Санкт-Петербургский Государственный Университет

Математико-Механический Факультет

Кафедра системного программирования

Инструментирование ядра Linux для сбора статистики об использовании файловых систем пользователем

Курсовая работа студента 345 группы

Новожилова Евгения Алексеевича

Научный руководитель к.т.н. Козловский В.С.

Санкт-Петербург

2013

Оглавление

1. Введение

В современном мире сильно распространен подход, основанный на отслеживании специфики использования продуктов компаний определенными группами пользователей для улучшения качества этих продуктов. Можно заметить, что такие компании, как Google и Amazon, имеют большие отделы сотрудников, занимающихся анализом пользовательского траффика. Компании, занимающиеся системами хранения данных также заинтересованы в анализе данных хранящихся на производимых ими устройствах. И на данный момент очень остро стоит проблема описания использования систем хранения данных. Сейчас, большие объемы информации, хранящиеся на массивах хранения данных описаны термином Big Data, и существует много исследований, посвященных анализу Big Data. Основной задачей данных исследований является изучить характер использования массивов хранения данных, чтобы улучшить их производительность и снизить операционные расходы. Большинство пользователей можно сгруппировать по конкретным промышленным отраслям. По статистике, каждая из этих групп использует ограниченный набор приложений, что дает возможность описать особенности этих групп и самих приложений с помощью существующих методов машинного обучения.

Данное исследование является первым прототипом для эмуляции системы хранения данных и первоначальных экспериментов для выявления закономерностей в поведении пользовательских приложений. Инструментирование позволит получить знания о использовании пользователями файловых систем и применить их в дальнейшем для анализа характера нагрузок.

Результаты, полученные после проведения данного исследования, должны позволить убедиться в правильности идеи о зависимости поведения приложения от его типа, что даст возможность в будущем охарактеризовать поведение приложений и отличать их друг от друга. Применение этой идеи в практике позволит охарактеризовать пользователей систем хранения данных и предлагать им системы хранения данных, сконфигурированные под их нужды.

Также, достаточно подробное знание поведения приложения позволит в ситуации отсутствия реальных приложений синтетически генерировать иммитационную нагрузку с заданными параметрами.

Цель работы - инструментировать ядро Linux для сбора информации об использовании файловых систем пользователями и провести базовый анализ корректности инструментирования.

2.Основные идеи

Как упомналось ранее, для последующих исследований необходимо извлечь из контроллера доступа к файловой системе набор данных, максимально отражающий поведение программы. Для достижения этой цели следует изучить стек ввода-вывода данных с физических носителей в ядре Linux и реализовать дополнительный модуль для сбора и экспорта данных.

Стоит отметить, что реализации всех файловых систем изначально находятся в пространстве ядра и вывод информации из него в пользовательское пространство преставляет собой отдельную задачу. Учитывая тот факт, что операции экспорта данных из пространства ядра являются ресурсо-затратными по времени, то для того, чтобы сохранить скорость работы операционной системы и общую производительность операций ввода-вывода необходимо найти минимальный репрезентативный набор параметров для экспорта.

Основными требованиями к рабочей системе являются:

1. **Расширяемость**

   Расширяемость системы является наиболее важной частью данного исследования, в связи с тем, что существующие решения этой задачи основаны на инструментировании каких-то определенных файловых систем (EXT2, NFS). Помимо прочего, существующие решения предоставляют крайне различающийся набор данных, который сложно применить в последующем анализе.

2. **Скорость работы**

   Система должна сохранить скорость работы для получения максимально правдивых результатов работы реальных приложений

3. **Удобство анализа**

   Данные, собранные после инстументирования, должны иметь возможность быть обработанными в любой среде. К примеру, статистические системы, такие как R, Matlab, GNU Octave, поддерживают работу с входными файлами как в бинарном формате, так и tab-separated формате.

