МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский ядерный университет «МИФИ»
(НИЯУ МИФИ)

На правах рукописи

УДК 004.75

Чуркин Кирилл Юрьевич

Разработка персональной распределённой файловой системы

Выпускная квалификационная работа магистра

Направление подготовки 09.04.01 Информатика и вычислительная техника

Выпускная квалификационная
работа защищена
«___»_____2024 г.
Оценка _____
Секретарь ГЭК _____

Москва 2024

Студент-дипломник _____ / Чуркин К.Ю. /
Руководитель работы _____ / Дюмин А.А. /
Рецензент _____ / Свиридов А.Э. /
Заведующий кафедрой №12 _____ / Иванов М.А. /

АННОТАЦИЯ

Оглавление

ВВЕДЕНИЕ 6
1 Обзор существующих распределенных файловых систем хранения 7
1.1 NFS (Network File System) 10
1.2 ZFS (Zettabyte File System) 12
1.2.1 Хранение в пулах (Pooled storage) 12
1.2.2 Копирование при записи (Copy-on-write) 13
1.2.3 Снапшоты (Snapshots) 13
1.3 GFS 14
1.4 HDFS 16
1.5 Ceph 18
1.6 Lustre 19
1.7 Сравнение 21
2 Концепция приложения 22
2.1 Требования 22
2.2 Особенности реализации и ограничения 23
2.3 Архитектура 24
2.4 Безопасность 24
2.5 Запуск ролей 25
2.6 Функциональность 25
2.6.1 Просмотр файловой системы 25
2.6.2 Изменения файловой системы 26
2.6.3 Фильтрация показываемых файлов 26
2.6.4 Репликация 26
2.6.5 Открытие файлов и работа с ними 27
2.7 Флаги запуска 30
3 Реализация 31
3.1 Конфигурация 31
3.1.1 StorageRole 31
3.1.2 RunnerRole 31
3.1.3 RouterRole 31
3.1.4 Roles 31
3.1.5 Log 31
3.1.6 User 31
3.1.7 FiltersSettings 31
3.1.8 ReplicationSettings 32

3.1.9 Settings 32
3.1.10 Config 32
3.2 Роли 32
3.2.1 Storage 32
3.2.2 Runner 32
3.2.3 Router 32
3.3 Сообщения 33
3.3.1 Оповещение (Notify) 33
3.3.2 Топология (Topology) 33
3.3.3 Информация о файле (FileInfo) 33
3.3.4 Директория файловой системы (FilesystemDirectory) 33
3.3.5 Запрос аутентификации клиента (ClientAuthRequest) 34
3.3.6 Запрос вставки (InsertRequest) 34
3.3.7 Запрос переименования (RenameRequest) 34
3.3.8 Запрос копирования/перемещения (CopyRequest/MoveRequest) 34
3.3.9 Запрос открытия/запуска файла (OpenRequest) 34
3.3.10 Ответ в строке состояния (StatusBarResponse) 34
3.3.11 Ответ открытия/запуска файла (OpenResponse) 35
3.3.12 Координаты и дельта координат (Coords/CoordsDelta) 35
3.3.13 Событие мыши (MouseEvent) 35
3.3.14 Событие клавиатуры (KeyboardEvent) 35
3.4 Сервер 36
3.5 Роль Storage 37
3.5.1 Уведомление роутера 37
3.5.2 Сбор данных о файловой системе 38
3.5.3 Обработчик добавления файла 39
3.5.4 Обработчик удаления файла 39
3.5.5 Обработчик получения файла 40
3.5.6 Обработчик копирования файла 41
3.5.7 Обработчик перемещения файла 42
3.5.8 Обработчик переименования файла 42
3.6 Роль Runner 43
3.6.1 Уведомление роутера 43
3.6.2 Обработчик открытия/запуска файлов 44
3.6.3 Обработчик получения изображения приложения 45
3.6.4 Обработчик события клавиатуры 45
3.6.5 Обработчик события мыши 46

3.6.6 Обработчик создания прямого стрима приложения 47
3.6.7 Обработчик создания стрима приложения....48
3.6.8 Стрим приложения 49
3.6.9 Выбор обработчика изображения 49
3.6.10 Захват изображения на ОС Windows 50
3.7 Роль Router 53
3.7.1 Сбор данных о файловой системе 53
3.7.2 Отслеживание устаревших инстансов 54
3.7.3 Репликация....55
3.7.4 Фильтрация файлов 57
3.7.5 Обработчик уведомлений 58
3.7.6 Обработчик открытия/запуска файлов 59
4 Методика тестирования 60
4.1 Просмотр директории файловой системы 60
4.2 Переход в новую директорию 60
4.3 Создание файла/директории 60
4.4 Удаление файла/директории 61
4.5 Копирование файла/директории 61
4.6 Перемещение файла/директории 61
4.7 Открытие файла 61
4.8 Результаты 62
ЗАКЛЮЧЕНИЕ 64
СПИСОК ЛИТЕРАТУРЫ 65

ВВЕДЕНИЕ

Распределенные файловые системы играют ключевую роль в современных информационных технологиях, обеспечивая доступ к данным из различных мест и устройств, а также обеспечение их надежности и безопасности. В условиях постоянного роста объема данных и необходимости их быстрого доступа, разработка эффективных и надежных распределенных файловых систем становится актуальной задачей как для исследователей, так и для практиков в области информационных технологий.

Целью данной дипломной работы является разработка концепции и прототипа распределенной файловой системы, способной обеспечить высокую производительность, надежность и безопасность хранения и обработки данных в сетевых средах. Для достижения поставленной цели необходимо решить ряд научно-технических задач, связанных с проектированием архитектуры системы, выбором подходящих алгоритмов и методик для обеспечения согласованности данных, а также реализацией механизмов защиты от несанкционированного доступа.

В работе будет проведен анализ существующих распределенных файловых систем, выявлены их сильные и слабые стороны, что позволит определить направления для дальнейшего развития и улучшения. Особое внимание будет уделено вопросам обеспечения отказоустойчивости и масштабируемости системы, а также ее адаптивности к изменяющимся условиям эксплуатации.

Разработанный прототип распределенной файловой системы будет проходить тестирование на соответствие требованиям к производительности, надежности и безопасности, а также будет оцениваться его применимость в реальных условиях использования. Результаты исследования и разработки будут представлены в виде подробного описания архитектуры системы, алгоритмов и методик, примененных при ее создании, а также результатов тестирования и анализа.

Данная работа призвана внести вклад в развитие теории и практики создания распределенных файловых систем, предложить новые подходы и решения, которые могут быть использованы в будущих проектах в этой области.

1 Обзор существующих распределенных файловых систем хранения

Распределённые файловые системы представляют собой сложные архитектурные конструкции, предназначенные для обеспечения совместного доступа к файлам и данным, хранящимся на разных компьютерах в сети. Они отличаются от традиционных сетевых файловых систем тем, что обеспечивают более глубокую интеграцию и согласованность данных между узлами сети, а также повышают доступность и надежность хранения данных за счёт распределения информации по нескольким физическим местам.

