Санкт-Петербургский Государственный Университет

Математико-Механический факультет

Кафедра системного программирования

Безопасное рабочее пространство пользователя

Multi-Cloud Desktop

Модуль внедрения автозапуска приложений в виртуальные машины

Курсовая работа студента 445 группы

Одерова Романа Сергеевича

Научный руководитель.. . . . . Баклановский М.В.

старший преподаватель

кафедры системного программирования

Санкт-Петербург

2013

Оглавление

1. Введение

В современном мире огромную роль играют компьютерные системы и информационные технологии в целом. За последние полвека мир IT очень бурно эволюционировал, порождая новые области науки и индустрии. Некоторые методы и подходы получили известность совсем недавно, а другие уже давным-давно считаются базовыми, и без них невозможно представить ни один современный компьютер. Виртуализация (в общем понимании) – одна из основополагающих технологий, взятых за основу при построении современных компьютерных систем.

1.1. Основные определения и концепции

Виртуализация – это совокупность технологий, для организации выполнения приложений в изолированной среде так, что:

- приложения не влияют на стабильность работы других приложений за пределами данной среды;
- приложения получают доступ к ресурсам системы так, будто они работают с ними напрямую;
- на одном ПК могут одновременно выполняться несколько групп приложений в отдельных средах [27].

Такие изолированные среды называются виртуальными машинами (ВМ), а программы, контролирующие их работу – менеджерами виртуальных машин[1] (Virtual Machine Manager, VMM). Любая ВМ включает в себя процессоры, память, жесткие диски, сетевые адаптеры, устройства ввода/вывода, предоставляемые менеджером виртуальных машин. Для поддержания взаимодействий с этими устройствами необходимы специальные технологии.

Не вдаваясь в детали эмуляции различных ресурсов компьютера, стоит упомянуть о возникающей задаче депривилегизации. В «кольцевой» архитектуре (Ring 0 – Ring 3) [12] с наивысшим уровнем привилегий может исполняться лишь базовая ОС или гипервизор. Таким образом возникает необходимость корректно обрабатывать команды гостевых ОС, не имеющих в реальной системе должных привилегий. Существует два подхода: бинарная трансляция и паравиртуализация. Первый подход заключается в том, что гипервизор «на

[1] Менеджеры виртуальных машин в современном мире виртуализации также называют гипервизорами

лету» анализирует команды гостевой ОС и преобразует их соответствующим образом. Это универсальная технология, требующая больших ресурсов. Второй подход – паравиртуализация – подразумевает перекомпиляцию ядра гостевой операционной системы таким образом, что все вызовы направляются в VMM. Поэтому такой метод применим лишь к ОС с открытым исходным кодом.

В 1973 году Р.Голдберг [4] выделил два типа VMM (см. Рис. 1): нативный (native/bare metal) и хостовый (hosted). Гипервизоры первого типа исполняются непосредственно на железе и управляют гостевыми ОС[2]. Примерами виртуальных сред с описанной архитектурой являются Oracle VM Server, Xen, KVM, VMware ESX/ESXi, Microsoft Hyper-V. Хостовые менеджеры ВМ запускаются в предустановленной (хостовой) ОС. Гипервизорами второго типа считаются VMware Workstation, Oracle VirtualBox.

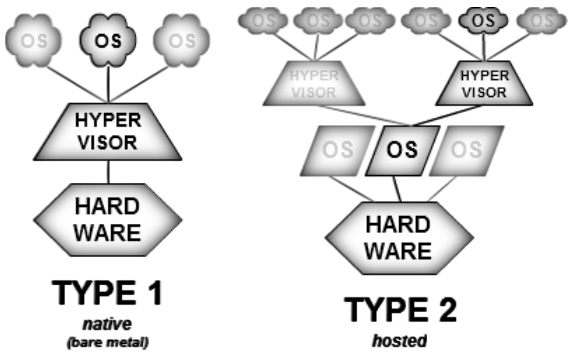

Рис. 1. Классификация Голдберга

Intel в свою очередь определяет несколько более узких классов VMM [27] (см. Рис.2):

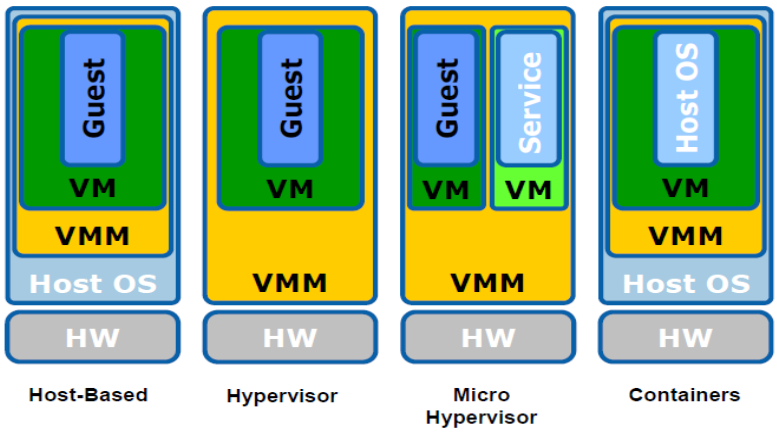

Рис. 2. Классификация гипервизоров

[2] Гостевая ОС – операционная система, установленная в виртуальной машине. Но иногда термин «виртуальная машина» употребляется в качестве синонима термина «гостевая ОС».

- host-based (VMware Server, Parallels Server)
    - запускается в базовой ОС
    - ВМ располагает всеми виртуальными устройствами
    - поддержка различных ОС
    - небольшая производительность
- hypervisor (VMware ESX Server)
    - особая ОС с драйверами для всех устройств
    - поддержка различных ОС
    - расходует дополнительную память для VMM
    - ограниченное количество поддерживаемых устройств
- micro-hypervisor (XEN, KVM, Citrix XenServer)
    - особая ОС
    - работа с устройствами с помощью сервисной ВМ
    - поддержка различных ОС
    - понижение производительности из-за взаимодействия с сервисной ВМ
- containers (Parallels Virtuozzo Containers)
    - «клонирует» базовую ОС
    - разделяет ресурсы между ВМ – изолированными контейнерами
    - не требует виртуализации большинства устройств
    - все ВМ исполняют одну ОС (базовую)

1.2.Применение виртуализации

Основные области использования виртуализации [18]:

- Консолидация серверов (см. Рис. 3)

  С применением виртуализации можно более эффективно использовать имеющиеся ресурсы. Установив специальное ПО (гипервизор), можно на одном сервере запустить несколько различных операционных систем. Это оптимизирует использование доступных ресурсов.

