Урок 8

Кроссплатформенность и виртуализация

Кроссплатформенность. Системные вызовы и трансляция системных вызовов. Эмуляция и виртуализация. Аппаратная виртуализация. Виртуализация на уровне ядра. Паравиртуализация.

- Введение
- Исполнение родного кода
  - Аналоги
  - Компиляция для разных платформ
  - Поставка в исходниках
  - Кросс-компиляция
  - Платформонезависимый код (скрипты, байт-код)
  - Поддержка среды
  - Проблема курицы и яйца. Раскрутка компилятора
  - Языки высокого и низкого уровня
  - Удалённый доступ, клиентские машины, тонкий клиент
  - Сопроцессор
- Трансляция кодов
  - Эмуляция
  - CISC- и RISC-процессоры

Трансляция вызовов

- Механизм поддержки API другой ОС
- Нативная поддержка нескольких API
- Поддержка файловых систем
- Реализация ОС

Виртуализация

- Динамическая трансляция
- Аппаратная виртуализация
  - Виртуальный 8086
  - Intel-VT и AMD-V
    - KVM
- Виртуализация на уровне ОС
    - Изоляция процессов: chroot для ФС и cgroups для процессов
    - Контейнерная виртуализация
    - OpenVZ, LXC, Docker
- Паравиртуализация

Практическое задание

Дополнительные материалы

Используемая литература

Введение

Простейший способ аппаратной виртуализации мы рассмотрели: приложения MS DOS, написанные на 16-битном машинном коде, могут работать на 32-битных машинах, использовать 32-битный код в самой операционной системе и других приложениях, задействовать механизм защиты памяти, а их память не ограничена 1 Мб. Каждое приложение MS DOS могло работать в своём пространстве 1 Мб памяти, исполняя 16-Мбитный код, но при этом операционной системе транслировались системные вызовы к DOS, работающие в 32-битном коде. Как вы помните, виртуальный режим 8086 – не что иное, как способ аппаратной виртуализации.

Рассмотрим способы запуска программ, написанных для одних операционных систем/архитектур процессора на других ОС/архитектурах, а также механизмы, которые позволяют осуществить такую работу. Эти механизмы могут пересекаться, комбинироваться, относиться одновременно к разным разделам приведённой классификации.

Исполнение родного кода

Аналоги

По сравнению с CP/M, по аналогии с которым была создана, MS DOS включала ряд UNIX-подобных утилит. Это относится и к архитектуре ОС (возможность организации конвейеров, встроенные команды), и к неописанным утилитам – например, more, fdisk и аналоги, но с другими именами. Важно, что это не кросс-компиляция: программы изначально пишутся для каждой ОС/архитектуры отдельно.

Компиляция для разных платформ

При компиляции для разных платформ код, написанный, например, на C, компилируется для каждой операционной системы отдельно. В частности, код также может быть написан по-разному для разных операционных систем и платформ.

Это связано и с наличием соответствующих библиотек, компиляторов, особенностей архитектуры. Так, код, написанный под 32-битные процессоры, работать будет и на 64-битных, но в 32-битном режиме. Для переноса кода в 64-битный режим его необходимо дорабатывать.

Поставка в исходниках

Поставка в исходниках возможна для семейства операционных систем (например, для разных дистрибутивов Linux). В этом случае rpm или deb-пакет содержит исходные файлы, которые необходимо скомпилировать на машине назначения.

Обычно для Linux-дистрибутивов это выглядит так:

```
# tar xvfz distrib.tar.gz
# cd distrib
# .configure
# make
# make install
```

Как правило, для популярных дистрибутивов ОС уже готовятся готовые пакеты, но иногда приходится использовать и самостоятельную сборку или править файл/заголовки, прежде чем ПО соберётся.

Для сборки пакетов на машине должны присутствовать инструментальные средства, уже скомпилированные для разных платформ.

