Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего

профессионального образования

«МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ (ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)»

ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ

КАФЕДРА ИНФОРМАТИКИ

(Специализация 010900 "Прикладные математика и физика")

Разработка алгоритмов оптимизации затрачиваемых вычислительных ресурсов виртуальным окружением и программная реализация с целью их обоснования

Выпускная квалификационная работа

студента 4 курса 073а группы

Капаева Евгения Олеговича

Научный руководитель:

д. ф.-м. н. Тормасов Александр Геннадьевич

г. Москва, 2014

Оглавление

1 Введение

Одной из наиболее значимых проблем, стоящих в настоящее время перед производителями и пользователями вычислительных средств, является управление энергопотреблением. Так, плата за энергопотребление мощных серверов, входящих в состав высокопроизводительных вычислительных комплексов (ВК), стала основной статьей затрат на их обслуживание, а в категории мобильных и встраиваемых систем эффективное управление энергопотреблением позволяет улучшить одну из главных характеристик – максимальное время функционирования в автономном режиме (от батареи).

Общепринятый подход к снижению уровня энергопотребления сформулирован в стандарте ACPI , который в ряде распространенных архитектур поддерживается на аппаратном уровне. Согласно этому стандарту для ВК определяется несколько наборов состояния энергосбережения. Наиболее общим является набор {S0;Sn}, характеризующий энергосберегающие состояния ВК в целом с учетом процессоров, памяти, шин, периферийных устройств. Любое состояние Si из этого набора определяется тремя значениями (Ci, Pi, Di), каждое из которых принадлежит одному из трех наборов состояний, соответственно {C0;Cn}, {P0;Pn} и {D0;Dn}. Здесь {C0;Cn} – набор состояний сна для процессорного ядра, {P0;Pn} – набор активных состояний с различной частотой для процессорного ядра, {D0;Dn} – набор энергосберегающих состояний шин, памяти, контроллеров ввода вывода и периферийных устройств, расположенных на материнской плате. В соответствии со стандартом переходы из одного состояния в другое инициируются операционной системой.
В связи с развитием облачной инфраструктуры и виртуализации, системы, работающие в виртуальном окружении, нуждаются в оптимизации расходуемых вычислительных ресурсов. Современные решения виртуализации направлены в первую очередь на поддержку требуемого уровня производительности виртуализируемых платформ. Из-за чего вопросы энергоэффективности обычно опускаются.

В данной работе разработан алгоритм, который позволяет повысить энергоэффективность виртуализованных систем семейства Linux при помощи гипервизора KVM(Kernel-Virtual-Machine). Реализованное программное решение может быть перенесено на другие платформы виртуализации, работающие по схожему с KVM принципу. Также рассматриваются общие вопросы энергосбережения в ядре Linux, алгоритмы и существующие эвристические модели улучшающие энергоэффективность

2 Анализ существующих решений

2.1 Стандарт ACPI

ACPI (англ. *Advanced Configuration and Power Interface*) — открытый промышленный стандарт управления питанием. Концепция ACPI предполагает, что управление питанием полностью передается операционной системе, предоставляя методы для прямого детализированного управления аппаратным обеспечением. Такая модель выгодно отличается от существовавшей до этого модели APM (Advanced Power Manager), в которой за управление питанием ответственен BIOS материнской платы, а возможности ОС в этом отношении сильно ограничены. В модели ACPI, BIOS предоставляет операционной системе методы для прямого детализированного управления аппаратным обеспечением. Таким образом, ОС получает практически полный контроль над энергопотреблением.

Другая важная часть спецификации ACPI - это предоставление на серверах и настольных компьютерах таких возможностей по управлению питанием, которые до того были доступны только на портативных компьютерах. Например, система может быть переведена в состояние чрезвычайно низкого энергопотребления, в котором питание подается лишь на оперативную память (а возможно, и она находится без питания), но при этом прерывания некоторых устройств (часы реального времени, клавиатура, модем и т. д.) могут достаточно быстро перевести систему из такого состояния в нормальный рабочий режим (то есть «пробудить» систему).

Помимо требований к программному интерфейсу ACPI также требует специальной поддержки от аппаратного обеспечения. Таким образом, поддержку ACPI должны иметь ОС, чипсет материнской платы и центральный процессор.

