Добрый день, тема моей работы звучит следующим образом: «Работа с файловыми системами в операционной системе Windows с использованием драйверов операционной системы Linux»

Начну с того, что напомню, зачем нужна ФС. Файловая система представляет собой способ организации, хранения и именования данных *и* определяет, где и как данные сохраняются на физическом устройстве и по каким правилам извлекаются обратно для использования в приложениях.

*Файловая система является ключевым компонентом любой операционной системы.* Для каждой ОС сущ-ет определенный набор ФС, поддержка которых изначально включена в ядро ОС. Такие файловые системы мы будем называть нативными. в случае Linux нам предоставляется возможность использовать широкий спектр разнообразных ФС. Windows таким качеством похвастаться не может.

При это стоит помнить, что сущ-ет целые классы программного обеспечения, к-ые в силу своей специфики нацелены на работу с широким количеством ФС. К ним можно отнести системы резервного копирования и восстановления данных, антивирусное ПО, Менеджеры жестких дисков и разделов, файловые обозреватели и т.д. любое ПО, которому требуется работать с ненативной для ОС файловой системой. В случае, если операционная система не поддерживает файловую систему, то проблему организации доступа должна решать сама программа, причем чем с большим количеством файловых систем программа способна работать, тем она более привлекательна для пользователя.

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? Есть 2 основных подхода.

Мы можем посмотреть на исходные драйвера Linux и портировать их, написать свою реализацию драйверов для работы Windows. В силу сложного устройства совеременных ФС, данный подход очень трудоемкий, сложный и дорогой по времени и долгий.

Тогда мы можем взять уже реализованные кем-то драйвера. К сожалению, на сегодняшний день таких драйверов мало, они поддерживают ограниченное кол-во файловых систем ОС Linux. Зачастую такие драйвера носят чисто экспериментальный характер, реализуют лишь ограниченный функционал ФС (обычно только чтение, изредка - запись).

Второй же подход заключается в том, что мы берем драйвера, написанные для Linux, и используем их, ничего не изменяя. Эти драйверов реализует полный доступ к ФС, обладают высокой надежностью и эффективностью, потому как разрабатываются и тестируются огромным Linux-сообществом. НО, для их использования мы должны реализовать в Windows окружение Linux. Сделаем это с помощью виртуальной машины.

Таким образом, архитектура выбранного решения выглядит следующим образом. Реализуется библиотека, предоставляющая Windows-приложению программный интерфейс для доступа к ФС. В своей работе библиотека использует клиент-серверное взаимодействие. В отдельном процессе запускается виртуальная машина, внутри которой загружается урезанный дистрибутив Linux с необходимыми драйверами и утилитами *для работы с ФС*. При запуске виртуальной машины к ней присоединяется диск (либо его образ) с ФС, доступ к которой нужно организовать. После загрузки Linux запускается специальный контролирующий демон, который монтирует файловую систему на присоединенном устройстве, после чего способен выполнять команды по чтению/записи, полученные от библиотеки, Общение с демоном осуществляется посредством сетевого взаимодействия.

Стоит сказать, то очень вовремя был обнаружен крайне полезный в рамках данной работы проект компании RedHat – проект libguestfs.

Данный проект представляет собой набор утилит для доступа и модификации образов дисков виртуальных машин. Предоставляемый функционал поистине велик и на первый взгляд напоминает что-то вроде этого:%

*Предоставляемые инструменты позволяют делать очень много вещей связанных с редактированием образом, просмоор содержимого, создания, модификации, переноса виртуальных машин на железо и обратно и т.д.*

Но нас больше интересует библиотека, реализуемая в рамках данного проекта.

Она инкапсулирует в себе базовую функциональность и в своей работе использует рассмотренную только что архитектуру*, используемую во всех генерируемых в рамках данного проекта утилитах и инструментах. предоставляет приложениям программный интерфейс для доступа и работы с практически с любыми файловыми системами и образами дисков. Библиотека реализована на языке C.*

В качестве виртуальной машины предоставляется возможность использовать qemu, qemu-kvm, uml. Для взаимодействия используется удаленным вызов процедур с помощью протокола XDR

Итак, задачи данной работы были сформулированы следующим образом:

* Выбрать виртуальную машину для запуска Linux в Windows
* Портировать исходный код библиотеки libguestfs на Windows
* Улучшить производительность работы библиотеки и сравнить её исходной, которая на Linux
* *Реализовать передачу файлов через разделяемую память*
* *Сравнить производительность с исходной (с той, которая на Linux)*

В качестве виртуальной машины наиболее выгодным было использование QEMU, поскольку: Это свободно распространяемая виртуальная машина, имеется возможность её сборки для работы в Windows

И поскольку в libguestfs уже включена поддержка QEMU

То почему бы этим не воспользоваться?