Кросс-компиляция

В нормальном режиме ПО собирается на машине, для которой она предназначена. Кросс-компиляция – возможность собрать ПО на машине с другой архитектурой. Это может быть удобно, когда нужно подготовить дистрибутивы для разных версий ОС/аппаратных архитектур.

Платформонезависимый код (скрипты, байт-код)

Платформонезависимый код относится к интерпретируемым скриптам (php, perl, python, bash, node.js) и компилируемым в байт-код (java, .NET, Форт). Его создание уже можно отнести к виртуализации. В роли виртуального процессора выступает либо сам интерпретатор (php, V8), либо соответствующие виртуальные машины, исполняющие байт-код (Java). В последнем случае перед нами уже нечто более похожее на виртуальный процессор, исполняющий аналог машинного кода (байт-код). Разница только в том, что виртуальная машина изначально разработана не как аналог физического процесса (см. эмуляция), а изначально как программная разработка.

Кроме того, сам интерпретатор или виртуальная машина уже должны быть разработаны, скомпилированы с учётом разных архитектур, на которых будет исполняться интерпретируемый скрипт или байт-код.

Поддержка среды

Ещё сюда же можно отнести поддержку среды, когда приложения выполняются в нативном коде аппаратной платформы и ОС, но при этом используют высокоуровневую среду, схожую со средой другой операционной системы: это наборы утилит, заголовочные файлы, компиляторы.

Среди них реализация POSIX для Windows – подсистема для приложений на базе UNIX, UNIX-подобная среда для Windows. Как отдельные компоненты сюда входят XMing – реализация X11-Server, MinGW – набор инструментов для компиляции.

Поддержка среды не позволяет запускать на этой же машине родные скомпилированные файлы для другой среды, но позволяет скомпилировать и исполнять программы, написанные для другой архитектуры ОС на этой.

Проблема курицы и яйца. Раскрутка компилятора

Проблема курицы и яйца состоит в следующем: мы несколько раз упомянули, что для исполнения или сборки кода необходим уже готовый код – но как начинается разработка? Если это новый процессор, то разработку начинают в машинных кодах. Если новая операционная система, для разработки может быть использован инструментарий родственной или похожей операционной системы. Так, Линус Торвальдс для разработки ядра Linux использовал ОС и окружение ОС Minix Эндрю Танненбаума.

Остро стоит вопрос создания новых компилируемых языков программирования. Такой процесс называется раскруткой компилятора (англ. bootstrapping). Для этого пишется начальный код на машинном коде либо имеющемся языке, затем пишется код, выполняющийся на нужном языке программирования, и компилируется так, чтобы заместить код, скомпилированный из исходного кода

на другом языке или непосредственно в машинном коде. Далее добавляется новый функционал и перекомпилируется.

Языки высокого и низкого уровня

Все моменты, связанные с созданием исполняемого кода для разных платформ (ОС или аппаратных платформ), обусловлены языками высокого уровня, которые либо компилируются в машинный код соответствующей платформы (отличающийся не только для разных процессоров, но и для разных ОС благодаря разным системным вызовам, прерываниям и т.д.), либо интерпретируются как переносимый код.

Ассемблеры и машинные коды как низкоуровневые средства обычно привязаны к архитектуре, и хотя перенос с одной машины на другой возможен, в таком случае требуется доработка под каждую конкретную архитектуру, для машинного кода сравнимая с написанием кода заново.

Удалённый доступ, клиентские машины, тонкий клиент

Особый случай – когда код выполняется на машине с родной ОС/архитектурой, но ввод-вывод осуществляется с другой машины (терминала). По большей части это касается всех веб-приложений (например, сервер работает на Linux, а браузер в Windows).

Более частным случаем является использование сессии ssh с целью доступа к Linux-машине для Windows, для чего используются аналоги: putty – для подключения к ssh, XMing – для реализации X-Server на Windows-машине. К этому же случаю можно отнести такие механизмы, как RDP (англ. Remote Desktop Protocol – протокол удалённого рабочего стола), позволяющий запускать приложения, физически выполняемые на Windows-машине, на других машинах.

