УДК 004.946

К. В. Кринкин
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

М. Ю. Кринкин
Санкт-Петербургский академический университет – научно-образовательный центр нанотехнологий РАН

Методы динамической трансляции для кроссархитектурной миграции процессов Linux

Виртуализация позволяет достичь высокой утилизации аппаратных ресурсов, снижения затрат энергии, обеспечивает изоляцию рабочих окружений. Статья содержит анализ существующих методов миграции процессов Linux. Предлагается оригинальный подход кроссархитектурной миграции на основе динамического транслятора QEMU.

Виртуализация, динамическая трансляция, миграция процессов

Миграция процессов – это перенос процессов между вычислительными узлами во время исполнения. Переносимое состояние включает в себя адресное пространство процесса, состояние процессора, внешние связи процесса и другие ресурсы. Эта технология позволяет использовать более гибкие механизмы распределения нагрузки, повысить устойчивость системы к ошибкам, упростить администрирование.

Несмотря на эти преимущества, до недавнего времени технологии миграции процессов не были широко распространены. Одна из причин этого – сложность реализации миграции процессов в современных операционных системах, изначально разрабатываемых без учета миграции процессов. В качестве примера приведем проект linux-cr по реализации механизма checkpoint/restart в ядре Linux. Исходные коды так и не были приняты в ядро Linux из-за слишком большого объема и сложности*.

Однако с современным уровнем развития технологий виртуализации, а также с учетом предыдущего опыта реализация миграции процессов становится более чем реальной**. Например, в проекте CRIU учли ошибки предшественника (linux-cr) и реализовали механизм checkpoint/restart для Linux преимущественно в пространстве пользователя с минимальной поддержкой со стороны ядра ОС.

Кроссархитектурная миграция процессов. Процесс является контейнером ресурсов ОС, причем состояние многих ресурсов не зависит от конкретной аппаратной архитектуры, поэтому в рамках исследования рассмотрим возможность кроссархитектурной миграции процессов, т. е. миграции процессов в условиях различных аппаратных архитектур исходного и целевого узлов миграции.

К ресурсам, не зависящим от аппаратной архитектуры, можно отнести сетевые соединения, так как современные распространенные сетевые протоколы спроектированы, чтобы быть независимыми от аппаратной платформы, и ресурсы файловой системы. К ресурсам, зависимым от аппаратной платформы, можно отнести память, так как на различных системах могут существо-

* Corbet J. Checkpoint/restart: it's complicated. http://lwn.net/Articles/414264/, November 2010.

** Emelyanov P. Checkpoint/restore mostly in the userspace. http://lwn.net/Articles/451916/, July 2011.

вать различные ограничения на выравнивание данных в памяти и доступ к невыровненным данным, а также различный порядок байт, и состояние процессора, так как разные процессоры обладают разным набором команд.

Для преодоления различий аппаратных платформ разработан большой набор средств – от аппаратных [1], [2] до программных. К программным средствам относятся различные виртуальные машины. Например, распространенные Java- и .NET-виртуальные машины позволяют создавать кроссплатформенные программы, не зависимые от конкретной аппаратной платформы и даже ОС. Другой пример – эмулятор QEMU, который позволяет исполнять нативные приложения, разработанные для одной архитектуры, на другой.

Динамический транслятор QEMU. QEMU – быстрый и переносимый динамический транслятор. В QEMU поддерживается 2 режима функционирования:

1. Полная системная эмуляция. В этом режиме QEMU эмулирует и CPU, и периферийные устройства. Этот режим используется, чтобы запускать несколько, возможно, различных ОС внутри одной системы, а также для отладки системного кода.

2. Эмуляция пользовательского режима. В этом режиме QEMU позволяет запускать программы, скомпилированные для одной архитектуры, на другой, но ОС должны совпадать. Этот режим будет использован в реализации миграции процессов.

Для повышения производительности QEMU использует динамическую трансляцию в нативный код – во время рассмотрения очередного блока кода (блока трансляции) QEMU преобразует его в эквивалентный код хост-платформы.

Начиная с версии 0.9.1 QEMU используется трансляция исходного кода в RISC-подобный набор команд внутреннего представления TCG. На уровне внутреннего представления можно проводить некоторые оптимизации, например произвести анализ живости и свертку констант, однако, так как QEMU транслирует программу небольшими блоками кода, все оптимизации ограничены этим блоком. Затем по внутреннему представлению генерируется блок кода хост-платформы, который помещается в кеш для дальнейшего использования.

