Санкт-Петербургский государственный университет

ФУНТ Дина Дмитриевна

Выпускная квалификационная работа

Криминалистический анализ файловой системы XFS с восстановлением данных из журнала

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2018 «Программная инженерия»*

Научный руководитель:
доцент кафедры системного программирования, к.т.н., Ю.В. Литвинов

Рецензент:
старший специалист по тестированию ООО «Техцентр Дойче Банка» С.М. Машарский

Санкт-Петербург
2022

Оглавление

Введение

Одной из наиболее важных задач в цифровой криминалистике является анализ данных с цифровых устройств. Такие устройства часто бывают связаны с расследуемыми преступлениями, а потому, находящаяся на них информация может являться важными материалами для следствия. Основным способом получения информации с устройства является изучение файловой системы. Однако ее ручной анализ является очень кропотливой и трудоемкой работой. Кроме того, это может оказаться и невозможным из-за поломки устройства. Таким образом, для анализа большого количества данных файловых систем нужны инструменты, которые будут структурировать эти данные и облегчать дальнейшую обработку экспертам.

Для автоматизации этого процесса существуют различные инструменты криминалистического анализа образов дисков устройств. В настоящее время создано значительное количество инструментов для анализа файловых систем, например, UFS Explorer [3], PhotoRec [1], The Sleuth Kit (TSK) [2] и др. Однако большая часть этих инструментов предназначена для анализа десктопных файловых систем, а на практике требуются также инструменты для анализа серверных файловых систем.

Одной из таких файловых систем, активно используемых на серверных устройствах, является XFS. Поскольку потребность в больших файловых системах становится все выше, распространенность XFS в настоящее время увеличивается. Эта система поддерживается в большинстве дистрибутивов Linux, таких как Arch, Debian, Fedora, Ubuntu и Oracle, а также используется по умолчанию в дистрибутивах на основе Red Hat [6].

Распространенность XFS и практически полное отсутствие доступных инструментов анализа данной файловой системы привело к необходимости создания инструментов, поддерживающих её анализ. Так, компанией Belkasoft, занимающейся разработкой инструментов криминалистического анализа, была поставлена задача поддержать чтение

памяти устройств с файловой системой XFS, а также восстановления удаленных данных.

1. Постановка задачи

Целью данной работы является создание инструмента, позволяющего выполнять чтение и восстановление удаленных файлов из образов файловых систем XFS.

Для достижения цели были поставлены следующие задачи.

1. Провести обзор существующих решений для анализа образов памяти и изучить особенности файловой системы XFS.
2. Исследовать возможность и предложить способы восстановления удаленных данных XFS.
3. Реализовать инструмент для чтения файловой системы и восстановления удаленных данных.
4. Проверить полученное решение на тестовых образах XFS.

2. Обзор существующих решений

В данной главе проводится обзор существующих решений, способных выполнять чтение файлов и восстановление удаленных данных.

2.1. Обзор существующих инструментов анализа XFS

В рамках работы был проведен обзор существующих инструментов анализа XFS, потенциально подходящих для решения поставленной задачи. Были найдены инструменты, способные отображать данные файловых систем в читаемом виде. Также некоторые из них оказались способны восстанавливать удаленные данные из файловых систем.

1. UFS Explorer

   UFS Explorer [3] позволяет доступ к неповрежденному содержимому хранилища, а также восстановление удаленных файлов из широкого спектра файловых систем, применяемых в Windows, и из интересующей нас XFS. UFS Explorer поддерживает восстановление файлов из XFS после удаления, после форматирования жесткого диска или программных сбоев. Также он способен восстанавливать удаленные файлы даже с реальными именами файлов.

   Однако данный продукт является коммерческим и его функциональность невозможно использовать в других продуктах.

2. ReclaiMe

   ReclaiMe [5] — это универсальный набор инструментов для восстановления данных практически из всех файловых систем, от файловых систем семейства Windows до Linux и MacOS. Он предназначен только для работы на Windows и позволяет читать и восстанавливать файлы XFS с дисков, подключенных компьютеру.

   Однако, возможность анализа образов дисков, извлеченных из поврежденных устройств, не предусмотрена данной утилитой. Этот продукт также является коммерческим.

3. X-Ways Forensics

   X-Ways Forensics [9] — это рабочая среда для компьютерных криминалистов. X-Ways Forensics поддерживает чтение файлов диска, но для восстановления удаленных данных используется карвинг (метод восстановления без метаданных, при котором проводится сканирование незанятой памяти дискового пространства и файлы восстанавливаются только на основе их структуры и содержимого). Восстановление данных этим методом является очень неточным, поскольку при сканировании всей области памяти проводится извлечение всех найденных файлов. Также восстановление данных файла возможно только из одного промежутка в памяти, что влечет некорректное восстановление разреженных файлов. Поэтому, для поддержки восстановления удаленных файлов необходимы более точные методы, нежели восстановление при помощи карвинга.

4. XFS_undelete

   Xfs undelete [10] — утилита для восстановления удаленных файлов из образов дисков XFS с открытым исходным кодом. Она не поддерживает чтение, однако в ней реализован алгоритм восстановления удаленных файлов на основе анализа внутренних структур организации индексных дескрипторов файловой системы. Метод, предложенный данным инструментом, не восстанавливает многие из удаленных файлов, поэтому данное решение является недостаточно точным и требует дополнения, путем анализа журнальных данных.

По итогам обзора, ни один из существующих инструментов не подошел для решения поставленной проблемы, потому как либо не поддерживает необходимую нам функциональность, либо является коммерческим продуктом. Что подтверждает актуальность поставленной задачи и необходимость реализации её решения.

2.2. Библиотека для анализа файловых систем TSK

Библиотека The Sleuth Kit (TSK) представляет собой набор инструментов командной строки и библиотеку C/C++ с открытым исходным кодом для криминалистической экспертизы. Sleuth Kit позволяет анализировать образы дисков или файловых систем, сформированных с помощью «dd» или аналогичных утилит, создающих необработанный образ.

Это удобный, расширяемый инструментарий имеющий базовые функции низкоуровневой обработки дисковых образов. TSK предоставляет API, удобное для поддержки других файловых систем. В нем уже поддержан разбор многих файловых систем, таких как ext2, ntfs, fat и т.д., кроме XFS.

На базе TSK построено много различных инструментов криминалистического анализа, он является самым мощным открытым инструментом анализа образов дисков, доступным для переиспользования. Поэтому было принято создать его расширение для поддержки анализа XFS.

3. Особенности файловой системы XFS

Перед началом работы с XFS необходимо выяснить, какие особенности имеет данная файловая система, а также изучить её внутреннее строение. В данном разделе произведен краткий обзор устройства XFS.