На подобной идее строится концепция тонкого клиента, когда машина (возможно, с устаревшим аппаратным обеспечением) служит для доступа к приложениям, выполняемым на сервер приложений.

Отдельно выделяют веб-приложения и веб-интерфейсы (Microsoft Word Online, Google Docs), а также удалённые файловые хранилища (Onedrive, Google.Drive, работающие совместно с вышеупомянутыми приложениями).

Сопроцессор

Сопроцессор – это интересный способ, позволяющий выполнять код для другой архитектуры, используя в качестве дополнительного процессора процессор с поддержкой нужной архитектуры.

Так, в компьютерах Apple II, построенных на базе процессора MOS Technology 6502, вместо доработки CP/M для используемого процессора разработали специальную плату расширения «Softcard» с дополнительным процессором Z80 для запуска системы CP/M и программ для неё.

Менее интересными примерами являются математические сопроцессоры, применяемые совместно с Intel-процессорами, вплоть до Intel 80386 (в 80486 реализовали инструкции с плавающей точкой, так что потребности в сопроцессоре отпала). Отдельно упомянем графические процессоры, обычно необходимые для расчета 3D-графики – это оказывается быстрее, чем вычисления на центральном процессоре, – и иногда используемые для математических вычислений.

Трансляция кодов

Помимо предназначенных для компиляции и интерпретации (объединяемых общим понятием «трансляция»), существуют компиляторы и интерпретаторы, которые служат для перевода кода с высокоуровневого языка на низкоуровневый либо непосредственно для его исполнения. Рассмотрим способы исполнения машинного кода на платформе, которая работает на другом наборе инструкций, за исключением аппаратной виртуализации.

Эмуляция

По сравнению с аппаратной виртуализацией (например, виртуальный режим процессора 8086), когда код тем не менее исполняется на настоящем процессоре, эмуляция программно воспроизводит аппаратный процессор. Инструкции эмулируемой машины не передаются на исполнение процессору, а обрабатываются программно – в результате фактически одна команда эмулируемого процессора порождает несколько команд на настоящем процессоре.

Программа-эмулятор обладает довольно сложной архитектурой, так как для выполнения машинного кода ей требуется имитировать и память, и устройства ввода/вывода. Составной частью эмуляции платформы является эмуляция процессора, которая схожа с процессом интерпретации. В общем случае эмулятор процессора – это своего рода интерпретатор.

Эмуляция применяется для сохранения наследия предыдущих эпох (существуют эмуляторы старых оригинальных компьютеров, таких, как ZX-Spectrum, PDP-11, Сетунь, игровых приставок, NES, SNES), а также для разработки программного обеспечения для других платформ. Интересно, что, когда представители Microsoft заключили контракт на поставку интерпретатора Basic для компьютера Альтаир, разработчики даже не обладали подобным компьютером. Пол Аллен написал эмулятор Альтаир на имеющемся PDP-11, используя всего лишь документацию для Альтаир.

Эмулятором является известный DOSBox, который не использует виртуальный режим 8086, а самостоятельно эмулирует инструкции процессора 8086. Точнее, DOSBox эмулирует и операционную систему DOS, и 8086-процессор. Это позволяет, с одной стороны, не зависеть от архитектуры процессора, с другой – исполнять код на 64-битных процессорах, которые не поддерживают режим 8086.

DOSEMU для Linux позволяет 32-битным машинам запускать DOS-код в виртуальном режиме (системный вызов vm86()) либо использовать собственный эмулятор 8086 (для 64-битных машин).

В MAC OS при переходе с аппаратной платформы Motorola 68k на PowerPC для совместимости использовался эмулятор 68k. В дальнейшем при переходе MAC OS с процессоров PowerPC на Intel вновь понадобился эмулятор, на этот раз Rosetta – эмулятор PowerPC, а также использование механизмов API (см. далее), позволяющих использовать платформонезависимый запуск приложений.