2.2 Технические особенности

Интерфейс ACPI организуется путём размещения в определенной области оперативной памяти нескольких таблиц, содержащих описание аппаратных ресурсов и программных методов управления ими. Каждый тип таблицы имеет определенный формат, описанный в спецификации. Кроме того, таблицы, содержащие методы

управления устройствами и обработчики событий ACPI, содержат код на языке AML (ACPI Machine Language) — машинно независимый набор инструкций, представленный в компактной форме. Операционная система, поддерживающая ACPI, содержит интерпретатор AML, который транслирует инструкции AML в инструкции центрального процессора, выполняя таким образом методы или обработчики событий.

Некоторые из этих таблиц полностью или частично хранят статические данные в том смысле, что от запуска к запуску системы, они не изменяются. Статические данные, как правило, создаются производителем материнской платы или BIOS и описываются на специальном языке ASL (ACPI Source Language), а затем компилируются в представление на AML.

Другие таблицы хранят динамические данные, которые зависят, например, от установок BIOS и комплектации материнской платы. Такие таблицы формируются BIOS на этапе загрузки системы до передачи управления ОС.

Роль ОС в этой модели заключается в том, что она переводит различные компоненты аппаратного обеспечения из одного состояния (например, нормальный режим работы) в другое (например, режим пониженного энергопотребления). Переход из одного состояния в другое происходит, как правило, по событию. Например, падение температуры на ядре процессора является событием, по которому ОСможет вызвать метод уменьшения скорости вращения вентилятора. Другой пример: пользователь дал явное указание перехода системы в спящее состояние с сохранением оперативной памяти на диск, а через некоторое время администратор сети произвёл включение системы с помощью функции Wake-on-LAN.

2.3 Глобальные состояния системы

Выделяют следующие основные состояния «системы в целом».

G0 (**S0**) (Working) — нормальная работа.

G1 (Suspend, Sleeping, Sleeping Legacy) — машина выключена, однако текущий системный контекст (system context) сохранён, работа может быть продолжена без перезагрузки. Для каждого устройства определяется «степень потери информации» в процессе засыпания, а также где информация должна быть сохранена и откуда будет прочитана при пробуждении и время на пробуждение из одного состояния до другого (например, от сна до рабочего состояния). Выделяют 4 состояний сна:

S1 — состояние при котором все процессорные кэши сброшены и процессоры прекратили выполнение инструкций. Однако, питание процессоров и оперативной памяти поддерживается; устройства, которые не обозначили, что они должны оставаться включенными, могут быть отключены;

S2 — более глубокое состояние сна, чем S1, когда центральный процессор отключен, обычно, однако, не используемое;

S3 («Suspend to RAM» (STR) в BIOS, «Ждущий режим» («Standby») в версиях Windows вплоть до Windows XP и в некоторых вариациях Linux, «Sleep» в Windows Vista и Mac OS X, хотя в спецификациях ACPI упоминается только как S3 и Sleep) — в этом состоянии на оперативную память (ОЗУ) продолжает подаваться питание и она остаётся практически единственным компонентом, потребляющим энергию. Так как состояние операционной системы и всех приложений, открытых документов и т. д. хранится в оперативной памяти, пользователь может возобновить работу точно на том месте, где он её оставил — состояние оперативной памяти при возвращении из S3 то же, что и до входа в этот режим. (В спецификации указано, что S3 довольно похож на S2, только чуть больше компонентов отключаются в S3.) S3 имеет два преимущества над S4: компьютер быстрее возвращается в рабочее состояние, и, второе, если запущенная программа (открытые документы и т. д.) содержит частную информацию, то эта информация не будет принудительно записана на диск. Однако, дисковые кэши могут быть сброшены на диск для предотвращения нарушения целостности данных в случае, если система не просыпается, например, из-за сбоя питания;

S4 («Спящий режим» (Hibernation) в Windows, «Safe Sleep» в Mac OS X, также известен как «Suspend to disk», хотя спецификация ACPI упоминает только термин S4) — в этом состоянии всё содержимое оперативной памяти сохраняется в энергонезависимой памяти, такой как жёсткий диск: состояние операционной системы, всех приложений, открытых документов и т. д. Это означает, что после возвращения из S4, пользователь может возобновить работу с места, где она была прекращена, аналогично режиму S3. Различие между S4 и S3, кроме дополнительного времени на перемещение содержимого оперативной памяти на диск и назад, - в том, что перебои с питанием компьютера в S3 приведут к потере всех данных в оперативной памяти, включая все несохранённые документы, в то время как компьютер в S4 этому не подвержен. S4 весьма отличается от других состояний S и сильнее S1-S3 напоминает G2 *Soft Off* и G3 *Mechanical Off*. Система, находящаяся в S4, может быть также переведена в G3 *Mechanical Off* (Механическое выключение) и все ещё оставаться S4, сохраняя

информацию о состоянии так, что можно восстановить операционное состояние после подачи питания.