3.1. Преимущества XFS

XFS имеет следующие преимущества.

1. Быстрое восстановление после сбоев.

   При возникновении проблемы в файловой системе XFS может быстро восстановиться, исключив процесс полной проверки, тогда как другие журнальные файловые системы проверяют всю файловую систему.

2. Быстрые транзакции.

   Древовидная структура организации XFS позволяет быстро искать и выделять свободную память, даже когда существует много файлов.

3. Большая масштабируемость.

   64-битная файловая система позволяет построить систему размером до 9 эксабайт.

4. Эффективное распределение.

   Это первая файловая система, которая обеспечивает отложенное выделение пространства с использованием записи в буфер. Таким образом, можно рационально распределить пространство.

5. Превосходная пропускная способность.

   Память файловой системы XFS разделена на множество подтомов, называемых группами выделения. Существует возможность распределения потоков ввода/вывода по этим группам, что обеспечивает эффективность обработки данных.

3.2. Технические характеристики

XFS — это журналируемая файловая система, которая была первоначально создана Silicon Graphics для ОС IRIX, была перенесена на ядро Linux и поддерживалась в большинстве дистрибутивов Linux [7].

Основными техническими характеристиками XFS являются:

- 64-битность файловой системы, что позволяет задействовать большие объемы памяти для хранения файлов на серверах.
- файловая система журналируемая, причем происходит журналирование только метаданных (если не задать иное параметрами).
- структуры Б-дерева для хранения данных файловой системы. Для списка блоков с inode[1], списка экстентов с содержимым файла, каталогов файлов, списков экстентов свободных блоков (свободные блоки проиндексированы и по размеру блока, и по расположению). Однако небольшой файл или каталог может быть размещен прямо внутри inode.
- динамическое выделение (по мере надобности) «индексных блоков» inode и освобождение неиспользуемых inode (высвобождая место для хранения данных).
- хранение всех данных файловой системы в big-endian порядке байтов, в отличие от журнала, который хранится в смешанном порядке: вспомогательные структуры — в little-endian, а остальное — в порядке, определенном носителем.
- транзакционная файловая система, что означает, что состояние информации, находящейся на диске, всегда является консистентным.

[1]inode (индексный дескриптор) — это структура данных в традиционных для ОС UNIX файловых системах. В этой структуре хранится метаинформация о стандартных файлах, каталогах или других объектах файловой системы, кроме непосредственно данных и имени. [8]

3.3. Организация хранения информации

Рассмотрим внутреннюю организацию хранения элементов файловой системы более подробно.

1. Группы выделения

Вся область памяти файловой системы делится на так называемые Группы Выделения (Allocation Groups), имеющие одинаковый размер. Каждую группу AG можно рассматривать как отдельную файловую систему, которая управляет своим собственным пространством. Это делает возможными одновременные файловые операции; только одна запись может происходить в AG в любое время, но в файловой системе могут выполняться несколько операций, каждая из которых происходит в разных AG.

Каждая из AG располагает собственными структурами данных для управления свободным местом и inodes в пределах своих границ (рис. 1).

В начале области памяти AG находится суперблок. Суперблок занимает ровно один сектор в памяти и содержит в себе основную информацию о файловой системе, а также информацию о количестве и размерах AG. В каждой группе выделения находится одинаковый суперблок, причем нулевая Группа Выделения располагается прямо в начале памяти файловой системы, а вместе с ней и нулевой суперблок, который считается основным суперблоком. Вторичные суперблоки используются в случае, когда первичный суперблок поврежден.

Помимо суперблока, каждая из AG располагает собственными структурами данных, содержащими информацию о свободных блоках, о выделенных и свободных индексных дескрипторах и свободных блоках для расширения Б-деревьев.

AGF — структура, содержащая ссылку на корень дерева свободных блоков AG.

AGI — содержащая ссылку на корень дерева выделенных и свободных индексных дескрипторов.

AGFL — ссылка на список блоков, выделенных для расширения би-

деревьев, необходимых для того, чтобы деревья со служебной информацией не были разреженными.

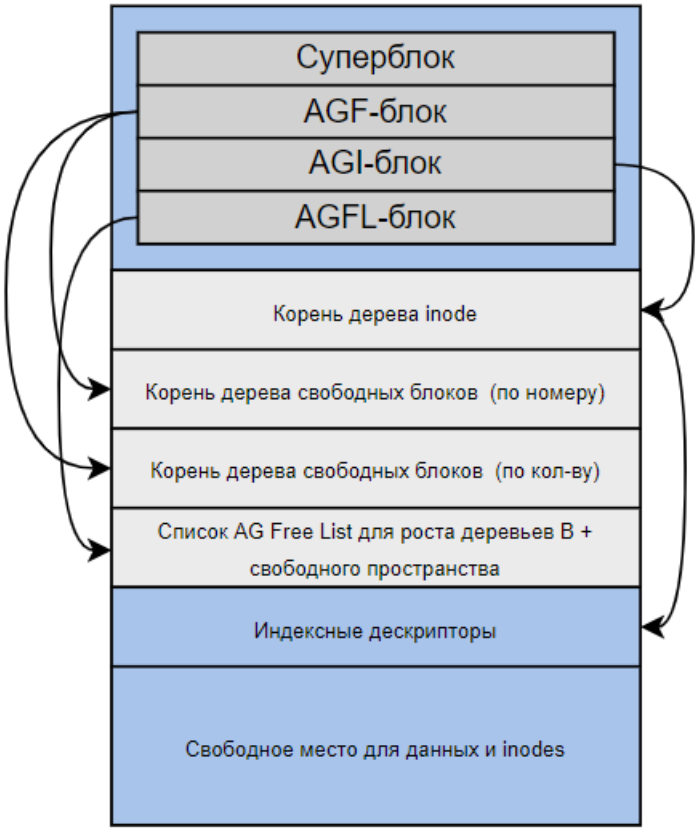

Рис. 1: Строение AG

2. Индексный дескриптор

Структура индексного дескриптора (inode) содержит в себе всю метаинформацию о файле или каталоге. Каждый объект файловой системы, будь то каталог или файл, описан ровно в одном индексном дескрипторе.

Разбиение на AGs также позволяет во внутренних структурах AG использовать относительные указатели на блоки и индексные дескрипторы, и удерживать разрядность этих указателей в пределах 32 bit, таким образом экономя свободную память.

Inode в XFS состоит из двух частей. Ядро имеет структуру, общую для всех дескрипторов, в нем содержится основная информация об описываемом элементе файловой системы, такая как тип элемента, время создания, обновления и последнего обращения. Область данных inode

зависит от типа дескриптора и содержит либо список областей памяти, занимаемых элементом файловой системы, либо указатель на блок со списком областей памяти, либо указатель на Б-дерево, хранящее список областей памяти.