Для эмуляции современных процессоров используется QEMU. Он позволяет эмулировать Intel, AMD-процессоры, а также ARM, MIPS, SPARC, PowerPC и т.д., и частично содержит компоненты, относящиеся не к эмуляции, а к виртуализации (использование KVM и механизмов аппаратной виртуализации при выполнении).

Примечательно, что эмуляторы могут быть написаны и на интерпретируемых языках (например, эмулятор Сетунь).

Понятие эмуляции включает в себя не только эмуляцию процессора. Существуют эмуляторы технологических устройств и оборудования, которые позволяет осуществлять обучение, исследования и отладку.

Перспективные аппаратные платформы, ещё не реализованные физически, также исследуются с помощью механизмов эмуляции, что позволяет разрабатывать алгоритмы и программы. Так, исследования по квантовым вычислениям возможны отчасти и потому, что реализованы программные эмуляторы/симуляторы квантовых компьютеров.

CISC- и RISC-процессоры

Современные процессоры тоже работают по подобному принципу: то, что исполняет процессор, также транслируется в набор низкоуровневых инструкций микрокода процессора, доступа к которому мы не имеем.

Процессоры делятся на:

- CISC-процессоры (Complex Instruction Set Computer) с полным набором команд – отличаются нефиксированной длиной команды;
- RISC-процессоры (Reduced Instruction Set Computer) с сокращённым набором команд. Фактически речь идёт не о сокращении набора команд (их может быть и больше), а о более низкоуровневых командах, которые быстро выполняются, но обладают более ограниченными функциями. Длина команд RISC-процессоров, как правило, фиксирована.

Машинный код современных процессоров (Intel, AMD) – это, как правило, CISC-код, кодируемый процессором в инструкции RISC-кода, который уже непосредственно исполняется на машине.

Трансляция вызовов

Если ПО предназначено для работы на той же аппаратной платформе, но стоит вопрос о её запуске на другой ОС, возникает ситуация, когда требуется транслировать системные вызовы программы в аналогичные вызовы соответствующей ОС.

Механизм поддержки API другой ОС

К таким механизмам можно отнести wine, позволяющий запускать Windows-приложения на Linux-машине. Wine осуществляет трансляцию системных вызовов WinAPI в системные вызовы Linux, причём для отображения графических приложений по факту используется X-Server, а при необходимости будут подгружены Mono (реализация среды .NET), движок браузера Gecko и т.д.

Возможна обратная ситуация: Linux-подсистема в Windows 10 (сравните с Wine: как выглядит имитация диска C: на Linux машине под Wine, как выглядит имитация корня / в приложении под Linux-подсистему). Тем не менее для запуска графических приложений тоже понадобится X-Server – например, X-Ming – а также Wine.

Прежде чем запускать приложение через трансляцию вызовов, лучше убедиться, нет ли для данной системы «родного» решения. Так, можно в учебных целях запустить Notepad++ через Wine в Linux, но надо иметь в виду, что реализация Notepad++ есть и для Linux, и такое решение будет удобнее. То же самое касается и Microsoft Office: его можно запустить через Wine, но можно запустить и Microsoft Word Online, используя браузер (см. выше об удаленном доступе).

Интересен HX DOS Extender, который не только является расширителем DOS для доступа к защищённому режиму (как мы помним, все расширители DOS реализуют DPMI-сервер, так что с оговоркой в этом пункте можно упомянуть и их), но, более того, позволяет запускать Windows PE-приложения: в частности, QEMU, DOSBox и даже старые игры, в том числе графические.

Нативная поддержка нескольких API

Несколько API могут быть реализованы и в рамках основной поставки операционной системы: так Windows 98 реализовывало запуск DOS-приложений (впрочем, благодаря механизму виртуального процессора 8086). Операционная система OS/2 позволяла запускать DOS и Windows 3.11 приложения.

Microsoft XP (развитие OS/2 и NT) позволяло запускать приложения Windows 95/98 (совсем другая операционная система) благодаря поддержке Win32, реализованного в Windows 9x.

