Санкт-Петербургский государственный университет

**Факультет прикладной математики – процессов управления**

**Кафедра технологии программирования**

**Иванищев Василий Олегович**

**Выпускная квалификационная работа бакалавра**

**Интроспекционный анализ динамики оперативной памяти операционных систем семейства Windows NT**

**Introspective analysis of Windows NT memory dynamics**

Направление 010400

Прикладная математика и информатика

Заведующий кафедрой,  
кандидат физ.-мат. наук,  
доцент Сергеев С. Л.

Научный руководитель,  
кандидат тех. наук,  
доцент Блеканов И. С.

Рецензент,  
кандидат физ.-мат. наук,  
доцент Шиманчук Д. В.

Санкт-Петербург

2015

**Содержание**

[Введение 3](#_Toc420246238)

[Актуальность работы 3](#_Toc420246239)

[Цель работы 6](#_Toc420246240)

[Задачи работы 7](#_Toc420246241)

[Глава 1. Обзор подходов к извлечению и анализу/мониторингу оперативной памяти 8](#_Toc420246242)

[1.1. Классификации методов 8](#_Toc420246243)

[1.2. Средства анализа образов памяти 9](#_Toc420246244)

[1.3. Средства мониторинга виртуальных систем 10](#_Toc420246245)

[Глава 2. Устройство операционных систем семейства Windows NT. Анализ дампов физической памяти машин под управлением ОС Windows 12](#_Toc420246246)

[2.1. Операционные системы и их типы 12](#_Toc420246247)

[2.2. Физическая и виртуальная память 13](#_Toc420246248)

[2.3. Режим ядра и пользовательский режим 13](#_Toc420246249)

[2.4. Трансляция виртуального адреса в физический 14](#_Toc420246250)

[2.5. Процессы и их потоки 15](#_Toc420246251)

[2.6. Реестр 16](#_Toc420246252)

[2.7. Сетевые соединения 16](#_Toc420246253)

[Глава 3. Архитектура программного комплекса 18](#_Toc420246254)

[3.1. Платформа виртуализации QEMU/KVM 18](#_Toc420246255)

[3.2. Теоретический алгоритм работы 26](#_Toc420246256)

[Заключение 30](#_Toc420246257)

[Результаты работы 30](#_Toc420246258)

[Перспективы развития 30](#_Toc420246259)

[Список литературы 31](#_Toc420246260)

# Введение

## Актуальность работы

В связи с прогрессом в области информационных технологий наблюдается рост программного и аппаратного обеспечения, позволяющего решать многие задачи в различных сферах деятельности человека. Люди стали активно использовать платёжные системы, WEB-ресурсы для передачи данных, запоминающее устройства для записи и хранения важной информации и т.п. Всё это – заслуга компьютерных технологий. Однако в настоящее время активная интеграция современных информационных технологий практически во все области человеческой деятельности привела к тому, что с помощью программно-аппаратных средств и систем всё чаще совершаются разного рода правонарушения и преступления: нарушения в области авторского и смежных прав, хищения денежных средств, мошенничество, лжепредпринимательство, продажи секретной информации и т.п.

Преступления, совершаемые с использованием компьютерных технологий, представляют серьезную угрозу как для коммерческих организаций, так и для государственных структур. Такого рода преступления принято называть компьютерными преступлениями [1]. Наука, занимающаяся исследованием таких преступлений, называется компьютерной криминалистикой (на англ. computer forensics) [1]. Сам термин, форензика, произошёл от латинского «foren», что значит «речь перед форумом». В русский язык это слово пришло из английского. Полная форма этого термина на английском языке звучит следующим образом: «computer forensic science», что дословно означает «компьютерная криминалистическая наука». Согласно определению, компьютерная криминалистика – это прикладная наука о раскрытии преступлений, связанных с компьютерной информацией, об исследовании цифровых доказательств, методах поиска, получения и закрепления таких доказательств, о применяемых для этого технических средствах [1].

Данная область криминалистики полноценно существует в развитых странах: издан ряд научных трудов (Windows Forensic Analysis Toolkit, Harlan Carvey; The Art of Memory Forensics, Michael Hale Ligh, Andrew Case, Jamie Levy, AAron Walters и т. д.) имеются учебные курсы, существуют официальные рекомендации, которым необходимо следовать при криминалистической экспертизе.

В других странах компьютерная криминалистика лишь начинает развиваться. К сожалению, Россия относится как раз к таковым, не смотря на то, что в ней масса компьютерных специалистов высокого уровня. Например, одним из показателей является серийный выпуск программно-аппаратных комплексов, специализированных для сбора, обработки и анализа цифровых доказательств, для обеспечения целостности данных при изъятии и исследовании (например, PCI карта Tribble, проект WindowsSCOPE, LiMe и т. п.). В России такого рода обеспечение только начинает производиться, но всё чаще - закупается. Данное положение дел объясняется несколькими обстоятельствами: слаборазвитыми теоретическим и прикладным основаниями компьютерно-криминалистической науки, малым количеством доступных публикаций по данному направлению, а также не адаптированной к эффективному обучению основам криминалистической науки системой (высшего) образования.

Центральными техническими задачами компьютерной криминалистики является извлечение и анализ информации, хранящейся в памяти компьютера [1]. При этом, первая задача состоит из извлечения данных из: (а) энергозависимой (оперативной) памяти (на англ. - memory acquisition) [2] и (б) энергонезависимых запоминающих устройств (накопителей) (на англ. - data carving) [3].

Следует отметить, что данные на компьютере могут храниться и модифицироваться в следующих частях компьютерной памяти:

* регистры процессора,
* иерархическая кэш-память процессора,
* оперативная память,
* внешняя память.

Первые два типа памяти являются специфичными и частично неадресуемыми для прикладного программного обеспечения, в связи с чем задача достоверного извлечения данных не имеет для них надёжного решения.

Тематика настоящей работы касается класса методов извлечения и анализа оперативной памяти (ОП).

В настоящее время всё большую популярность набирают технологии виртуализации. Растёт пропускная способность физических интерфейсов компьютеров, а также ёмкость и отзывчивость систем хранения данных. В результате, возникает ситуация, когда мощности одного физического сервера можно эффективно отобразить в виртуальную среду на множество эмулированных серверов, гибко масштабируя, таким образом, инфраструктуру под конкретные бизнес-задачи коммерческих организаций и государственных учреждений [4].