Существует три основных типа inode, описывающих элементы: локальный (local), расширенный (extended), Б-дерево (b-tree). Для описания файлов возможны только расширенный и Б-дерево, а для директорий все три.

3. Файлы

Файлы, как и другие объекты файловой системы, описываются в индексном дескрипторе. Раздел данных индексного дескриптора хранит ссылку на блок файловой системы, содержащий отсортированный список областей памяти, занятых файлом. Чем больше файл, тем больше блоков необходимо для размещения файла. Файловая система стремится записать блоки так, чтобы файл занимал в памяти непрерывную область, однако не всегда можно найти место для размещения необходимого размера. Тогда файловая система делит файл на области, а файл в таком случае называется разреженным.

Список областей очень разреженных файлов может не поместиться в блок файловой системы. В таком случае, хранение списка организуется в виде Б-дерева, а тип индексного дескриптора меняется на тип Б-дерева.

При удалении файлов в ядре индексного дескриптора обнуляются некоторые поля, тогда как в области памяти ничего не меняется (рис. 2).

4. Директории

Директории, так же, как и файлы, и ссылки, описываются в индексном дескрипторе. Данные, хранящиеся в индексном дескрипторе, состоят из записей об именах файлов/директорий, хранящихся в описываемой директории, а также ссылок на индексные дескрипторы этих элементов.

Структура директорий в XFS представляет собой дерево. В суперблоке хранится ссылка на единственную корневую директорию, в кото-

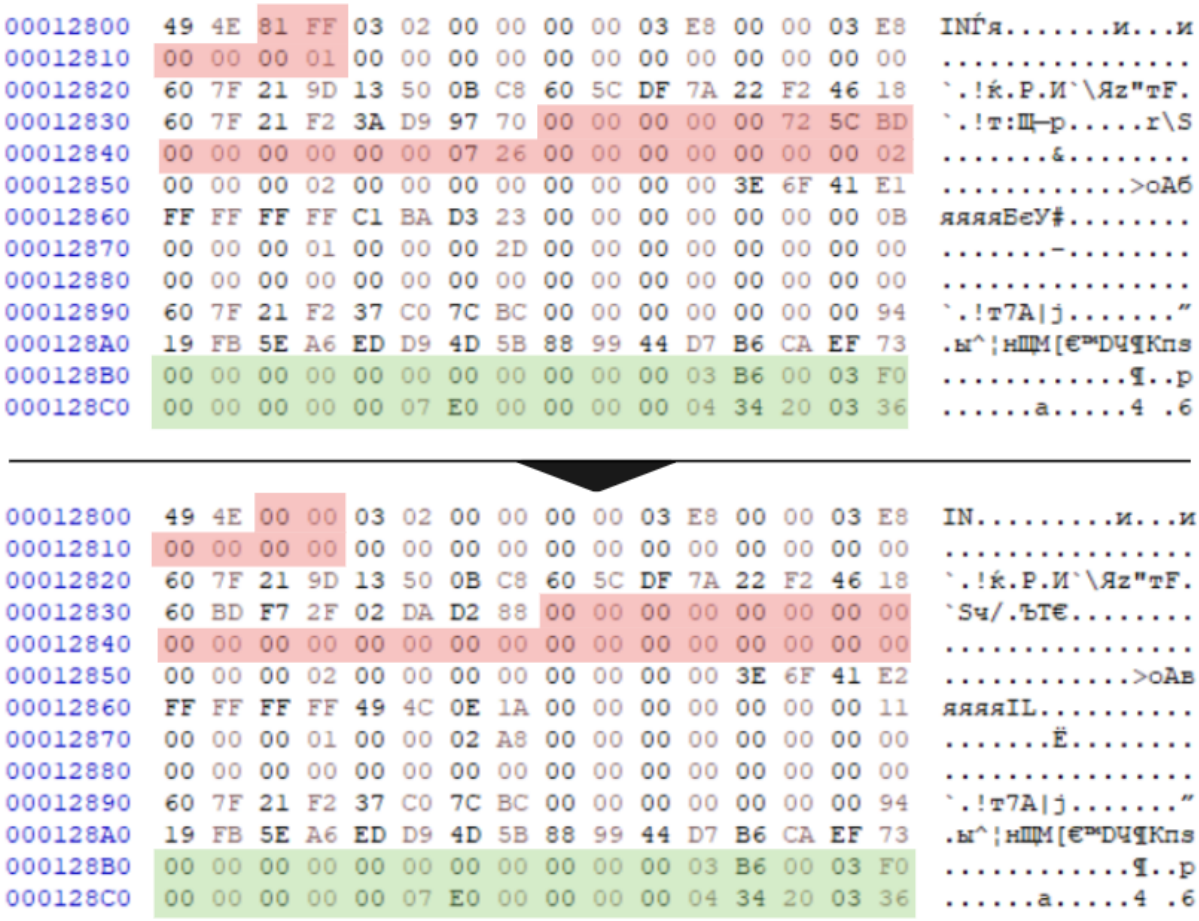

Рис. 2: Изменения в индексном дескрипторе при удалении файла. Красным выделены стертые байты полей ядра индексного дескриптора. Зеленым — область данных, байты которой не изменились.

рой хранятся все остальные директории, файлы и ссылки.

Существует несколько типов организации директорий, определяющихся в зависимости от количества содержащихся элементов.

a. Short Form Directories

Самый простой тип директорий, в котором данные о файлах, содержащихся в них, хранятся прямо в области данных индексного дескриптора. Количество файлов, которое может вместить такой тип директории, зависит от размера индексного дескриптора, длин имен записей и расширенных атрибутивных данных.

При добавлении файла в директорию, список записей в индексном дескрипторе увеличивается. В случае, если для записи о файле недостаточно места, тип директории меняется на блочный, под записи выде-

ляется отдельный блок директорий, все данные о файлах переносятся в этот блок, а в данных индексного дескриптора записывается ссылка на выделенный блок.

При удалении файла из директории локального формата происходит сдвиг хвостовой части списка файлов на место, которое ранее было занято ссылкой на удаленный файл (рис. 3). Данные записи о файле полностью затираются и больше не могут быть восстановлены.