При переходе с MAC OS 9 на MAC OS X (результат дальнейшего развития UNIX-подобной системы NextSTEP) в целях совместимости было реализовано API Carbon, позволяющее запускать программы на обеих системах. В качестве подобного механизма использовалась и JAVA – более того, поддержка JAVA мобильными устройствами позволяет использовать такой интерфейс для платформонезависимого запуска приложений.

Существует проект Longene – особого ядра Linux, включающего поддержку системных вызовов Windows.

Поддержка файловых систем

Для поддержки служат драйверы, модули ядра для файловых систем: это драйверы NTFS для Linux и DOS, драйверы VFAT для DOS и Linux.

Несмотря на то, что обращение к системе происходит через системные вызовы запущенной операционной системы, файлы создаются/читаются в формате файловой системы, разработанной для другой ОС.

Аналогично приложениям, которые могут быть запущены на машине с родной ОС, а доступ к ним будет осуществляться с другой, могут функционировать файловые системы. Это обеспечивается сетевым доступом к файловым системам (SMB, NFS). Ту же цель преследуют такие сервисы, как DROPBOX, Onedrive, Google Disk, Yandex Disk (в какой-то степени это относится к использованию удаленного доступа – см. выше).

Реализация ОС

Отдельно отметим реализацию всей среды ОС для запуска в ней написанных программ. Примером могут служить и вышеупомянутые эмуляторы DOSBox, которые не требуют DOS (однако они эмулируют и процессор). Тем не менее, если Wine – интерфейс под Linux, эмулирующий отдельный слой, существуют и проекты по воссозданию DOS: FreeDOS – свободная реализация DOS, ReactOS – попытка реализации всех функций Windows XP. Среди прочего, разработчики ReactOS сотрудничают с разработчиками Wine, а в разработке используются наработки Wine, Samba (для доступа к SMB), NTFS-3G (драйвер для Linux файловой системы NTFS).

Более того, сам проект GNU/Linux в известной мере родился как свободная реализация среды UNIX – правда, на более высоком уровне, без осуществления совместимости ядра.

Отдельно упомянем Longene – проект по созданию объединённого ядра Linux (англ. Linux Unified Kernel, Longene), рассчитанного на бинарную совместимость приложений и драйверов устройств, используемых в Microsoft Windows и Linux, без использования виртуализации или эмуляции. Longene использует системные вызовы Windows и Linux и их соответствующие таблицы. Приложения Windows могут вызывать программное прерывание «int 0x2e», в то время как приложения Linux используют табличный вызов через «int 0x80».

Виртуализация

Виртуализация позволяет гостевой операционной системе выполнять код на процессоре хоста. Такой способ более производителен, чем эмуляция, и позволяет добиться более корректной работы программ, чем трансляция вызовов API, благодаря использованию оригинальной операционной системы.

Существуют несколько подходов к виртуализации.

Динамическая трансляция

Код гостевой машины выполняется на хост-машине, но при этом код проверяется на безопасность гипервизором и заменяется на безопасный. Это отчасти похоже на эмуляцию, но соотношение кода гостевой машины и хост-машины составляет практически один к одному. Гостевая программа не знает, что выполняется в виртуальной машине. Для разделения кода гостевой машины и хоста на 32-битных системах используется сегментная адресация памяти.

Такой подход используется в VMWare. VirtualBox также использует динамическую трансляцию.

Аппаратная виртуализация

Виртуальный 8086

Виртуальный 8086 использовался в MS DOS (при переносе ядра в расширенную память менеджером EMM386), в Windows 3 и Windows 95/98 для запуска DOS-приложений. Процессор аппаратно запускал виртуальные DOS-машины, при этом самостоятельно исполняя код в защищенном режиме. В 64-битных процессорах виртуальный режим 8086 не поддерживается, поэтому такие системы, как DOSBox, вместо виртуализации используют эмуляцию.

Intel-VT и AMD-V

Intel-VT и AMD-V – современные методы поддержки виртуализации, защищающие в том числе и устройства ввода-вывода. Механизмы работы у них разные. Обе технологии поддерживаются виртуализациями XEN и KVM.

