ЮГО-ЗАПАДНОЕ ОКРУЖНОЕ УПРАВЛЕНИЕ ОБРАЗОВАНИЯ
ДЕПАРТАМЕНТА ОБРАЗОВАНИЯ
ГОРОДА МОСКВЫ

ГБОУ ЛИЦЕЙ № 1533 (ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ)

ВЫПУСКНАЯ РАБОТА

(специальность ”Прикладное программирование”)

учащегося группы 11.3
Захарова Ильи Александровича

Разработка гипервизора Jinet

Научный руководитель: Байков Б. К.,
ведущий системный разработчик,
«Т-Платформы»

Консультанты: Потёмкин А.В
Гиглавый А.В.
Завриев Н.К.
Труфанов Д.С.

Москва — 2017

Содержание

1 Введение

Гипервизор – это программа, обеспечивающая разделение ресурсов компьютера на несколько виртуальных машин, и запуск на каждой виртуальной машине своей операционной системы.

Такая программа обеспечивает либо разделяемый, либо монопольный доступ виртуальных машин к каждому из аппаратных устройств компьютера. Создаются виртуальные устройства, конфигурация которых может отличаться от конфигурации устройств аппаратных.

Виртуализация позволяет эффективно и безопасно разделять работающие приложения друг от друга. Виртуальные машины не могут влиять на ход работы друг друга, и, если скомпрометирована одна виртуальная машина, все остальные остаются в безопасности. Поэтому она используется в самых разных областях IT. Зачастую виртуализацию также используют для эффективного сегментирования ресурсов компьютера на отдельные виртуальные машины.

Была поставлена задача написать минимальный учебный демонстрационный гипервизор. Исходный текст проекта выпущен под лицензией MIT. `git`-репозиторий гипервизора располагается по адресу `https://github.com/jinet-vm/vmm`.

2 Актуальность

С каждым годом технологии виртуализации всё глубже и глубже входят в мир информационных технологий, находя применения в самых разных областях IT:

1. изоляция серверных систем для обеспечения их безопасности

2. эффективное сегментирование ресурсов компьютера

3. одновременное использование разных ОС на настольном компьютере

4. отладка гостевых систем через вывод всех выходов из ВМ

Для создания систем виртуализации требуются специалисты, способные понимать принципы и механизмы программной и аппаратной виртуализации. Для меня, как автора диплома, целью написания диплома стало погружение в мир виртуализации, вычислений и прямого программирования железа. Удалось узнать, как работает виртуализация изнутри и научиться писать код управления механизмами

обслуживания виртуализации.

3 Постановка задачи

Идея проекта была предложена руководителем с целью изучения архитектуры x86, включая механизмы изоляции приложений и виртуальных машин друг от друга путём создания монитора виртуальных машин, он же гипервизор. Для меня это стало уникальной возможностью углубить знания архитектуры процессоров Intel и навыки системного программирования.

Цель работы – это создание минимального монитора виртуальных машин (гипервизора) с использованием механизмов аппаратной виртуализации архитектуры x86-64 (AMD64).

Были поставлены следующие задачи:

1. Обеспечить загрузку из бутсектора (bootsector)
2. Настроить режим работы видеоадаптера для отображения вывода текстовых данных
3. Настроить память в 32-битной плоской модели и осуществить переход в 32-битный защищённый режим (`Protected Mode`)
4. Настроить механизмы страничной трансляции памяти и переход в 64-битный режим (`Long Mode`)
5. Настроить обработку исключений и прерываний
6. Настроить расширения аппаратной виртуализации Intel VT-i и включение режима VMX
7. Обеспечить функционирования вызовов (VMCall) из виртуальных машин в гипервизор (монитор виртуальных машин) и возвратов обратно в виртуальные машины

4 Анализ предметной области

4.1 Виртуализация

Виртуализация – техника предоставления исполняемой программе набора вычислительных ресурсов, абстрагированная от их аппаратной реализации. Виртуализация была предметом изучения информатики на протяжении многих лет: так, например, советские инженеры решали проблему портирования программного обеспечения с платформ имеющих другие интерфейсы, нежели физический компьютер, на котором программа исполнялась.