4. **Деперсонификация данных**

Инструментируя стек ввода-вывода, мы получаем доступ к данным пользователя, которые нельзя раскрывать. Полученные данные могут содержать личную информацию пользователей, к примеру, в таких местах как названия файлов. Тестовая система должна деперсонифицировать пользовательские данные, сохранив паттерны приложений, для того, чтобы избежать проблем с законодательствами стран в будущем.

3. Решение задачи

В качестве экспериментальной установки был выбрал дистрибутив OpenSUSE 12.1 и ядро Linux версии 3.4.4. Данный дистрибутив был установлен на виртуальную машину, на которой производилось тестирование.

Исследование архитектуры стека файлового ввода-вывода

Реализация стека файлового ввода-вывода в ядре Linux состоит из 4 основных компонентов, которые обеспечивают работу данной подсистемы, таких как обработчик системный вызовов, виртаульная файловая система, драйверы реальной файловой системы, и драйверы блочного устройства. Подробнее устройство подсистемы показано на рисунке ниже.

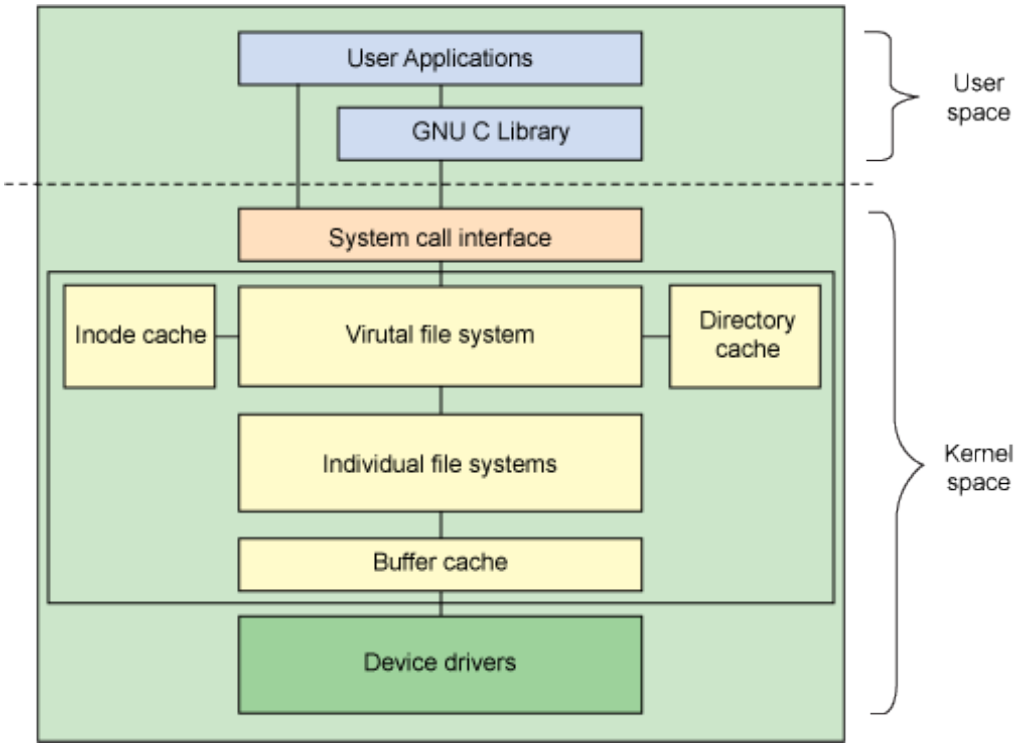

Рис.1

Инструментирование драйверов устройств блочного уровня не подходит для реализации цели из-за отсутствия расширяемости данного решения в принципе, потому что данный уровень является самым низким в архитектуре. В драйвере блочного устрройства практически невозможно отследить обращения к какой-либо определенной файловой системе, Также на

данном уровне статистика о работе приложения будет искажена в связи с использованием кэша в устройствах хранения данных.