G2 (**S5**) (soft-off) — *мягкое (программное) выключение*; система полностью остановлена, но под напряжением, готова включиться в любой момент. Системный контекст утерян.

G3 (mechanical off) — *механическое выключение* системы; блок питания ATX отключен.

Дополнительно — технология OnNow от Microsoft (Расширения S1-S4 состояния G1). Также Windows 7 поддерживает "Гибридный спящий режим", сочетающий в себе преимущества S1/S3 (быстрота пробуждения) и S4 (защищённость от сбоев электропитания).

2.4 Состояния центрального процессора

Выделяют четыре состояния функционирования процессора (от C0 до C3).

C0 — оперативный (рабочий) режим.

C1 (известно как *Halt*) — состояние, в котором процессор не исполняет инструкции, но может незамедлительно вернуться в рабочее состояние. Некоторые процессоры, например Pentium 4, также поддерживают состояние Enhanced C1 (C1E), для более низкого энергопотребления.

C2 (известно как *Stop-Clock*) — состояние, в котором процессор обнаруживается приложениями, но для перехода в рабочий режим требуется время.

C3 (известно как *Sleep*) — состояние, в котором процессор отключает собственный кэш, но готов к переходу в другие состояния.

2.5 Состояния устройств

Выделяют четыре состояния функционирования других устройств (монитор, модем, шины, сетевые карты, видеокарта, диски, флоппи и т. д.) — от D0 до D3.

D0 — полностью оперативное состояние, устройство включено.

D1 и **D2** — промежуточные состояния, активность определяется устройством.

D3 — устройство выключено.

2.6 Состояния производительности

Пока процессор или устройство функционирует (D0 и C0, соответственно), он может находиться в одном или нескольких состояниях производительности. Эти состояния зависят от конкретной реализации. Так, P0 - всегда наивысший уровень производительности; с P1 до P*n* последовательное снижение уровня производительности, до предела реализации где *n* не превышает 16.

P-состояния также известны как SpeedStep в процессорах Intel, как PowerNow! или Cool'n'Quiet в процессорах AMD, и как PowerSaver в процессорах VIA.

P0 максимальная производительность и частота

P1 меньше чем *P0*, напряжение/частота урезаны

P2 меньше чем *P1*, напряжение/частота урезаны

...

Pn меньше чем *P(n-1)*, напряжение/частота урезаны

3 Драйвер CPUfreq

3.1 Интерфейс CPUfreq

Важнейшая подсистема ОС Linux, которая позволяет изменять динамически частоту процессора в зависимости от текущей загруженности системы путем перевода процессора в одно из так называемых, P-state. Режимы P-state — это рабочие состояния, задающие тактовую частоту и напряжение питания процессора. Чем больше номер режима, тем ниже частота и напряжение питания процессора. Основные составные компоненты данной системы:

CPUfreq модуль, который предоставляет общий интерфейс различным низкоуровневым архитектурно-зависимым технологиям и высокоуровневым управляющим политикам

Архитектурно-зависимые драйверы, реализуют различные технологии для изменения частоты, такие как Intel SpeedStep Technology, Enchances Intel SpeedStep Technology.

3.2 Регуляторы

Внутриядерные регуляторы – специальные системы, которые определяют наиболее подходящее состояние процессора, ориентируясь на различные критерии, например текущую нагрузку на процессор, некоторые из них динамически изменяют частоту при изменении входных данных от системы или пользователя. В подсистеме CPUFreq в Linux Kernel доступно для использования пять регуляторов:

Регулятор performance (производительность): самая высокая частота. Регулятор производительности статично настраивает процессор на наивысшую

возможную частоту. По умолчанию этот регулятор не предоставляет возможностей экономии энергии, хотя в нем можно изменить частоту, которую он выбирает.