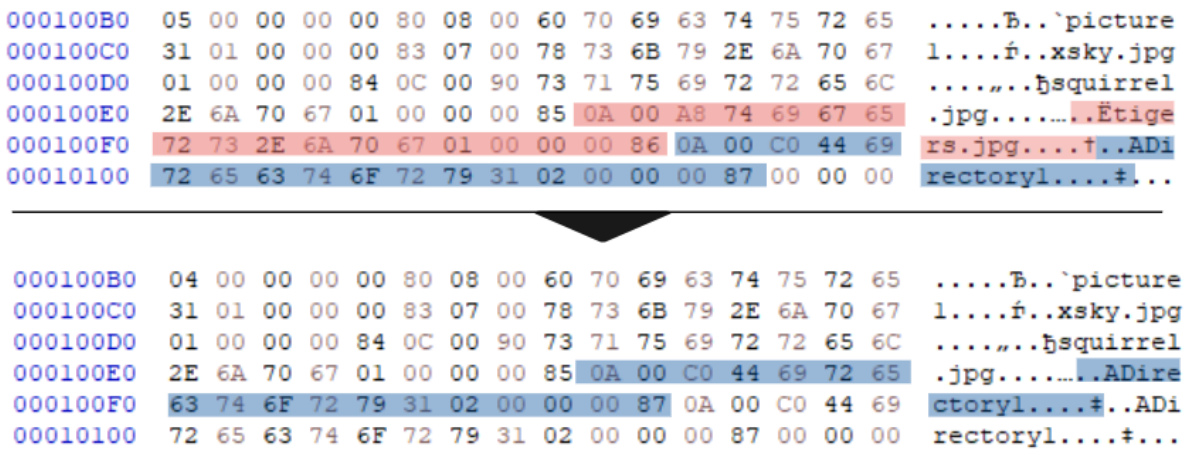

Рис. 3: Сдвиг списка файлов в локальной директории после удаления файла tigers.jpg

b. Block Directories

Список файлов так же хранится списком, но в отдельном блоке файловой системы, на который указывает индексный дескриптор директории. Размер блоков директорий одинаковый и определяется файловой системой и находится в данных суперблока. Блок делится на заголовок с метаинформацией (сигнатура, номер блока в файловой системе, номер индексного дескриптора, список свободных областей в области для записей и т.д.), область для записей, область, содержащую информацию о хэшах записей, и хвост с информацией о количестве записей (рис. 4).

В отличие от директорий локального типа, в этом случае среди файлов директории хранятся записи о текущей (“.”) и родительской (“..”) директориях.

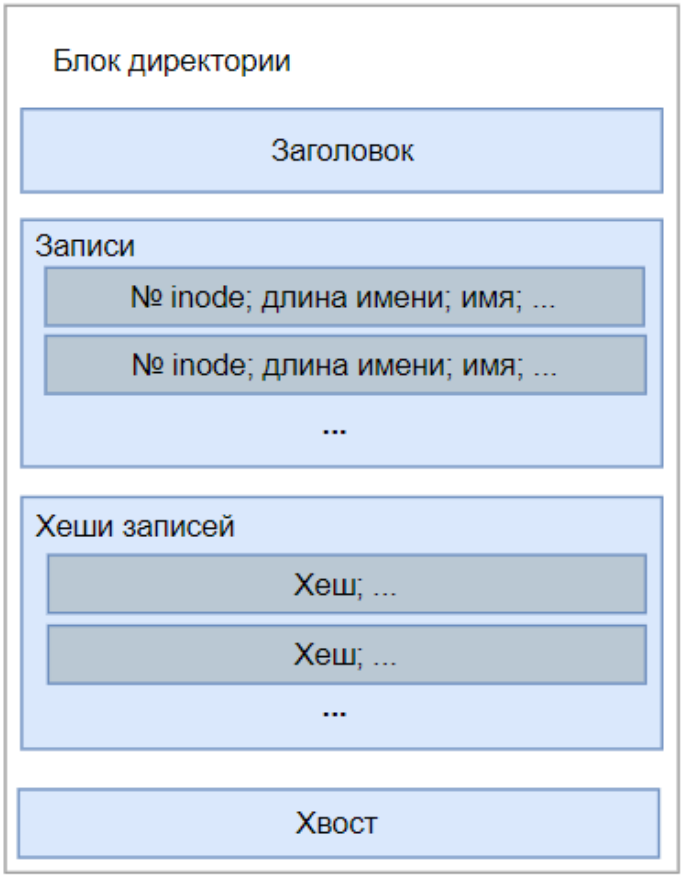

Рис. 4: Строение блока директории

Запись файлов также проводится на свободные места до тех пор, пока достаточно места в блоке, далее директория преобразуется в листовую директорию.

При удалении данных из директории место записи о файле помечается как свободное (рис. 5). Флаг, сигнализирующий о свободном промежутке памяти, записывается прямо на месте, где хранится номер индексного дескриптора удаленного файла. После флага идет длина области, занимаемой записью. Флаг вместе со смещением занимают половину длины индексного дескриптора.

c. Leaf Directories

Как только блок директории заполняется записями, данные каталога преобразуются в другой формат. Листовой формат отличается от блочной директории тем, что в области данных дескриптора теперь содержится не одна ссылка на блок директории, а список ссылок на блоки

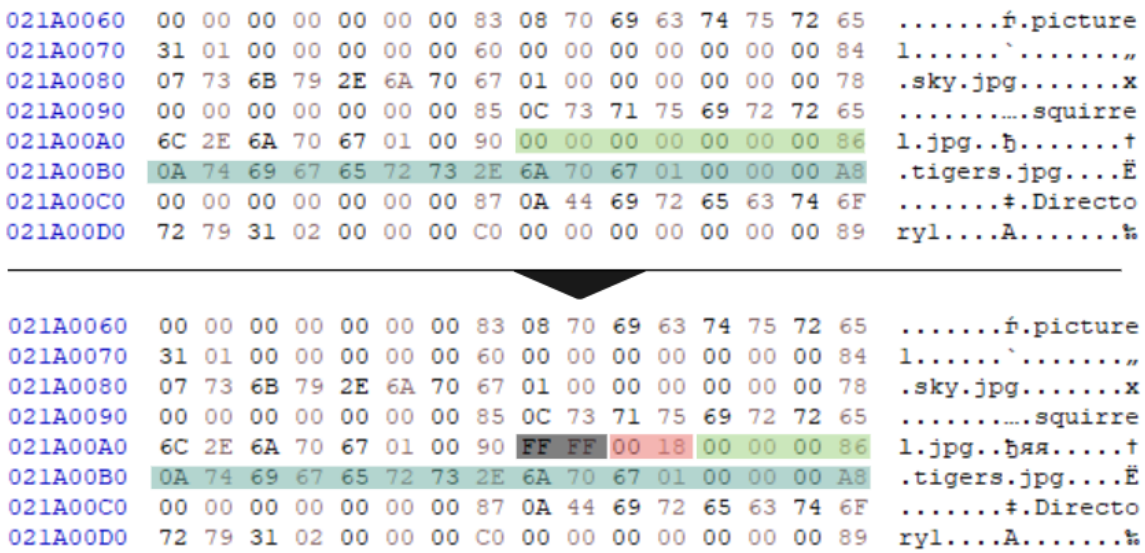

Рис. 5: Удаление файла tiger.jpg из блочной директории. Перезапись области inode флагом (0xFFFF) и размером освободившегося промежутка (0x0018)

и указатель на отдельный блок, в который теперь выносятся записи хэшей. Кроме списка хешей из заголовка выносится список наилучших свободных подблоков для новых записей.