В рамках этой работы мы будем говорить не столько о виртуализации ресурсов, сколько о работе гипервизора – программы, занимающейся разделением работы ресурсов одной физической машины (хозяин (*англ.* host)) на множество виртуальных машин (гость (англ. guest)). Внутри каждой виртуальной машины исполняется своя ОС, ход работы которой не влияет на работу других ВМ.

Гипервизор, в отличие от эмулятора, выполняющего программную эмуляцию команд, лишь перехватывает управление у виртуальных машин в случае необходимости. Код виртуальных машины выполняется аппаратно в процессоре. Так как принцип работы гипервизора предполагает изоляцию виртуальных машин, для более эффективной его работы необходима аппаратная поддержка виртуализации. Первой в этой области была компания IBM с мэйнфреймами System/360, System/370, созданными на рубеже 60-70-х годов прошлого века. Современные процессоры Intel также поддерживают расширения аппаратной виртуализации (VT-i, VT-d), что значительно ускоряет процесс виртуализации.

После появления первых гипервизоров появилась необходимость создания формальных критериев виртуализации. В 1974 статья Джеральда Попека и Роберта Гольдберга их сформировала.

4.1.1 Классический критерий виртуализуемости

Требования к монитору виртуальных машин (то же, что и гипервизор) состоят из трёх пунктов:

1. **Изоляция**. Каждая ВМ имеет доступ только к своим ресурсам. Виртуальные машины не влияют на работу друг друга, если одна виртуальная машина оказалась зараженной вирусом, другая продолжает работу.

2. **Эквивалентность**. Виртуальная машина ведёт себя так же, как и настоящий компьютер с аналогичными характеристиками. Единственное различие заключается в скорости исполнения, вирту-

альная машина работает медленнее компьютера.

3. **Эффективность**. Статистически преобладающее подмножество инструкций виртуального процессора должно исполняться напрямую хозяйским процессором, без вмешательства монитора ВМ. Управление передаётся гипервизору только в случае привилегированной операции – той, которая может нарушить изоляцию машин.

4.1.2 Типы гипервизоров

Гипервизоры делят по их устройству на две большие группы: гипервизоры I и II типа.

- **Гипервизоры I типа** исполняются непосредственно на компьютере и имеют полный доступ к его устройствам. Компьютер загружается в софт гипервизора, и монитор ВМ, подобно операционной системе, начинает работы с устройствами. Примеры: VMWare ESXi, Xen. (см. рис. 1)

- **Гипервизоры II типа** загружаются внутри ОС и получают доступ к устройствам через её интерфейсы. Зачастую при установке гипервизор II типа устанавливает в ядро ОС свой модуль, с помощью которого он может обращаться к низкоуровневым процессорным расширениям виртуализации. Примеры: KVM, VirtualBox. (см. рис. 2) Гипервизор Jinet - гипервизор I типа: так легче

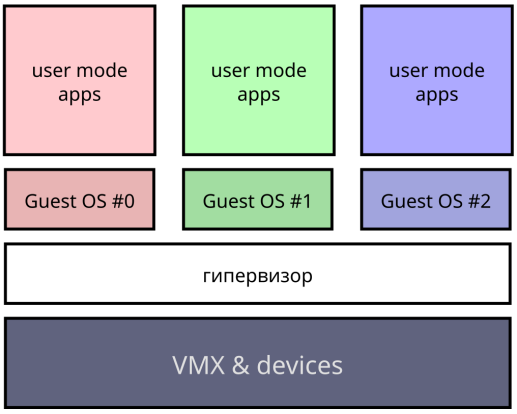

Рис. 1: I тип гипервизоров

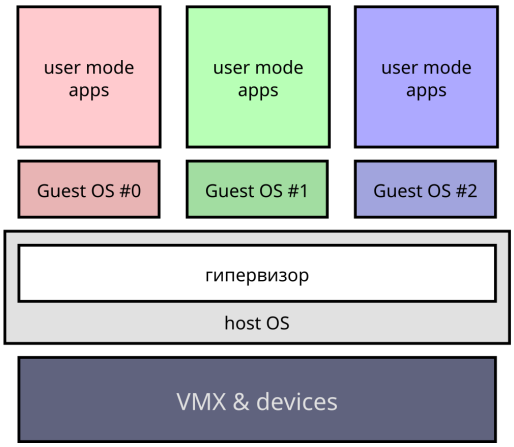

Рис. 2: II тип гипервизоров

обращаться к ресурсам компьютера.