KVM

KVM (Kernel-based Virtual Machine) поддерживается на уровне ядра операционной системы. Она состоит из загружаемого модуля ядра (kvm.ko), предоставляющего базовый сервис виртуализации, процессорно-специфического загружаемого модуля kvm-amd.ko либо kvm-intel.ko и компонентов пользовательского режима (модифицированного QEMU). Компонент ядра, необходимый для работы KVM, включён в основную ветку ядра Linux начиная с версии 2.6.20. KVM также портирован на FreeBSD как модуль ядра. Программа, работающая в пространстве пользователя, использует интерфейс /dev/kvm для настройки адресного пространства гостя виртуальной машины, через него же эмулирует устройства ввода-вывода и видеоадаптер. KVM позволяет виртуальным машинам использовать немодифицированные образы дисков QEMU, VMware и других, содержащих операционные системы.

Для управления KVM-машинами через веб-интерфейс может использоваться ProxMox.

Виртуализация на уровне ОС

Изоляция процессов: chroot для ФС и cgroups для процессов

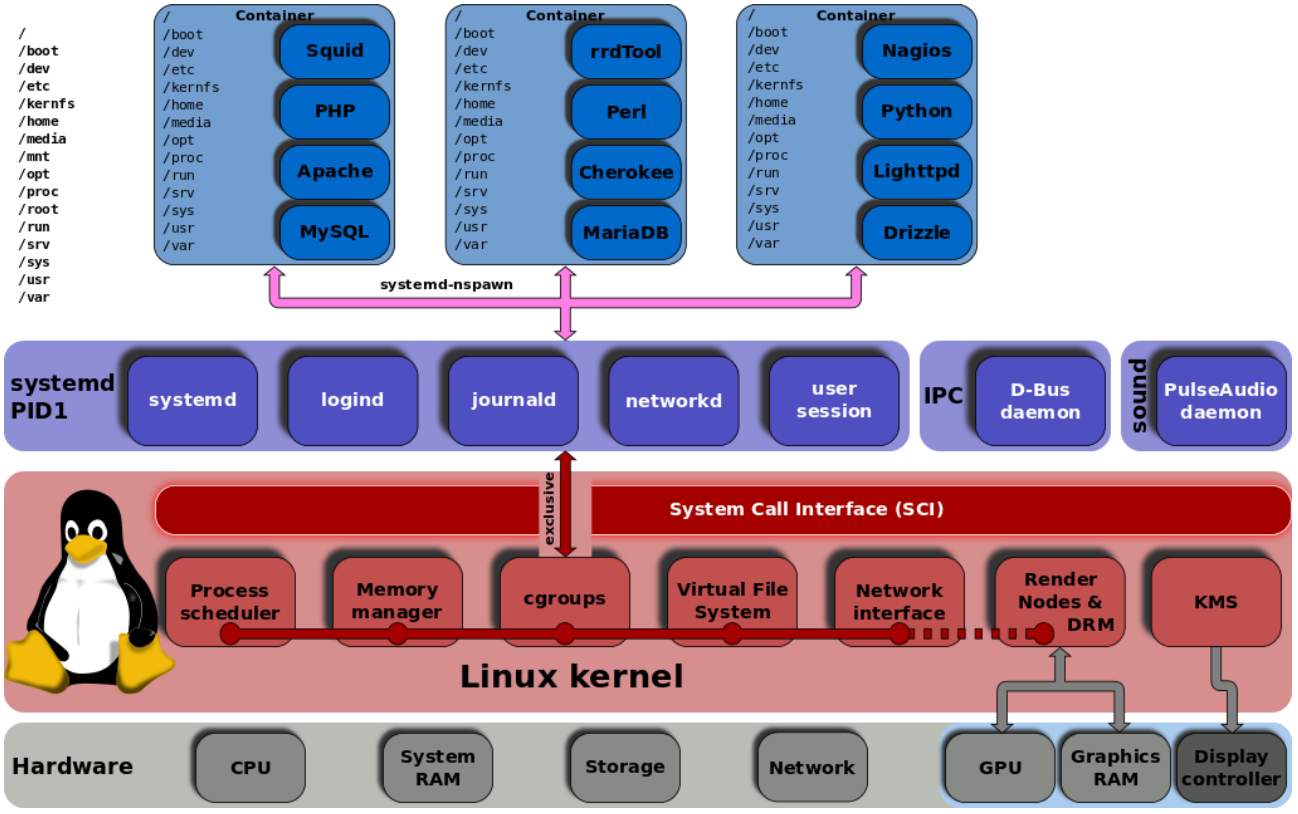
Простейший способ ограничения, основанный на подмене корня VFS некоторой директорией, – chroot.