Основные характеристики распределённых файловых систем:

- ***Масштабируемость***

  Распределённые файловые системы должны обладать возможностью масштабирования в случае увеличения нагрузки. Это означает, что система должна быть способна адаптироваться к росту объема данных и числа пользователей, добавляя новые компоненты или узлы в сеть без значительных нарушений работы.

- ***Мобильность файлов***

  В распределённых файловых системах предусмотрена возможность перемещения файлов из одного месторасположения в другое на работающей системе. Это позволяет пользователям и приложениям легко переносить данные между различными узлами сети, что увеличивает гибкость и удобство использования системы.

- ***Сетевая прозрачность***

  Одной из ключевых особенностей распределённых файловых систем является сетевая прозрачность. Это означает, что пользователи и приложения должны иметь возможность обращаться к удалённым файлам так, как будто они находятся локально, без необходимости знать о сетевой топологии или физическом расположении файлов.

- ***Прозрачность размещения***

  Имя файла в распределённых файловых системах не должно определять его местоположения в сети. Это обеспечивает гибкость в управлении файлами и позволяет им перемещаться между различными узлами сети без изменения их идентификаторов.

- ***Независимость размещения***

  Имя файла не должно меняться при изменении его физического месторасположения. Это гарантирует, что пользователи и приложения могут обращаться к файлам по неизменным адресам, даже когда сами файлы перемещаются или реплицируются на другие узлы.

- ***Мобильность пользователя***

  Пользователи должны иметь возможность обращаться к разделяемым файлам из любого узла сети. Это обеспечивает удобство использования и доступность данных для пользователей, находящихся в разных географических местах.

- ***Устойчивость к сбоям***

  Распределённые файловые системы должны быть устойчивы к сбоям отдельных компонентов, таких как серверы или сегменты сети. В случае отказа одного из узлов, система должна продолжать функционировать, возможно, с некоторым снижением производительности или доступа к некоторым данным, но без полного прекращения работы.

Реализационные аспекты распределённых файловых систем касаются внутренней структуры и механизмов, которые обеспечивают функционирование системы на уровне программного обеспечения. Эти аспекты включают в себя способы обработки запросов, управления данными, обеспечения безопасности и координации между узлами сети.

Ключевые аспекты реализации распределённых файловых систем:

- ***Монтирование удалённых файловых систем***

  В ОС UNIX и подобных системах, распределённые файловые системы часто реализуются через монтирование удалённых файловых систем к каталогам локальной файловой системы. Это позволяет пользователям и приложениям обращаться к удалённым файлам так, как будто они являются частью локальной файловой системы.

- ***Использование файлов***

  При разработке распределённых файловых систем важно учитывать, как система будет использоваться. Исследования показывают, что большинство файлов имеют размер менее 10КБ, чтение встречается чаще, чем запись, и большинство файлов имеют короткое время жизни. Эти факторы влияют на выбор стратегий кэширования, упреждающего чтения и группировки операций.

- ***Файловый сервис и Сервис директорий***
  - Файловый сервис – процесс, который реализует файловый сервис, предоставляемый системой. Файловый сервер обычно является обычным пользовательским процессом, и в системе могут быть различные файловые серверы, предоставляющие различный сервис.
  - Сервис директорий – управляет структурой каталогов и обеспечивает доступ к ним. В распределённых файловых системах сервис директорий играет ключевую роль в обеспечении сетевой прозрачности и удобства навигации по файлам.

- ***Совместное резервирование файлов***

  При открытии файла клиент определяет тип доступа, который требует для себя, и тип доступа, в котором сервер должен отказать всем прочим клиентам. Это обеспечивает контроль над доступом к файлам и предотвращает конфликты при одновременном доступе к ресурсам.

- ***Интерфейс файлового сервера***
  Файловый сервис может базироваться на модели удаленного доступа, где каждый запрос обрабатывается на лету, или на модели загрузки/разгрузки, где данные периодически синхронизируются между узлами.

1.1 NFS (Network File System)

Сетевая файловая система позволяет пользователям получать доступ к файлам, хранящимся на удаленных серверах. Кроме того, пользователи не осознают, что файлы находятся на удаленном сервере, и обращаются к ним как к локальным файлам. Кроме того, NFS также обеспечивает безопасность и масштабируемость.

Сетевая файловая система реализована с использованием клиент-серверной архитектуры. В этой архитектуре один компьютер выступает в качестве сервера, а другие компьютеры - в качестве клиентов. Клиентский компьютер выдает запросы на данные, которые должен выполнить серверный компьютер. Существует два типа запросов, выдаваемых клиентскими компьютерами: запросы на чтение и запись.

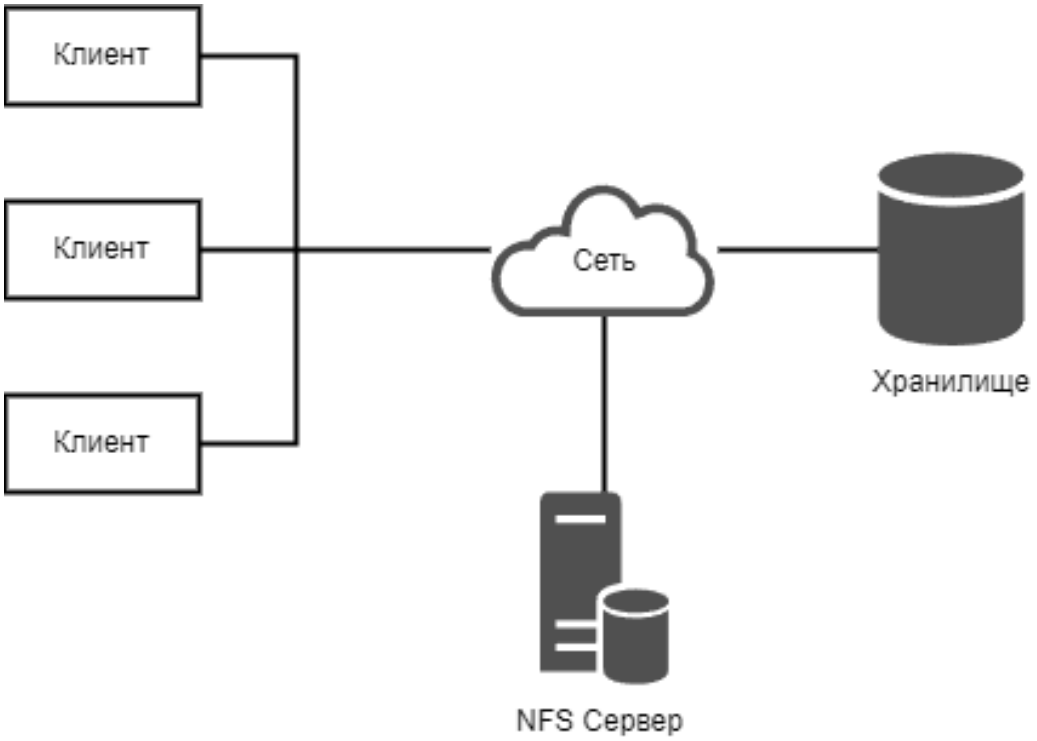

Рис. 1.1.1 – Архитектура NFS

Клиентский компьютер отправляет запрос на чтение серверу для чтения данных. Кроме того, клиентский компьютер может также отправить запрос на запись серверному компьютеру для записи данных. Запросы на чтение и запись реализуются с использованием стандартных операций чтения/записи. Компьютер-сервер выполняет запрос, используя соответствующий протокол. Затем он возвращает данные на компьютер клиента.