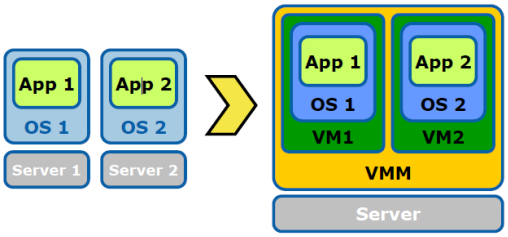

Рис. 3. Консолидация серверов

- Изоляция сервисов (см. Рис. 4)

  Программы выполняются на одной машине, но в разных изолированных средах. В случае появления критических ошибок в одном из процессов, другие остаются в безопасности.

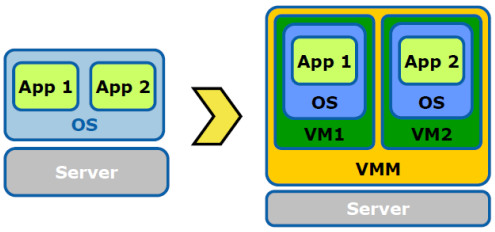

Рис. 4. Изоляция сервисов

- Тестирование ПО

  Из предыдущего пункта следует возможность создания тестовой системы. Программисты запускают опасные приложения в виртуальных машинах без страха «уронить» всю систему целиком.

- Отказоустойчивость

  Важная область применения методов виртуализации. О ней пойдет речь далее.

1.3.Отказоустойчивость

Отказоустойчивость – это способность системы продолжать корректно работать при отказе (механической или алгоритмической причине ошибки) одной или нескольких подсистем, от которых она зависит [32]. Среди отказов могут быть: ошибки ПО, непредусмотренные usecase'ы (пользовательские ошибки) и др.

Высокую доступность отказоустойчивых систем чаще всего выражают в процентах uptime (непрерывного времени доступности пользователям) в год. Например, «6 девяток» означает, что система доступна 99.9999% времени работы (т.е. около 0,6 секунды в год система позволяет себе «не откликаться» клиенту).

Для достижения отказоустойчивости можно использовать два подхода.

1. Избыточность системы [10].

   Внедрение дополнительных модулей/ресурсов и специальных алгоритмов, позволяющих системе корректно выходить из экстренных ситуаций.

   Избыточность невыгодна по экономическим причинам: необходимо дублировать (троировать…) дорогостоящее оборудование и каналы связи.

2. Виртуализация

   Использование VMM, позволяющего нескольким ОС изолированно выполняться на одной машине и не оказывать влияния друг на друга в случае возникновения отказов. Эта технология повышает время, в течение которого система способна откликаться пользователю.

   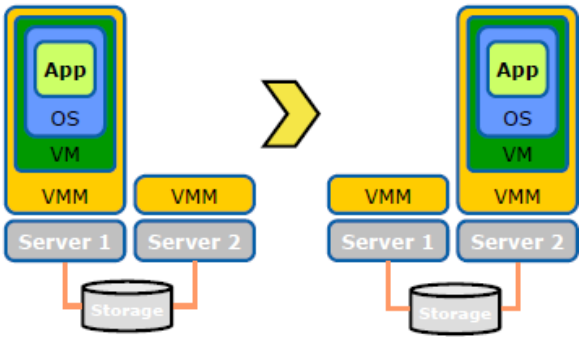

   Рис. 5. Отказоустойчивость

   Не требует существенных материальных вложений.

   Развитием идеи применения виртуализации для достижения отказоустойчивости можно считать технологию миграции, заключающейся в переносе последнего работоспособного состояния ОС на другой хост и восстановлении работы без заметного простоя (см. Рис. 5).

2. Первоначальная задача

2.1.Отказоустойчивая виртуализация на основе гипервизора Xen

Как упоминалось в предыдущем пункте, при достижении отказоустойчивости можно руководствоваться двумя подходами: избыточностью и использованием виртуализации. Т.к. избыточность невыгодна, была поставлена задача реализации отказоустойчивой системы с использованием технологии виртуализации. А именно, был рассмотрен гипервизор Xen, и на его основе построена соответствующая отказоустойчивая система. Подробные сведения о проделанной работе можно посмотреть в [26], [30], [31].

2.2.Дальнейшее развитие тематики

После построения отказоустойчивой системы на основе Xen было решено провести обзор направлений развития данной тематики. Важным требованием было сохранить полученные результаты и применить уже имеющийся опыт работы с Xen впоследствии.

2.2.1. Тестирование

Основной целью являлось проведение тестирования уже построенной системы под нагрузками и в критических ситуациях (выключение энергии на хосте, вывод компонент системы из строя, имитация ошибок ПО и прочее). Результатом мог стать детальный отчет о проведенном исследовании [26].

2.2.2. Профайлинг Remus

В этой задаче подразумевалось исследование механизма расстановки чек-поинтов и восстановления системы модуля Remus. Возможным развитием могло стать улучшение процесса трансфера контрольных точек и повышение степени их сжатия для ускорения репликации. Результатом могло стать сопоставление реализаций различных алгоритмов сжатия. Однако было найдено исследование коллег из University of British Columbia, рассмотревших алгоритмы сжатия информации и сравнивших их реализации в Remus [20].

2.2.3. *Автомасштабирование ресурсов*

Целью работы являлось создание модуля балансировки нагрузки на отказоустойчивую систему, что было в дальнейшем развито до идеи создания кластера на основе Xen [26].

2.2.4. *Qubes-OS*

Qubes-OS – opensource ОС, предоставляющая повышенную безопасность при работе с различным ПО. Основана на Xen. Первая версия – сентябрь 2012 года. Второй релиз в разработке: обещается поддержка Windows [17].

Общая идея Qubes-OS – это изоляция приложений друг от друга в специальных ApplicationVM. Пользователь может создать несколько доменов (легковесных ВМ, запущенных на Xen) и определить их уровни безопасности. Основная задача – обеспечение строгой изоляции доменов так, что в результате атаки на приложения одной AppVM другие остаются нетронутыми.

Детали Qubes-OS подробно описаны в документации [21] и в [26].

Идея разработки такой архитектуры системы, базирующейся на разделении привилегий процессов и их изоляции, заслуживает упоминания.

Выделим два пути развития.

1. Исследование уязвимостей Qubes-OS [21].
2. Перенос архитектурной идеи Qubes-OS в облачную инфраструктуру.

Идея разделения приложений в изолированных ВМ может быть доработана следующим образом.

3. Проблематика и постановка задачи обеспечения безопасного рабочего пространства пользователя

3.1. Общее описание проблемы

Более детальное описание можно увидеть в [26]. Остановимся лишь на основных аспектах и поясним на примере.

Допустим, пользователь на своей локальной машине работает с некоторыми конфиденциальными данными. Если во время нахождения этих данных на локальном компьютере он использует, например, интернет-браузер (или любое другое приложение, в котором в том или ином виде могут существовать уязвимости), не исключены ситуации внедрения вредоносного кода в систему или в ПО, эксплуатация их уязвимостей и, как результат, хищение важной информации.