Портирование библиотеки началось с выбора подходящей среды разработки,

Конечно же, было желание использовать «родной» для Windows *интегрированную среду разработки* Visual C++, *поставляемую как часть Visual Studio*. К сожалению, данный инструмент не подошел по ряду причин:

* Поскольку Libguestfs изначально нацелен для работы Linux и компилируется gcc, в коде библиотеки используется специальные расширения языка компилятора, аналогов которым в Visual C++ нет
* Используя Visual С++ у нас не получится интегрироваться в систему сборки libguestfs

Все этого диктовало необходимость использования такого окружение, такого набора инструментов, в рамках которых предоставляется достойный Windows-аналог gcc-компилятора и желательно поддерживается система сборки GNU. Вообщем, проекты Cygwin и MinGW.

Наиболее подходящей средой разработки оказался Cygwin и MinGW.

Первоначально было реализована возможность запуска библиотеки в Cygwin. Портирование библиотеки началось с реализации возможности её запуска в Cygwin-е.

поскольку Cygwin предоставляет практически полное POSIX окружение и позволяет запускать Unix-приложение в Windows практически без изменения их исходного кода. Это оказалось верным и в случае libguestfs.

Однако то факт, что после установки всех необходимых пакетов размер Cygwin-овского дистрибутива стал неприлично велик (порядка 1 ГБ), и желание устранить зависимость портируемой библиотеки от Cygwin-овского окружения, послужило причиной портировать libguestfs нативно используя инструменты проекта MinGW.*компилятор и систему сборки, предоставляемую в рамках проекта MinGW.*

В рамках нативного портирования весь платформозависимый код был локализован и скрыт внутри простых кроссплатформенных интерфейсов, оформленных в виде небольшой библиотеки. Также потребовалось реализовать Windows-аналоги некоторых Unix-функции, специфичных для Unix. Собственно, ряд модулей Libguestfs были полностью переписаны для работы в Windows.

Значительна часть усилий была потрачена на разрешений внешних зависимостей от сторонних библиотек – поиск и сборку вспомогательных библиотек, от которых зависит libguestfs. В конечном итоге, процесс сборки получившейся портированной библиотеки был интегрирован в уже существующую систему сборки libguestfs.

Результаты тестирования производительности libguesfs *с использованием Cygwin и MinGW* показаны на слайде*.* Видно, что по сравнению с Linux скорость работы библиотеки упала, но, например, отказ от использования Cygwin *всё же* оказал некоторый положительный эффект на производительность.

Да, Поскольку наблюдаемая скорость чтения и записи всё равно далека от той, которую мы могли получить, работая с файловой системой непосредственно, нативно, без использования промежуточного звена в виде виртуальной машины, был предпринят ряд оптимизаций (мер) увеличить скорость выполнения операций чтения и записи, а именно

* Замена протокол взаимодействия: XDR на Protobuf
* Использование разделяемой памяти

Замена XDR на Protobuf не дала особого эффекта в увеличении производительности.

А вот использование разделяемой памяти оказалось более успешным. Изначально в Libguestfs все данные команд и результаты их выполнения передавались через сокеты. Собственно, файлы передавались частями по 8 КБ в каждом пакете. *Учитывая все дополнительные издержки связанные с пересылкой данных через сокеты, мы получаем очень медленную скорость чтения и записи, особенно больших файлов.* Использование разделяемой памяти позволило передавать файлы частями гораздо большего размера, а также избавиться от лишнего копирования происходящего при передаче через сокеты.

При реализации общей памяти особое внимание стоило уделить организации доступа к ней со стороны демона, который работает внутри виртуальной машины и ничего не знает о windows и его файловых отображениях.

Доступ был реализован с помощью представления памяти в виде особого PCI-устройства.

А В дистрибутив Linux был дополнительно включен специальный драйвер для работы с этим устройством.

Полученные после этого результаты производительности очень вдохновляют. Операции чтения и записи выполняются в разы быстрее, чем раньше, что не может не радовать.

Резюмируя всё вышесказанное, подведем итог результаты работа.

Итого, в рамках данной работы была реализованы библиотека, позволяющая приложениям Windows работать с нативными для Linux файловыми системами. Для доступа к ФС использовались нативные Linux-драйвера, для работы которых окружение Linux эмулируется с помощью виртуальной машины. И Была добавлена возможность передачи файлов через общую памяти для повышения скорости выполнения операций чтения и записи.