Однако, при всех достоинствах виртуальных окружений, имеются и ощутимые недостатки - расследование противоправных действий, совершаемых с использованием виртуализации, часто сопряжено с заметными проблемами сбора допустимых доказательств, обусловленными отсутствием криминалистических средств для работы с виртуальными средами. Аналогично, не в полной мере удовлетворена потребность в средствах мониторинга комплексов виртуализации с целью криминалистически правильного фиксирования инцидентов в контексте компьютерно-технических экспертиз. Обобщая, можно сказать, что проблематика анализа информационной динамики в виртуальных окружениях является одной из самых актуальных сфер компьютерной криминалистики.

В данной работе уделяется внимание одному из аспектов указанной проблематики, а именно - способам извлечения и анализа данных ОП виртуальных машин. Также мы обсудим методы мониторинга памяти виртуальных машин, как побочный продукт технологии анализа ОП.

Направленность работы состоит в формулировании предложений по практическому применению озвученных способов и методов к виртуальным машинам под управлением 64-битных операционных систем семейства Windows NT. (Данный выбор обусловлен тем, что эти операционные системы популярны среди большинства пользователей.)

Как упоминалось выше, существует множество инструментов для задач извлечения/анализа ОП в классической постановке. На первый взгляд, их можно было бы использовать и в нашем случае. К сожалению, это не возможно из-за ряда ограничений, которые являются для нас принципиальными. К ним относятся высокая стоимость, отсутствие открытого исходного кода под либеральными лицензиями и, как следствие, невозможность модификации программных компонентов под определённые задачи, платная техническая поддержка и т.п.

## Цель работы

Целью данной работы является создание архитектуры криминалистического программного комплекса, интегрированного с платформами виртуализации, в контексте проблемы интроспекционного анализа/мониторинга динамики оперативной памяти гостевых операционных систем.

## Задачи работы

Для достижения поставленной цели был сформулирован и решен ряд задач:

* исследование внутреннего устройства физического и виртуального пространства памяти в операционных системах семейства Windows NT (на примере Windows 7, x86\_64);
* обзор известных методов и подходов снятия и анализа образов оперативной памяти;
* разработка архитектуры программного комплекса.

# Глава 1. Обзор подходов к извлечению и анализу/мониторингу оперативной памяти

## 1.1. Классификации методов

Существует множество методик для извлечения и анализа необходимых данных оперативной памяти. Однако, не все из них являются подходящими для корректного сбора доказательств с точки зрения компьютерной криминалистики. В этой связи можно выделить следующие классы методов:

* криминалистически правильный,
* криминалистически сомнительный,
* криминалистически неправильный.

Также, безусловно, важен факт приостановки функционирования оборудования, подвергаемого криминалистическому исследованию, что вводит вторую классификацию:

* методы (средства), не требующие остановки целевой системы (online),
* методы (средства), требующие остановки целевой системы (offline).

Когда речь идёт о технологиях виртуализации, а именно о серверах, являющихся основой виртуальной системы, нецелесообразно рассматривать подходы второй группы по очевидным причинам – система должна функционировать непрерывно и остановка для анализа её данных недопустима.

Ещё один немаловажный аспект – инвазивность исследовательских действий. Если при снятии или анализе информации память целевой системы модифицируется, то данный метод криминалистики недопустим. Такие методы, в лучшем случае, можно отнести к криминалистически сомнительным, хотя, как правило, они являются криминалистически неправильными. Таким образом, вводится третья классификация имеющихся методик – по признаку инвазивности производимых действий:

* инвазивные,
* неинвазивные.

Рассмотрим и классифицируем несколько средств, используемых при снятии физической памяти систем.

*FireWire-копирование.*

FireWire (IEEE 1394A, IEEE 1394B) последовательная высокоскоростная информационная шина, используемая, как правило, устройствами хранения данных и имеет возможность горячего подключения. При снятии данных через интерфейс FireWire внедрение кода в память операционной системы не происходит, поскольку доступ физической памяти осуществляется напрямую, в обход центрального процессора. В этом смысле, метод относится к неинвазивным и online-средствам. Однако, он является криминалистически-сомнительным: копирование одних участков памяти происходит на фоне изменения других, что принципиально не позволяет получить моментальную копию содержимого памяти и нарушает требование к целостности данных при сборе цифровых доказательств.

*Копирование средствами гипервизора QEMU/KVM.*

Данная методика применяется к извлечению памяти виртуальных машин. Она не требует остановки системы, поэтому ее можно отнести к классу online-средств. Кроме того, она неинвазивна, в силу того, что память гостевой операционной системы не подвергается модификации. Наконец, методика криминалистически правильная, т. к. гипервизор гарантирует целостность снимаемого образа.

## 1.2. Средства анализа образов памяти

Одной из центральных задач нашего комплекса является анализ образов оперативной памяти. Есть несколько программных инструментов, которые решают данную проблему (Belkasoft Evidence Center, MANDIANT Memoryze и т. п.). Но ввиду их недостатков – высокой стоимости, отсутствия открытого исходного кода – использовать их не представляется возможным.

Отдельно хотелось бы отметить программное решение – Volatility Framework. Оно производит анализ образов оперативной памяти и, что немаловажно, является бесплатным программным обеспечением с открытым исходным кодом. Однако, есть ряд причин, из-за которых Volatility подходит только в качестве обучающего инструмента, а не базового функционального блока в сторонних проектах. Во-первых, это лицензия, под которой разрабатывается Volatility – GNU GPL v2 – она имеет ряд ограничений по сравнению с более либеральными MIT или BSD 2-clause «Simplified». Во-вторых, данное обеспечение написано на языке программирования Python и не позволяет использовать его нативный API в приложениях, написанных на других языках. Также, в ходе тестирования данного инструмента, было выявлено, что он не полностью приспособлен к работе с дампами (на англ. dump) памяти, получаемыми при помощи библиотеки libvirt.