4.2 Архитектура процессоров Intel x86

4.2.1 Загрузка из бутсектора (bootsector) и работа в Real Mode

BIOS передаёт управление на физический адрес памяти 0x7C00, после загрузки туда первого сектора (512 байт) загрузочного устройства. Машина начинает работать в 16-битном Real Mode. Программисту доступны для вычислений регистры битности 8, 16, 32. Адресация осуществляется с помощью сегментной модели памяти. Физический адрес равен `(эффективный адрес) + (соответствующий сегмент) * 16`. Таким образом, т.к. и сегментный регистр, и регистр с логическим адресом – 16-битные, в Real Mode доступно около 1 МБ логической памяти.

Загружается указатель стека (указатель стека растёт в сторону меньших адресов). Выполнив инициализацию устройств, программа может догружать свой основной код и, после прыжка в 32-битный режим, продолжает работу в нём.

4.2.2 Настройка видеоадаптера

С целью не настраивать работу с графикой напрямую через видеокарту (в таком случае, для каждой видеокарты необходимо разрабатывать свой драйвер), было решено разрабатывать драйвера под известные стандарты – интерфейсы к низкоуровневым подсистемам видеоадаптера, предоставляемые BIOS. В данной работе было изучено две такие технологии:

- **VGA**. Эта система существует с самого начала существования IBM PC-совместимых компьютеров. Разработчику предоставляются текстовый режим 80x25 (буфер записи - 0xB8000, 16 цветов фона и текста), графический режим 320x200 (буфер записи - 0xA0000, 256 цветов).

- **VESA**. Версии этого стандарта были разработаны в последнее десятилетие прошлого века ассоциацией Video Electronics Standards Association под названием VESA BIOS Extensions (VBE). Предоставляются режимы разных разрешений с разной глубиной цвета. В рамках данной работы используется VESA-режим 1280x1024 с глубиной цвета 8 бит (VBE 2.0).

4.2.3 Настройка 32-битной GDT и переход в Protected Mode

Сегментная модель памяти используется и в 32-битном режиме, Protected Mode, но несколько в другом качестве. Теперь сегменты характеризуются, в первую очередь, смещением и размером. Физический

адрес равен сумме смещения и линейного адреса. Характеристики сегментов регулируются глобальной таблицей дескрипторов - GDT. Для загрузки в 32-битный режим необходимо подготовить 32-битную GDT. В рамках данного проекта настраивается 32-битная GDT с flat-моделью: смещения регистров равны нулю, и эффективный адрес равен логическому. Загружается GDT, выполняется загрузка теневой части сегментных регистров.

4.2.4 Настройка 64-битных таблиц страничной трансляции и переход в Long Mode

Механизм сегментации остаётся в Long Mode (64-битный режим) в рудиментарном качестве: большинство сегментов обязательно являются flat-сегментами, исключение делается для служебных структур, например TSS. Основным и обязательным механизмом управления виртуальной памятью становится страничная трансляция: виртуальная память делится на страницы разного размера, которым в соответствие ставится участок физической памяти; при обращении к странице, процессор находит физический адрес странице по логическому, добавляет смещение и происходит обращение.

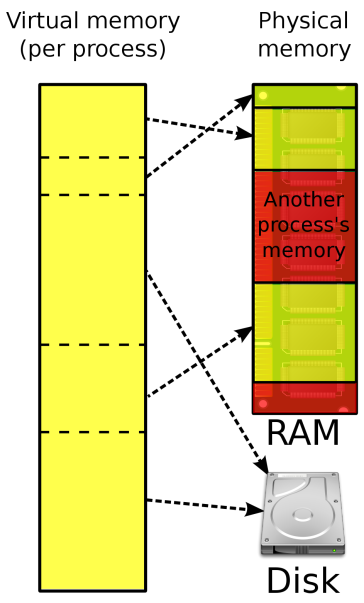

Рис. 3: Механизм работы виртуальной памяти

В основе виртуальной памяти лежит система из 4-х уровней таблиц страничной трансляции: PML4, PDP, PD, PT. PT - самая младшая таблица, страницы которой указывают непосредственно на физическую память. Ячейки других таблиц могут указывать на непрерывные куски памяти или на таблицу младшего уровня. Из-за особенностей устройства логической памяти на Intel процессорах (48-битная адресация), память делится на две половины. В рамках проекта гипервизор перенесён во вторую половину логической памяти.