Операции записи и удаления для этого типа одинаковы с блочным.

d. Node Directories

Когда место в листовой директории, в блоке, содержащем хеши, заканчивается, файловая система снова трансформирует директорию в узловую. Список наилучших свободных подблоков для новых записей выделяется в свой собственный блок, а хранение блоков с хешами записей организуется в Б-дерево.

Список блоков данных остается неизменным, потому операции записи и удаления для этого типа одинаковы с блочным.

e. B+tree Directories

Когда список блоков записей выходит за пределы пространства индексного дескриптора, его формат изменяется на Б-дерево. Индексный дескриптор теперь содержит указатель на корневой блок файловой системы Б-дерева для блоков каталога.

Поскольку листовые блоки дерева записей организованы так же, как

и в предыдущих типах директорий, операции записи и удаления все еще одинаковы с блочным.

3.4. Журналирование в XFS

Журнал в XFS занимает непрерывную область в памяти, расположение и размер которой описывается в суперблоке файловой системы.

Журнал представляет собой серию записей; каждая запись журнала содержит часть или всю транзакцию. А каждая транзакция состоит из серии заголовков операций журнала (элементы журнала), структур форматирования и необработанных данных, измененных операциями транзакции.

Каждая транзакция имеет заголовок фиксированного размера с полями, определяющими количество включаемых в транзакцию элементов журнала и размер занимаемой транзакцией памяти. Первая операция в транзакции устанавливает идентификатор транзакции, а последняя операция — это запись фиксации. Операции, записанные между операциями запуска и фиксации, представляют собой изменения метаданных, внесенные в рамках данной транзакции. Если операция фиксации отсутствует, транзакция не завершена и не может быть восстановлена [4].

Операции, записанные последовательно в области памяти транзакции, также имеют заголовок фиксированной длины. Существует несколько типов операций, каждая из которых определяется сигнатурой, находящейся в начале области памяти операции.

Наибольший интерес для осуществления целей данной работы представляют следующие две операции:

- обновление информации в индексном дескрипторе

  При любом изменении в индексном дескрипторе, в журнале появляется запись, содержащая состояние inode на момент после записи, уже с проведенными изменениями. Изменения в любом из полей из разделов дескриптора отражаются в журнале, будь то

время последнего изменения файла или же количества занимаемых блоков, или изменение количества ссылок на inode из директорий.

- запись части буфера файловой системы

  В буфер файловой системы попадает измененная информация из блоков файловой системы или блоков директорий. По мере заполнения буфера, система выгружает информацию в журнал и продолжает работу с очищенным буфером.

Стоит также отметить, что важной особенностью журнала является его цикличность, т.е. когда операции в журнале выходят за границу журнальной области, запись продолжается с начала области, стирая информацию о более старых операциях файловой системы. При этом, номер цикла журнала увеличивается, и в поле из заголовка транзакции, содержащее номер цикла, начинает записываться новое число. Таким образом, начало самой старой операции в журнале не определено и может находиться как в самом начале области, так и в самом её конце.

4. Алгоритмы восстановления удаленных данных

Когда файл удаляется, место в памяти, содержащее данные файла, помечается как свободное, а также создается запись в журнале об операции удаления файла. В большинстве случаев, участок памяти, занимаемый удаленным файлом, не перезаписывается до тех пор, пока в это место не будет назначен другой файл. Таким образом, остается возможность восстановления удаленных файлов путем анализа записи в журнале.

В данной главе будет рассмотрен алгоритм восстановления данных на основе анализа деревьев индексных дескрипторов, предлагаемый инструментом xfs_undelete. А так же предложен новый алгоритм восстановления удаленных файлов XFS путем анализа журнала и подробно рассмотрены его шаги.

4.1. Восстановление файлов из деревьев индексных дескрипторов

В каждой AG содержится дерево, хранящее в себе все индексные дескрипторы, выделенные в области памяти, занимаемой AG. Ссылка на корень дерева хранится в структуре AGI (рис.1).

Листья деревьев содержат структуры, состоящие из указателей на группы индексных дескрипторов, зарезервированных файловой системой. В каждой такой структуре, кроме указателей, хранится битовая карта, по биту на каждый индексный дескриптор группы, для обозначения свободных и занятых дескрипторов.

Для определения файлов, подлежащих восстановления, необходимо выполнить обход по дереву и получить список свободных индексных дескрипторов. Далее просканировать область данных каждого дескриптора и извлечь список областей памяти, занимаемых файлами.

Данный метод имеет некоторые ограничения. Поскольку файловая система динамически освобождает и выделяет индексные дескрипто-

ры, при полностью свободной группе выделенных дескрипторов система удаляет запись о ней из дерева. Удаленные файлы в таком случае становятся недоступны для восстановления этим методом.

Также при удалении файла в ядре индексного дескриптора очищается информация о длине файла. Поэтому, при извлечении файла из памяти, могут быть извлечены лишние байты, пустые или из других файлов, что способно привести к повреждению файла и невозможности его чтения.

Еще одним ограничением метода является невозможность восстановления имен файлов, так как имена объектов хранятся отдельно от их индексных дескрипторов, данным методом их не извлечь.

4.2. Восстановление файлов из журнала

В рамках данной работы был разработан алгоритм восстановления файлов, на основе информации о событиях, записанной в журнал файловой системы, во время удаления элемента.

В начале необходимо извлечь из суперблока информацию, нужную для дальнейшего анализа журнала. На этом этапе требуется получить размер и расположение области журналирования.

На следующем этапе выполняется непосредственно анализ области журналирования. Список удаленных файлов получается путем чтения всех элементов транзакций журнала и поиска среди них записей определенного типа, сигнализирующих об изменениях в памяти, выделенной под файл.

В процессе анализа области журналирования извлекается список файлов, подлежащих восстановлению, с информацией, необходимой для дальнейшего извлечения каждого файла, такой как адрес смещения области в памяти, занимаемой файлом, фактический размер, количество экстентов, выделенных под каждый удаленный файл.