## 1.3. Средства мониторинга виртуальных систем

Вторая, более глобальная задача, включающая в какой-то степени первую, – это наблюдение (или мониторинг) виртуальных машин. В данной области тоже имеется множество программных средств (Veeam One, Naumen и т. п.). Но, к сожалению, перед большинством из них стоит иная цель – наблюдение за производительностью систем. Ещё один недостаток этих средств – многие из них способны работать только с определёнными гипервизорами. Задача же настоящей работы состоит в том, чтобы создать архитектуру универсального комплекса, позволяющего, среди прочего, наблюдать за состоянием виртуальных машин с целью выявления нестандартных активностей в оперативной памяти.

Поэтому в данной работе будет уделено внимание созданию именно такого криминалистического программного инструмента, интегрированного с платформами виртуализации (прежде всего QEMU/KVM).

# Глава 2. Устройство операционных систем семейства Windows NT. Анализ дампов физической памяти машин под управлением ОС Windows

Перед непосредственным проектированием программного комплекса следует предоставить общие сведения по устройству операционных систем семейства Windows NT и методике анализа образов их памяти.

**2.1. Операционные системы и их типы**

Операционная система — это комплекс взаимосвязанных программных компонентов, посредством которого пользователи и их программы взаимодействуют с аппаратным обеспечением. Она предоставляет интерфейсы для пользователей и выполняющихся приложений. Программы пользователей, служебные программы, различные приложения взаимодействуют с аппаратурой компьютера только через обращение к операционной системе (точнее, к ядру операционной системы). Большинство операционных систем строятся из множества программных модулей. Существует группа «основных» управляющих модулей, вместе с некоторыми структурами системных данных она образует ядро операционной системы – центральную и основную часть системы [5].

В зависимости от структурных особенностей ядра, операционные системы бывают различных типов: микроядерные и макроядерные (монолитные). В микроядерных операционных системах само ядро очень компактно, а остальные модули вызываются из ядра как сервисные. В отличие от микроядерной архитектуры, в макроядерных операционных системах главная «управляющая» часть включает в себя большое количество модулей и использует гораздо больше памяти [5]. Ядро Windows NT сочетает в себе элементы как микроядерной архитектуры, так и монолитной. Однако, Windows NT часто называют микроядерной операционной системой, хотя это не совсем так. Данное ядро слишком велико, чтобы отнести его к типу микроядер, да и работа компонент ядра в одном адресном пространстве свойственна системам с монолитным ядром. С другой стороны, расположение и взаимодействие элементов ядра происходит точь-в-точь как в микроядерных системах. Такой тип организации ядра называется «гибридное ядро». Именно к этому, третьему, типу относится семейство операционных систем Windows NT [6].

## 2.2. Физическая и виртуальная память

В данной работе будут использоваться два термина – виртуальная память и физическая память. Под физической памятью подразумевается оперативная память, представляющая из себя упорядоченный набор однобайтовых ячеек, каждая из которых имеет свой уникальный адрес. Совокупность адресов в физической памяти, используемых для идентификации хранимых в памяти данных, называется физическим адресным пространством [7]. Виртуальная память – это абстрактное хранилище, созданное самой операционной системой для удобства управления физической памятью. Виртуальная память позволяет программам считать, что им предоставлена вся теоретически доступная память, организованная в непрерывное адресное пространство.

В операционной системе Windows виртуальная память имеет сегментно-страничную организацию. Адресное пространство процесса представляется в виде набора сегментов переменного размера. При этом каждый сегмент делится на страницы – блоки фиксированного размера, в то время как физическая память делится на блоки того же размера – фреймы (кадры) [7].

## 2.3. Режим ядра и пользовательский режим

Чтобы предотвратить ситуации, когда пользовательские приложения получают доступ или вносят изменения в важные системные данные в Windows используются два процессорных режима доступа: пользовательский режим и режим ядра. Пользовательская программа функционирует в пользовательском режиме, а код операционной системы (например, системные службы и драйверы устройств) запускается в режиме ядра. Режим ядра — режим работы процессора, в котором имеется неограниченный доступ к системной памяти и внешним устройствам [8].

Каждая страница в виртуальной памяти имеет пометку, показывающую, в каком режиме доступа должен быть процессор для чтения или записи этой страницы. Таким образом, происходит разделение виртуальной памяти операционной системы: пространство ядра (зарезервировано для работы ядра, его расширений и некоторых драйверов устройств) и пространство пользователя (в нём функционируют все приложения пользователя).

Как было сказано выше, большинство пользовательских приложений работает в режиме пользователя, однако, они могут осуществить переключение из пользовательского режима в режим ядра с помощью системных вызовов. Такой переход осуществляется за счёт использования специальной инструкции процессора. Перед тем, как вернуть управление пользовательскому приложению, процессор переключается в прежний режим [8].

## 2.4. Трансляция виртуального адреса в физический

При анализе оперативной памяти необходимо уметь восстанавливать адресное пространство процесса, то есть производить преобразование известных виртуальных адресов в соответствующие им физические, получив, тем самым, данные всех страниц виртуальной памяти из фреймов оперативной памяти.

Для операционных систем Windows 7 x86-64 трансляция виртуального адреса в физический производится посредством четырёхуровневой табличной структуры преобразований. Виртуальный адрес разделен на пять частей: селектор четвертого уровня отображения страницы, селектор указателя каталога страниц, селектор таблицы страниц, селектор записи таблицы страниц и байтовое смещение. При помощи этих указателей осуществляется преобразование адресов. В процессе преобразования могут попадаться недостоверные записи в таблицах страниц, поэтому необходимо учитывать специфику состояний найденных страниц [9].

Анализ образа оперативной памяти предполагает поиск артефактов (сущностей) операционной системы, который осуществляется путём паттерн-анализа (на англ. pattern – образец). Его суть заключается в поиске определённого набора байтовых паттернов в образе памяти. Выбор паттерна зависит от сущности, которую необходимо обнаружить. Их классическими примерами являются список процессов, деревьев реестра и список сетевых соединений.