Иерархия выбранной директории будет восприниматься запущенным приложением как корневая иерархия VFS. Для полноценной работы необходимо создать нужные файлы виртуальных файловых систем внутри chroot (файлы устройств и т.д.).

Работа в chroot является рекомендуемой для работы DNS-сервера Bind и ряда других приложений.

С помощью chroot можно создать «песочницу» для запуска браузера.

Механизм cgroups, добавленный в ядро Linux, позволяет изолировать и группы процессов.



Изоляция cgroups

Автор: Shmuel Csaba Otto Traian, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=31700051

Контейнерная виртуализация

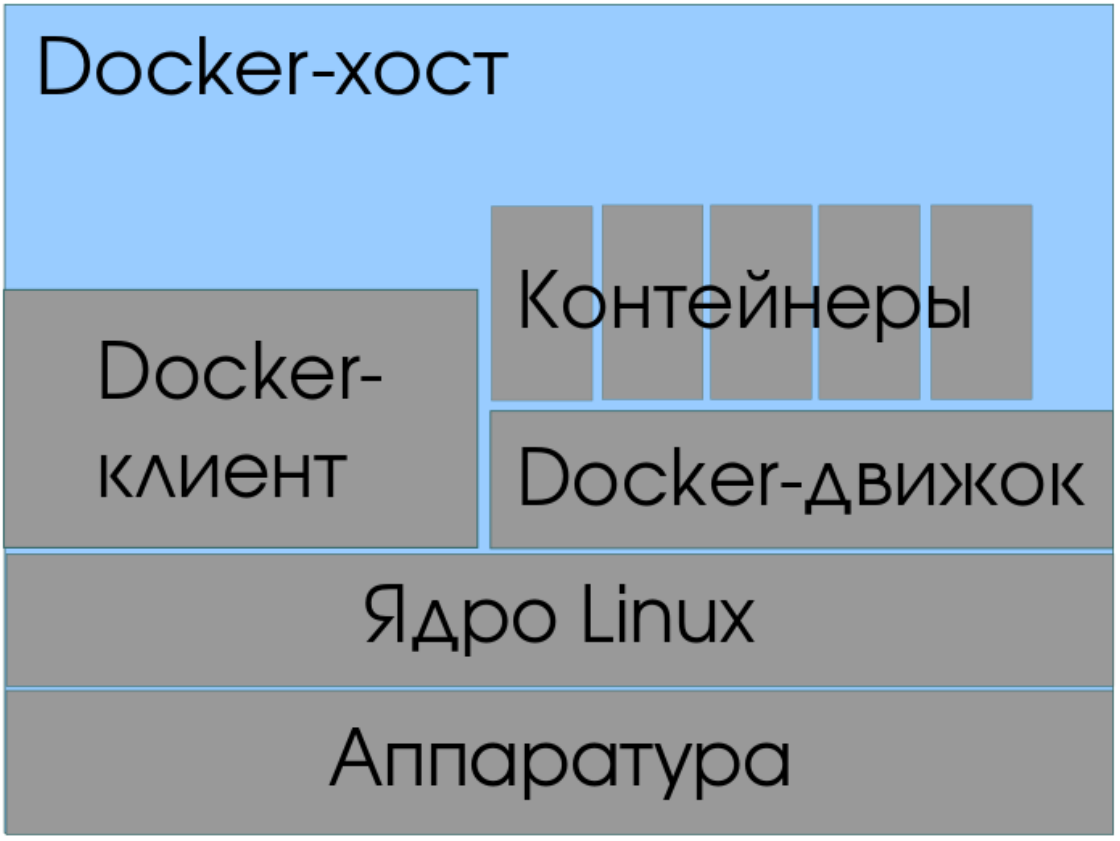
Если изолировать процессы и создать иерархию файловой системы на диске, внутри операционной системы можно запускать несколько «виртуальных операционных систем». Полноценная реализация такого подхода позволяет запускать на базе ядра Linux другие версии Linux, использующие то же самое ядро. Ограничением будет невозможность из гостевой машины запускать и удалять модули ядра – в остальном функциональность в этом случае та же, что и на хост-машине.