Регулятор powersave (энергосбережение): самая низкая частота. Регулятор powersave, статически настраивает процессор на самую низкую из доступных частот. В данном случае на производительность влияет тот, что система никогда не будет использовать более высокую частоту, независимо от степени занятости процессора. На практике этот регулятор часто не дает экономии энергии, поскольку самая большая экономия достигается за счет использования режимов C-state. Использование регулятора powersave увеличивает время работы процесса, поскольку он будет использовать наименьшую частоту; в результате система дольше не перейдет в состояние простоя и потому не получит экономию от вхождения в режим C-state.

Регулятор Userspace: ручная настройка частоты пользователем. Userspace предоставляет возможность вручную настраивать, выбирать и выставлять частоту. Этот регулятор также работает совместно с демонами частоты процессора, запущенными в пользовательском пространстве для регулирования частоты.

Регулятор Ondemand: изменение частоты по уровню загрузки процессора.

Регулятор ondemand проверяет уровень использования процессора, и если определённый порог превышен, регулятор выставляет наиболее высокую частоту из доступных. В случае, если процент использования ниже порога, регулятор понижает частоту до следующей возможной отметки.

Регулятор conservative: более плавный вариант ondemand

Регулятор conservative базируется на регуляторе ondemand. Он динамически настраивает частоты в зависимости от загрузки процессора, однако ведёт себя немного по-другому и даёт возможность увеличивать мощность постепенно. Регулятор conservative проверяет процент использования процессора и в случае, если значение выходит за нижний или верхний порог, постепенно повышает или понижает частоту до следующей возможной, вместо того чтобы сразу установить наивысшую частоту, как это делает регулятор ondemand.

4. Драйвер CPUIdle

4.1. Общее описание

Основной составляющей, которая обеспечивает энергоэффективность, является подсистема cpuidle. Этот компонент ОС Linux позволяет переводить систему в энергосберегающие C-состояния (C-states) . Режимы C-state, за исключением режима *C0*, когда процессор находится в работающем состоянии, — это режимы бездействия, когда процессор перестаёт подавать тактовые импульсы отдельным своим компонентам и отключает их для экономии энергии. Чем выше уровень режима C-state, тем больше компонентов он охватывает (остановка тактирования ядра процессора, прекращение подачи прерываний и т.д.) и тем больше экономия. Во время простоя системы эти режимы помогают сэкономить энергию.

Различные процессоры имеют различные энергосберегающие характеристики и используют разные алгоритмы входа и выхода в энергосберегающие состояния. Драйвер CPUIdle позволяет разделять интерфейс управления энергосбережением на две части: архитектурно-зависимую часть(которая является уникальной для каждой отдельно взятой архтектуры) и архитектурно-независимую часть, которая управляет политиками энергосбережения.

4.2 Интерфейс драйвера

Рассмотрим подробнее основные элементы подсистемы CPUIdle. На верхнем уровне драйвер устроен достаточно просто:

```
#include <linux/cpuidle.h>
   struct cpuidle_driver {
      char                name[CPUIDLE_NAME_LEN];
      struct module       *owner;
   };
```

```
int cpuidle_register_driver(struct cpuidle_driver *drv);
```

Эти действия позволяют драйверу быть доступным в sysfs. Ядро драйвера CPUIdle требует, чтобы только один драйвер был зарегистрирован в системе. Как только драйвер зарегистрирован в системе можно зарегистрировать драйвер отдельно для каждого процессора (вполне возможно что они могут иметь различные характеристики). Далее, необходимо описать С-состояния процессора:

```
struct cpuidle_state {
    char        name[CPUIDLE_NAME_LEN];
    char        desc[CPUIDLE_DESC_LEN];
    void        *driver_data;

    unsigned int    flags;
    unsigned int    exit_latency; /* in US */
    unsigned int    power_usage; /* in mW */
    unsigned int    target_residency; /* in US */

    unsigned long long      usage;
    unsigned long long      time; /* in US */

    int (*enter)    (struct cpuidle_device *dev,
                struct cpuidle_state *state);
};
```

Рассмотрим наиболее важные поля cpuidle-state. Доступные флаги:

- CPUIDLE_FLAG_TIME_VALID - должен быть установлен там, где возможно точно измерить время проведенное в данном состоянии.

- CPUIDLE_FLAG_TIME_STOP - флаг указывает на то что в данном состоянии таймер должен быть остановлен.

- CPUIDLE_FLAG_COUPLED - флаг указывает на то, что переход в данное состояние не может быть осуществлен независимо от состояний на других процессорах. Данный флаг в основном используется при описании состояний ARM процессоров.