В наше время рассматриваемая проблема может быть решена путем использования разных машин для работы с программами различного уровня доверия. Этот метод ужасно неудобен, и поэтому предлагается создать систему, в рамках которой можно работать с приложениями на одной машине, принимая во внимание повышенные требования к безопасности.

Основная идея состоит в изоляции клиентской машины от доступа в интернет, переместив конфиденциальную информацию в специальное защищенное хранилище. Также предлагается отказаться от локального запуска программ и «разбросать» их по облакам с учетом необходимого уровня доверия. Клиент получает только состояния приложений.

3.2.Идея решения

Казалось бы, с помощью технологии виртуализации описанная задача решается в лоб: устанавливаем гипервизор и запускаем в разных ВМ разные приложения на одной локальной машине. Однако существуют методы внедрения вредоносного кода в систему в обход виртуальных машин [5], [9], [13], [14].

Но можно усовершенствовать эту идею и создать систему Multi-Cloud Desktop.

3.3.Multi-Cloud Desktop

Опишем общую архитектуру системы.

Пользователь запускает приложения, используя специальное предустановленное ПО. Программы выполняются в привычном режиме, взаимодействие с ними происходит с помощью клавиатуры и мыши, как обычно. Однако на самом деле приложения запускаются в облаке на специальных серверах, а на клиентский ПК «прилетает» видео-поток, отображающий состояние приложений.

Для удовлетворения повышенных требований к безопасности необходимо отключить пользователю доступ в интернет с локальной машины, на которой будет функционировать только тонкий клиент. Он реализует протокол общения с облаком приложений.

Приложения в облаке запускаются в изолированных средах – виртуальных машинах. Этот подход повышает безопасность и обеспечивает отказоустойчивость. Важная особенность нашей системы заключается в возможности одновременной работы с приложениями для различных ОС (Linux, Windows, Mac OS...) на одной машине.

Серверы в облаке имеют несколько уровней доверия. Наиболее безопасным должен быть сервер для запуска программ, работающих с конфиденциальными данными (желательно отрезать ему доступ в Интернет и ограничить к нему доступ физический сотрудников организации). На рисунке 6 представлена общая схема архитектуры.

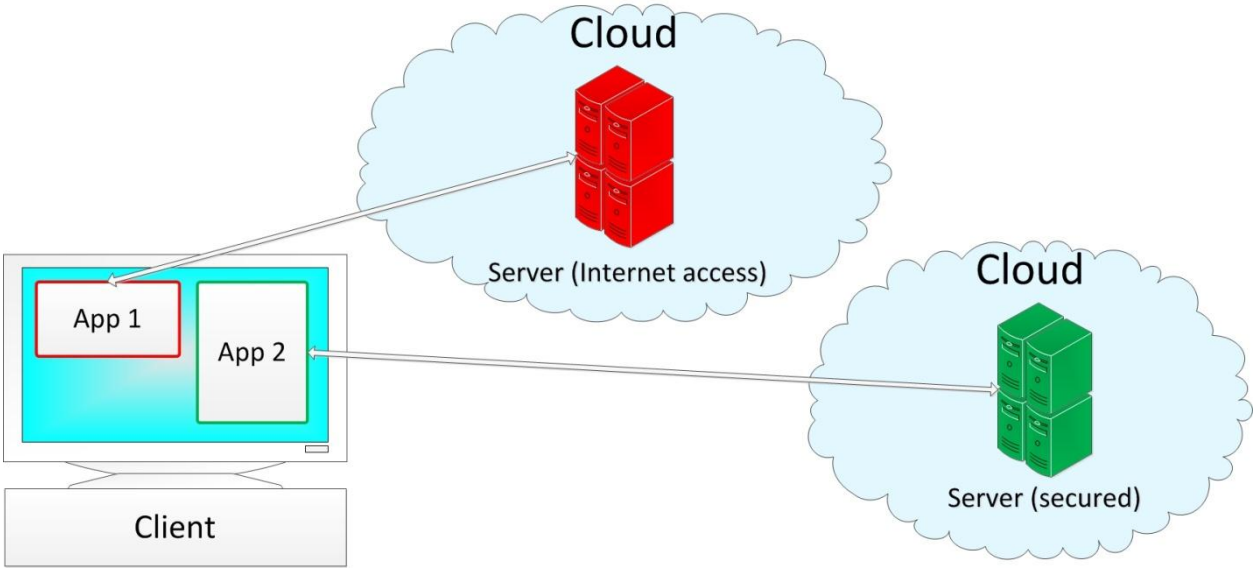

Рис. 6. Общая архитектура Multi-Cloud Desktop

Более детальное описание можно найти в [26].

3.4. Обзор существующих решений

Был проведен анализ существующих решений. В результате не было найдено продукта, целиком отвечающего нашим требованиям. Ближайшим конкурентом по функциональности является Citrix XenApp [7].

Citrix XenApp – коммерческое решение компании Citrix, использующее технологию виртуализации, централизованно управляемое и осуществляющее доставку Windows-приложений клиенту по требованию на любое устройство: планшет, ноутбук, десктоп… Доставка программ производится с помощью проприетарного протокола ICA.

Описание продукта, схемы его работы и пр. можно найти в официальных документах Citrix [6] и в [19].

3.5. Описание основных технологий

Для конфигурирования рассматриваемой в пункте 3.3 Multi-Cloud Desktop предлагается воспользоваться гипервизором Xen, а точнее его «облачной версией» Xen Cloud Platform.

3.5.1. Xen Cloud Platform

XCP – это по сути почти обычный Xen, снабженный Domain 0 системой CentOS 5 и утилитами для работы с облаком.

Подробное описание архитектуры XCP и всех ключевых понятий можно найти в [26].

Вкратце отметим, что главные элементы XCP – хосты. На них запускаются и работают виртуальные машины. Хосты можно объединять в пулы, в рамках которых есть возможность проводить миграцию ВМ. Все ВМ хранятся в Storage Repository (SR) – логической абстракции, реализованной в основном для упрощения механизма миграции (в случае SR, построенного на сетевом общем хранилище). Образы дисков виртуальных машин называются Virtual Disk Image (VDI) – это «слепки» соответствующих файловых систем; Virtual Block Device (VBD) реализуют доступ к VDI из ВМ [23], [34].

3.5.2. X11 (X Window System)

X Window System – это открытая кроссплатформенная оконная система, то есть “серверно-клиентское” программное обеспечение, которое позволяет управлять оконным графическим интерфейсом пользователя в распределенных сетях.