Следующим возможным уровнем для инструментирования является непременно драйвер файловой системы. Большинство прежних исследований, относящихся к анализу поведения приложений, основывалось на сборе данных с этого уровня. Инструментирование драйвера файловой системы намного проще драйвера блочного устройства, но у такого решения остается недостаток в виде привязки к определенному типу файловой системы.

Третий уровень абстракции от физического устройства, который подвергся инструментированию в данной работе, называется Виртуальная Файловая Система (Virtual File System - VFS). Она предоставляет интерфейс к различным файловым системам для системных библиотек. VFS перенаправляет через себя всевозможные операции, среди которых больший интерес для данного исследования представляют операции чтения, записи, открытия и закрытия файлов. Таким образом, через данный интерфейс проходят все запросы к файловым системам от различных реализаций системных библиотек, таких как glibc, eglibc, newlib и т.д. Основные понятия, которыми оперирует VFS при обращениям к файлам и которые использованы в данной работе являются :

- **file** – объект, содержащий всю информациб о файле в файловой системе, такую как
    - Точка монтирования файловой системы с данным файлом
    - **inode** – уникальный идентификатор файла в файловой системе. Понятие inode является одним из основных в области файловых систем
    - **dentry** – идентификатор каталога в файловой системе, которому принадлежит данный файл
    - **file_operations** – методы, реализованные для работы с этим файлом
- **super_block** – объект, содержащий самую важную информацию о файловой системе, как ее тип, размер и текущий статус
- **offset** – смещение указателя в файле, по которому производится запись/чтение
- **count** – количество байт, к которм производится доступ, по известному смещению

Для отслеживания были выбраны самые часто используемые операции при обращениям к файлам, такие как:

- **Открытие/закрытие файла**
- **Чтение из файла**
- **Запись в файл**

Реализация

После проведенного исследования архитектуры стека ввода/вывода данных с физических носителей в ядре Linux было принято решение о добавлении модуля-монитора к виртуальной файловой системе, задачей которого было передавать данные из простанства ядра в пространство пользователей.

Операционная система Linux предоставляет некоторые интерфейсы для обмена данными между пространством ядра и простарнством пользователя. Данные интерфейсы используются для предоставления пользователю системной информации, к примеру такой, как информация об оперативной памяти в компьютере, которую можно получить из /proc/meminfo. Существующие интерфейсы можно разделить на 2 группы, различающиеся по методу предоставления информации. Первая группа предоставляет доступ к данным по модели обращения к файлу. Такое решение возможно благодаря созданию виртуальных файловых систем в ядре Linux, где в будущем создаются файлы, в которые процессы ядра экспортируют специфичные для них данные. Примеры такой реализации - файловые системы /sys и /proc. В основе другого подхода к экспорту информации лежит идея BSD сокетов, которые передают информацию к получетелю по моменту ее появления. Данную идею реализует компонент ядра Linux netlink sockets.

Реализации виртуальных файловых систем обладают недостатком в виде отсутствия генерации сигналов о появившейся информации. А также возможности экспорта информации при помощи виртуальных файловых систем ограничены по объему экспортируемой информации. В силу аритекрутрных решений в ядре Linux, данные варианты не позволяют экспортировать более одной страницы памяти за раз. Описанные интерфейсы были созданы для хранения информации в человеко-читаемом виде создаваемой один раз.

В отличие от виртуальных файловых систем, netlink сокеты поддерживают генерацию сигналов о появившейся информации, что позволяет получать информацию из ядра по мере ее появления. Адресация сообщений происходит на основе идентификаторов процессов в ОС GNU\Linux. Для обмена сообщениями в ходе работы был определен собственный протокол netlink сокетов – NETLINK_INFOCOLL.

Для того, чтобы отфильтровывать данные, касающиеся только целевой файловой системы, была реализована проверка на наличие ключа монтирования у файловых систем '–o infocoll'.