Блок трансляции ограничивается, как правило, инструкцией перехода, т. е. содержит один базовый блок, поэтому блоки трансляции могут быть достаточно маленькими, что, вместе со статическим выделением регистров, ограничивает возможности для оптимизации генерируемого кода, однако сама трансляция получается довольно простой и быстрой.

Реализация кроссархитектурной миграции. В рамках исследований был реализован механизм кроссархитектурной миграции процесса Linux с архитектуры x86 на архитектуру ARM. Совместили подход проекта CRIU и получили состояние процесса исключительно средствами пространства пользователя, с эмулятором QEMU для получения кроссархитектурной трансляции. При реализации миграции процесса ограничились только двумя ресурсами: памятью процесса и состоянием процессора.

Причина такого ограничения заключается в том, что перенос даже не зависимых от аппаратной платформы ресурсов является технически сложной задачей. Дело в том, что ресурсы процесса глубоко интегрированы в ОС Linux, и сделать снимок состояния ресурсов исключительно в пространстве пользователя не представляется возможным, как и восстановить состояние ресурсов из сделанного снимка. Например, сокеты имеют сложное внутреннее состояние, зависящее от используемого сетевого протокола. Кроме того сокеты как интерфейс сетевой подсистемы ОС связывают между собой 2 процесса. Соответственно перенос состояния сетевого соединения затрагивает не только переносимый процесс. Аналогичная ситуация возникает, например, с каналами, которые связывают 2 процесса.

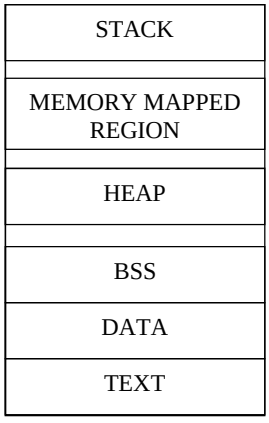
Другой важный ресурс, который в статье не рассматривается, – файловые дескрипторы. В UNIX все есть файл, поэтому файловые дескрипторы являются одними из самых важных ресурсов, даже сокеты имеют подобный файловому интерфейс. Файлы могут представлять регулярные файлы, именованные каналы или устройства, каждый тип файла требует особой обработки, поэтому ограничимся в данной статье только базовыми ресурсами.

Процедура миграции процесса состоит из двух частей – создания образа и восстановления процесса из образа (checkpoint/restart). Для создания образа процесса сначала необходимо остановить процесс. Для этого в ОС Linux используется системный вызов ptrace. Данный вызов служит для отладки, он позволяет останавливать и возобновлять процесс, читать и изменять память процесса, содержимое регистров, перехватывать раз-

личные события, например системные вызовы или сигналы*. Далее необходимо сохранить адресное пространство процесса и состояние регистров процессора.

Восстановление процесса осуществляется в эмуляторе QEMU. Для восстановления процесса необходимо восстановить копию адресного пространства процесса в памяти QEMU, затем восстановить состояние процессора.

Типичное адресное пространство процесса Linux изображено на рисунке. В самых верхних адресах адресного пространства располагается стек процесса, стек растет вниз. Ниже стека располагается memory mapping region. В этот регион памяти обычно загружаются динамические библиотеки, также в нем выделяется память с помощью системного вызова mmap, этот регион растет вниз. Далее идет куча процесса, эта память используется для динамического выделения памяти, например с помощью malloc или оператора new языка C++. Куча процесса растет вверх, верхняя граница определяется с помощью системных вызовов brk и sbrk. Нижние адреса занимает код и данные, память для которых была выделена на этапе компиляции. Таким образом, адресное пространство процесса используется неравномерно, т. е. незанятые регионы памяти располагаются в середине адресного пространства процесса.



Чтобы получить содержимое памяти процесса, в реализации используется файловая система proc. Список занятых регионов адресного пространства процесса доступен в файле /proc/<pid>/maps. Каждый регион памяти описывается диапазоном адресов, правами доступа (на чтение, запись и исполнение), а также описанием (например, [HEAP], [STACK], /usr/lib/libc.so), т. е. чем занят данный регион памяти. Формат описания регионов памяти поддерживается постоянным для обеспечения обратной совместимости.

Для считывания непосредственно памяти процесса используется файл /proc/<pid>/mem. Смещения в этом файле соответствуют адресам в адресном пространстве процесса, т. е. чтобы получить всю память процесса, необходимо прочитать данные из этого файла согласно описаниям регионов памяти из файла /proc/<pid>/maps.

Восстановление адресного пространства процесса в эмуляторе QEMU осуществляется с помощью системного вызова mmap. Память восстанавливается с сохранением относительных смещений регионов памяти, так как эмулятор не поддерживает сложные механизмы преобразования адресов в режиме эмуляции одного процесса.