Преимущества NFS:

- NFS проста в использовании и не требует никакой настройки.
- Обеспечивает безопасность своих пользователей. Протокол использует строгую аутентификацию для защиты от несанкционированного доступа к данным.
- NFS обладает высокой масштабируемостью и может интегрировать данные из удаленных местоположений в локальное хранилище.
- Включает аварийное восстановление.

Недостатки NFS:

- NFS не обеспечивает никакой синхронизации между клиентом и сервером. Это означает, что данные на удаленном компьютере не синхронизированы с данными на компьютере клиента.
- Время доступа к файлу зависит от скорости сети, т.к. когда пользователь хочет получить доступ к файлу на сервере, файл должен пройти по сети на его компьютер.
- Если сервер выходит из строя, доступ к файлам невозможен.
- NFS поддерживает только один хост, что не позволяет обмениваться файлами между разными серверами, поскольку структура сети NFS допускает только один сервер.
- Не поддерживает иерархическое управление хранилищем. Если пользователь хочет хранить данные централизованно, он не сможет сделать это с помощью NFS.

1.2 ZFS (Zettabyte File System)

Эта система проектировалась с очень большим запасом по параметрам, на основе совершенно справедливого прогноза огромного роста данных, подлежащих хранению в распределенных системах в будущем.

Функции ZFS:

- Разделение системы хранения на пулы (Pooled storage).
- Копирование при записи (Copy-on-write).
- Снапшоты (Snapshots) системы.
- Верификация целостности данных и автоматическое исправление данных.
- Автоматическая замена на запасной диск (Hot spare).
- Максимальный размер файла 16 эксабайт.
- Объем хранения 256 квадриллионов зеттабайт.

1.2.1 Хранение в пулах (Pooled storage)

В отличие от большинства файловых систем, ZFS объединяет функции файловой системы и менеджера томов (volume manager). Это означает, что ZFS может создавать файловую систему, которая будет простираться по многим группам накопителей ZVOL или пулам. Более того, можно добавлять емкость в пул простым добавлением нового накопителя. ZFS сама выполнит партицию и форматирование нового накопителя.

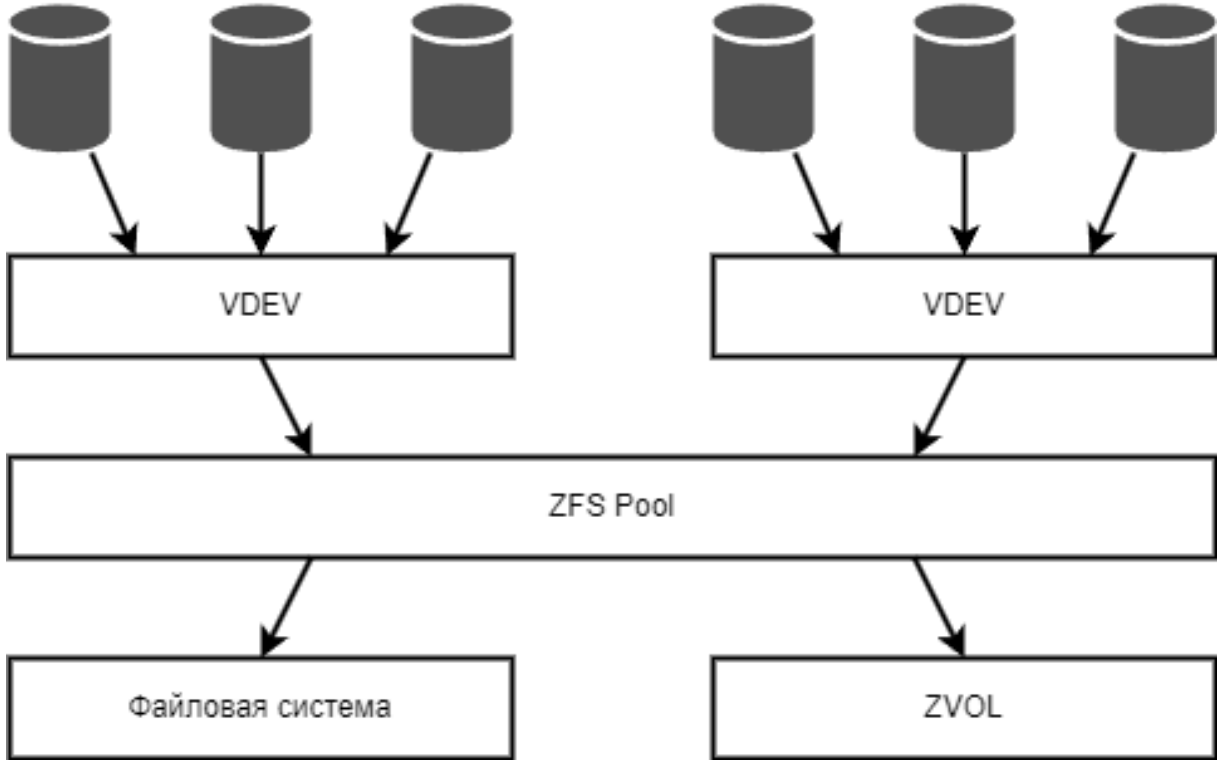

Рис. 1.2.1 – Архитектура ZFS

1.2.2 Копирование при записи (Copy-on-write)

В большинстве файловых систем при перезаписи данных на то же физическое место носителя ранее записанные там данные теряются навсегда. В ZFS новая информация пишется в новый блок. После окончания записи метаданные в файловой системе обновляются, указывая на местоположение нового блока. При этом, если в процессе записи информации с системой что-то происходит, старые данные будут сохранены. Это означает, что не нужно запускать проверку системы после аварии.

1.2.3 Снапшоты (Snapshots)

Copy-on-write закладывает основу для другой функции ZFS: моментальных снимков системы (снапшотов). ZFS использует для отслеживания изменений в системе.

Снапшот содержит оригинальную версию файловой системы, и в «живой» файловой системе присутствуют только изменения, которые были сделаны с момента последнего снапшота. Никакого дополнительного пространства не используется. Когда новые данные записываются в «живую» систему, выделяются новые блоки для сохранения этих данных.

Если файл удаляется, то ссылка на него в снапшоте тоже удаляется. Поэтому снапшоты в основном предназначены для отслеживания изменений в файлах, а не для добавления или создания файлов.

1.3 GFS

GFS является наиболее, наверное, известной распределенной файловой системой. Надежное масштабируемое хранение данных крайне необходимо для любого приложения, работающего с таким большим массивом данных, как все документы в интернете. GFS является основной платформой хранения информации в Google. GFS — большая распределенная файловая система, способная хранить и обрабатывать огромные объемы информации.