Глубина состояния описывается полями: exit_latency, target_residency и power_usage. Поле exit_latency хранит информацию о том сколько времени нужно чтобы вернуться в полностью функциональное состояние. Поле target_residency характеризует минимальное время, которое процессор должен провести в данном состоянии чтобы окупить с энергетической точки зрения окупить переход в это состояние. Поле power_usage показывает, сколько энергии тратится во время нахождения процессора в данном состоянии.

В структуре cpuidle_state также хранится указатель на функцию в которой описывается процесс перехода в данное состояние. Описания этих полей достаточно для запуска и работы драйвера CPUIdle.

4.3 Политики энергосбережения.

Основным элементом управления энергосбережением драйвера CPUIdle являются так называемые регуляторы. Ядро Linux позволяет описывать множество регуляторов, но в системе одновременно не может работать не более одного. В Linux kernel 3.11 в подсистеме CPUIdle доступны для использования два стандартных регулятора.

4.4 Регулятор Ladder

Данный регулятор обеспечивает плавный переход в более экономичные режимы энергосбережения, переводя систему в одно из соседних состояний на некоторое фиксированное время. Предположим, что система находится в состоянии Cx, где x – это номер состояния от 1 до n. Если не происходит прерывания и система находится в этом состоянии ожидаемое время, тогда происходит прыжок в состояние Cx+1. В противном случае, после срабатывания прерывания, система переходит в Cx-1 состояние.

У такого подхода есть ряд недостатков, которые могут существенно повлиять на производительность системы. Каждый C-state характеризуется временем выхода (latency time) из этого состояния в рабочее: у более глубоких режимов latency time больше. Поэтому пробуждение системы из состояний с высоким номером может быть довольно долгим, а если рассматривать драйвер в виртуальном окружении, то накладные расходы на выход из этого состояния могут быть катастрофически неприемлемыми.

4.5 Регулятор Menu

Данный регулятор работает по несколько иному принципу. На принятие решений о переходе в другое состояние menu влияют несколько факторов: “порог энергетической окупаемости перехода” (Energy break even point), влияние на производительность(performance impact) и время выхода. Остановимся поподробнее на каждом из этих факторов.

Каждый переход в C-state характеризуется определенной энергетической стоимостью входа и выхода и временем, которые должна провести в этом состоянии система, чтобы окупить этот переход с энергетической точки зрения. Соответственно, необходимо хорошее предсказание как долго система будет простаивать. Так как существуют еще и другие способы пробуждения (например прерывания) помимо обычного простаивания определенное время, полученная оценка может быть довольно оптимистичной. Для получения более реальной оценки используется “корректирующий фактор”, который вычисляется на основе предыдущего поведения. Например, если в прошлый раз прерывание сработало по истечении половины ожидаемого времени

простаивания, то коэффициент составит 0.5. Регулятор menu для дальнейших операций использует усредненное значение корректирующего фактора. Однако нельзя опираться только на значение корректирующиего фактора. Э то связано с тем, что соотношение зависит от порядка ожидаемой величины. Например, если ожидаемое время составляет 500 миллисекунд, вероятность раннего прерывания намного выше, чем если бы ожидаемое время было равно 50 микросекундам.

C-состояния, особенно, те которые имеют большое время выхода, оказывают значительное негативное воздействие на производительность системы в целом. Для борьбы с этим эффектом используется так называемый множитель производительности (performance multiplier). Если время задержки, умноженное на performance multiplier, больше чем предполагаемое время, данное C-состояние исключается из числа кандидатов на переход из-за большого влияния на производительность. Соответственно, чем больше множитель, тем дольше система должна бездействовать, чтобы перейти в более глубокое C-состояние. Таким образом, чем меньше загружена система, тем в более глубокое C-состояние она перейдет.

4.6 Алгоритм предсказания следующего времени засыпания

Для предсказания ближайшего интервала простоя используются предыдущие n=8 значений. Данная константа требует дополнительного изучения и, возможно, с помощью ее изменения можно улучшить энергоэффективность, однако на данный момент используются данное значение как универсальное, проверенное на многих архитектурах. Для начала получаем значение времени, через которое произойдет прерывание таймером. В ранних версиях драйвера использовалось только это значение, умножалось на 0,5 и затем использовалось для выбора подходящего состояния для перехода. Затем алгоритм претерпел некоторые изменения. После получения данного значения времени, оно помещается в одну из 6 корзин согласно таблице 1. Номер корзины служит индексом в массиве корректирующих факторов. Таким образом учитывается тот факт, что вероятность раннего прерывания у длительных пауз выше, чем у коротких.