Получить состояние регистров процессора можно посредством команды PTRACE_GETREGSET, системного вызова ptrace. В реализации ограничились только базовым набором регистров, т. е. регистры общего назначения, сегментные регистры, указатели стека и команд.

Восстановление состояния процессора в эмуляторе – тривиальная задача, каждый поддерживаемый процессор в эмуляторе QEMU описывается структурой языка C, а восстановление сводится к заполнению соответствующих полей структуры.

Ограничения разработанного подхода. Рассмотренный подход к реализации кроссархитектурной миграции обладает рядом ограничений. В данной статье авторы намеренно ограничились самым базовым набором ресурсов: памятью и состоянием процессора. Но для полноценной миграции процессов необходимо обеспечить перенос многих других ресурсов, например файловых дескрипторов или сокетов, без которых миграция процессов не имеет прикладной ценности.

Как было показано ранее, память процесса используется неравномерно, что приводит к ограничению адресного пространства. Так как QEMU требует восстановления памяти процесса без относительных смещений, в адресном пространстве QEMU просто не хватает места для восстановления типичного процесса Linux.

Кроме этого использование динамического транслятора QEMU приводит к серьезной потере производительности. В некоторых тестах потеря производительности достигала двух порядков (таблица), что неприемлемо для промышленного использования кроссархитектурной миграции процессов.

* Padala P. Playing with ptrace. Part I. http://www.linuxjournal.com/article/6100, October 2002. Padala P. Playing with ptrace. Part II. http://www.linuxjournal.com/article/6210, November 2002.

Тест	Потери трансляции	Нативное исполнение
Доступ к памяти	0.009	0.007
Целочисленное деление	0.129	0.008
Файловый вывод	1.330	0.424

Направления дальнейших исследований. Рассмотренный подход показывает, что реализация кроссархитектурной миграции процессов возможна, однако разработка промышленной реализации требует решения ряда задач.

QEMU позиционируется как быстрый и переносимый транслятор, однако он почти не поддерживает оптимизаций над внутренним представлением*. Поэтому интересным направлением является использование более продвинутого бинарного транслятора, поддерживающего большее количество оптимизаций над промежуточным представлением. В качестве основы для такого транслятора можно использовать LLVM [3], который поддерживает большое число аппаратных платформ и различных оптимизаций.

Процессы в современных ОС работают в сложном окружении, важной частью которого являются разделяемые библиотеки. У популярных библиотек существуют версии под различные архитектуры (например, стандартная библиотека языка С), что открывает возможность для оптимизаций. Вместо трансляции кода динамической библиотеки, как это сейчас делает QEMU, можно использовать нативную для целевой платформы версию библиотеки, собранную с помощью оптимизирующего компилятора.

Ограничения, накладываемые неравномерным использованием адресного пространства процесса, серьезно снижают применимость описанного подхода. Чтобы преодолеть эти ограничения, можно использовать поддержку со стороны ядра операционной системы для управления адресным пространством процесса.

СПИСОК ЛИТЕРАТУРЫ

1. Karaki H., Akkary H., Shahidzadeh S. X86-ARM Binary Hardware Interpreter // IEEE Intern. Conf. on Electronics, Circuits and Systems, Beirut, Lebanon, December 11–14, 2011.

2. Ezzedine M., Karaki H., Akkary H. Ubiquitous Computing Platform via Hardware Assited ISA Virtualization” // IEEE Intern. Conf. on Innovations in Information Technology, Al Ain, UAE, March 17–19, 2013.

3. Lattner C., Adve V. LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation // Proc. of the 2nd IEEE/ACM Intern. Symp. on Code Generation and Optimization, San Jose, CA, USA, March 20-24, 2004.

K. V. Krinkin
Saint-Petersburg state electrotechnical university «LETI»

M. Yu. Krinkin
St. Petersburg Academic University – Nanotechnology Research and Education Centre of the Russian Academy of Sciences (Russia)

DYNAMIC TRANSLATION FOR CROSS-ARCHITECTURAL PROCESS MIGRATION

Virtualization enables high hardware resource utilization, reduce energy consumption and ensures work environment isolation. This paper includes overview of process migration techniques for OS Linux. We propose approach to cross-architectural process migration based on dynamic translation.

Virtualization, dynamic translation, process migration

* Fabrice Bellard, “Tiny Code Generator”, http://git.qemu.org/?p=qemu.git;a=blob_plain;f=tcg/README;hb=HEAD.