Тезисы, мысли

Есть нативные файловые системы. В Linux их много, на любой вкус и цвет. В Windows мало. ПО, работающее с различными файловыми системами. Пример сервера, совместно используемый диск с файловой системой. Для одного нативная, для другого нет. Требуется доступ к ненативным файловым системам из приложений операционной системы.

Цель

Текущее положение дел. Драйверы. Есть, но не для всех. Портировать - очень долго и сложно. Лучше использовать нативные драйвера.

Создание окружения для нативных драйверов. Эффективно и тяжело - реализация окружения ядра одной ОС в другой. Просто и результативно - виртуальная машина.

Архитетура. Libguestfs.

? Libguestfs ?

Задачи

? Libguestfs ?

изучить внутреннее устройство

портировать на windows

портирование системы сборки

локализация кода внутри кроссплатформенных интерфейсов

реализовать разделяемую память

сравнить производительность

портирование на Cygwin. Позволил понять, насколько всего много

портирование с использованием visual studio

портирование на Mingw

? замена протокола ?

Портирование системы сборки для Windows. Autotools. Отключение зависимостей. Libtool.

Тесты

Реализация общей памяти.

QEMU, ivshmem

Крутые тесты

Файловая система представляет собой способ организации, хранения и именования данных на носителях информация. Без наличия файловой системы, вся информация размещалась бы на устройстве хранения данных произвольным образом, в виде одного непрерывного блока, без возможности определения, в каком месте заканчивается один блок информации и начинается другой. Файловые система использует определенный набор правил и структур данных для контроля за тем, где и как файлы сохраняются на физическом устройстве и извлекаются обратно для использования в работе приложений.

Каждая файловая система изначально поддерживает работу с определенным набором файловых систем. Такие файловые системы называются нативными для данной операционной системы. в зависимости от рассматриваемой операционной системы Количество нативно поддерживаемых файловых систем сильно варьируется. Так, например, основное ядро операционной системы Linux поддерживает широкий спектр разнообразных файловых систем, в то время как в операционной системе Windows включена поддержка сравнительно небольшого количества файловых систем. Рассмотрим их более подробно.

Существуют целые классы программного обеспечения, которым в силу своей специфики требуется работать с широким количеством файловых систем. К ним относятся:

* Системы резервного копирования и восстановления данных
* Системы защиты данных, антивирусное программное обеспечение
* Менеджеры жестких дисков и разделов
* Файловые менеджеры – программы для просмотра и управления содержимым файловой системы, реализующие возможность копирования, перемещения, удаления файлов и каталогов внутри файловой системы

Также зачастую при решении той или иной задачи может быть наиболее выгодно использовать конкретную файловую систему, например, из-за предоставляемых её определенных преимуществ, даже если выбранная файловая система не является нативной в используемой операционной системе. При этом не всегда удобно отказаться от используемой операционной системы в пользу той, в которой требуемая файловая система поддерживается в качестве нативной. -> драйвер до цели

Файловая система является ключевым компонентом любой операционной системы..

Текуще положение дел -> драйвер до цели

Цель:

Предоставить приложениям операционной системы Windows возможность работать с файловыми системами, нативными для операционной системы Linux и поддержка которых изначально отсутствует в Windows.

Для работы с файловой системой операционной системе необходимо предоставить определенный программный компонент – драйвер файловой системы. Драйвер – программный модуль, который умеет интерпретировать внутренние структуры файловой системы и предоставляет приложениям удобный иерархический вид её содержимого.

Сложность файловых систем

Откуда взять драйвер файловой системы Linux для Windows?

Портировать драйвер файловой системы Linux: взять и написать свой

Использовать исходные нативные драйвера файловой системы, создав в Windows специальное окружение операционной системы Linux

|  |  |
| --- | --- |
| Исходнык драйвера Linux  Качетсвенные , надежные  Разрабатываются ведущими IT-компаниями вместе с файловыми системами  Тестриуемые (прежду чем попасть в основное ядро Linux – долгий этап тестирования и исправления багов)  Поддержка со стороны разработчиков  Открытый доступ | Экспериментальные  Отсутсвие поддержки  Неполная поддержка всех возможной файловой системы, недоработанные  Ненадежные, используем на свой страх и риск  Тяжелая разработка  Для каждой файловой системы – отдельная реализация драйвера |

Портироватьк каждый драйвер – не подходит. Создадим окружение Linux и будем использовать нативные драйвера.

Архитектура решения

Модель работы разрабатываемой библиотеки представляет собой клиент-серверное взаимодействие, в котором библиотека, предоставляющая приложению определенный программный интерфейс, выступает в роли клиента, а запускаемая ею виртуальная машина

Разрабатываемая библиотека имеет клиент-серверную архитектуру.