4.2.5 Настройка обработка исключений и прерываний

Исключения и прерывания приостанавливают поток исполнения, чтобы обработать ошибку процессора или внешнее событие. Для написания таких обработчиков определяется таблица дескрипторов прерываний – IDT. Процессор получает информацию от устройств через прерывания. В этой задаче ему помогает контроллер прерываний APIC: вы нажимаете клавишу на клавиатуре, сигнал попадает в контроллер прерываний (8259 PIC или более новый APIC), а он передаёт данные на шину процессора. В гипервизора Jinet используется APIC.

Для передачи сигнала о прерывании используется Local APIC, который передаёт сигнал с внутренней шины внутрь процессора (в наше время Local APIC обыкновенно находится внутри процессора). Контроллер I/O APIC собирает сигналы с внешних устройств и передаёт сигнал на шину. Такое устройство системы делает возможной работы многоядерных и многопроцессорных систем.

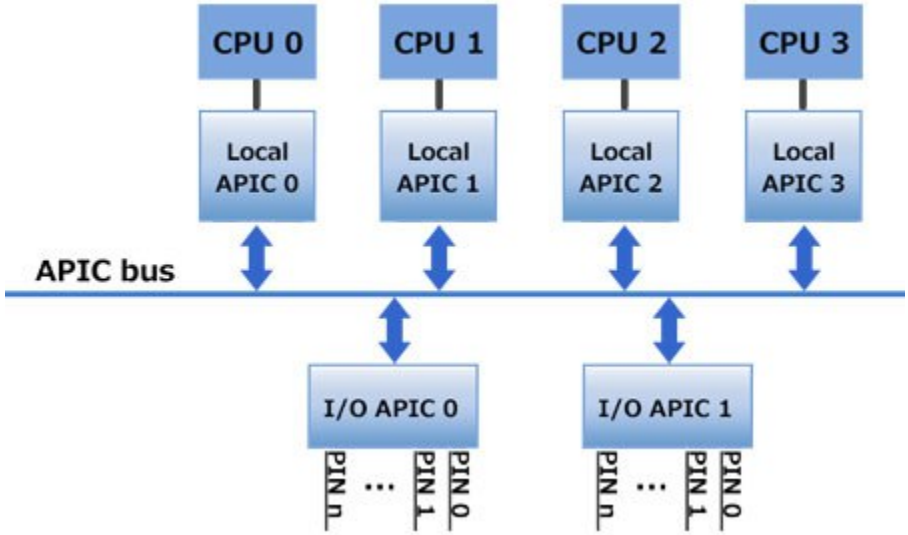

Рис. 4: Логическая схема работы APIC

4.2.6 Настройка аппаратной виртуализации

В рамках данной работы используются расширения аппаратной виртуализации Intel VT-i. AMD реализует на своих процессорах аналогичную технологию под название SVM.

Работа виртуальных машин регулируется структурой VMCS – управляющая структура виртуальных машин. Обращаясь к ней через интерфейс VMX-а (команды `vmread`, `vmxon`, `vmread`), мы можем её инициализировать и настраивать поля-регистры виртуальной машины. Для запуска виртуальной машины необходимо заполнить около сотни разных полей, регулирующих её поведение и характеристику. После такой настройки включается виртуальная машина с помощью команды `vmlaunch`.

4.2.7 Обработка вызовов из виртуальной машины

После включения виртуальной машины логический процессор входит в режим VMX non-root operation. В случае исполнения привилегированной операции происходит VMExit, процессор возвращается в VMX root operation и управление передаётся на обработчик VMexit, адрес которого задаётся полем VMCS.

Через соответствующие поля в VMCS обработчик может узнать, что стало причиной VMExit, чи-

тается поле VM Exit Basic Reasons (см. [1, том 3D]). В случае, если произошёл VMCall, который вызывается из гостевой системы одноимённой командой, вызывается особый обработчик. Иначе гипервизор обрабатывает причину вызова vmexit.