Алгоритм обмена сообщениями

Идея обмена сообщениями на основе сокетов заключатся в наличии двух или более субъектов, между которыми происходит обмен. В текущей реализации системы существует 2 субъекта:

- Клиент – приложение, запущенное в пространстве пользователя, которое обрабатывает сообщения, полученнные из ядра и записывает их в файл
- Источник – модуль в ядре Linux, в котором описаны методы для сжатия информации об операциях и отправке сообщений клиенту

Также в системе существует 6 типов сообщений между клиентом и ядром:

- INFOCOLL_BEGIN – инициализация соединения между клиентом и ядром
- INFOCOLL_END – завершение соединения между клиентом и ядром
- INFOCOLL_OPEN – сообщение об открытии файла
- INFOCOLL_CLOSE – сообщение о закрытии файла
- INFOCOLL_READ – сообщение о чтении из файла
- INFOCOLL_WRITE – сообщение о записи в файл

Для инициализации системы обмена сообщениями необходимо первоначально запустить клиент. При запуске клиент посылает в ядро (идентификатор процесса ядра - 0) по протоколу

NETLINK_INFOCOLL сообщение типа INFOCOLL_BEGIN, содержащее в себе информацию о идентификаторе процесса клиента. Обработчик входящих сообщений в модуле ядра обновляет информацию о подключенном клиенте, сохраняя информацию о его идентификаторе процесса, для последующей отправки сообщений клиенту. При любом действии с файловой системой В случае, если у источника существует подключенный клиент, то происходит вызов компоновщика исходящих сообщений и последующая отправка сообщения клиенту.

Каждое передаваемое сообщение состоит из 4 полей, таких как:

- Тип сообщения
- Время в наносекундах от загрузки
- Полезная нагрузка

Полезная нагрузка представляет собой поле, состоящее из 5 байт, в каждый из которых может быть записана информация, специфичная для определенной операции. В текущей реализации системы эти поля используются только при передаче информации о чтении/записи, в которых хранится

- inode идентификатор файла, к которому производился доступ
- Количество байт в операции
- Смещение в байтах от начала файла
- Итоговый размер файла

Клиент, получая сообщение, извлекает данные из бинарного пакета и записывает их в файл. Формат выходного файла выбирается при старте клиента. На текущий момент реализована запись в файл в бинарном и tab-separated форматах.