Номер корзины	Время до срабатывания таймера (uS)
1	<10
2	<100
3	<1000
4	<10000
5	<100000
6	>100000

Таблица 1. Зависимость номера корзины от времени до срабатывания таймера.

Далее вычисляются среднее значение по последним 8 интервалам и среднеквадратичная ошибка. Если в данной выборке есть выбросы, удаляем их и повторяем вычисления среднего и среднеквадратичной ошибки до тех пор пока не получим набор без выбросов или пока в выборке не останется ¼ часть от первоначального количества значений. В качестве предсказанного значения используется среднее от оставшейся выборки. Такой подход позволяет справиться с нагрузками которые происходят через длинные паузы, потому что, по сути, регулятор не допускает слишком оптимистичных оценок времени паузы, тем самым предотвращая переход в глубокие состояния в необоснованных с точки зрения производительности ситуациях.

После получения предсказания происходит процесс выбора подходящего C-состояния. Учитываются параметры состояния target_residency и exit_latency. Таким образом подбирается наиболее глубокое состояние, которое удовлятворяет предсказанной длительности паузы. Далее регулятор обновляет данные, в том числе сохраняет уже вычисленное значение интервала времени.

5 Постановка задачи

5.1 Особенности виртуального окружения

Описанный выше подход к энергосбережению, применяющийся в реальных операционных системах семейства Linux, совершенно не подходит для виртуальных систем. В первую очередь это связано с тем, что гостевой системе мы имеем дело с виртуальными процессорами для которых не определены энергосберегающие состояния, соответственно hardware cpuidle драйвер не работает в этих системах. Драйвер, тем не менее, может функционировать в виртуальном окружении, но ни о каком сбережении энергии не может быть и речи. Даже если hardware драйвер и запустится, он будет тратить процессорное время впустую. Кроме того, в ходе экспериментов выяснилось, что для запуска настоящего драйвера в виртуальной системе необходимо дополнительно дорабатывать архитектурно-зависимую часть подсистемы cpuidle, что связано с неполной реализацией архитектуры процессоров и несоответствии ей cpuidle драйвера. В ходе работы было решено отказаться от настройки hardware драйверов под виртуальное окружение, что связано прежде всего с разнообразием архитектур процессоров и виртуальных оболочек.

Вместо этого, было принято решение о разработке cpuidle драйвера для виртуальных Linux систем, работающих под управлением KVM (kernel virtual machine). Программное обеспечение KVM состоит из загружаемого модуля ядра (называемого kvm.ko), предоставляющего базовый сервис виртуализации, процессорно-специфического загружаемого модуля kvm-amd.ko либо kvm-intel.ko, и компонентов пользовательского режима (модифицированного QEMU). Все компоненты ПО KVM являются ПО с открытым исходным кодом. Компонент ядра, необходимый для работы KVM, включен в основную ветку Linux начиная с версии 2.6.20. Сам по себе KVM не выполняет эмуляции. Вместо этого программа, работающая в пространстве пользователя, использует интерфейс /dev/kvm для настройки адресного пространства гостя виртуальной машины. KVM превращает ядро Linux в гипервизор (в тот момент, когда происходит инсталляция модуля ядра kvm). Так как гипервизор является стандартным ядром Linux, он получает преимущества от изменений в стандартном ядре (поддержка

памяти, планировщик и т.д.). Оптимизация этих компонент Linux (таких как планировщик работающий за O(1) в ядре 2.6) дает преимущества как гипервизору (базовая операционная система), так и гостевой операционной системе Linux. Но KVM – не первая машина, которая делает это. С ядром, работающим как гипервизор, возможно затем запускать другие операционные системы, такие как другие ядра Linux или Windows. В нашей задаче для экспериментов в качестве гостевой системы использовалось ядро Linux версии 3.11. Здесь нужно обратить внимание на то что, KVM представляет собой программный решение, поддерживающее аппаратную виртуализацию. Необходимость поддержки аппаратной виртуализации заставила производителей процессоров несколько изменить их архитектуру за счет введения дополнительных инструкций для предоставления прямого доступа к ресурсам процессора из гостевых систем. Этот набор дополнительных инструкций носит название Virtual Machine Extensions (VMX). VMX предоставляет следующие инструкции: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXON и VMXOFF.