5 Обзор аналогов

В данной работе представляется маленький гипервизор. Подобные ему большие аналоги писали десятки программистов много месяцев. Гипервизор Jinet сейчас позволяет запускать простейший код в изолируемом окружении BM виртуальной машины. У всех из предложенных ниже гипервизоров исходный текст находится в открытом доступе. Приведённые аналоги были написаны специалистами в области информационной безопасности и системного программирования. Некоторые из предложенных гипервизоров первого типа, а некоторые – второго.

Таблица 1: Аналоги

Гипервизор	Тип VMM[1]	Платформы (если II тип)	Размер исходного текста	Назначение
SimpleVisor	II	Win8.1; частично UEFI	162 KB	использование новейшего VMX
Ramooflax	I	–	2.26 MB	анализ/отладка /контроль BM
HyperPlatform	II	Win7; Win8.1; Win10: x86 & x64	7.64 MB	анализ работы Windows
Jinet	I	–	421 KB	учебный гипервизор

[1] см. 4.1.2

6 Ход работы

В ходе работы над проектом было реализовано множество задач (■ и + означают, что задача решена; □ и − означают, что задача ещё не решена):

6.1 Изучение Real Mode ассемблера (16bit)

+ Загрузочный сектор: написание своего загрузочного кода
+ Написание клеточного автомата в загрузочном секторе
+ BIOS - прерывания: int 13h, int 15h
+ Работа с VideoBIOS: работа с int 10h
+ Начало работы с эмуляторами Bochs, QEMU

6.2 Изучение Protected Mode (32bit)

+ Управление памятью: глобальная таблица дескрипторов (GDT)
+ Изучение Big Real Mode
+ Изучение структуры исполняемого файла ELF
+ Изучение компоновки кода на Assembler и кода на языке C
+ Начало разработки в Protected Mode на языке C
+ Чтение карты доступной физической памяти
+ Виртуальная память: настройка страничного отображения (Paging)
+ Обработка прерываний: таблица прерываний в защ. режиме (IDT)
+ Перенос кода ядра и таблиц выше 1 МБ

+ Изучение механизмов аппаратной многозадачности. Структура TSS

6.3 Изучение топологии ядер и процессоров (ACPI: SRAT, SLIT)

+ Изучение APIC, XAPIC (LAPIC, IOAPIC, ACPI: MADT)

6.4 Изучение Long Mode (64 bit, x86-64)

+ Изучение особенностей TSS и IDT в 64-битном режиме

+ Изучение особенностей страничной адресации памяти (PAE)

6.5 Изучение механизмов организации ядра

+ Сборка проекта с помощью утилиты `make`

+ Портирование проекта с собственного загрузчика на GRUB

+ Настройка VGA, VBE как выводов терминала системы

+ Управление памятью

    + Высокоуровневый доступ к структурам paging-а

    + Выделение физической памяти (binary buddy allocator)

    + Куча (heap)

+ Доступ к отладочной консоли через Intel LPSS UART[2]

+ Обработка ACPI таблиц: MADT, DBGP

+ Инициализация многоядерности: отправление SIPI, init-IPI

+ Разработка планировщика для задач на BSP (round-robin)

[2] определяется в DBGP, PCI номер устройства: вендор 8086h, id устройства A166h

* Написание документации проекта на Doxygen [in process]

6.6 Изучение аппаратных механизмов виртуализации VMX (VT-i, VT-d)

+ Подготовка и включение VMX-режима
+ Подготовка управляющих структур для виртуальной машины
+ Создание обработчика VMCal
+ Создание обработчиков событий
– Создание расширенных таблиц страничной трансляции (EPT)
– Создание BIOS для виртуальных машин
– Создание виртуальных устройств:
  – Клавиатура и мышь
  – Жёсткий диск (виртуальный int 13h)
  – Виртуальный текстовый дисплей
  – Виртуальный VGA дисплей
– Создание механизма проброса устройств (pass-through)
– Создание партиции гипервизора и её структуры
– Создание инструментов управления

В ходе работы над дипломным проектом было изучено большое количество документации по процессорам Intel ([1]) и AMD (как первосоздателя x86-64: [2]), документация `gnu make` ([3]), `gnu ld` ([4]), `gcc` ([5]), `fasm` ([6]), формата ELF ([7]), таблиц ACPI ([8])

7 Программная реализация

Гипервизор Jinet был написан на языках программирования ассемблер (диалекты `fasm` и `as`) и C (компилятор `gcc`). Сборка проекта осуществляется с помощью сборщика `gnu ld` и утилиты `gnu make`. В качестве системы контроля версий используется `git` и хостинг GitHub.