Файлы в GFS организованы иерархически, при помощи каталогов, как и в любой другой файловой системе, и идентифицируются своим путем. С файлами в GFS можно выполнять обычные операции: создание, удаление, открытие, закрытие, чтение и запись.
Более того, GFS поддерживает резервные копии, или снимки (snapshot). Можно создавать такие резервные копии для файлов или дерева директорий, причем с небольшими затратами.

Архитектура GFS

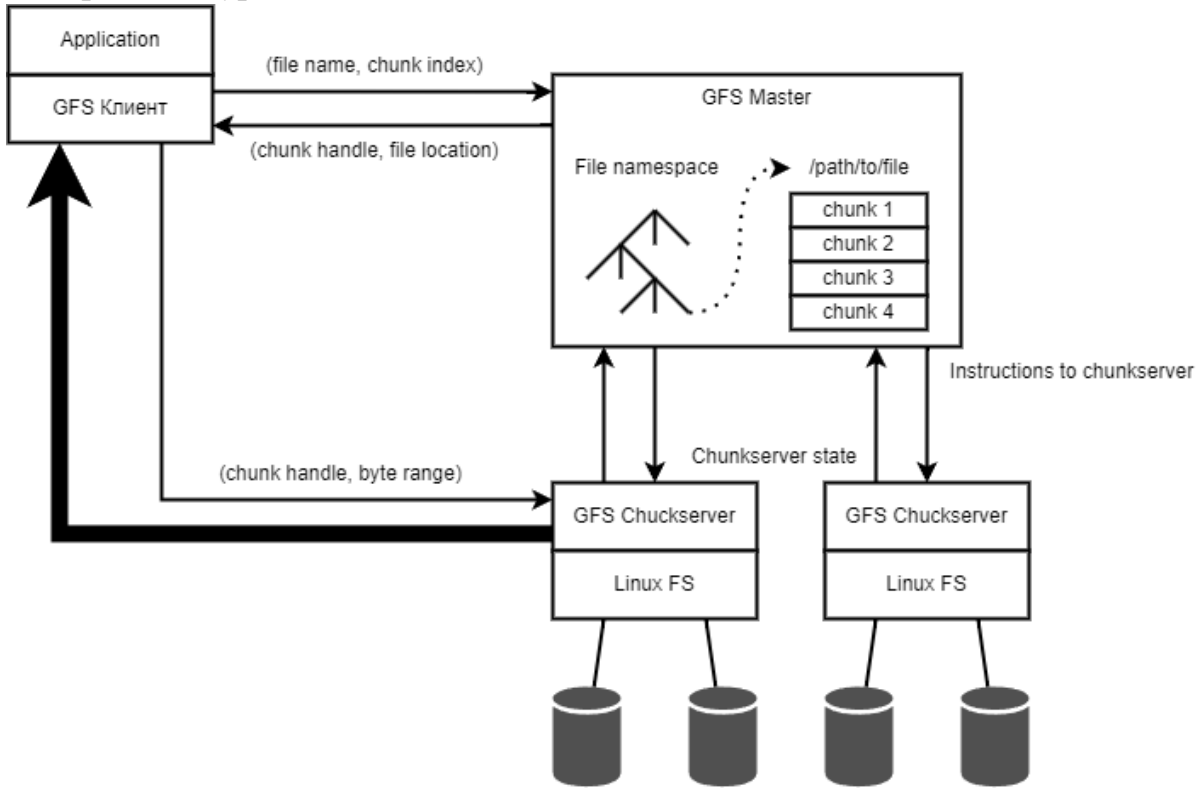

Рис. 1.3.1 – Архитектура GFS

В системе существуют мастер-сервера и чанк-сервера, собственно, хранящие данные. GFS кластер состоит из одной главной машины мастера (master) и множества машин, хранящих фрагменты файлов чанк-серверы (chunkservers). Клиенты имеют доступ ко всем этим машинам. Файлы в GFS разбиваются на куски — чанки (chunk). Чанк имеет фиксированный размер, который может настраиваться. Каждый такой чанк имеет уникальный и глобальный 64 — битный ключ, который выдается мастером при создании чанка. Чанк-серверы хранят чанки, как обычные Linux файлы, на локальном жестком диске. Для надежности каждый чанк может реплицироваться на другие чанк-серверы.

Мастер отвечает за работу с метаданными всей файловой системы. Метаданные включают в себя пространства имен, информацию о контроле доступа к данным, отображение файлов в чанки, и текущее положение чанков. Также мастер контролирует всю глобальную деятельность системы такую, как управление свободными чанками, сборка мусора и перемещение чанков между чанк-серверами. Мастер постоянно обменивается сообщениями с чанк-серверами, чтобы отдать инструкции, и определить их состояние.

Клиент взаимодействует с мастером только для выполнения операций, связанных с метаданными.

Использование одного мастера существенно упрощает архитектуру системы. Позволяет производить сложные перемещения чанков, организовывать репликации, используя глобальные данные. Казалось бы, что наличие только одного мастера должно являться узким местом системы, но это не так. Клиенты никогда не читают и не пишут данные через мастера. Вместо этого они спрашивают у мастера, с каким чанк-сервером они должны контактировать, а далее они общаются с чанк-серверами напрямую.

1.4 HDFS

HDFS (Hadoop Distributed File System) — распределенная файловая система Hadoop для хранения файлов больших размеров с возможностью потокового доступа к информации, поблочно-распределённой по узлам вычислительного кластера, который может состоять из произвольного аппаратного обеспечения. Hadoop Distributed File System, как и любая файловая система – это иерархия каталогов с вложенными в них подкаталогами и файлами.

Архитектура HDFS

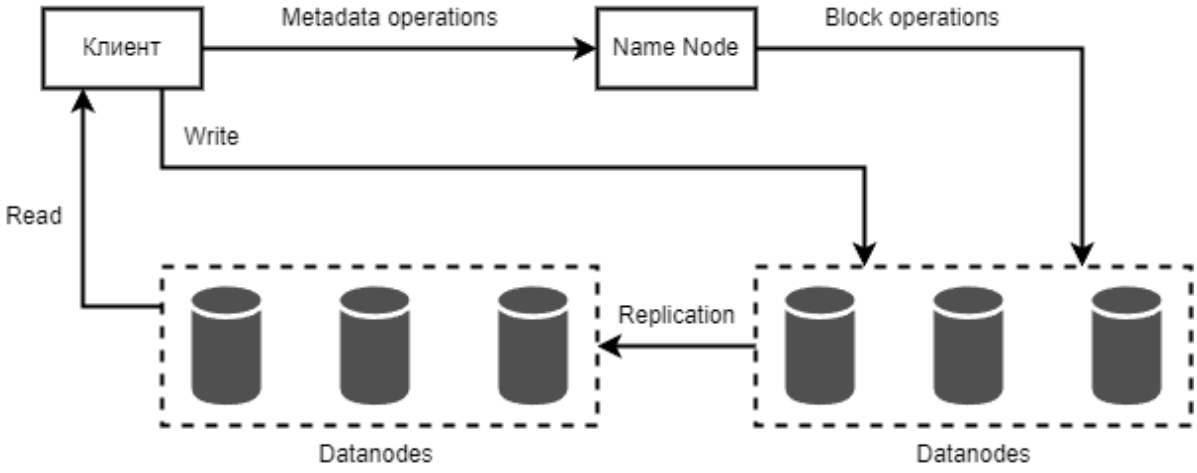

Рис. 1.4.1 – Архитектура HDFS

Управляющий узел (NameNode) – отдельный, единственный в кластере, сервер для управления пространством имен файловой системы, хранящий дерево файлов, а также метаданные файлов и каталогов. NameNode – обязательный компонент кластера HDFS, который отвечает за открытие и закрытие файлов, создание и удаление каталогов, управление доступом со стороны внешних клиентов и соответствие между файлами и блоками, дублированными на узлах данных.