Дальнейшая работа

Портирование и использование UML в качестве виртуальной машины

Портирование всего проекта libguestfs.

UML - вариант ядра Linux, который позволяет запускать линукс в качестве обычного user-приложения. Грубо говоря, это линукс, который исполняется в пространстве пользователя и ожидает увидеть не голое железо, а другой линукс, возможности которого и использует дл своей работы.

Добрый день, тема моей работы звучит следующим образом: «Работа с файловыми системами в операционной системе Windows с использованием драйверов операционной системы Linux»

Начну с того, что напомню, зачем нужна ФС. Файловая система *представляет собой способ организации, хранения и именования данных на носителях информация. Она* определяет, где и как данные сохраняются на физическом устройстве и по каким правилам извлекаются обратно для работы в приложениях.

Файловая система является ключевым компонентом любой операционной системы. Для каждой ОС сущ-ет определенный набор ФС, поддержка которых изначально включена в ядро ОС. Такие файловые системы мы будем называть нативными. В зависимости от ОС количество и виды таких ФС сильно варьируется. Так, в случае ОС Linux нам предоставляется возможность использовать широкий спектр разнообразных ФС. ОС Windows таким качеством похвастаться не может.

При это стоит упомянуть, что сущ-ет целые классы программного обеспечения, к-ые в силу своей специфики нацелены на работу с широким количеством ФС. К ним можно отнести системы резервного копирования и восстановления данных, антивирусное ПО, Менеджеры жестких дисков и разделов, файловые обозреватели и т.д. любое ПО, которому требуется работать с ненативной для ОС файловой системой. В случае, если операционная система не поддерживает файловую систему, проблему организации доступа должна решать сама программа и чем с большим количеством файловых систем программа способна работать, тем она более привлекательна для пользователя.

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? Есть 2 основных подхода.

Мы можем посмотреть на исходные драйвера Linux и портировать их, написать свою реализацию драйверов для работы Windows. В силу сложного устройства совеременных ФС, данный подход очень трудоемкий, сложный и дорогой по времени и долгий.

Тогда мы можем взять уже реализованные кем-то Windows-аналоги этих драйверов. К сожалению, на сегодняшний день таких драйверов мало, они поддерживают ограниченное кол-во файловых систем ОС Linux. Зачастую такие драйвера носят чисто экспериментальный характер, поддерживают лишь ограниченный функционал ФС (обычно только чтение, изредка - запись).

Второй же подход заключается в том, что мы берем драйвера, написанные для Linux, и используем их, ничего не изменяя. Эти драйверов реализует полный доступ к ФС, обладают высокой надежностью и эффективностью, потому как разрабатываются и тестируются огромным Linux-сообществом. НО, для их использования мы должны реализовать в Windows окружение Linux. Сделаем это с помощью виртуальной машины.

Таким образом, архитектура выбранного решения выглядит следующим образом. Реализуется библиотека, предоставляющая Windows-приложению программный интерфейс для доступа к ФС. В своей работе библиотека использует клиент-серверное взаимодействие. В отдельном процессе запускается виртуальная машина, внутри которой загружается урезанный дистрибутив Linux с необходимыми драйверами и утилитами *для работы с ФС*. При запуске виртуальной машины к ней присоединяется диск (либо его образ) с ФС, доступ к которой нужно организовать. Внутри виртуальной машины он выглядит как реальное устройство хранения данных ФС, с нативной для Linux. После загрузки Linux запускается специальный контролирующий демон, который монтирует файловую систему на присоединенном устройстве, после чего способен выполнять команды по чтению/записи ФС, полученные от библиотеки, и возвращать результаты их исполнения. Общение с демоном осуществляется посредством сетевого взаимодействия.

Стоит сказать, то очень вовремя был обнаружен крайне полезный в рамках данной работы проект компании RedHat – проект libguestfs.

Данный проект представляет собой набор утилит для доступа и модификации образов дисков виртуальных машин. Предоставляемый функционал поистине велик и на первый взгляд напоминает что-то вроде этого:%

*Предоставляемые инструменты позволяют делать очень много вещей связанных с редактированием образом, просмоор содержимого, создания, модификации, переноса виртуальных машин на железо и обратно и т.д.*

Но нас больше интересует библиотека, реализуемая в рамках данного проекта.