## 2.5. Процессы и их потоки

Первое, с чего следует начать анализ дампа оперативной памяти операционной системы – это поиск списка активных процессов. Каждый процесс в ОП представлен структурой \_ERPOCESS, хранящейся в адресном пространстве ядра. У данной структуры есть множество полей и подструктур, первая из них – подструктура \_KPROCESS – блок управления процессом, а он, в свою очередь, начинается с субструктуры \_DISPATCHER\_HEADER. Значения некоторых полей, которые она содержит, постоянны для всех процессов данной операционной системы. Точнее, такими полями являются Type и Size. Таким образом, поиск процессов в оперативной памяти сводится к поиску соответствующего байтового паттерна в адресном пространстве ядра.

Список результатов будет гораздо длиннее реального списка активных процессов, но после проверки определённых условий на адреса потоков процесса и поля DTB, которое совпадает с значением управляющего регистра CR3, часть «кандидатов» будут исключены.

После нахождения смещения структур \_EPROCESS не составит труда найти информацию о процессах (уникальный идентификатор, имя, время создания и т. п.) и их потоках (число активных потоков, список всех потоков и т. п.).

## 2.6. Реестр

Реестр – иерархически организованная база данных параметров операционной системы, содержащая информацию об аппаратном и программном обеспечении и его настройки, профили пользователей и т. д. [10]. Реестр представляет собой набор файлов в формате Windows NT Registry File (REGF), которые называются хайвами (от англ. hive). Каждый хайв содержит дерево реестра и представлен в памяти структурой \_CMHIVE. В данной структуре содержится множество метаданных, ценных для анализа. Для поиска данной структуры можно воспользоваться выше упомянутым методом поиска соответствующего байтового паттерна в адресном пространстве ядра. В данном случае байтовым паттерном будет сигнатура, с которой начинается подструктура \_HHIVE структуры \_CMHIVE.

Каждый хайв устроен таким образом, что данные внутри него хранятся в ячейках (на англ. cells). По мере загруженности системы, реестр Windows разрастается, и, т. к. всё это происходит в реальном времени, в физической памяти отображение реестра хранится не равномерно, а с промежутками, т. е. фрагментировано. Поэтому для того, чтобы получить полную информацию о реестре необходимо осуществить «трансляцию» адресов ячеек [8].

## 2.7. Сетевые соединения

Информация о сетевых соединениях включает в себя идентификатор процесса, время создания, установленные соединения, локальный и удаленные адреса, локальный и удаленный порты. Извлечение такой информации осуществляется аналогично описанным выше случаям – с помощью байтового поиска соответствующих структур. В этом случае объектами исследования будут структуры TCB и TcpEndpoint. В роли байтового паттерна используется сигнатура этих структур. Произведя поиск данных сущностей, мы получим необходимую информацию [11].

# Глава 3. Архитектура программного комплекса

Перед тем, как перейти к разработке архитектуры комплекса требуется изучить характеристики и особенности платформы виртуализации QEMU/KVM.

## 3.1. Платформа виртуализации QEMU/KVM

KVM (Kernel-based Virtual Machine) – это программное решение, обеспечивающее поддержку технологий виртуализации в среде Linux. Проект KVM зарекомендовал себя как надёжное программное обеспечение и входит в основную ветку разработки ядра Linux с 2007 года. Сам по себе KVM не выполняет эмуляции, он просто позволяет использовать аппаратную виртуализацию на уровне ядра.

Для непосредственной эмуляции виртуальной машины необходим гипервизор. Для данной цели подходит QEMU (Quick Emulator) – бесплатный гипервизор с открытым исходным кодом, поддерживающий аппаратную виртуализацию. Первоначально он был разработан французским программистом Фабрисом Белларом (Fabrice Bellard). Сегодня проект развивается при поддержке обширного международного сообщества и тесно связан с проектом KVM.

В совокупности, данные два продукта позволяют использовать виртуализацию и выполнять эмуляцию виртуальных машин. В дальнейшем, в данной работе, под «гипервизором» будет пониматься совокупность двух данных программных решений.

Далее, рассмотрим подробнее работу гипервизора.

Виртуальная машина, или таргет-платформа (от англ. target – целевая), создаётся путём запуска процесса QEMU. Теоретически, QEMU позволяет запустить одновременно любое число виртуальных машин, ограниченное исключительно ресурсами хостa (от англ. host). Каждая из виртуальных машин с гостевой операционной системой будет определяться своим QEMU-процессом на хост-платформе (см. рис. 3.1.).

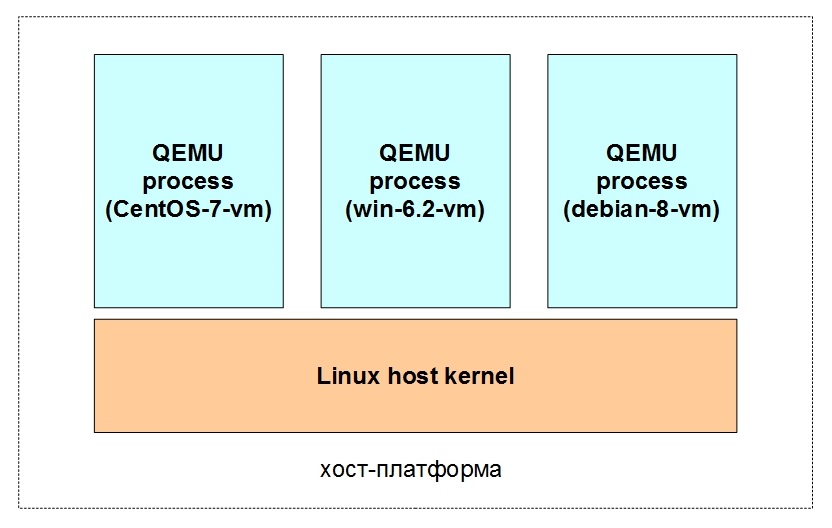


Рис. 3.1. Процессы QEMU на хост-платформе.

Когда виртуальная машина выключается, процесс QEMU завершается. Перезагрузка гостевой ОС выполняется без перезапуска процесса QEMU.

После того, как запускается QEMU-процесс, происходит выделение и резервирование фиксированного блока виртуальной памяти (адресной арены) хост-процесса под оперативную память виртуальной машины, т.е. «физической» памяти для гостевой ОС (см. рис. 3.2.).