Secondary NameNode — вторичный узел имен, отдельный сервер, единственный в кластере, который копирует образ HDFS и лог транзакций операций с файловыми блоками во временную папку, применяет изменения, накопленные в логе транзакций к образу HDFS, а также записывает его на узел NameNode и очищает лог транзакций. Необходим для быстрого ручного восстановления NameNode в случае его выхода из строя.

Узел или сервер данных (DataNode, Node) – один их множества серверов, отвечающих за файловые операции и работу с блоками данных. Данные проходят с остальных узлов кластера к клиенту мимо узла NameNode.

Клиент (Client) – пользователь или приложение, взаимодействующий через специальный интерфейс с распределенной файловой системой. Создавая файл, клиент может явно указать размер блока файла (по умолчанию 64 Мб) и количество создаваемых реплик (по умолчанию значение равно 3-ем).

Особенности:

- большой размер блока по сравнению с другими файловыми системами (>64MB), поскольку HDFS предназначена для хранения большого количества огромных (>10GB) файлов.
- ориентация на недорогие и, поэтому не самые надежные сервера – отказоустойчивость всего кластера обеспечивается за счет репликации данных;
- репликация происходит в асинхронном режиме – информация распределяется по нескольким серверам прямо во время загрузки, поэтому выход из строя отдельных узлов данных не повлечет за собой полную пропажу данных.
- HDFS оптимизирована для потоковых считываний файлов, поэтому применять ее для нерегулярных и произвольных считываний нецелесообразно, файлы пишутся однократно, что исключает внесение в них любых произвольных изменений.
- принцип WORM (Write-once and read-many, один раз записать – много раз прочитать) полностью освобождает систему от блокировок типа «запись-чтение». Запись в файл в одно время доступен только одному процессу, что исключает конфликты множественной записи.
- сжатие данных и рациональное использование дискового пространства позволило снизить нагрузку на каналы передачи данных, которые чаще всего являются узким местом в распределенных средах.
- самодиагностика — каждый узел данных через определенные интервалы времени отправляет диагностические сообщения узлу имен, который записывает логи операций над файлами в специальный журнал.

Недостатки:

- сервер имен является центральной точкой всего кластера и его отказ повлечет сбой системы целиком.
- отсутствие полноценной репликации Secondary NameNode.
- отсутствие возможности дописывать или оставить открытым для записи файлы в HDFS, за счет чего в классическом дистрибутиве Apache Hadoop невозможно обновлять блоки уже записанных данных.
- отсутствие инструментов для поддержки ссылочной целостности данных, что не гарантирует идентичность реплик. HDFS перекладывает проверку целостности данных на клиентов. При создании файла клиент рассчитывает контрольные суммы каждые 512 байт, которые в последующем сохраняются на сервере имен. При считывании файла клиент обращается к данным и контрольным суммам. В случае их несоответствия происходит обращение к другой реплике.

1.5 Ceph

Ceph – распределенная масштабируемая система хранения с дизайном без единой точки отказа. Система разрабатывалась как открытый проект с файловой системой высокой масштабируемости до уровня эксабайт и рассчитанная на работу на стандартном коммерчески доступном серверном оборудовании COTS (Commercial Off The Shelf). Ceph приобретает все большую популярность в облачных системах хранения данных.

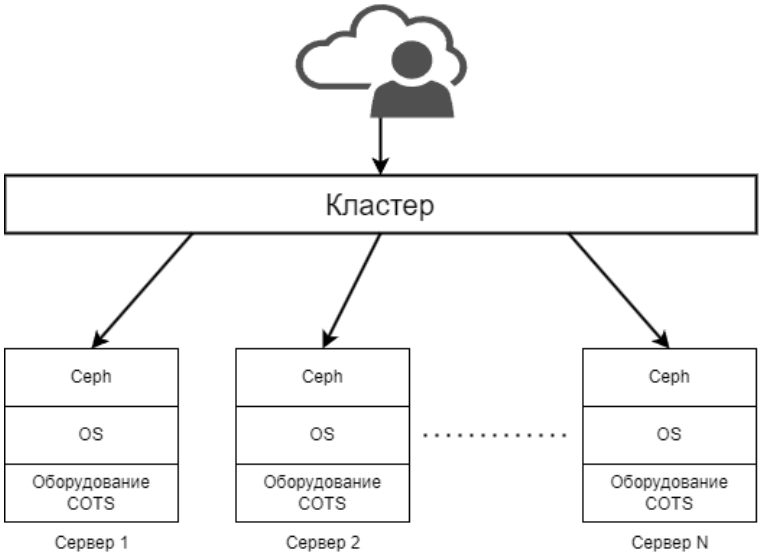

Рис.1.5.1 – Архитектура Ceph

Основные принципы построения Ceph:

- Масштабируемость всех компонентов.
- Отсутствие единой точки отказа.
- Использование открытого ПО (Linux).
- Использование стандартного оборудования COTS (Commercial Off The Shelf).
- Автономность в управлении компонентами.

Ceph для клиента выглядит как обычная файловая система с папками и файлами, организованными по принципу иерархии.

Ceph реплицирует данные, за счет чего файловая система устойчива к отказам при использовании обычного стандартного компьютерного оборудования COTS, которое не требует специального обслуживания или администрирования. Поэтому она хорошо подходит для организаций, которые не обладают экспертизой в IT, и в которых недостаточно средств для содержания штата IT-специалистов.

1.6 Lustre

Lustre – это кластерная архитектура для СХД, центральным элементом которой является файловая система Lustre, работающая на ОС Linux и обеспечивающая POSIX-интерфейс для файловой системы UNIX.

Lustre способна масштабировать емкость и производительность практически до любых требуемых величин, устраняя надобность в развертывании многих отдельных файловых систем в каждом компьютерном кластере. Кроме агрегирования емкости хранения многих серверов, пропускная способность ввода-вывода также агрегируется и масштабируется при добавлении серверов – в том числе в динамическом режиме.

Lustre также не очень подходит для сетевых моделей peer-to-peer, где клиенты и серверы работают на одном узле хранения, и каждый занимает небольшую область емкости, из-за недостаточной репликации на уровне ПО Lustre. При этом, если один клиент или сервер отказывает, то данные этого узла будут недоступны, пока узел не перезапустится.

Архитектура Lustre

Сервер метаданных MDS (Metadata Server) – обеспечивает доступ к метаданным файловой системе и управляет запросами клиентов.

Узел метаданных – хранит метаданные файловой системы.

Сервер хранения – хранит данные.