Библиотека, работающая на стороне приложения, в отдельном (дочернем) процессе запускает виртуальную машину, внутри которой загружается дистрибутив Linux. В состав загружаемого дистрибутива входят нативные драйвера файловых систем и определенный набор вспомогательных утилит для работы с ними. После загрузки операционной системы Linux в ней запускается специальный контролирующий демон, способный выполнять команды, полученные от библиотеки, по взаимодействию с требуемыми файловыми системами. Таким образом, библиотека, предоставляющая приложению определенный программный интерфейс по работе с файловой системой, выступает в роли клиента, а запущенная ею виртуальная машина с загруженной внутри операционной системой Linux и демоном выступает в роли сервера.

При запуске виртуальной машины к ней присоединяется физическое устройство (либо образ диска) с файловой системой, доступ к которой необходимо организовать. Внутри виртуальной машины присоединенное устройство (либо образ диска) выглядит как некоторое устройство хранения данных (например, как жесткий диск) с нативной для операционной системы Linux файловой системой.

Благодаря наличию в операционной системе Linux необходимых драйверов, работающий внутри виртуальной машины демон монтирует это устройство, после чего способен взаимодействовать с монтированной файловой системой и осуществлять требуемые операции чтения и записи в соответствии с командами, полученными от библиотеки, работающей на стороне приложения.

Стоит заметить, что в случае, если вместо реального физического устройства к виртуальной машине присоединяется образ диска, то виртуальная машины сама транслирует доступ ядра Linux к «физическому» устройству в доступ к образу диску.

Библиотека, выступающая в роли клиента, «общается» с демоном с помощью механизма удаленного вызова процедур (RPC – Remote Procedure Call). Библиотека отправляет демону определенные команды для осуществления операций чтения и записи файловой системы и получает обратно от демона результаты их выполнения.

Рассмотренный механизм работы позволяет осуществить доступ к любой нативной файловой системе операционной системы Linux, расположенной на реальном физическом устройстве либо образе диска, поддержка которой включена в используемый в виртуальной машине дистрибутив Linux.

**Проект libguestfs**

Библиотека libguestfs.

Задачи.

Выбор виртуальной машины.

Портирование на Cygwin. Устранение зависимостей. Кроссплатформенное сетевое взаимодействие. Результаты – proof of concept, конечно медленее.

Портирование на Windows. Protobuf

Портирование на MinGW. Реализация кроссплатформенных интерфейсов.

Реализация разделяемой памяти. Qemu, ivshmem. Proto.c , shared\_memory

Собственная система сборки с Autotools.

Дальнейшая работа: UML

Для решения задачи организации доступа к нативным файловым системам операционной системы Linux в операционной системе Windows был выбран подход, заключающийся в использовании нативных драйверов Linux с эмуляцией окружения операционной системы Linux в операционной системе Windows посредством использования виртуальной машины. Кратко перечислим основные преимущества данного подхода по сравнению с остальными (рассмотренными ранее):

* отсутствие затрат на разработку драйвера для каждой из файловой систем, доступ к которым необходимо организовать
* сложность реализации данного подхода никак не зависит от количества файловых систем, возможность работы с которыми нужно предоставить в операционной системе Windows
* нативные драйвера операционной системы Linux реализуют полный доступ к файловой системе, поддерживают все её функции. Обладают высокой производительностью и надёжностью, активно развиваются и поддерживаются разработчиками, лишены ошибок и недоработок, зачастую встречающихся в их портированных аналогах, реализованных сторонними компаниями и отдельными программистами
* драйвера операционной системы Linux, как и сама операционная система Linux, относятся к свободно распространяемому программному обеспечению, следовательно, отсутствуют материальные затраты, связанные с их использованием
* полное переиспользование исходного кода операционной системы Linux, отсутствие затрат на портирование

Одним из главных недостатков данного подхода является существенные потери в производительности, а именно скорости выполнения операций чтения и записи. Также происходит высокое потребление системных ресурсов, связанных с тем, что осуществляется запуск целой виртуальной машины, внутри которой работает полноценная операционная система Linux. Существует несколько способов устранения данных недостатков, о них будет рассказано далее более подробно

Реализация выбранного подхода будет выполнена в виде динамической библиотеки операционной системы Windows, написанной на языке программирования C. Выбор языка C в качестве языка разработки является наиболее целесообразным, поскольку:

Архитектура решения

Реализуем библиотеку, которая предоставляет приложениям определенный программный интерфейс по доступу к файловым системам, расположенным на физическом устройстве или образе диска.

Работа библиотеки представляет основывается на клиент-серверном взаимодействии. Библиотека выступает в роли клиента, виртуальная машина – в роли сервера.

Библиотека в отдельном процессе запускает виртуальную машину. Внутри виртуальный машины запускается урезанный дистрибутив Linux, с необходимы набором драйверов и утилит по работе с файловыми системами. Внутри Linux работает специальный контролирующий демоном, с которым взаимодействует библиотека – отправляет команды по чтению/записи файловых систем. Общение с демоном осуществляется посредством сетевого взаимодействия.