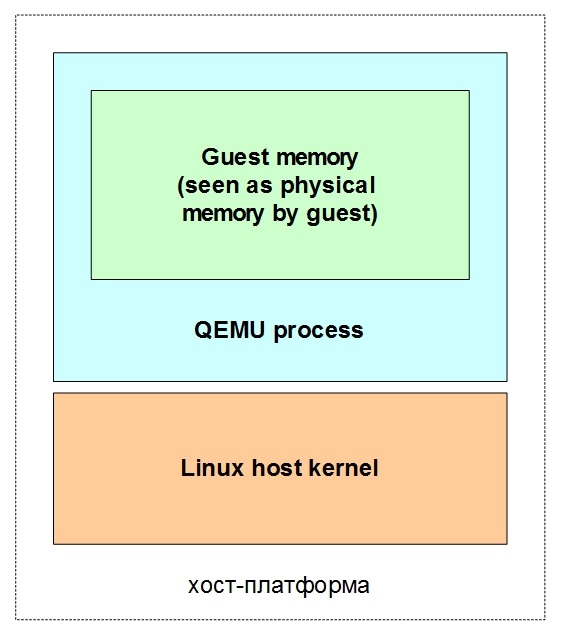


Рис. 3.2. Физическая память гостевой операционной системы.

Виртуальная память гостевой ОС в свою очередь отображается на вышеупомянутую «физическую», штатным для гостевой системы способом.

Понятно, что в случае нескольких виртуальных машин у каждой будет своя «физическая память», каждая из которых будет отображаться на общую физическую память хост-платформы (см. рис. 3.3.).

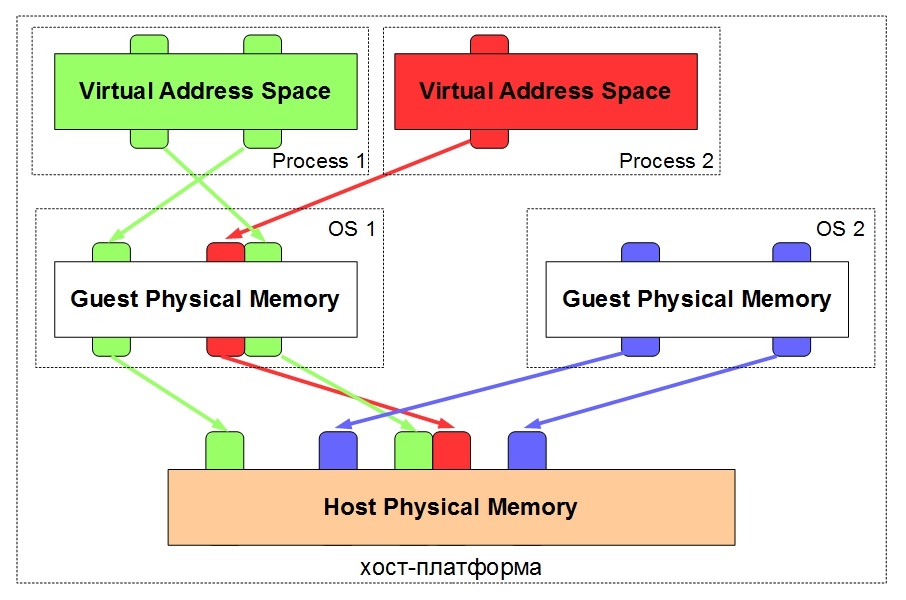


Рис. 3.3. Отображение памяти в случае нескольких виртуальных машин.

QEMU поддерживает два вида записи байтов (порядка байтов) целевых платформ: от старшего к младшему (на англ. big-endian или «тупоконечный») и от младшего к старшему (на англ. little-endian или «остроконечный»). Поэтому осуществляя доступ к памяти гостевой ОС необходимо учитывать байтовый порядок. Endian-преобразования выполняются вспомогательными функциями, а не путём доступа напрямую к оперативной памяти виртуальной машины.

Как было сказано выше, KVM – это функция виртуализации в ядре Linux, которая позволяет таким программам, как QEMU безопасно выполнять код гостевых систем непосредственно в физической памяти сервера виртуализации. Это возможно только тогда, когда архитектура целевой платформы поддерживается процессором хостa.

Для того, чтобы выполнить код гостевой ОС с помощью KVM, процесс QEMU открывает директорию /dev/kvm и совершает системный вызов KVM\_RUN ioctl (input/output control – системный вызов потока ввода/вывода). Ядро модуля KVM использует аппаратные расширения виртуализации, поддерживаемые современными процессорами Intel и AMD для того чтобы напрямую исполнять код гостевой машины. Когда «гость» получает доступ к устройству хост-системы, процессор виртуальной машины останавливает свою работу (или выполняет специальные операции), а KVM передаёт управление QEMU. В этот момент QEMU может сэмулировать желаемый результат операции или просто ждать следующего прерывания сеанса гостевой ОС в случае остановки виртуального процессора.

Всё вышесказанное можно проиллюстрировать фрагментом псевдокода. Основной поток гостевого процессора выглядит следующим образом:

open("/dev/kvm")

ioctl(KVM\_CREATE\_VM)

ioctl(KVM\_CREATE\_VCPU)

for (;;) {

ioctl(KVM\_RUN)

switch (exit\_reason) {

case KVM\_EXIT\_IO: /\* ... \*/

case KVM\_EXIT\_HLT: /\* ... \*/

}

}

Для хост-платформы QEMU – это обычный процесс, такой же как sshd или bash, которые используют общие ресурсы хост-системы. При создании серверов виртуализации, однако, процессы QEMU все же выделяют в отдельную группу по:

* приоритету доступа к ресурсам,
* смежному расположению адресных пространств процессов.

Поскольку система эмуляции QEMU разработана так, что запущенная виртуальная машина располагается целиком и непрерывно (без разделения на функциональные блоки) в адресном пространстве процесса QEMU, такие детали, как, например, запущенные процессы внутри гостевой ОС не видны приложениям хост-платформы. Это сделано из соображений безопасности и оптимизации производительности виртуальных машин.

Каждая виртуальная машина имеет так называемый VCPU-поток для каждого эмулируемого процессора. Выделенный поток iothread запускает цикл обработки событий (event loop) для обработки пакетных операций сетевых соединений, дисковых операций ввода/вывода и т. п. (см. рис. 3.4.).