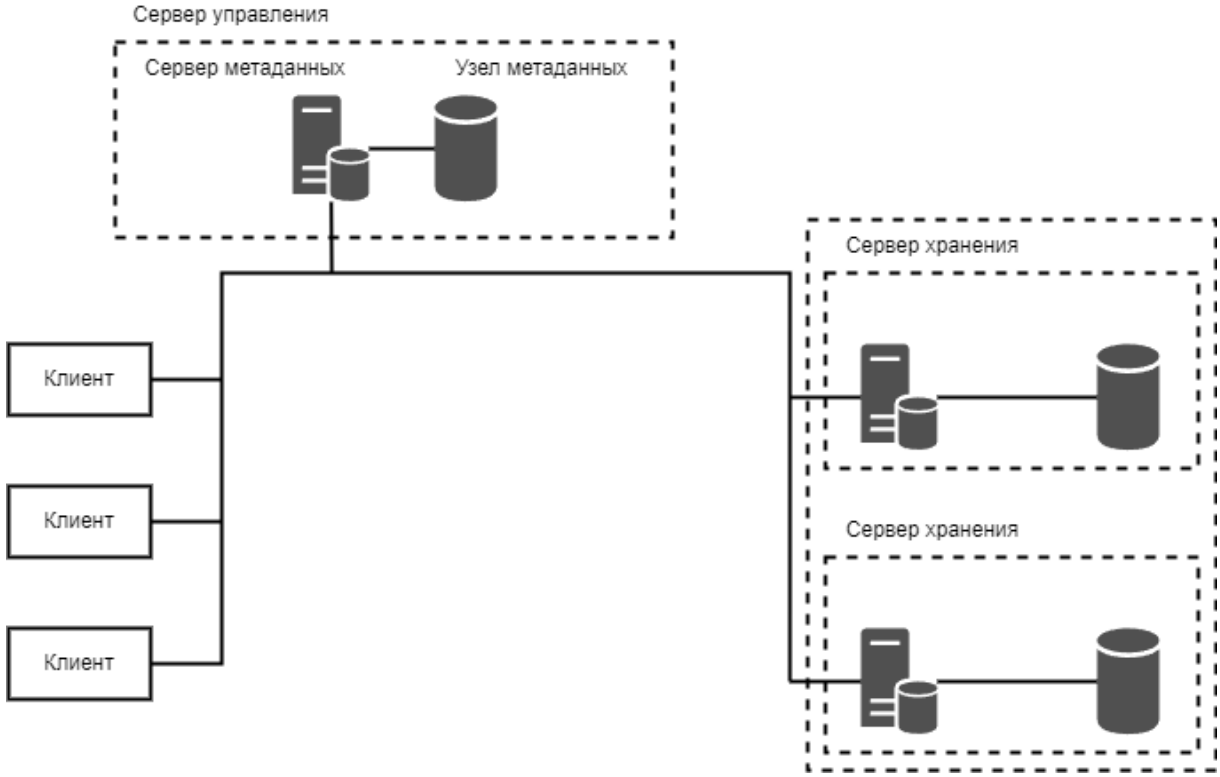

Рис. 1.6.1 – Архитектура Lustre

Архитектура серверов хранения

Сервер объектного хранения – обрабатывает запросы данных.

Целевой узел объектного хранения – диск с данными.

Любой узел может быть доступен для многих серверов, но сервер может иметь доступ только к одному узлу в какой-то момент времени.

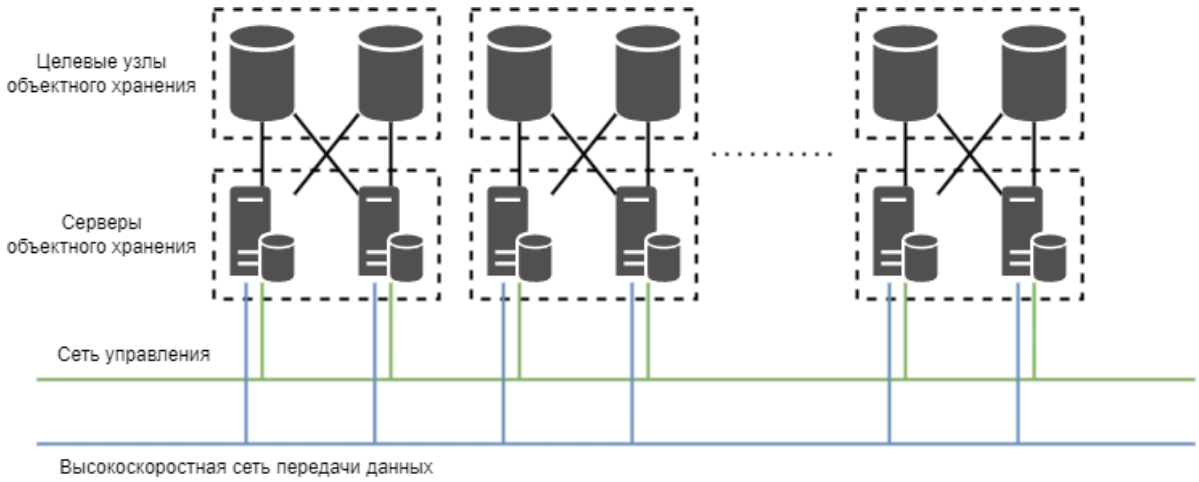

Рис. 1.6.2 – Архитектура серверов хранения Lustre

1.7 Сравнение

Самым главным отличительным плюсом всех рассмотренных архитектур является практически бесконечная расширяемость хранилищ данных. При этом все файловые системы используют сеть для соединения хранилищ между собой.

Ниже приведена таблица сравнения рассмотренных систем, основанная на удобстве эксплуатации.

	Наличие единой точки отказа	**Требование специального оборудования**	**Зависимость от платформы**
NFS	+		
ZFS			
GFS	+		
HDFS			
Ceph		+	+
Lustre	+		+

Так же стоит отметить, что большинство файловых систем поддерживают механизм репликации, который повышает вероятность сохранения данных при физических поломках оборудования и в некоторых случаях снижает нагрузку на основное хранилище данных.

Стоит отметить, что в текущем контексте отсутствие плюса в пункте “Наличие единой точки отказа” подразумевает, что каждое хранилище является отдельным модулем и при его отказе пользователь не сможет получить доступ только к данным из этого хранилища. Сохранить доступ к данным при отказе хранилища без настроенного механизма репликации не получится.

2 Концепция приложения

2.1 Требования

Из обзорной части очевидно, что такие решения рассчитаны на уровень использования компаниями и для развёртывания такой системы необходимо закупать дополнительное оборудование. Однако все это не подходит для обычного пользователя, имеющего в своем распоряжении всего лишь несколько персональных устройств, что заставляет его использовать одновременно несколько устройств и передавать информацию через различные буферы в виде жестких накопителей информации. Поэтому первое и основное требование к приложению – **не использовать/требовать дополнительное оборудование, кроме персональных устройств самого пользователя**.