OpenVZ, LXC, Docker

Примерами контейнерной виртуализации служат OpenVZ и LXC (Linux Container). Обе технологии основаны на cgroups и позволяют запускать контейнеры с Debian, Ubuntu, Centos в качестве гостевых.

ProxMox также позволяет управлять контейнерами LXC через веб-интерфейс.

Ещё одним инструментом виртуализации является Docker. Изначально рассчитанный на возможности LXC, сейчас Docker использует собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux, – libcontainer.



Docker Автор: Роман Сузи - собственная работа, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=46127762

Паравиртуализация

Если для виртуализации требуется изменение ядра гостевой операционной системы, такой подход называется паравиртуализацией. Вместо обращения к оборудованию гостевая система использует системные вызовы операционной системы хоста. В качестве системы, использующей паравиртуализацию, часто упоминают XEN.

Похожим образом работает coLinux (но только для 32-битных систем; сейчас проект остановлен).

Практическое задание

Для работы понадобится установленная в VMWare или VirtualBox система GNU/Linux.

1. Установить и запустить Windows-приложение с помощью Wine на Linux. Сложность определяется программой, которую вы хотите запустить (от Notepad++ до Microsoft Office).
2. * Создать виртуальную машину с помощью KVM.
3. * Установить поддержку Linux для Windows 10 и XMing, запустить Linux-приложение.
4. * Запустить приложение (например, Chromium) в песочнице.
5. * Установить и настроить систему с Proxmox.
6. * Установить и настроить систему с Docker.

Примечание. Задания со звёздочкой предназначены для тех, кому требуются более сложные задачи.

Дополнительные материалы

1. https://geekbrains.ru/events/474 — Сергей Сизов. Знакомимся с основными возможностями Docker
2. https://ru.wikipedia.org/wiki/Транслятор
3. https://ru.wikipedia.org/wiki/Раскрутка_компилятора
4. https://ru.wikipedia.org/wiki/QEMU
5. https://mkdev.me/posts/osnovy-virtualizatsii-i-vvedenie-v-kvm
6. http://citforum.ru/operating_systems/virtualization/part2.shtml

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Эмулятор Сетунь http://trinary.ru/projects/setunws
2. http://s.arboreus.com/2006/11/blog-post_16.html
3. https://ru.wikipedia.org/wiki/Подсистема_для_приложений_на_базе_UNIX
4. https://ru.wikipedia.org/wiki/Windows_Subsystem_for_Linux
5. HX DOS Extender http://old-dos.ru/index.php?do=show&id=3788&mode=files&page=files
6. Longene https://ru.wikipedia.org/wiki/Объединённое_ядро_Linux
7. https://ru.wikipedia.org/wiki/CP/M
8. https://en.wikipedia.org/wiki/Z-80_SoftCard
9. https://ru.wikipedia.org/wiki/Cgroups
10. https://ru.wikipedia.org/wiki/Docker
11. http://www.ibm.com/developerworks/ru/library/l-virtualization-colinux/index.html