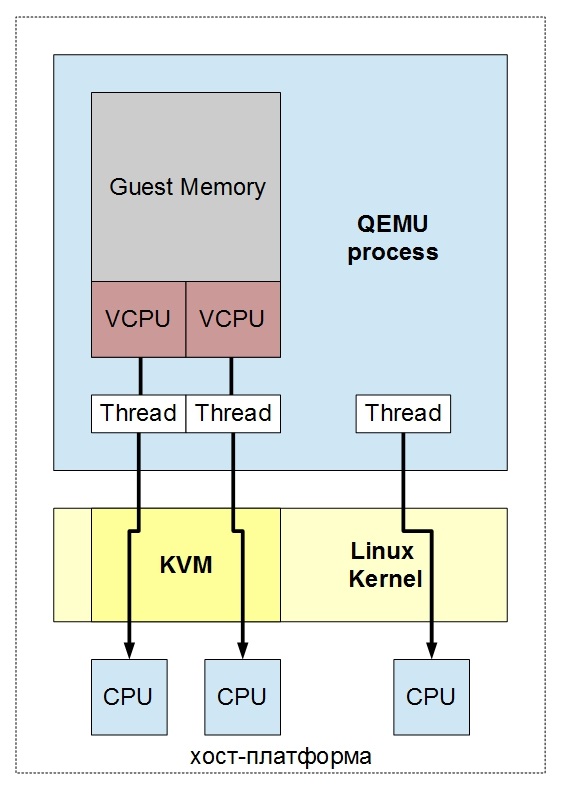


Рис. 3.4. Особенности эмуляции процессоров KVM.

Следующая схема иллюстрирует, как виден QEMU процесс для хост-платформы в общем случае:

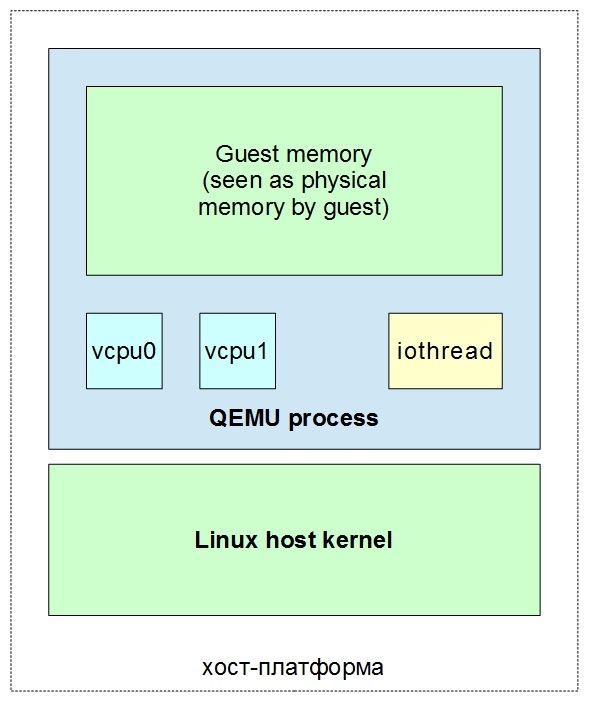


Рис. 3.5. QEMU процесс для хост-платформы в общем виде.

Запуск гостевой ОС включает выполнение гостевого кода, обработки таймеров, выполнение ввода/вывода и ответы на команды монитора?. Делая все эти вещи сразу требуется архитектура, способная опосредовать ресурсы безопасным способом, не останавливая работу гостевой ОС в случае, когда диск ввода/вывода или монитор команд требуют длительного времени для выполнения. Есть две популярных архитектуры программ, которые должны реагировать на события из нескольких источников:

• Параллельная архитектура разделяет работу по процессам или потокам, которые могут выполняться одновременно. Эта архитектура будет называться «потоковая архитектура»

• Событийная архитектура реагирует на события путём выполнения главного цикла, что направляет в обработчики событий. Это обычно осуществляется с помощью системных вызовов select(2) или poll(2) для ожидания нескольких файловых дискрипторов

QEMU на самом деле сочетает в себе событийно-управляемое программирование с потоками. Это имеет смысл потому что цикл событий не может воспользоваться несколькими ядрами, так как он имеет только один поток исполнения. Кроме того, иногда проще написать специальный поток, чтобы разгрузить одну конкретную задачу, а не интегрировать его в архитектуру управляемых событий. Тем не менее, ядро QEMU является событийно-управляемым и большинство кода выполняется в этой среде.

Событийно-управляемая архитектура сосредоточена вокруг цикла событий, который отправляет события в функции-обработчики. Основной цикл событий QEMU это main\_loop\_wait() , он выполняет следующие задачи:

• Ожидает файловые дискрипторы, чтобы стать возможным для чтения/записи. Файловые дискрипторы играют решающую роль, потому что файлы, сокеты, ???, и другие различные ресурсы являются файловыми дискрипторами. Файловые дискрипторы могут быть добавлены с помощью qemu\_set\_fd\_handler().

• Запускает ??? таймеры. Таймеры могут быть добавлены с помощью qemu\_mod\_timer().

• Запускает bottom-halves, которые, как таймеры, истекают сразу???. BHs используются чтобы избежать повторного входа и переполнения стека вызовов. Они могут быть добавлены с помощью qemu\_bh\_schedule ().

Когда дескриптор файла будет готов, таймеры и BH планируются, цикл событий вызывает функцию обратного вызова, которая реагирует на события. Обратная связь имеет два простых правила относительно их окружения:

• Ни один другой код ядра не будет выполнятся поскольку в синхронизации нет необходимости. Обратные вызовы выполняются последовательно и атомарно по отношению к другому коду ядра. Существует только один поток управления, который выполняет код ядра в любой момент времени

• Не блокируемые системные вызовы или долго выполняющиеся вычисления должны быть выполнены??? Так цикл событий ждёт обратного вызова для того, чтобы вернуться прежде чем продолжить работу с другими событиями, важно не тратить неограниченное количество времени на функции обратного вызова. Нарушение этого правила влечёт к остановке гостевой ОС и монитор становится неспособным отвечать на запросы???