X11 – протокол позволяющий запускать приложения на удаленных компьютерах (серверах) с помощью X-Client и отображать эти приложения на локальном компьютере пользователя (клиенте) с помощью X-Server [33].

В MCD X11 позволяет доставлять приложения пользователю. На клиентском ПК работает X-Server, отображающий окна, обрабатывающий события устройств ввода и общающийся с X-Client на стороне сервера, где запускается необходимое приложение.

На текущий момент в протокол X11 входит передача окон приложений с Linux на Linux, с Linux на Windows, но отсутствуют варианты Windows→Linux, Windows→Windows [33].

Поэтому для создания единой системы передачи графики приложений необходимо реализовать недостающие ветви протокола (см. далее пункт 3.6.1.3).

3.6.Постановка задачи

На данный момент первостепенная задача заключается в построении тестовой системы, Proof of Concept, способной подтвердить возможность реализации проекта. Таким образом, некоторая функциональность далее рассматривается в упрощенном виде.

3.6.1. Основные задачи

Выделим основные задачи, без которых в принципе невозможно создание описанной выше системы Multi-Cloud Desktop.

3.6.1.1. Конфигурация тестовой системы

Для начала необходимо создать соответствующую среду для работы, а именно построить небольшое облако. Текущий вариант тестовой системы представляет собой следующую конфигурацию.

Два идентичных хоста соединены с помощью Ethernet. На каждом установлен Xen Cloud Platform версии 1.6. Они объединены в единый пул, первый из подключенных к пулу хостов объявлен мастером. Для каждого из хостов сконфигурирован локальный SR, в котором хранятся привязанные к хосту виртуальные машины. Установка гостевых ОС может быть осуществлена как по сети, так и с использованием установочного диска, вставленного в привод хоста с целевой виртуальной машиной. С помощью утилиты xe можно управлять виртуальными машинами, мигрировать их с одного хоста на другой, выбирать мастера пула и др.

3.6.1.2. Автозапуск приложений по требованию

Начнем с серверной части. Так как требуемое пользователем приложение запускается в виртуальной среде на специальном сервере, необходимо продумать, каким образом внедрять нужную программу в соответствующую гостевую ОС. Было выдвинуто предложение о предварительном создании и хранении образов операционных систем, укомплектованных доступными приложениями, и внедрении в них механизма автозапуска необходимых клиенту программ. То есть когда пользователь вызывает некоторое приложение A, отправляется запрос на сервер, на котором выбирается соответствующая укомплектованная виртуальная машина VM_A, в ее образ внедряется автозапуск программы A, и происходит загрузка этой гостевой операционной системы. Таким образом происходит запуск программ по требованию.

3.6.1.3. Передача графики. Модификация протокола X11

После запуска приложения на сервере необходимо каким-то образом передать его окно клиенту. Подобную задачу решает существующий протокол X11 (см. пункт 3.5.2), однако в нем не реализована половина требуемой нам функциональности: в нем есть поддержка передачи графического представления программ из Unix-подобных систем в любую другую ОС, но он «не умеет» передавать графику приложений ОС Windows.

Таким образом, задача заключается в реализации передачи графики приложений на стороне X-клиента MS Windows посредством протокола X11, а также в получении и обработке событий, приходящих со стороны X-сервера.

3.6.1.4. Протокол управления MCD

Следующий шаг – взаимодействие клиента и сервера в построенном облаке, т.е. реализация соответствующего протокола. Этот механизм в первом приближении должен включать в себя возможность запуска/остановки приложения.

Пользователь посылает сигнал управляющей утилите на стороне сервера с запросом на запуск определенного приложения. Сервер выполняет необходимые действия для старта требуемого приложения в нужной виртуальной машине, инициирует передачу изображения окна программы пользователю и начинает обрабатывать управление, приходящее со стороны клиента. Остановка происходит с точностью до наоборот [26].

3.6.2. Конкретная задача

В данной курсовой работе рассматривается решение задачи реализации модуля внедрения автозапуска приложений в виртуальную машину. Соответственно, возникшая задача связана с реализацией следующей последовательности действий: модификация файла ВМ, внедрение автозапуска, сохранение файла ВМ, запуск ВМ средствами XCP.

Будут рассмотрены основные идеи реализации, описаны механизмы работы и прочее.

4. Решение задачи автозапуска приложений

4.1. Возможные способы автозапуска приложений

В операционной системе существуют способы автоматического запуска приложений в определенный момент, например, при старте системы или в определенное время.

Ниже будут перечислены некоторые из них.

Для операционной системы Microsoft Windows существует два наиболее распространенных способа внедрения автозапуска.

Первый способ – размещение ярлыка требуемого приложения (либо некоторого .bat-файла с требуемыми командами) в StartUp-директории. Такие действия приводят к тому, что при следующем запуске Windows запускаются и все приложения, ярлыки которых лежат в указанной выше папке.

В Windows 7 ярлыки можно разместить как в StartUp-директории для общего пользования (“C:\ProgramData\Mirosoft\Windows\Start Menu\Programs\Startup”), так и определить запускаемые программы для конкретной учетной записи (“C:\Users\The_User_Name\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup”).

Второй способ заключается в модификации определенных записей реестра Windows [11] и внесения в них информации о запускаемом приложении. Для автозапуска в рестре существует 7 разделов:

1. `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`
2. `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`
3. `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce`
4. `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce`
5. `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices`
6. `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce`
7. `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup`

В каждой версии Windows есть свои правила и ограничения на использование этих разделов, однако общая суть заключается в следующем. Разделы с 1 по 4 обрабатываются при регистрации пользователя в системе. Run RunOnce отличаются тем, что прописанные в них программы запускаются при каждом старте системы или при первом старте системы соответственно[3]. 5 и 6 разделы обрабатываются в фоновом режиме при появлении диалогового окна входа в систему[4]; раздел 7 выполняется как часть процедуры программы установки, или при использовании мастера установки/удаления программ [29].

Есть, конечно, еще несколько способов, например, можно поместить в папку “C:\Windows\” программу с именем explorer.exe (удалив при этом стандартный explorer, естественно), и она будет выполняться при каждом старте системы.

Для операционной системы Linux дела обстоят по-другому. Здесь существуют специальные скрипты, которые обрабатываются в определенные моменты времени.

4.2.Генерация .lnk-файлов

Как упоминалось выше, существует два основных способа автозапуска приложений Windows: помещение ярлыка программы в Startup-директорию и модификация файлов реестра. Было решено воспользоваться первым способом ввиду его относительной простоты и минимальной модификации файловой системы. .lnk файл создан для поддержки запуска приложений и ссылок на файлы.