Процессор с поддержкой виртуализации может работать в двух режимах root operation и non-root operation. В режиме root operation работает специальное программное обеспечение, являющееся «легковесной» прослойкой между гостевыми операционными системами и оборудованием — монитор виртуальных машин (Virtual Machine Monitor, VMM), носящий также название гипервизор (hypervisor).

Чтобы перевести процессор в режим виртуализации, платформа виртуализации должна вызвать инструкцию VMXON и передать управление гипервизору, который запускает виртуальную гостевую систему инструкцией VMLAUNCH и VMRESUME (точки входа в виртуальную машину). Virtual Machine Monitor может выйти из режима виртуализации процессора, вызвав инструкцию VMXOFF. Процедура запуска виртуальных машин Процедура запуска виртуальных машин. Каждая из гостевых операционных систем запускается и работает независимо от других и является изолированной с точки зрения аппаратных ресурсов и безопасности.

Классическая архитектура программной виртуализации подразумевает наличие хостовой операционной системы, поверх которой запускается платформа виртуализации, эмулирующая работу аппаратных компонентов и управляющая аппаратными ресурсами в отношении гостевой операционной системы. Реализация такой платформы достаточно сложна и трудоемка, присутствуют потери производительности, в связи с тем, что

виртуализация производится поверх хостовой системы. Безопасность виртуальных машин также находится под угрозой, поскольку получение контроля на хостовой операционной системой автоматически означает получение контроля над всеми гостевыми системами.

В отличие от программной техники, с помощью аппаратной виртуализации возможно получение изолированных гостевых систем, управляемых гипервизором напрямую. Такой подход может обеспечить простоту реализации платформы виртуализации и увеличить надежность платформы с несколькими одновременно запущенными гостевыми системами, при этом нет потерь производительности на обслуживание хостовой системы. Такая модель позволит приблизить производительность гостевых систем к реальным и сократить затраты производительности на поддержание хостовой платформы.

5.2 Описание алгоритма энергосбережения для виртуального окружения

Во время простоя гостевая система может тратить процессорное время, что негативно сказывается на энергоэффективности. Так как виртуальное окружение не может самостоятельно переводить реальный процессор в энергосберегающие состояния, нужно переложить ответственность за эти действия на хостовую операционную систему. Поэтому нужно сообщить хостовой операционной системе о том, что гостевая система будем простаивать некоторое время и можно сэкономить энергию, переведя процессор в глубокие C-состояния. Для этих целей будем использовать операцию VMcall, которая вызывается в ходе исполнения кода гостевой системы. В гипервизоре происходит обработка в ходе которой определяется, какое время гостевая система будет простаивать.

На первом этапе был испробован метод, когда в случае предсказанных коротких пауз переход из виртуальной системы в хостовую систему не происходит. Однако, выяснилось, что у данного подхода есть некоторые проблемы связанные с производительностью. Прежде всего во время коротких пауз гостевая система занимает процессор. Как следствие хостовой системе, возможно, нужно будет запустить что-то еще, а процессорное время будет занято виртуальной системой.

В то же время некоторые программы в виртуальной системе могут не получить своего процессорного времени. Было принято решение в короткие паузы тоже вызывать VMcall Всего есть две возможности. В случае коротких времен засыпания происходит вызов функции msleep на время порядка 1 миллисекунды. Для длинных пауз, логично предоставить возможность хостовой системе хотя бы раз за 1 период сна перейти в самое глубокое состояние. В этом как раз и заключается эффект длинных пауз. На современных процессорах время выхода из самого глубокого состояния составляет порядка 50 миллисекунд, поэтому в качестве времени засыпания для длинной паузы будем использовать это значение. Кроме того, нужно учесть, что существует потенциальная опасность задержки передачи управления от хостовой системы к гостевой во время сна. Для этого вместо функции msleep используется функция msleep_interruptible. Данная функция перед засыпанием устанавливает для текущего потока флаг TASK_INTERRUPTIBLE, что означает возможность завершения сна по сигналу.

5.3 Результаты экспериментов

Для проведения тестов использовалась утилита "PowerTOP” которая предоставляет возможность измерять среднюю нагрузку на процессор, количество переходов в энергосберегающие состояния. Первый эксперимент - CPUidle драйвер на виртуальной машине не запущен, второй эксперимент - драйвер запущен, третий эксперимент виртуальная система не запущена. Эксперименты производились в течение 2 часов каждый. Большее время запуска программы не оказывает реального воздействия на итоговые значения. Результаты приведены в таблице №2.