Второе правило иногда трудно соблюдать и есть код QEMU который это блокирует???. На самом деле есть ещё цикл, вложенный в qemu\_aio\_wait(), который ждёт подмножество событий, которые обрабатывает цикл событий верхнего уровня. Надеемся, эти нарушения будут исправлены в будущей реструктуризации кода. Новый код практически никогда не имеет законных оснований, чтобы использовать блокировку, и одно из решений заключается в использовании специализированных рабочих потоков чтобы разгрузить длительное выполнение или блокировку кода.

…

После того, как известны общие принципы работы QEMU/KVM, необходимо преступить к описанию других архитектурных особенностей программного комплекса.

## 3.2. Теоретический алгоритм работы

Для начала следует упомянуть, что управлять виртуальными машинами, используя команды QEMU напрямую представляется нерациональным ввиду: (а) перспективы использования в качестве бекенд-гипервизора не только QEMU/KVM; (б) автоматизации менеджмента виртуальными окружениями; (в) унификации бизнес-логики анализа и мониторинга. Поэтому было решено применить библиотеку управления виртуализацией – libvirt, т. к. на данный момент она поддерживает большинство известных гипервизоров и унифицирует управление ими через обобщенный API. Благодаря этому, программный комплекс становится универсальным и независящим от конкретного гипервизора.

Таким образом, наш программный инструмент, решая первую задачу – снятие дампа физической памяти, будет использовать интерфейс libvirt. Она, в свою очередь, используя непосредственно команды гипервизора, будет управлять виртуальными машинами.

Алгоритм действий разрабатываемого программного комплекса может быть описан следующим образом:

1. *Снятие дампа памяти.*

Снятие дампа физической памяти виртуальной машины производится при помощи команды libvirt - create snapshot. Стоит подчеркнуть, что данное действие производится «наживую», т. е. без приостановки гостевой операционной системы.

1. *Подготовка дампа памяти к анализу.*

На этом этапе выделяется пространство в оперативной памяти хост-платформы. Далее происходит загрузка дампа в данную область оперативной памяти. Фактически, снятие дампа происходит непосредственно в оперативную память хоста. Это позволяет добиться высоких результатов производительности.

1. *Паттерн-анализ дампа памяти.*

На данном этапе происходит поиск сущностей операционной системы в дампе памяти с помощью методов, описанных ранее в данной работе.

1. *Анализ результатов поиска.*

Подразумевается, что на входе мы имеем ряд правил, которым данный комплекс будет следовать. Поэтому на этапе анализа производится выявление нестандартных активностей и проверка условий, с целью получения дальнейших инструкций системе.

Следует заметить, что все действия производятся неинвазивно, поскольку всё взаимодействие с гостевой ОС происходит через гипервизор. Поэтому такая система будет корректной с точки зрения компьютерно-криминалистического анализа.

Разработанная архитектура программного комплекса изображена на рис. 3.6.

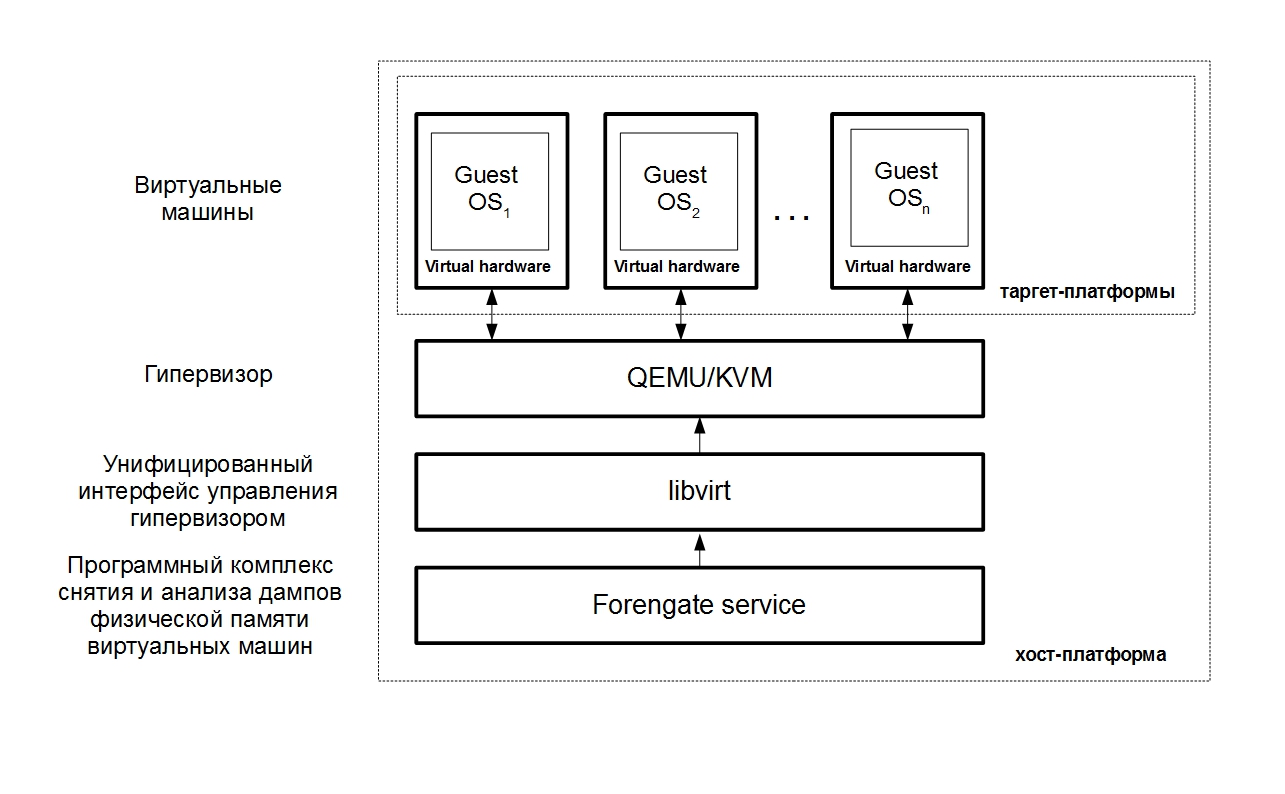


Рис. 3.6. Архитектура программного комплекса на первом этапе работы.