Она инкапсулирует в себе базовую функциональность и в своей работе использует рассмотренную только что архитектуру*, используемую во всех генерируемых в рамках данного проекта утилитах и инструментах. предоставляет приложениям программный интерфейс для доступа и работы с практически с любыми файловыми системами и образами дисков. Библиотека реализована на языке C.*

*Причем в своей работе библиотека использует ту архитектуру, о которой я только что говорил.* В качестве виртуальной машины предоставляется возможность использовать qemu, qemu-kvm, uml. Для взаимодействия используется удаленным вызов процедур с помощью протокола XDR

Итак, задачи данной работы были сформулированы следующим образом:

* Выбрать виртуальную машину для запуска Linux в Windows
* Портировать исходный код библиотеки libguestfs на Windows
* Повысить производительность работы библиотеки и сравнить её исходной, которая на Linux
* *Реализовать передачу файлов через разделяемую память*
* *Сравнить производительность с исходной (с той, которая на Linux)*

В качестве виртуальной машины наиболее выгодным было использование QEMU, поскольку:

QEMU – свободно распространяемая виртуальная машина, имеется возможность использования в Windows

И поскольку в libguestfs уже включена поддержка QEMU

То почему бы этим не воспользоваться?

Портирование библиотеки началось, естественно, с выбора подходящей среды разработки, *окружения, инструментов, с помощью которых будет осуществляться портирование.*

Первым делом, конечно же, было желание использовать «родной» для Windows *интегрированную среду разработки* Visual C++, *поставляемую как часть Visual Studio*. К сожалению, данный инструмент не подошел по ряду причин:

* Поскольку Libguestfs изначально нацелен на Linux и компилируется gcc, в исходной коде библиотеки используется специальные расширения компилятора, аналогов которым в Visual C++ нет
* Используя Visual С++ у нас не получится интегрироваться в систему сборки libguestfs

Все этого диктовало необходимость использования такого окружение, такого набора инструментов, в рамках которых предоставляется достойный Windows-аналог gcc-компилятора и желательно поддерживается система сборки GNU. Вообщем, проекты Cygwin и MinGW.

Таким образом, портирование библиотеки началось с реализации возможности её запуска в Cygwin-е. поскольку Cygwin предоставляет практически полное POSIX окружение и позволяет запускать Unix-приложение в Windows практически без изменения их исходного кода. Это оказалось верным и в случае libguestfs.

Однако то факт, что после установки всех необходимых пакетов размер Cygwin-овского дистрибутива стал неприлично велик (порядка 1 ГБ), и желание устранить зависимость портируемой библиотеки от Cygwin-овского окружения, послужило причиной портировать libguestfs нативно используя инструменты проекта MinGW.*компилятор и систему сборки, предоставляемую в рамках проекта MinGW.*

*Cygwin – проект, предоставляющий Unix-среду в Windows. Ключевой частью Cygwin является его динамически подключаемая библиотека (DLL) реализует значительную часть стандарта POSIX на основе системных вызовов Win32.*

*Поскольку в Cygwin-е реализована значительна часть POSIX API, портирование потребовало минимум изменений с точки зрения исходного кода. Большая часть усилий ушла на изучение системы сборки, использующейся в проекте, и разрешение зависимостей от внешних библиотек, наличие которых требуется для сборки libguestfs.*

*Причинами, послужившими толчком и дальнейшему нативному портированию библиотеки с помощью MinGW, стали следующие факты:*

*Желание устранить зависимость библиотеки от Cygwin - нет необходимости использовать эмуляцию полного POSIX окружения*

*После установки всех необходимых библиотек с помощью встроенного в Cygwin менеджера пакетов размер Cygwin-дистрибутива оказался достаточно большим – порядка 1 ГБ, это послужило причиной более детально разобраться с внешними зависимостями и устранять их вручную по мере необходимости*

*Главной целью портирования было оценить сложность и возможность портирования. Полученные результаты свидетельствовали о том, что нативное портирование более чем возможно.*

*В данном случае портирование потребовало минимального количества изменений, сделанные в способе запуска виртуальной машины и организации сетевого взаимодействия. Больше сил ушло изучение системы сборки, на разрешение сторонних зависимостей и собственной сборки библиотеки.*

*Результаты тестирования представлены на слайде. Очевидно, что в результате данного портирования библиотека показывает менее привлекательные результаты, но главной целью данного портирования была необходимость оценить сложность и в принципе возможность портирования данной бибилотеки без её полного переписывания.*

*После этого я приступил к нативному портированию библиотеки.*

В рамках нативного портирования весь платформозависимый код был локализован и скрыт внутри простых кроссплатформенных интерфейсов, оформленных в виде небольшой библиотеки. Также потребовалось реализовать аналоги Unix-функции, *предоставляемые Unix-системами,* для их работы в Windows. Некоторые отдельные модули Libguestfs были полностью переписаны для работы в Windows.