Для начала, рассмотрим формат .lnk-файла (сам формат называется shell link binary format)

```
SHELL_LINK = SHELL_LINK_HEADER [LINKTARGET_IDLIST] [LINKINFO]
             [STRING_DATA] *EXTRA_DATA
```

`SHELL_LINK_HEADER` содержит информацию по идентификации и различных флагах, определяющих наличие дополнительных секций структуры.

`LINKTARGET_IDLIST` – опциональная структура, определяющая цель, на которую указывает ярлык. Ее присутствие специфируется битом HasLinkTargetIDList в секции ShellLinkHeader.

[3] В разделах 3 и 4 можно добавить символ ‘!’ перед именем параметра, в результате чего запись из реестра удалится не сразу, а лишь после завершения выполнения команды запуска соответствующей программы.
[4] Замечание: разделы 5 и 6 необходимы для запуска фоновых процессов и выполняются только при начальной загрузке системы

LINKINFO – опциональная структура, содержащая информацию для определения цели ярлыка, если он не найден в стандартном местоположении.

STRING_DATA – ноль или более опциональных структур, используемых для взаимодействия пользовательского интерфейса и идентифицирующей путь к файлу информации.

EXTRA_DATA – ноль или более дополнительных структур.

Описание полей и структур можно найти в спецификации Microsoft [2]. Однако не все поля документированы хорошо. В некоторых абсолютно не понятно, в каком виде должны быть описаны данные. Поэтому пришлось провести дополнительное исследование рассматриваемого формата shell link.

Опишем кратко ход исследования и построения тривиального lnk-файла. Т.к. работа по созданию ярлыка будет проводиться в Linux путем внесения изменений в образ файловой системы Windows, то пользоваться стандартными средствами/утилитами не получится и необходим собственный генератор. Целью исследования было максимальное сокращение размеров lnk-файла путем выкидывания наибольшего количества опциональных структур.

1. Для начала с официального сайта MSDN был взят пример ярлыка, ссылающегося на файл "C:\test\a.txt" [1]. Он работал корректно, что и следовало ожидать.
2. Затем была предпринята попытка сократить некоторые секции, объявленные как необязательные в спецификации Microsoft. После удаления некоторых из них (таких как EXTRA_DATA) ярлык продолжал работать корректно.
3. Раздел, описывающий путь к целевому файлу имеет следующую структуру:
   IDListSize – размер списка (IDList) элементов пути к файлу
   IDList структура описывает непосредственно путь.

   IDLIST = *ITEMID TERMINALID

   *ITEMID – список элементов ItemID.
   ITEMID представляет собой ItemIDSize (размер поля) + Data (непосредственно элемент пути к файлу)
   TERMINALID – 2 нулевых байта.

Применительно к тестовому примеру этот раздел выглядел так:

```
IDListSize: (2 байта, смещение 0x004C), 0x00BD, размер IDList.
IDList: (189 байт, смещение 0x004E) IDList структура:
     ItemIDList: (187 байт, смещение 0x004E), ItemID структуры:
         ItemIDSize: (2 байта, смещение 0x004E), 0x0014
         Data: (12 байт, смещение 0x0050), <18 байт данных> [computer]
         ItemIDSize: (2 байта, смещение 0x0062), 0x0019
         Data: (23 байт, смещение 0x0064), <23 байт данных> [c:]
         ItemIDSize: (2 байта, смещение 0x007B), 0x0046
         Data: (68 байт, смещение 0x007D), <68 байт данных> [test]
         ItemIDSize: (2 байта, смещение 0x00C1), 0x0048
         Data: (68 байт, смещение 0x00C3), <70 байт данных> [a.txt]
     TerminalID: (2 байта, смещение 0x0109), 0x0000 конец IDList.
```

Здесь названия элементов пути “c:\test\a.txt” заполнены некоторой дополнительной информацией (чаще всего нулями) неизвестного предназначения. Были удалены неизвестные биты (с сокращением размеров элементов списка), не описанные в спецификации Microsoft, но присутствующие в тестовом примере ярлыка. Ярлык продолжил работать правильно.

4. Сравнивая сгенерированные системой ярлыки для обычных windows-файлов, можно увидеть, что перед элементом “test” всегда располагается одинаковое количество символов. Зануляем их и по-прежнему наблюдаем корректную работу.
5. Теперь пробуем вручную добавить в структуру ярлыка один уровень иерархии в путь: переносим файл в “c:\test\test\a.txt”.

   Ярлык способен работать, если «запихать» весь путь целиком в один элемент IDList. Если же разбивать аналогично на несколько элементов, то не получается.

   Замечаем, что в начале каждого Item стоит некий уникальный идентификатор. После рассмотрения аналогичных мест в разных ярлыках, созданных системными средствами, становится ясно, что идентификатор 0x31 ставится, если текущий элемент – это директория, и 0x32 – если это файл.

После некоторых дальнейших упрощений был получен требуемый результат и написан Python-скрипт, генерирующий ярлык по полному пути к файлу.

4.3. Автозапуск в скриптах Linux

В Linux существуют специальные runlevel, определяющие основные этапы работы системы. В соответствующих каталогах /etc/rc?.d, где ? – номер определенного runlevel, присутствуют символические ссылки на скрипты, подлежащие выполнению в рамках этого runlevel. Например, 0 уровень предполагает действия по выключению системы. 1 – однопользовательский режим (single user mode), предназначенный для различных действий по восстановлению системы. На этом уровне система полностью «инициализирована», но ни один сервис не запущен, и работать может только root-пользователь. 2 – используется редко, не включает поддержку сети, аналогичен уровню 3. 3 и 5 режимы соответствуют консольному и графическому многопользовательским режимам. 4 – обычно не используется в работе. 6 – уровень для рестарта системы [3].

Для запуска скрипта myscript.sh[5] необходимо поместить его в директорию /etc/init.d/ и прописать его в требуемые runlevel. Это можно сделать с помощью команды update-rc.d.

Существует файл /etc/rc.local, который можно использовать для выполнения скриптов при старте системы. Достаточно лишь прописать в него требуемые команды.

Например, для создания файла при старте системы необходимо модифицировать rc.local следующим образом:

```
#!/bin/sh –e
/bin/touch /path_to_file/filename
exit 0
```

Для автозапуска приложений можно создать специальные конфигурационные файлы формата .desktop (описание формата можно посмотреть в спецификации) [8], где будут описаны определенные параметры запуска. Эти файлы располагаются в /home/user_name/.config/autostart/.

Пример такого файла для запуска приложения Mozilla Firefox в Ubuntu:

```
[Desktop Entry]
Type=Application
Terminal=false
Exec=/usr/bin/firefox
Name=Firefox
X-GNOME-Autostart-enabled=true
```

[5] Замечание: скрипт должен быть исполняемым. Это можно сделать, выполнив команду sudo chmod +x myscript.sh

Для внедрения в образ виртуальной машины автозапуска необходимо уметь модифицировать этот образ (например, .VDI или .VMDK), который представляет собой обычный файл определенной структуры. Все действия будут производиться в XCP Domain 0, которой является CentOS 5.