Конечно же пользователь может иметь в своём распоряжении и дополнительное оборудование, например, различные накопители и сервера, не подходящее для полноценного использования, но все еще имеющее возможности хранить большое количество данных или имеющее малое количество памяти, но достаточно мощные характеристики железа для обработки данных или не имеющее ни того, ни другого, но при этом очень портативное, чтобы было удобно иметь доступ откуда угодно. Поэтому стоит определить некоторые **роли запущенного приложения на определенном оборудовании**, а именно:

- **Storage** – устройство, обладающее только возможностью хранить данные.
- **Runner** – устройство, позволяющее запускать приложения без сильных потерь производительности. Имеет только копию данных, связанных с запущенным приложением.
- **Router** – определяет устройство, имеющее интерфейс доступа к данным. Выполняет сбор информации о файловой системе со стораджей и распределяет нагрузку на раннеры.

Таким образом, одно устройство может иметь от 1 до 3 ролей одновременно, так как они никак не конфликтуют друг с другом по функциональности.

Очевидно, что пользователь может обладать устройствами с различной операционной системой (Windows, MacOS, Linux, Android и другие Unix системы), поэтому следующим требованием является **независимость от операционной системы**.

Обычный пользователь с большой вероятностью не будет обладать достаточной экспертизой в настройке таких систем, следовательно **настройка должна происходить автоматически и с минимальным количеством параметров**.

Для сохранения данных с неустойчивых хранилищ **необходимо предоставить гибкий и точечный механизм репликации**, так как

пользователь с наибольшей вероятностью не обладает достаточно большим количеством памяти, чтобы иметь реплику всей файловой системы.

2.2 Особенности реализации и ограничения

Исходя из универсальности использования самым логичным будет **предоставить пользователю веб-интерфейс** для взаимодействия с файловой системой, переложив системные особенности взаимодействия с файлами на саму операционную систему.

Так же основываясь на автоматизации настройки приложения необходимо ввести ограничение на **использование только локальной сети** для существующий устройств пользователя, хорошим примером здесь будет использование роутера, который предоставляет пользователю как доступ к глобальной, так и локальной сетям одновременно.

Безопасность выполнения операций на уровне ОС так же ложится на **пользователя операционной системы,** под которым запущен инстанс приложения, позволяя гибко настроить доступ к данным на каждом устройстве.

2.3 Архитектура

Для минимально работающей системы необходимо, чтобы была запущена только одна роль – Router, т.к. она обеспечивает клиента интерфейсом. Но в таком случае никаких данных увидеть не получится.

Чтобы увидеть данные необходимо либо добавить роль Storage в уже запущенное приложение, либо запустить дополнительное с этой ролью. Теперь клиенту становится доступна возможность изменения состояния файловой системы устройства, на которой запущена эта роль. Если запустить ещё на одном устройстве приложение с этой ролью, то клиент сможет увидеть объединённую файловую систему двух хранилищ.

Однако, имея только вышеуказанные две роли, клиент не сможет изменять сами файлы. Для получения такой возможности необходимо добавить роль Runner. Она позволяет открывать файлы через предназначенные для них приложения и изменять их.

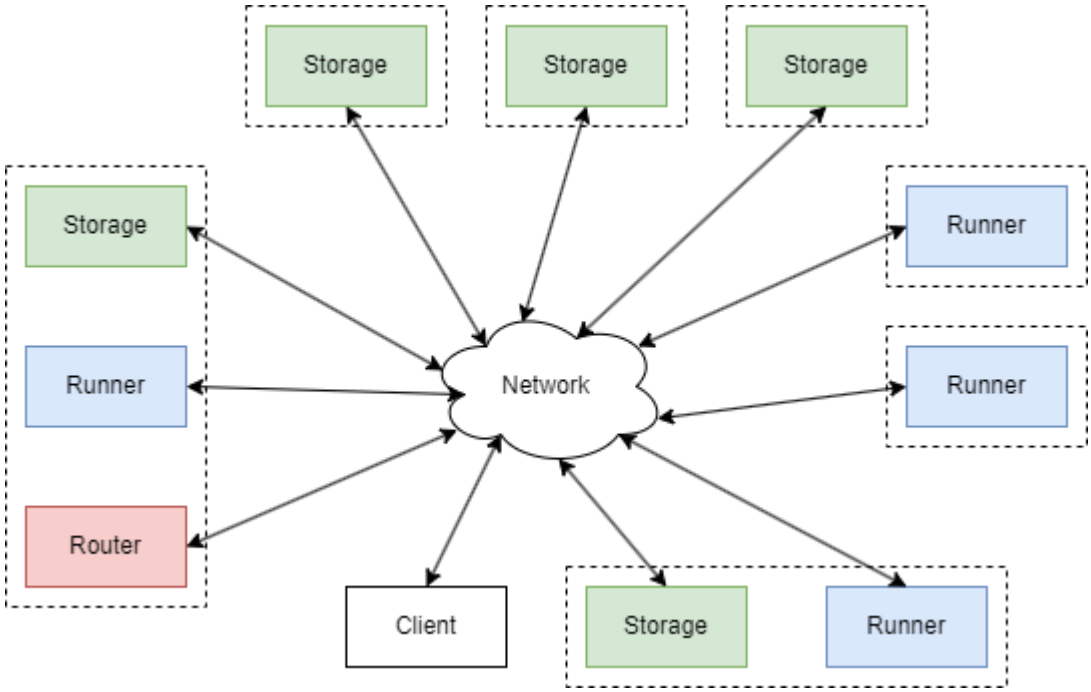

Рис. 2.3.1 – Архитектура приложения

2.4 Безопасность

В данной концепции приложения вопрос безопасности данных очень важен. Любой человек подключившийся к сети, зная правильные запросы, может с лёгкостью удалить все существующие данные или занести различные вредоносные программы. Поэтому используется два механизма безопасности.

Все приложения используют протокол https для общения. При надобности пользователь так же можешь перегенерировать сертификаты и заменить их.

Роль роутера поддерживает механизм аутентификации. При запуске приложение случайно генерирует достаточно большой токен на основе логина и пароля, которые указаны в конфигурационном файле или флагах запуска приложения. Все эндпоинты приложения включают в себя этот токен, поэтому,

не имея логина и пароля, становится невозможно отправить хотя бы какой-то запрос.

Таким образом каждый эндпоинт будет описываться шаблоном url/token/role/path.

2.5 Запуск ролей

Независимо от порядка запуска ролей каждая выполняет постоянные и периодические действия по поддержанию актуальной информации о кластере.

Роутер будет периодически отправлять широковещательные запросы в сеть.

Остальные роли при получении таких запросов будут пытаться пройти аутентификацию и получить токен, а после этого отправят оповещение на специальный эндпоинт.

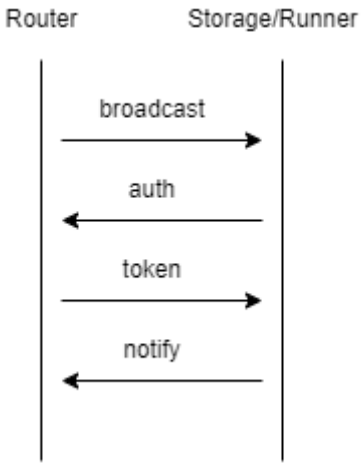

Рис. 2.5.1 – Алгоритм запуска роли

При запросе оповещения storage/runner передают следующую информацию: тип роли, свой адрес, имя хоста, свой токен, название платформы ОС и текущую метку времени (таймстамп).