После получения списка файлов на восстановление необходимо второй раз проанализировать журнал для поиска записей об изменениях в блоках и индексных дескрипторах директорий, для извлечения имен

удаленных файлов.

Последний шаг — восстановление. Для получения непосредственно ссылок на области памяти, занимаемые файлом, необходимо также проанализировать область данных индексных дескрипторов удаленных файлов. Дальнейшее извлечение файлов проводится на основе информации, собранной на предыдущих шагах, такой как имя, путь, размер и начальное смещение в дисковом хранилище.

Далее будет более подробно рассмотрен каждый из шагов процедуры восстановления.

4.3. Детальный разбор восстановления файлов из журнала

4.3.1. Анализ суперблока

Перед началом анализа журнала необходимо выяснить расположение и размер занимаемой им области памяти. Это можно вычислить по параметрам, описанным в структуре суперблока.

Для проведения расчетов потребуются поля, в которых содержится номер блока, с которого начинается журнал и количество блоков, зарезервированных под него. А также для вычисления смещения журнала от начала образа в байтах нужно считать информацию о количестве блоков в группе, размере сектора в байтах.

С учетом особенностей нумерации блоков и секторов надо вычислить смещение журнала и его размер.

4.3.2. Анализ журнала

После получения доступа к области журналирования необходимо её просканировать. Так как журнал ведется циклично и нет информации о том, где начинается самая старая запись, необходимо определить смещение относительно начала журнала, с которого начинать чтение.

В заголовках транзакций хранится информация о номере цикла, в который была записана транзакция. Поскольку в области журналиро-

вания может находиться одновременно только два различных номера цикла, необходимо найти первую транзакцию с начала, в которой стоит меньший номер.

Для начала нужно определить, какие номера циклов находятся в журнальной области. Заголовок первой и последней транзакции, описанных в журнале, будут иметь разные номера циклов, поэтому необходимо найти их смещения относительно начала и конца области журналирования и извлечь номера циклов из заголовков.

Далее, когда определен номер более старого цикла, необходимо узнать адрес смещения первой сохранившейся его транзакции. Это просто сделать с помощью алгоритма двоичного поиска по блокам файловой системы, входящим в область журнала, учитывая, что адрес смещения транзакции в журнале всегда кратен размеру сектора файловой системы, указанному в суперблоке.

После определения адреса смещения начала цикла, идет этап чтения журнала и выделения из него записей об удаленных файлах. Чтение журнала проводится путем последовательного сканирования заголовков операций, анализа содержимого тела текущей операции и дальнейшей итерации по операциям. Поскольку операции в памяти расположены непрерывно друг за другом, адрес смещения конца предыдущей операции будет адресом начала следующей.

В области журнала нередко бывают поврежденные данные, как в заголовках операций, так и в их областях данных. Например, встречаются нулевые длины операций, хранящиеся в заголовках. Заголовок каждой операции в транзакции начинается с 4 байтов — идентификатора транзакции, который описан в заголовке транзакции. В случае, если длина операции установлена некорректно, это можно определить путем проверки заголовка следующей операции и скорректировать длину.

Когда становится ясно, каким образом можно сканировать записи в журнале, надо выяснить, какие записи в журнале означают удаление файлов. Поскольку документация к XFS не описывает принципы формирования последовательности записей в журнале в результате каких-либо событий в системе, возникла потребность в изучении поведения

журнала. Исследование изменения состояния журнала показало, что в случае удаления файла в элементах транзакции создается запись об изменении в inode с измененной структурой inode в теле элемента. Маркером записи об удалении файла служит поле di_nlink (количество ссылок на индексный дескриптор из различных элементов файловой системы, в частности из директорий). В случае удаления файла, он перестает быть доступен отовсюду, и количество ссылок соответственно обнуляется.

Так как определить имя файла обладая только информацией из индексного дескриптора невозможно (имена файлов хранятся в дескрипторах директорий), следующим шагом является извлечение имени файла из журнала.

4.3.3. Извлечение имени файла

Для криминалистического анализа имя файла может быть так же важно, как и его содержимое. В каком-то смысле, имя файла может объяснять содержимое файла, а наличие расширения облегчает криминалистам работу по определению инструментов для чтения файла.

Так как имя восстанавливаемого объекта файловой системы не хранится в данных индексного дескриптора, описывающего файл, извлечению имени необходимо уделить отдельное внимание.

Файловая система хранит имена файлов в данных директорий, содержащих этот файл, но при удалении файла эта информация больше не содержится в директориях, включавших удаленные файлы. В случае различных типов директорий информация стирается разными способами, как было показано в предыдущих главах. Из директорий мы больше не можем получить имена файлов. Однако при каждом изменении в метаданных, в том числе и директориях, файловая система создает запись об изменении в журнале. Поэтому для получения информации об имени файла необходимо провести повторное сканирование журнала.

На втором сканировании журнала нас будут интересовать не только записи об изменениях в индексных дескрипторах, но и записи об изменениях в буфере. Исследование поведения журналирования в XFS

показало, что среди записей журнала содержатся затронутые удалением части блоков директорий. Причем блоки хранят информацию в том виде, в котором она была в файловой системе до операции удаления.

При итерации по записям журнала необходимо обращать внимание на индексные дескрипторы локального типа директорий, поскольку в их области данных содержится список файлов. Для извлечения имен файлов необходимо пройтись по списку элементов директории и сравнить номера дескрипторов элементов с номерами дескрипторов из списка файлов, полученного на предыдущем шаге. При совпадении номеров можно считать, что мы нашли имя одного из восстанавливаемых файлов.

Так же необходимо обработать и буферные записи. Поскольку блоки директорий разного типа имеют определенную сигнатуру, то мы можем определить, какой тип директории содержит в себе область буфера. В соответствии с типом выбрать алгоритм перебора элементов директории, и точно так же сравнить номера дескрипторов с номерами из списка файлов на восстановление.

Однако, на данном этапе не исключено, что какие-либо записи журнала затерты новыми транзакциями или и вовсе содержат некорректные данные и по этой причине пропущены при итерации. Отсюда возникают файлы, для которых не удалось извлечь имена.

4.3.4. Извлечение файла

Теперь, когда получен список удаленных файлов, прочитаны индексные дескрипторы для каждого из файлов и получили список областей, занимаемых файлами, остается только считать эти области памяти, последовательно проходя по записям из списка.

Поскольку в XFS ведется журналирование метаданных, в большинстве случаев у индексного дескриптора в журнал сохраняется только ядро. Информация о расположении файла в памяти хранится в части данных индексного дескриптора. Получить её можно, обратившись по адресу смещения индексного дескриптора. Если файловая система не назначила на место этого индексного дескриптора новый элемент, то

область данных будет содержать все ту же информацию, что и раньше. Узнать, принадлежит ли дескриптор все тому же файлу или нет, можно, сравнив время создания из дескриптора в журнале.