	Средняя нагрузка на процессор	Время в C-состояниях
1 эксперимент	1,1%	15%
2 эксперимент	1,0%	52%
3 эксперимент	0,5%	99%

Таблица 2. Результаты экспериментов

В ходе второго эксперимента нагрузка на процессор оказалось на 10% ниже, а также существенно увеличилось время, которая система проводит в энергосбергающих состояниях. Безусловно, идеальным было бы значение в 100%, как в третьем эксперименте на машине без запущенной виртуальной системы, но, тем не менее, удалось существенно увеличить количество переходов в C-состояния, что, конечно же, положительно сказывается на энергоэффективности.

Стоит отметить, что данное программное решение легко перенести на другие платформы виртуализации, в частности Parallels Desktop и VMware. Это связано с особенностями архитектуры данных платформ виртуализации. Гипервизор данных платформ это компонент, работающий в одном кольце с ядром основной ОС (кольцо 0). Гостевой код может выполняться прямо на физическом процессоре, но доступ к устройствам ввода-вывода компьютера из гостевой осуществляется через второй компонент, обычный процесс основной ОС, который называется монитором уровня пользователя. Такой подход имеет свои плюсы с точки зрения производительности по сравнению с лекговесным гипервизором KVM.. Реализованный драйвер позволит монитору виртуальной машины ограничить количество переходов в хостовую систему, тем самым потенциально избегая переключения контекстов и повышая энергоэффективность всей системы в целом.

6 Заключение

В данной работе был разработан алгоритм, который позволяет повысить энергоэффективность виртуализованных систем семейства Linux при помощи гипервизора KVM(Kernel-Virtual-Machine). Также были рассмотрены общие вопросы энергосбережения в ядре Linux, алгоритмы и существующие эвристические модели улучшающие энергоэффективность. Были рассмотрены некоторые проблемы данных решений и предложены пути их решения как для виртуальных систем так и для систем, работающих на реальном оборудовании. Кроме того, реализованное программное решение может быть перенесено на другие платформы виртуализации, например, Parallels Desktop. Применение это решения потенциально позволит повысить энергоэффективность данной виртуальной платформы.

Хотелось бы поблагодарить ведущего разработчика компании Parallels Мелехову Анну за неоценимую помощь в работе.

7 Список литературы

1. Advanced Configuration and Power Interface Specification, Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd, Toshiba Corporation, Revision 4.0a, April 5, 2010
2. Venkatesh Pallipadi, Shaohua Li, Adam Belay. «cpuidle – Do nothing, efficiently …», Proceedings of the Linux Symposium 2007, Volume Two, p. 119.
3. Venkatesh Pallipadi, Alexey Starikovskiy. The Ondemand Governor: Past, Present, and Future. Proceedings of the Linux Symposium 2006, p. 657.
4. Len Brown, Anil Keshavamurthy, David Shaohua Li, Robert Moore, Venkatesh Pallipadi, Luming Yu. ACPI in Linux: Architecture, Advances, and Challenges. Proceedings of the Linux Symposium 2005, p. 51.
5. Suresh Siddha, Venkatesh Pallipadi, Arjan Van De Ven. Getting Maximum Mileage out of Tickless. Proceedings of the Linux Symposium 2007, Volume Two, p. 201. Performance Tuning Towards a KVM-based Embedded Real-time Virtualization System Ruhui Ma, Haibing Guan ∗, Alei Liang, Department of Computer Science and Engineering, Shanghai key laboratory of scalable computing and systems, Shanghai Jiao, Tong University, Shanghai, 200240, China
6. M. Rosenblum and T. Garfinkel, Virtual Machine Monitors: Current Technology and Future Trends, Computer, vol.38, pp.39-47, 2005.
7. I. Habib, Virtualization with KVM, Linux Journal, vol.2008 n.166, 2008.
8. Jenifer Hopper, Reduce Linux power consumption, IBM, 2008
9. Mudit Vats, Ishu Verma, Linux Power Management IEGD Considerations March, 2010
10. KVM Main Page. http://www.linux-kvm.org/page/Main_page
11. Real-Time Linux Wiki, http://rt.wiki.kernel.org.
12. Patrick Mochel, Linux Kernel Power Management, Open Source Development Labs, 2003
13. Linux Cross Reference http://lxr.free-electrons.com/