Значительна часть усилий была потрачена на разрешений внешних зависимостей – поиск и сборку вспомогательных библиотек, от которых зависит libguestfs. В конечном итоге, процесс сборки получившейся портированной библиотеки был интегрирован в существующую систему сборки libguestfs.

*Нативное портирование потребовало реализации аналогов Linux-функций для работы Windows*

*Весь платформозависимый код был локализован внутри простых кроссплатформенных интерфейсов и оформлен в виде небольшой библиотеки*

*Также были внесены изменения в процедуру запуска виртуальной машины*

Результаты тестирования производительности libguesfs *с использованием Cygwin и MinGW* показаны на слайде. *Здесь приведены результаты тестирования производительности библиотеки по выполнению операций чтения и записи.* Видно, что по сравнению с Linux скорость работы библиотеки упала, но, например, отказ от использования Cygwin *всё же* оказал некоторый положительный эффект на производительность.

Да, Поскольку наблюдаемая скорость чтения и записи всё равно далека от той, которую мы могли получить, работая с файловой системой непосредственно, нативно, без использования промежуточного звена в виде виртуальной машины, был предпринят ряд попытки увеличить скорость выполнения операций чтения и записи, а именно

* Заменить протокол взаимодействия: XDR на Protobuf
* Использование разделяемой памяти

Замена XDR на Protobuf не дала должно эффекта в увеличении производительности.

Использование разделяемой памяти оказалось более успешным. Изначально в Libguestfs все данные команд и результаты их выполнения передавались через сокеты. Собственно, файлы передавались частями по 8 КБ. Учитывая дополнительные издержки связанные с организацией сетевого взаимодействия, мы получаем очень медленную скорость чтения и записи, особенно больших файлов. Использование разделяемой памяти позволило передавать файлы частями гораздо большего размера, а также избавиться от лишнего копирования происходящего при передаче через сокеты.

При реализации передачи файлов посредством общей памяти особое внимание стоило уделить организации доступа к ней со стороны демона, который работает внутри виртуальной машины и ничего не знает о windows и его файловых отображениях.

Доступ был реализован с помощью представления памяти в виде особого PCI-устройства.

А В дистрибутив Linux был дополнительно включен специальный драйвер для работы с этим устройством.

Полученные после этого результаты производительности очень вдохновляют. Операции чтения и записи выполняются в разы быстрее, чем раньше, что не может не радовать.

Резюмируя всё вышесказанное, в рамках данной работы была реализованы библиотека, позволяющая приложениям Windows работать с нативными для Linux файловыми системами. Для доступа к ФС использовались нативные Linux-драйвера, для работы которых эмуляция окружение Linux осуществлялась с помощью виртуальной машины. Была добавлена возможность передачи файлов посредством механизма разделяемой памяти для повышения скорости выполнения операций чтения и записи.

Дальнейшая работа

Портирование и использование UML в качестве виртуальной машины

Портирование всего проекта libguestfs.

*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 защиты данных, антивирусное программное обеспечение
* 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

..

Текуще положение дел -> драйвер до цели

Цель:

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.

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

Разраьа

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.

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

Тестирование

Очевидно, что быстрее теперь. Но скорость далека от нативной

Результаты

Библиотека, реализованная на языке C, позволяющая осуществлять доступ к нативным файловым системам.

Возможность передачи файлов посредством разделяемой памяти

Доступ к любой нативной файловой системе операционной системы Linux

Низкая скорость чтения файлов

Высокие издержки памяти связанные с запуском виртуальной машины

Дальнейшая работы

Использование UML в качестве виртуальной машины

Что такое Cygwin? Cygwin – проект компании RedHat, представляет собой Unix-подобную среду.

Кроме того, Cygwin включает в себя инструменты разработки GNU для выполнения основных задач программирования, а также и некоторые прикладные программы, эквивалентные базовым программам UNIX

Таким образом, Cygwin представляет собой удобный инструмент для портирования программного обеспечения Unix в Windows.

Прежде чем приступить к реализации данного решения был обнаружен очень интресный проект

Хочется сказать, что оченб вовремя был обраружен

Окружение Cygwin

И поддерживать минимальные изменения, чтобы поддерживать fork ()