После получения областей памяти, занимаемых файлом, остается только скопировать байты из памяти в извлеченный файл.

5. Расширение библиотеки TSK

5.1. Структурные особенности библиотеки

Архитектура этого инструмента предоставляет различные уровни абстракции, которые используются при работе с различными сущностями файловой системы (рис. 6).

Самый нижний уровень, базовый уровень, содержит общие функции и структуры данных, которые можно применять ко всем уровням. Здесь определяются обработка ошибок и общие типы.

Следующим уровнем является Disk Image Layer, который предоставляет возможность работы с различными типами дисковых образов. Этот слой скрывает детали, связанные с разделенными, сжатыми и зашифрованными файлами изображений от других слоев. Все образы дисков должны быть сначала открыты функциями слоя образа диска, прежде чем они смогут быть обработаны другими слоями.

Следующий уровень — это уровень файловой системы. Этот уровень фокусируется на обработке данных в виде файловой системы, такой как FAT или NTFS. Файловые системы могут быть расположены в разделе или могут быть полным файлом образа диска. Этот набор функций позволяет читать произвольные данные из файловой системы, список файлов и открывать файлы.

Для реализации поддержки XFS было проведено расширение верхнего уровня File System, в то время, как остальные уровни были доступны как инструменты, позволяющие упростить работу с дисковым образом.

Уровень file system содержит набор функций, который позволяет читать произвольные данные из файловой системы, файлов списков и открытых файлов. Этот уровень уже содержит в себе реализации расширений для некоторых других файловых систем, таких как ext2, NTFS и тд. Необходимо было добавить на этом уровне поддержку XFS.

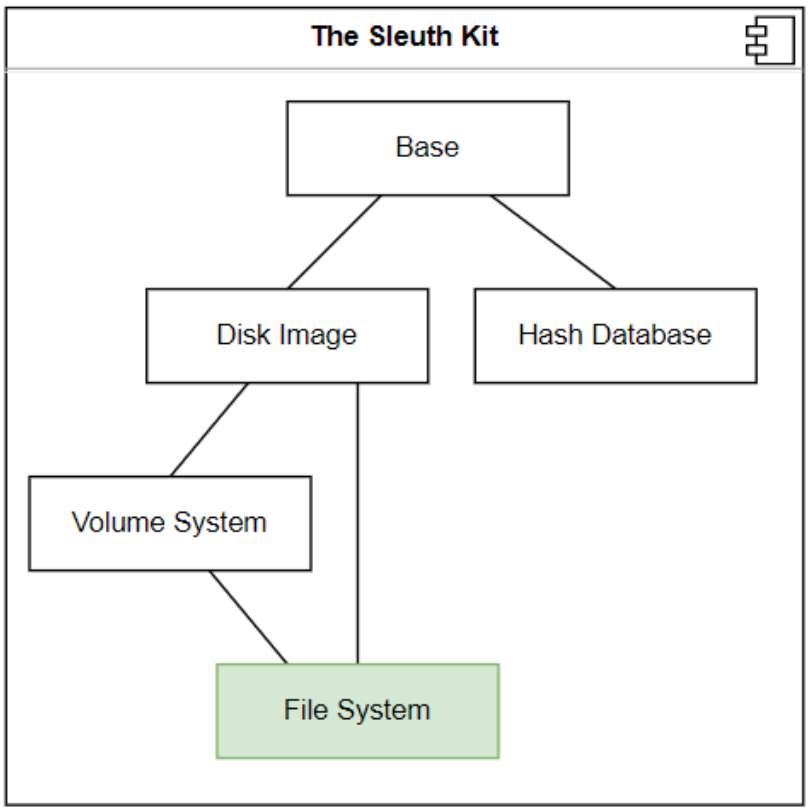

Рис. 6: Архитектура TSK

5.2. Детали реализации алгоритма чтения

Для реализации расширения был выбран язык C, потому как расширяемая библиотека также реализована на C. Уровень file system библиотеки предоставляет интерфейс для расширений, функции которого необходимо реализовать для поддержки возможности чтения данных диска с файловой системой XFS.

Каждый раз работа с дисковым образом начинается с его открытия методами дискового уровня. Функция xfs_open заполняет структуру TSK_FS_INFO, содержащую общую информацию о файловой системе, все данные из суперблока, размер блока файловой системы, ссылку на корневую директорию, на журнал, ссылки на реализацию методов интерфейса для работы с элементами конкретной обрабатываемой файловой системы.

Методы обработки файловой системы реализованы с учетом спе-

цифики данной файловой системы. Основными алгоритмическими методами интерфейса для реализации чтения являются методы итерации по директориям и сбор информации о файлах, хранящихся в них (dir_open_meta), а также сбор информации о файле путем анализа его индексного дескриптора для последующего извлечения его из памяти (xfs_load_attrs).

Метод итерации по директориям (dir_open_meta) получает информацию о расположении корневой директории из заранее инициализированной TSK_FS_INFO. Далее производится чтение индексного дескриптора корневой директории; если в корневой директории находятся ссылки на другие директории, то метод вызывается на вложенных директориях. По итогам работы для каждой директории инициализируется структура TSK_FS_DIR, содержащая список имен файлов и ссылок на них.

Для обработки конкретного файла в директории и получения списка областей памяти, занятых файлом, используется метод xfs_load_attrs. Данный метод анализирует область данных индексного дескриптора файла и заполняет структуру TSK_FS_META, содержащую список областей памяти файла.

Непосредственным извлечением файла из памяти на основе структуры TSK_FS_META занимается также метод уровня файловой системы — tsk_fs_file_walk. Метод итерируется по списку областей памяти файла, применяя для каждой из них переданный ему параметром обратный вызов для записи байтов в память.

5.3. Детали реализации алгоритма восстановления

Для осуществления возможности восстановления удаленных файлов в TSK было добавлено консольное приложение на основе описанных в предыдущей главе алгоритмов. Для него были реализованы методы итерации по записям журнала и анализа их содержимого каждой записи, а так же метод обхода деревьев дескрипторов.

Было реализовано оба описанных алгоритма и проведено сравнение

результатов восстановления для каждого из них. По результатам которого было решено объединить алгоритмы, и в конечном решении оба из них применяются последовательно.