Пример полученных данных

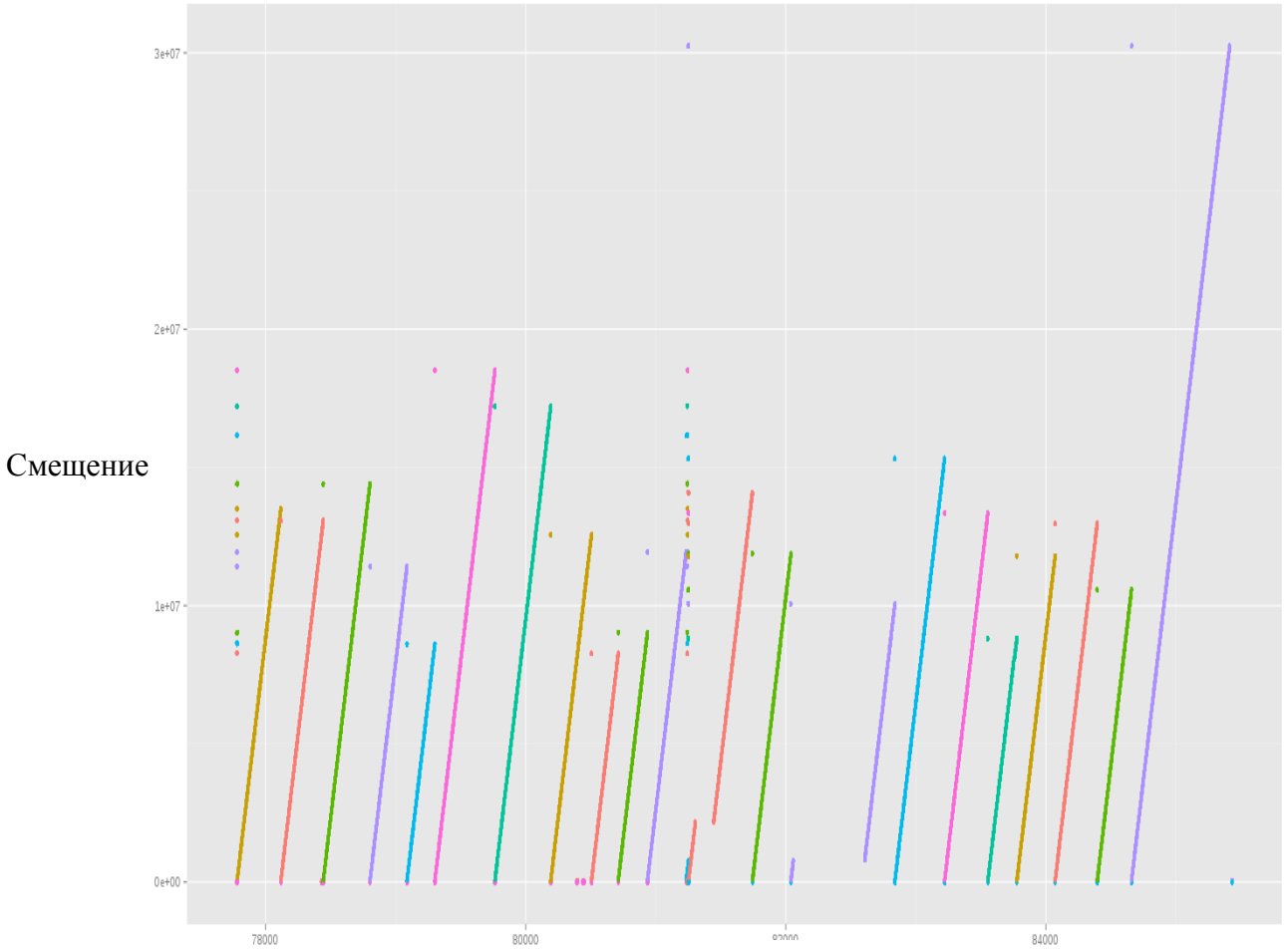

Рис.2

На графике выше показана работа приложения, воспроизводящего аудио-файлы с физического носителя. Данный график наглядно иллюстрирует корректность работы монитора об обращениях к файловой системе.

4.Заключение

В результате данной работы был реализован гибкий монитор операций обращения к файловым системам в ОС GNU\Linux. Для его реализации был исследован стек ввода-вывода с физических носителей в ядре ОС GNU\Linux, инструментированы методы доступа к файлам в виртуальной файловой системе и решена задача вывода данных из пространства ядра в пространство пользователя. По сравнению с существующими решениями, полученная система обладает такими преимуществами, как сбор данных об использовании любой файловой системы, единый формат выходных данных, который поддерживается большинство сред обработки данных. Полученная система предоставляет огромные возможности для изучения поведения приложений в будущем и развития этого направления анализа пользовательских данных.

Весь исходный код, который был написал во время выполнения курсовой работы выложен в общедоступный репозиторий https://github.com/ujohnny/infocoll

5.Список источников

1. Anatomy of the Linux file system, IBM DeveloperWorks, http://www.ibm.com/developerworks/linux/library/l-linux-filesystem/

2. Anatomy of the Linux virtual file system switch, IBM DeveloperWorks, http://www.ibm.com/developerworks/linux/library/l-virtual-filesystem-switch/

3. A File System Tracing Package for Berkeley UNIX, http://techreports.lib.berkeley.edu/accessPages/CSD-85-235.html

4. Communication between kernel and user-space in Linux using Netlink sockets, Software Practice And Experience, 2010, 00:1-7

5. Overview of the Linux Virtual File System, Linux Kernel Documentation, https://www.kernel.org/doc/Documentation/filesystems/vfs.txt

6. Passive NFS Tracing of Email and Research Workloads, http://www.eecs.harvard.edu/sos/papers/ellard-fast03.pdf