4.4. Модификация файлов виртуальной машины

В XCP используются файлы виртуальных машин в формате .VDI. Однако простой импорт существующего файла VDI [6] затруднителен.

Создание новой виртуальной машины происходит из файла формата .raw, который представляет собой простой бинарный образ диска и является легко портируемым.

Таким образом нужно уметь «залезать» внутрь файла с целью внесения изменений в файловую систему. Для этого его нужно смонтировать с использованием какой-нибудь утилиты. Такую возможность предоставляет программа VdFuse.

VdFuse[7] – утилита, позволяющая из образов vdi, vmdk, vhd, raw создавать файловую систему с помощью модуля FUSE[8]. Использует библиотеку Virtual Box для работы с соответствующими форматами.

Пример команды vdfuse (-w разрешает модификацию диска всем пользователям, -f определяет сам файл образа):

```
vdfuse -w -f /path_to_image/image /mnt_point
```

Таким образом, после создания файловой системы из доступного формата необходимо смонтировать ее в систему с помощью команды mount с опцией “loop” для последующей модификации файлов.

После проведения необходимых изменений необходимо размонтировать образ файловой системы с помощью команды umount.

[6] Virtual Disk Image – формат образа виртуальной машины, разработанный компанией Oracle для VirtualBox Подробнее см. https://forums.virtualbox.org/viewtopic.php?t=8046
[7] https://github.com/jonathanxavier/vdfuse
[8] Filesystem in Userspace – модуль для создания пользовательских файловых систем. Подробнее: http://fuse.sourceforge.net/

Затем для работы в XCP необходимо преобразовать файл образа в формат .raw. В этом могут помочь VBoxManage или QEMU.

VBoxManage – интерфейс командной строки VirtualBox. С помощью него можно полностью контролировать работу VirtualBox [22].

QEMU – эмулятор и виртуальная машина, позволяющая запускать полноценную операционную систему как обычный процесс на компьютере пользователя [15]. Поддерживает различные типы образов дисков: raw, cloop, cow, qcow, qcow2, vmdk, vdi. QEMU имеет специальную команду qemu-img для создания и управления образами дисков [16].

Соответствующие команды перевода файла исходного формата в файл .raw представлены ниже:

VBoxManage:

```
VBoxManage clonehd input_image.format --format raw output_image.raw
```

QEMU:

```
qemu-img convert input_image.format -O raw output_image.raw
```

4.5.Доказательство работы описанного подхода в Oracle VirtualBox

Итак, рассмотрев основные аспекты работы, можно теперь соединить их вместе и применить полученный результат к некоторому реальному примеру.

Возьмем за основу продукт Oracle VirtualBox, создадим в нем виртуальную машину Windows XP. При создании выберем любой тип файла образа диска, например vmdk.

Предположение: работа по модификации образов дисков производится из-под операционной системы Ubuntu.

Задача: проверить работу автозапуска при внесении необходимых изменений в файл образа диска Windows XP.

Решение:

1. «Создаем» файловую систему из файла XP.vmdk в папке /mnt_XP
   `vdfuse -w -f /XP.vmdk /mnt_XP`
   В папке /mnt_XP создаются файлы EntireDisk и Partition1
2. Монтируем в систему
   `mount -o loop /mnt_XP/Partition1 /mntpoint_XP`
3. Теперь в /mntpoint_XP открывается доступ к файловой системе нашей ОС Windows XP
4. Запускаем скрипт-генератор для создания ярлыка CreateLNK, например, для калькулятора и помещаем созданный ярлык test_calc.lnk в директорию StartUp.
   Замечание: путь в StartUp-папку Windows XP сейчас выглядит следующим образом (относительно Ubuntu): /mntpoint_XP/Documents and Settings/All Users/Start Menu/Programs/Startup
5. Можно также сгенерировать bat-файл и положить его в ту же директорию Автозапуска. В нем пропишем вызов программы Блокнот, которая при старте системы и откроет нам этот bat-файл. ☺

Итак, запускаем нашу виртуальную машину с модифицированного образа:

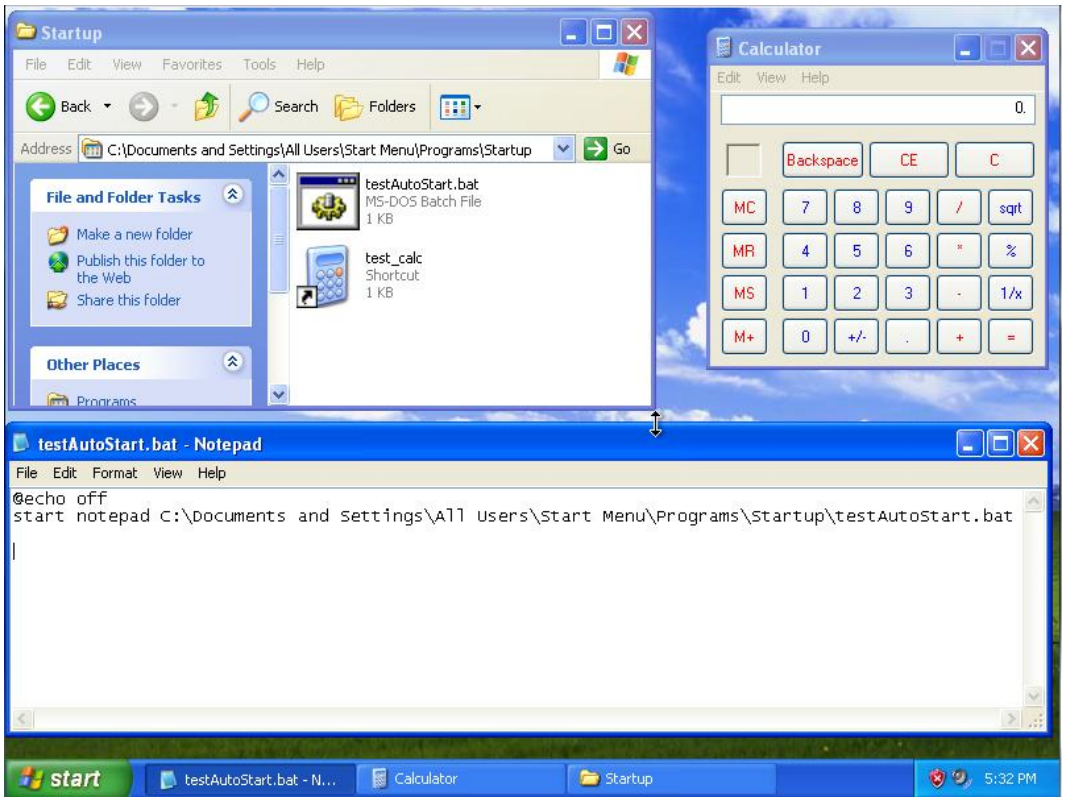

Рис. 7. Автозапуск в WinXP-BM

Что и требовалось получить.