Для вёрстки данной дипломной работы использовался LaTeX, а презентация для защиты создавалась в Microsoft PowerPoint.

8 Пример использования

Рис. 5: Работа программы (VESA-режим)

На рис. 5 изображена работа гипервизор Jinet. На данный момент гиперивизор поддерживает работу виртуальной машины, которая выводит сообщение о своём успешном запуске с помощью vmcall: `Hello! I am VM1!`.

9 Результат

Была изучена архитектура процессоров Intel x86. За время работы над гипервизором пришлось изучить самые разные аспекты работы современных ПК: от работы в 16-битном режим до опыта неуспешной отладки на разработческих платах.

Был получен важный опыт работы с технической документацией. Большая часть документации, изученной во время работы над дипломным проектом, была написана на техническом английском.

При работе с такой низкоуровневой технологией, как аппаратная виртуализация, была важно консультироваться со справочниками и документацией Intel.

Был изучен инструментарий современного системного программиста. Основы статического компонования, формат исполняемых файлов `elf`, особенности встроенного ассемблера gcc – самые разные технологии системного программирования были изучены при написании данной работы. Изучались также утилиты для сборки: были написаны скрипты для сборки проекта утилитами `gnu make` и `gnu ld`, а также Python-скрипт для конфигурирования проекта.

Были изучены основы работы с языком ассемблер и работы с большими проектами на C. Были изучены три диалекта языка ассемблер: `masm`, `fasm` и `gnu as`. В работу вошёл код, написанный на последних двух. Также были изучены принципы написания интерфейсов и безопасного программирования на C.

10 Выводы

Создан гипервизор Jinet. Доказано, что можно написать гипервизор силами одного человека в короткие сроки. Многое ещё требует реализации и улучшения:

1. Доработка SeaBIOS для поддержки запуска 16-битных операционных систем
2. Поддержка ряда устройств для запуска одного из вариантов DOS
3. Поддержка гипервизором работы на нескольких ядрах, процессорах
4. Поддержка NUMA-систем
5. Поддержка технологии Intel GVT-g (представлена в 2015), делающей возможной аппаратную виртуализацию видеоадаптеров для ВМ

11 Благодарности

В рамках данного проекта использовался исходный код реализации функции printf из проекта tinyprintf: `https://github.com/cjlano/tinyprintf`.

12 Список литературы

[1] Intel. Intel® 64 and IA-32 Architectures Software Developer Manual. 2017.

[2] AMD. AMD64 Architecture Programmer's Manual, Volume 2: System Programming. 2017.

[3] GNU Project and Free Software Foundation. GNU Make Manual.

[4] GNU Project and Free Software Foundation. GNU Linker Manual.

[5] GNU Project and Free Software Foundation. GCC Manual.

[6] Grysztar Tomasz. flat assembler 1.71.

[7] Portable Formats Specification Version 1.1. Executable and Linkable Format (ELF).

[8] Toshiba HP Intel Microsoft Phoenix. Advanced Configuration and Power Interface Specification, Revision 5.0a.

[9] Popek G. J., Goldberg R. P. Formal requirements for virtualizable third generation architectures // Communications of ACM. 1974.

[10] @Atakua. Аппаратная виртуализация. Теория, реальность и поддержка в архитектурах процессоров. `https://habrahabr.ru/company/intel/blog/196444/`. 2013.

[11] М. Чурдакис. The Infamous Trilogy: CPU internals, Virtualization, Raw multicore programming. `https://www.codeproject.com/articles/45788/the-real-protected-long-mode-assembly-tutorial-for`. 2015.

[12] OSDev Wiki. `http://wiki.osdev.org/Main_Page`.

[13] В. Садовников. Начала программирования в защищённом режиме. `http://e-zine.excode.ru/online/1/Introduction.html`. 2006.