6. Эксперименты

6.1. Алгоритм создания тестовых образов

Для проведения экспериментов над полученным расширением было создано несколько тестовых образов XFS следующим образом:

1. В качестве основной среды для создания образа был использован инструмент Oracle VirtualBox с установленным на нем дистрибутивом Linux Ubuntu 20.04 LTS. Хостовой ОС выступала Windows 10.

2. Был создан пустой файл, размером 5 Мб.

3. Полученный файл отформатирован в файловую систему XFS при помощи утилиты mkfs.

4. Далее в ОС был создан каталог для создания точки монтирования.

5. Файл был примонтирован к созданному каталогу и далее каталог заполнялся файлами.

6. С помощью скрипта на bash один из каталогов примонтированной файловой системы был заполнен файлами. Таким образом было создано несколько типов директорий, с учетом особенностей формирования блока директории.

   Было создано по образу для каждого типа директории, с 4, 256 и 2000 файлами соответственно.

7. Также из каждого образа были удалены некоторые файлы с различными ситуациями расположения, в корневой директории, на первом уровне вложенности и на втором.

8. После проведения необходимых манипуляций с каталогом он был размонтирован.

9. Полученный файл был перемещен на хост для проведения следующих экспериментов.

6.2. Тестирование и результаты эксперимента

Полученная функциональность была апробирована различными способами, в соответствии с особенностями реализации.

1. Чтение

Было проведено тестирование с использованием нескольких образов на каждый тип и на различных уровнях вложенности директории, файла и символьной ссылки, в ходе которого были корректно считаны файлы с тестовых образов.

Корректность устанавливалась побитовым сравнением исходных и результирующих файлов.

2. Восстановление

Операция восстановления удаленных файлов сильно зависит от размера журнала и событий, происходивших во время подготовки тестовых образов. Невозможно однозначно утверждать, какие записи об удалении файлов все еще содержатся в журнале, и какие файлы должны быть восстановлены в результате исполнения. Помимо зависимости от событий, восстановление данных зависит от корректности данных журнала, однако при работе с записями журнала, было выявлено много неточностей, которые приводят тому, что обработка записи может быть пропущена.

Проверка восстановления проводилась на созданных тестовых образах, были удалены файлы и директории из образов различных размеров и содержания. И на образах дисков, с неизвестным содержимым, предоставленных Belkasoft. В результате эксперимента получены следующие данные (таблица 1).

№	удалено файлов, шт.	извлечено файлов, шт.	извлечено файлов без имени, шт.	извлечено файлов, AGI анализ	извлечено уникальных файлов, комбинация методов
1	неизвестно	4	0	0	4
2	неизвестно	4	2	240	242
3	неизвестно	212	10	16	220
4	9	9	0	2	9
5	1	1	0	1	1
6	3	3	0	2	3
7	2	-	-	1	1

Таблица 1: Результаты восстановления файлов на различных образах

В образах 4-6, созданных для целей тестирования, удаление файлов являлось последними событиями, перед размонтированием образа, потому доступными из журнала оказались все удаленные файлы. В образе 7 при удалении файла произошел сбой, область журнала отформатировалась в первоначальное состояние, восстановление файлов не производилось.

Образы 1-3 были предоставлены Belkasoft, их содержимое не было известно перед тестированием. Восстановленными оказались некоторые метаданные операционной системы, логи, хеши и временные файлы. В образе 3 наблюдались также и файлы с пользовательским содержимым, изображения, текстовые документы и т.д.

Таким образом, установлено, что реализованный алгоритм анализа журнала является рабочим и способен восстанавливать доступные файлы, данные о которых хранятся в области журнала, и индексный дескриптор которых не очищен файловой системой.

Также было проведено сравнение двух реализованных алгоритмов восстановления данных. Можно видеть, что восстановление файлов на

основе анализа AGI деревьев файловой системы в основном восстанавливает файлы, отличные от тех, что были восстановлены методом анализа журнала, что отражают результаты эксперимента комбинациии методов.

Заключение

В рамках данной работы были получены следующие результаты.

1. Проведен обзор существующих решений UFS Explorer, ReclaiMe, X-Ways Forensics, xfs_undelete в ходе которого не было обнаружено подходящих решений поставленной задачи, кроме алгоритма из xfs_undelete. Также была изучена библиотека анализа файловых систем TSK, удобная для использования в данной работе, которая впоследствии была взята за основу. Были изучены особенности файловой системы XFS.

2. Разработан алгоритм восстановления удаленных данных XFS на основе анализа журнала файловой системы.

3. Реализовано расширение библиотеки TSK на языке C с закрытым исходным кодом для чтения файловой системы и восстановления удаленных данных на основе изученных особенностей XFS и с применением разработанного алгоритма, а так же с применением алгоритма, предлагаемого инструментом xfs_undelete.

4. Проведено тестирование алгоритма чтения из реализованной библиотеки на образах файловой системы с различными имеющимися ситуациями расположения файла. Расширение оказалось способно читать данные из всех типов директорий и файлов. Также был поставлен эксперимент над предложенным алгоритмом, в ходе которого было показано, что файлы, удаленные недавно, восстанавливаются и имеют первоначальное имя, а файлы, удаленные давно, либо теряют свое имя, либо оказываются недоступны. Проведено сравнение полученных результатов с результатами применения существующего алгоритма, в итоге которого было принято решение комбинировать два алгоритма в полученном расширении.

Список литературы

[1] CGSecurity. PhotoRec, Digital Picture and File Recovery. — URL: https://www.cgsecurity.org/wiki/PhotoRec (online; accessed: 2020-12-16).

[2] Carrier Brian. Autopsy overview page. — URL: http://www.sleuthkit.org/autopsy/ (online; accessed: 2020-12-16).

[3] Explorer UFS. UFS Explorer Standard Recovery Main page. — URL: https://www.ufsexplorer.com/ufs-explorer-standard-recovery.php (online; accessed: 2020-12-16).

[4] Inc. Silicon Graphics // XFS Algorithms & Data Structuress. — 2006.

[5] ReclaiMe. ReclaiMe Home page. — URL: https://www.reclaime.com/default.aspx (online; accessed: 2020-12-16).

[6] Red Hat. The XFS file system. — URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/storage_administration_guide/ch-xfs (online; accessed: 2020-12-16).

[7] SGI. XFS Papers and Documentation. — URL: https://xfs.org/index.php/XFS_Papers_and_Documentation (online; accessed: 2020-12-16).

[8] Wikipedia. Inode. — URL: https://en.wikipedia.org/wiki/Inode (online; accessed: 2020-12-16).

[9] X-Ways. X-Ways Forensics: Integrated Computer Forensics Software. — URL: http://www.x-ways.net/forensics/ (online; accessed: 2020-12-16).

[10] xfs_undelete. — https://github.com/ianka/xfs_undelete.