4.6.Модификация файла ВМ и запуск гостевой ОС в XCP

Для выполнения всех описанных выше действий по модификации образов виртуальных машин необходимо установить требуемые утилиты в Domain0: VdFuse, VBoxManage/QEMU.

Допустим, мы имеем необходимый файл – образ диска image.format. Его необходимо модифицировать (см. выше), затем привести к формату .raw.

На сервере создаем шаблон виртуальной машины с помощью стандартных утилит XCP, например, команда

```
xe vm-create name-label=<name for VM> [name-description=<description string for VM>]
```

создает новую виртуальную машину с параметрами по умолчанию. Команда не создает запускаемую виртуальную машину!

В результате выдается UUID (template-UUID), который будет использован далее при выполнении команды vm-install.

Замечание: не обязательно создавать виртуальную машину вручную – можно воспользоваться существующим клиентом для управления облаком Citrix XenCenter [25].

Получаем UUID созданной ВМ, например, с помощью команды:

```
xe vm-list
```

Конфигурируем PV/HVM загрузчик с помощью команды:

```
xe vm-param-set uuid=$VMUUID PV-bootloader=<…>
```

или

```
xe vm-param-set uuid=$VMUUID HVM-boot-params=<…>
```

Делаем загрузочным жесткий диск:

```
xe vbd-param-set uuid=$(xe vbd-list vm-uuid=$VMUUID userdevice=0 --minimal) bootable=true
```

Находим идентификатор VDI-диска только что созданной ВМ, например:

```
xe vm-disk-list uuid=$VMUUID

Disk 0 VBD:
    uuid ( RO) : 50adb0d9-xxxx-xxxx-xxxx-f8f64e5c4f19
    vm-name-label ( RO): FreeBSD
    userdevice ( RW):  0
Disk 0 VDI:
    uuid ( RO) : af85b950-xxxx-xxxx-xxxx-b5203ba45aae
    name-label ( RW): FreeBSD
    sr-name-label ( RO): Local storage
    virtual-size ( RO): 21474836480
```

Переходим в окружение, в котором доступен диск ВМ:

```
/opt/xensource/debug/with-vdi $VDIID /bin/bash
```

Устройство /dev/$DEVICE – это диск ВМ, и осталось залить туда наш начальный образ:

```
/opt/xensource/libexec/sparse_dd -src /var/run/sr-mount/.../FreeBSD.raw -dest /dev/$DEVICE \  -size 21474836480 -prezeroed
```

```
$ exit
```

Теперь можно запускать виртуальную машину [24].

5. Итоги и результаты

Итак, в результате работы сначала были изучены концепция виртуализации и принципы построения систем, способных «пережить» различные отказы. Изучены существующие решения, построена и протестирована отказоустойчивая система на основе гипервизора Xen.

Затем был проведен поиск дальнейших задач с погружением в предметную область, вследствие чего найден и изучен один из наиболее интересных представителей этой сферы – Qubes-OS.

Идея, заложенная в Qubes-OS, была переработана концептуально на уровне физической архитектуры и дополнена различными ограничениями, что позволило занять «нишу» на рынке виртуализации и предложить решение для задачи обеспечения безопасности рабочего пространства пользователя в условиях повышенного риска потери конфиденциальной информации.

В рамках новой идеи идет работа над построением Proof of Concept – построением тестового примера работающей системы, обладающей основной функциональностью.

В данной курсовой работе была решена одна из важнейших задач проекта, заключающаяся в реализации механизма автозапуска приложений в существующих виртуальных машинах. Рассмотрены основные способы автозапуска приложений в ОС Windows и Linux; для Windows дополнительно создан генератор .lnk-файлов, необходимых для работы одного из методов автозапуска. Написан скрипт, автоматизирующий процесс модификации файлов виртуальных машин. Протестирована работа описанного метода внедрения автозапуска в ВМ.

6. Дальнейшие направления развития проекта

6.1.Графический интерфейс пользователя (GUI)

Первоочередной дальнейшей задачей мы считаем создание GUI для нашей системы, который в первом приближении должен обладать следующей основной функциональностью:

- для пользователя:
  1. Отображение списка доступных пользователю приложений
  2. Запуск и работа с необходимым пользователю приложением
- для администратора:
  1. Управление приложениями системы
  2. Управление пользователями и их правами доступа
  3. Отображение статистики по системе: нагрузка хостов, нагрузка виртуальных машин, количество активных пользователей и др.
  4. Управление облачной инфраструктурой: виртуальными машинами, пулами, хранилищами и др.

6.2.Разделение привилегий

Вторым важным шагом развития нашей системы является реализация модели доступа к файлам. В качестве искомой модели доступа нами была выбрана Mandatory access control (MAC) – мандатное управление доступом. В этой модели разграничение доступа субъектов к объектам основывается на назначении метки конфиденциальности для информации, содержащейся в объектах, и выдаче официальных разрешений субъектам на обращение к информации соответствующего уровня конфиденциальности [28]. Т.е. программам и файлам присваиваются различные уровни доступа, а клиенту – уровень привилегий. Доступ организуется иерархически: пользователь с определенным уровнем привилегий имеет возможность работы с информацией максимум того же уровня, и не подозревает о существовании более секретных данных и приложений.

Например, в нашей системе все ресурсы (приложения, файлы) могут быть помечены администратором некоторыми значениями, обозначающими уровень доступности этого

ресурса или его роль, – условно цифрами от 0 до 3. Каждому пользователю системы администратор выдает права доступа – максимальный уровень «доверия». Например, начальник предприятия имеет доступ ко всем ресурсам: браузеру (уровень доступности 0), отчетам бухгалтерии (пусть им выдан уровень доступности 1), к самой системе бухучета (уровень 2) и к секретным документам (с уровнем доступности 3). В свою очередь бухгалтеры имеют доступ к системе бухучета, своим отчетам, браузеру, но не подозревают о существовании секретных файлов.

6.3.Шифрование

Чтобы повысить уровень безопасности нашей системы, можно внедрить сертифицированные методы шифрования на уровне общения тонкого клиента и облачной инфраструктуры.

6.4.Интеграция реализованных модулей в единую систему

Самым важным этапом развития нашего проекта является объединение всех её разрозненных модулей (облачная инфраструктура, модуль доставки окон, менеджер привилегий) в единую систему.

Также есть возможность повысить уровень защищённости выбранной платформы Xen Cloud Platform и сертифицировать её.