Проект libguestfs – проект компании RedHat, активно развивается с 2009 года. мощная библиотека-набор утилит для доступа и модификации образов дисков виртуальных машин. Предназначена для работы в Linux и на первый взгляд напоминает что-то вроде этого:%

Предоставляемые инструменты позволяют просматривать и редактировать файлы внутри гостевых систем, создавать образы для виртуальных машин, модифицировать таблицу разделов, управлять конфигурационными файлами, переносить «железные» машины в виртуальную среду, переносить виртуальные машины с одного образа на другой, переносить виртуальные машины из образа на железо. Для выполнения своих операций libguestfs не требует прав суперпользователя, что является несомненным преимуществом.

Особый интерес в рамках данной работы представляет реализуемая в рамках проекта бибилотека, написанная на языке C, которо

Особый интерес вызывает реализованная в проекте libguestfs библиотека, написанная на C. Она инкапсулирует в себе базовую функциональность, используемую во всех генерируемых в рамках данного проекта утилитах и инструментах. редоставляет приложениям программный интерфейс для доступа и работы с практически с любыми файловыми системами и образами дисков. Библиотека реализована на языке C.

В своей работе библиотека ипользует как раз ту самую клиент-серверную архитектуру, которая только что рассматривалась на предыдущем слайде. В качестве протокола взаимодействия используется XDR, в качестве виртуальной машины есть возмоность использовать

В качестве виртуальной машины предоставляется возможность использовать qemu и uml. Также есть возможность использовать libvirt либо подключиться к уже запущенному внутри виртуальной машины демону.

Взаимодействие библиотеки с демоном осуществляется с помощью сетевого взаимодействия через сокеты посредством удаленного вызова процедур по протоколу XDR. В большинстве случаев каждому вызову функции, предоставляемой библиотекой приложениям, соответствует отправка определенной команды демону.

Таким образом, задачами данного проекта являются

Выбор архитектуры

Выбор виртуальной машины

Портирование Libguestfs на Windows

Портирование системы сборки

Кросс-платформенный протокол взаимодействия клиента и сервера

Локализация кода, зависимого от типа операционной системы внутри реализаций кросс-платформенных интерфейсов

Выбрать подходящую виртуальную машину для запуска Linux

Сравнить производительность с Linux

Выбор QEmu

В качестве виртуальной машины было решено использовать QEMU, поскольку:

Виртуальна машина с открытым исходным кодом

Эмулирует большое количество платформ

Работает в пространстве пользователя

Имеется возможность использовать в Windows. Есть дистрибутивы для запуска в Windows

Портирование на Cygwin

Много сторонних зависимостей. Минимальное количество изменений, proof of concept, изменения в запуске виртуальной машины, изменения в сетевом взаимодействии.

Результаты

Тестировалась производительность библиотеки. Сравнивалось врем выполнения операций чтения и записи. Cygwin очевидно работает медленней, зато работает.

Нативное портирование

Реализация кроссплатформенных интерфейсов: conn-socket, pipe, create process – fork

Реализация Windows-аналогов некоторых функций на основе предоставленной в сети интернет документации

Замена XDR на Protobuf.

Реализации конвертеров из Protobuf в XDR

Результаты

Linux XDR Protobuf. Быстрее, но все равно медленнее Linux. Быстрее, чем Cygwin

Портирование с MinGW  
Интеграция в систему сборки Autotools.

Реализация общей памяти

Как уже упоминалось ранее, всё взаимодействие с демоном осуществляется посредством сетевого взаимодействия. Все данные, передаваемые демону и получаемые обратно от него, передаются через сокеты. В частности, команды и результаты их выполнения также передаются через сокеты.

Например, в случае выполнения операции чтения содержимое файлов пересылается частями, каждая часть размеров максимум в 8 КБ. Учитывая, что пересылка каждой из частей сопровождается дополнительными издержками, связанными с организацией сетевого взаимодействия, такими как копирование данных для пересылки, упаковка данных в пакет, передача пакета, извлечение данных и т.д., время чтения файла размеров несколько мегабайт и выше существенно возрастает, скорость сильно замедляется и далеко от нативной. То же самое касается и операции записи.

Собственно, для выполнения операции операций чтения и записи было принято решение организовать передачу файлов с помощью разделяемой памяти.

Использование разделяемой памяти позволяет:

Избавиться от лишнего копирования данных, происходящий при их передаче через сокеты.

Собственно, передавать файл частями гораздо большего размера, порядка несколько мегабайт и даже гигабайт, в зависимости от доступной оперативной памяти.

Доступ к разделяемые файлы можно получить через его имя.

Реализация доступа к разделяемой памяти через ivshmem

Внутри qemu разделяемая память выглядит как память специального pci-устройства. Реализуем драйвер для этого устройства. Используем драйвер для работы с устройством. Существует реализация для Linux (nahanni project). Модуль ivshmem. Портирован для использования в Windows как ivshmem-windows.c

Результаты

Очевидно, что быстрее теперь. Но скорость далека от нативной