На первом этапе разработки архитектуры программного комплекса был реализован доступ к физической памяти гостевых машин последовательно, через libvirt и QEMU/KVM. Разумеется, хотелось бы получить доступ к памяти напрямую с хост-платформы, не используя никаких «прослоек». С одной стороны, мы могли бы сделать форк (на англ. fork - системный вызов, создающий новый процесс, который является практически полной копией процесса-родителя) QEMU процесса и работать с ним. Но ввиду сегодняшней имплементации QEMU, а именно многопоточности процессов, мы не можем использовать данный подход, поскольку процесс-потомок, получившийся в результате форка, может быть «мёртвым»???. Поэтому самый простой способ решить данную проблему – это воспользоваться командами QEMU – что и было реализовано на первом шаге разработки программного комплекса.

Поэтому в дальнейшем планируется проанализировать устройство отображения памяти гостевых ОС на память сервера виртуализации, что позволит нам делать одномоментные снимки памяти напрямую с хост-платформы.

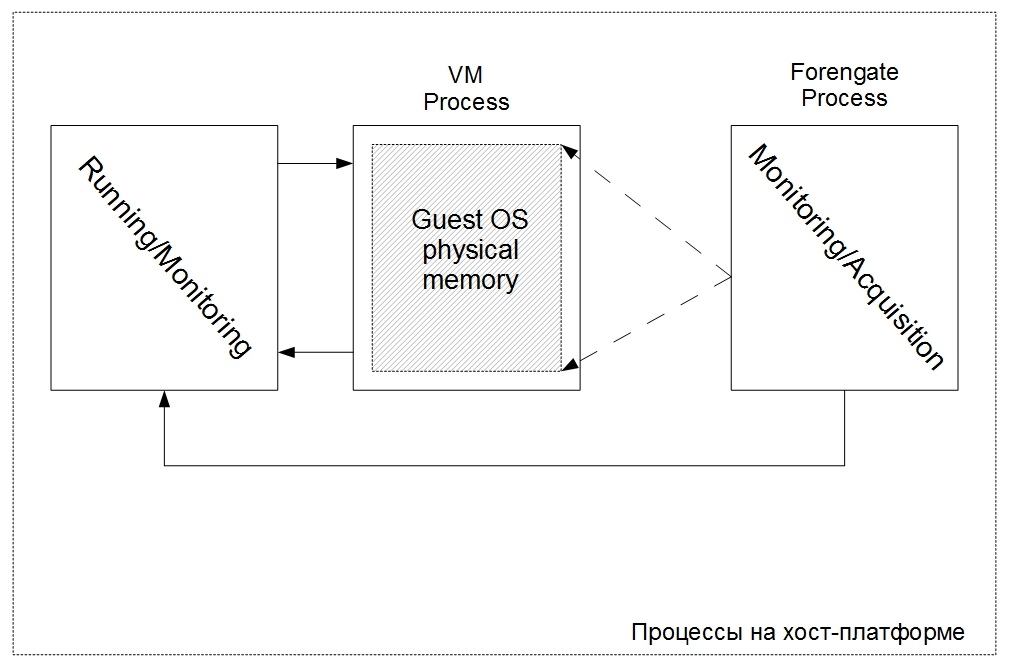


Рис. 3.7. Схема работы комплекса в виде процессов на хост-платформе. Связи пунктирными линиями – планируется реализовать.

# Заключение

## Результаты работы

В данной работе рассмотрены и классифицированы подходы и средства извлечения и анализа оперативной памяти. Также было исследовано внутреннее устройство операционных систем семейства Windows NT с последующим анализом дампов их оперативной памяти.

Наконец, разработана архитектура программного комплекса, интегрированного с платформами виртуализации и позволяющего производить криминалистически правильный неинвазивный анализ динамики оперативной памяти гостевых операционных систем.

## Перспективы развития

В работе был представлен подход к анализу оперативной памяти средствами гипервизора. Действительно, этот метод имеет место быть, поскольку он проверен и не вызывает сомнений. Но это лишь первый шаг к разработке унифицированного программного комплекса.

В дальнейшем, после изучения отображения физической памяти гостевой операционной системы на память сервера виртуализации, появится возможность избежать использования средств гипервизора для снятия образов. Это значительно увеличит производительность комплекса и сделает его независимым от каких-либо интерфейсов и прослоек в виде библиотек управления гипервизором и самого гипервизора. Таким образом, появится возможность снимать одномоментные образы памяти в обход гипервизора.

# Список литературы

1. Федотов Н.Н. Форензика – компьютерная криминалистика. Москва: «Юридический мир», 2007. 360 с.

2. Michael H. L., Andrew C., Jamie L., AAron W. The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Indianapolis: John Wiley & Sons, Inc., 2014. 886 p.

3. Digambar P, Bhadran V.K., Forensic Data Carving, Springer Berlin Heidelberg, 2011. pp 137-148.

4. Обзор методов виртуализации, архитектур и реализаций. [https://www.ibm.com/developerworks/ru/library/l-linuxvirt/index.html](http://www.ibm.com/developerworks/ru/library/l-linuxvirt/index.html)

5. Гордеев А. В. Операционные системы: Учебник для вузов. 2-е изд. СПб: Питер, 2007. 416 с.

6. MS Windows NT Kernel-mode User and GDI White Paper. <https://technet.microsoft.com/library/cc750820.aspx>

7. Виртуальное адресное пространство процесса. <http://www.intuit.ru/studies/courses/1089/217/lecture/5601>

8. Russinovich M., Solomon D., Ionescu A. Windows Internals, Part 1, Sixth Edition. Redmond, Washington: Microsoft Press, 2012. 726 p.

9. Russinovich M., Solomon D., Ionescu A. Windows Internals, Part 2, Sixth Edition. Redmond, Washington: Microsoft Press, 2012. 645 p.

10. Сведения о реестре Windows для опытных пользователей. <https://support.microsoft.com/ru-ru/kb/256986>

11. Lijuan. X, Lianhai. W, Shuhui. Z, Hengjian. L. A Method to Analyze Memory Images of 64-bit Windows 8, pp 304-312.