Возможно дальнейшее расширение функционала по мере развития проекта.

Список литературы

[1] **[MS-SHLLINK]: Shell Link (.LNK) Binary File Format, Shortcut to a file** [В Интернете] / авт. Microsoft // MSDN. - http://msdn.microsoft.com/en-us/library/dd871375.aspx.

[2] **[MS-SHLLINK]: Shell Link (.LNK) Binary File Format, Structures** [В Интернете] / авт. Microsoft // MSDN. - http://msdn.microsoft.com/en-us/library/dd891253.aspx.

[3] **An introduction to run-levels** [В Интернете] / авт. Debian // Debian Administration. - http://www.debian-administration.org/articles/212.

[4] **Architectural principles for virtual computer systems** [Report] / auth. Goldberg Robert P.. - Harvard : Harvard University, 1973.

[5] **Attacks on Virtual Machine Emulators** [Доклад] / авт. Ferrie Peter. - [б.м.] : SYMANTEC.

[6] **Citrix XenApp Comparative Feature Matrix** [В Интернете] / авт. Citrix // Citrix. - http://www.citrix.com/content/dam/citrix/en_us/documents/products/Citrix_XenApp_6.5_Comparative_Feature_Matrix.pdf.

[7] **Citrix XenApp Overview** [В Интернете] / авт. Citrix // Citrix. - http://www.citrix.com/products/xenapp/overview.html.

[8] **Desktop Entry Specification** [В Интернете] / авт. Preston Brown Jonathan Blandford, Owen Taylor, Vincent Untz, Waldo Bastian // standards.freedesktop.org. - http://standards.freedesktop.org/desktop-entry-spec/latest/.

[9] **Detection of VM-Aware Malware** [Журнал] / авт. David Yu Zhu Erika Chin. - Berkley : University of California, 2007 г..

[10] **Fault-tolerance and fault-intolerance: complementary approaches to reliable computing** [Конференция] / авт. Avizienis Algirdas. - Los Angeles : University of California, CS department, 1975.

[11] **INFO: Описание разделов реестра Run, RunOnce, RunServices, RunServicesOnce и Startup** [В Интернете] / авт. Microsoft // Microsoft Support. - http://support.microsoft.com/kb/179365/ru.

[12] **Intel 64 and IA-32 Architectures Software Developper's Manual** [Книга] / авт. Intel. - Т. 1 Basic Architecture.

[13] **Methods for Virtual Machine Detection** [В Интернете] / авт. Omella Alfredo Andres // S21sec. - 2006 г.. - http://charette.no-ip.com:81/programming/2009-12-30_Virtualization/www.s21sec.com_vmware-eng.pdf.

[14] **On the Cutting Edge: Thwarting Virtual Machine Detection** [В Интернете] / авт. Tom Liston Ed Skoudis. - http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf.

[15] **QEMU** [В Интернете] / авт. Wikibooks // Wikibooks. - http://en.wikibooks.org/wiki/QEMU.

[16] **QEMU/Images** [В Интернете] / авт. Wikibooks // Wikibooks. - http://en.wikibooks.org/wiki/QEMU/Images.

[17] **Qubes-OS wiki** [В Интернете] / авт. Qubes OS Team // Qubes-OS. - http://www.qubes-os.org/trac.

[18] **Reasons to Use Virtualization** [Online] / auth. Oracle // Oracle documentation. - Oracle. - http://docs.oracle.com/cd/E26996_01/E18549/html/BHCJAIHJ.html.

[19] **Reference Architecture–Based Design for Implementation of Citrix XenDesktop on Cisco Unified Computing System, Citrix XenServer, and NetApp Storage** [В Интернете] / авт. Cisco. - 2010 г.. - http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/Virtualization/ucs_xd_xenserver_ntap.pdf.

[20] **Remus: High Availability via Asynchronous Virtual Machine Replication** [Журнал] / авт. Brendan Cully Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, Andrew Warfield. - Vancouver : The University of British Columbia, Department of Computer Science.

[21] **System Documentation for Developers** [В Интернете] / авт. Joanna Rutkowska Rafal Wojtczuk // Qubes-OS. - Qubes-OS. - http://files.qubes-os.org/files/doc/arch-spec-0.3.pdf.

[22] **VBoxManage** [В Интернете] / авт. Oracle VirtualBox // VBox Manual. - http://www.virtualbox.org/manual/ch08.html#idp21429600.

[23] **Xen Cloud Platform** [В Интернете] / авт. Xen // Xen Project. - 2009 г.. - http://www.xen.org/files/XenCloud/reference.pdf.

[24] **Xen Cloud Platform: Как поселить чужую VM в новый дом** [В Интернете] // Habrahabr. - http://habrahabr.ru/post/177679/.

[25] **XenCenter** [В Интернете] / авт. Citrix Community // Cirix Developer Network. - http://community.citrix.com/display/xs/XenCenter.

[26] **Безопасное рабочее пространство пользователя MCD. Конфигурация системы. Построение протокола взаимодействия компонентов** [Доклад] / авт. С.А. Серко. - СПб : СПбГУ, 2013.

[27] **Виртуализация. Технология, изменяющая индустрию IT** [В Интернете] / авт. Intel. - world-it-planet.org/upload/VTx.pdf.

[28] **Мандатное управление доступом** [В Интернете] / авт. Wikipedia // Wikipedia. - http://ru.wikipedia.org/wiki/Мандатное_управление_доступом.

[29] **Назначение разделов "Run" системного реестра Windows** [В Интернете] / авт. Microsoft // Microsoft Support. - http://support.microsoft.com/kb/137367/RU.

[30] **Отказоустойчивая виртуализация на основе гипервизора Xen** [Конференция] / авт. Одеров Р.С. Серко С.А., Чередник К.Е. // Материалы конференции "Актуальные проблемы организации и технологии защиты информации". - СПб : ИТМО, 2012. - Т. 2.

[31] **Отказоустойчивая виртуализация на основе гипервизора Xen** [Конференция] / авт. Одеров Р.С. Серко С.А., Чередник К.Е. // Материалы научной сессии МИФИ-2013. - Москва : НИЯУ МИФИ, 2013.

[32] **Построение отказоустойчивой (fault tolerant) системы** [В Интернете] // Habrahabr. - http://habrahabr.ru/post/118496/.

[33] **Разработка и реализация X-клиента для платформ ОС Windows** [Доклад] / авт. А.О. Малыгин. - СПб : СПбГУ, 2013.

[34] **Цикл статей "Xen Cloud Platform в условиях предприятия"** [В Интернете] // Habrahabr. - http://habrahabr.ru/post/104025/, http://habrahabr.ru/post/104881/, http://habrahabr.ru/post/105262/, http://habrahabr.ru/post/105568/.