом CERES. Centers of Excellence for young RESearchers (Reg.no. 544137-TEMPUS-1-2013-SK-JPHES),

Список использованных источников:

1. Гультяев, А. К. Виртуальные машины – несколько компьютеров в одном / А. К. Гультяев. – Санкт-Петербург: Питер, 2006. – 224 с.
2. Intel 64 and IA-32 Architectures Software Developer's Manual – Intel Corp., 2014.
3. Википедия: Виртуальная машина. [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Виртуальная_машина. – Дата доступа: 25.03.2017.
4. Википедия: Архитектура набора команд. [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Архитектура_набора_команд. – Дата доступа: 25.03.2017.
5. International Organization for Standardization: ISO/IEC 14882:2014. [Электронный ресурс]. – Режим доступа: https://www.iso.org/standard/64029.html. – Дата доступа: 25.03.2017