После этого роли считаются соединёнными, и каждая знает необходимые токены других ролей, поэтому все остальные запросы будут выполняться без прохождения аутентификации.

Важно отметить, что перед отправкой уведомления роутеру инстанс проверяет совпадение сертификатов. Так же и со стороны роутера проверяется совпадение сертификатов от инстанса. Таким образом, невозможно будет подделать инстанс приложения для получения каких-либо данных, так как запрос будет отклонён на https уровне.

2.6 Функциональность

2.6.1 Просмотр файловой системы

При просмотре файловой системы роутер выполняет запрос информации на все известные хранилища. В ответ хранилища передают информацию о всех известных по указанному пути файлах и папках.

В передаваемую информацию входят имя, название хоста источника, название платформы источника, время последнего изменения и размер.

2.6.2 Изменения файловой системы

Все изменения производятся локально на хранилище, и запросы отправляются только к определённому хранилищу, основываясь либо на полученных метаданных файла, либо на выборе самого пользователя.

В список изменений файловой системы входят: создание, копирование, перемещение, удаление и переименование файлов.

2.6.3 Фильтрация показываемых файлов

Пользователь имеет возможность работать с чёрным и белым списками, в которых будут указаны пути директорий и файлов или шаблоны выражений (например, регулярные выражения).

2.6.4 Репликация

Пользователь имеет возможность настраивать частную репликацию файлов, указывая периодичность, хост источник и целевой хост, а также соответствующие пути к этим файлам.

Репликация данных — это процесс поддержания одинаковых данных на нескольких узлах распределенной системы. Она позволяет обеспечить отказоустойчивость, повышение доступности данных и балансировку нагрузки. Виды репликации данных включают:

- ***Синхронная репликация***

  В синхронной репликации изменения данных немедленно отражаются во всех репликах. Это обеспечивает высокую степень согласованности данных между репликами, но может привести к задержкам в обработке запросов из-за необходимости ожидания подтверждения операций на всех узлах.

- ***Асинхронная репликация***

  В асинхронной репликации изменения данных передаются между узлами без ожидания подтверждения операций. Это позволяет уменьшить задержку и увеличивает пропускную способность системы, но может привести к временным несогласованностям данных между репликами.

- ***Односторонняя репликация***

  Односторонняя репликация позволяет многократно производить отправку изменений данных либо из родительской реплики в дочернюю, либо из дочерней реплики в родительскую. Это может быть полезно для создания резервных копий или для разделения нагрузки между различными узлами.

- ***Двухсторонняя репликация***

  Двухсторонняя репликация позволяет отправлять изменения данных из родительской реплики в дочернюю и наоборот. Это обеспечивает двустороннюю синхронизацию данных между репликами, что может быть необходимо для обеспечения высокой доступности и отказоустойчивости в критически важных системах.

- ***Репликация триггерами***

  Репликация с использованием триггеров позволяет автоматически обрабатывать изменения данных и синхронизировать их между различными системами или базами данных. Это может включать преобразование данных, мониторинг состояния и другие операции, что обеспечивает большую гибкость и независимость от конкретных решений по хранению данных.

Для обычного пользователя, учитывая его вычислительные мощности, лучшим выбором будет асинхронная односторонняя репликация.

2.6.5 Открытие файлов и работа с ними

Открытие файлов происходит на инстансе с ролью runner в соответствующем ему установленном в системе приложении.

Если инстанс имеет две роли storage и runner, то приложение будет приоритетно открыто на нём. В случае ошибки открытия или отсутствия роли runner, запрос на открытие будет отправлен другому хосту с соответствующей платформой, а файл будет скопирован для его локального открытия. После завершения работы устаревший файл на хосте источнике будет заменён файлом открытом на хосте с ролью runner, а временный рабочий файл удалён.

Так же для работы с файлом будет запущен стрим приложения, которое открыло этот файл, в отдельном веб-окне. Все нажатия мыши, клавиатуры, нажатия на экран мобильного устройства и прокрутка будут переданы в соответствующее приложение.

Для отображения изображений наиболее популярными являются три формата: PNG, JPEG, и относительно новый формат WebP.

2.6.5.1 PNG

Формат PNG (Portable Network Graphics) представляет собой растровый графический формат, разработанный как преемник формата GIF в 1995 году. Этот формат широко используется для веб-графики благодаря своей способности обеспечивать высокое качество изображений с поддержкой прозрачности и гамма-коррекции, что позволяет корректно отображать цвета на разных устройствах и платформах. PNG также поддерживает сжатие данных без потерь качества, что делает его идеальным для использования в интернете и на сайтах.

Основные характеристики и преимущества формата PNG включают:

- Прозрачность: поддержка альфа-каналов позволяет изображениям иметь переходную прозрачность, что особенно полезно для веб-графики.
- Гамма-коррекция: улучшает отображение изображений на различных устройствах путем коррекции яркости.
- Сжатие без потерь: PNG использует алгоритмы сжатия, такие как deflate, для достижения высокого уровня сжатия изображений без потери качества.

- Универсальность: из-за стандартизации и отсутствия патентных ограничений, PNG поддерживается большинством графических редакторов и библиотек, что обеспечивает совместимость между различными приложениями.

2.6.5.2 JPEG

Формат JPEG (Joint Photographic Experts Group) является одним из наиболее популярных и широко используемых форматов изображений для цифровой фотографии и веб-разработки. Разработанный международной группой экспертов по фотографическим технологиям в конце 1980-х и начале 1990-х годов, JPEG стал стандартом для хранения и передачи фотографических изображений в Интернете благодаря своему методу сжатия с потерями, который позволяет уменьшить размер файла за счет некоторой потери качества изображения.

Основные характеристики JPEG:

- Сжатие с потерями: JPEG использует алгоритмы сжатия, которые удаляют информацию, которая менее заметна для человеческого глаза, таким образом уменьшая размер файла. Это может привести к некоторой потере качества изображения, но уровень качества обычно приемлем для большинства приложений.
- Поддержка цветовой палитры: JPEG поддерживает цветовые пространства RGB и CMYK, а также различные уровни глубины цвета до 24 бит на пиксель.
- Метаданные EXIF: большинство JPEG-файлов содержат метаданные EXIF, которые предоставляют дополнительную информацию о файле, такую как дата и время создания, настройки камеры, геолокация и др.
- Общепринятость: благодаря своей универсальности и широкой поддержке, JPEG является одним из самых распознаваемых форматов изображений, совместимым с большинством браузеров, программного обеспечения и мобильных приложений.

Преимущества JPEG:

- Маленькие размеры файлов: сжатие с потерями позволяет JPEG-файлам быть значительно меньше по размеру по сравнению с форматами без потерь, что упрощает их загрузку и передачу.
- Большая совместимость: JPEG поддерживается практически всеми устройствами и платформами, что делает его идеальным выбором для обмена изображениями.
- Легкость обработки: из-за сжатия с потерями JPEG часто используется для быстрого просмотра изображений, поскольку они занимают меньше места и открываются быстрее.

2.6.5.3 WebP

Формат WebP был разработан компанией Google в 2010 году как современный альтернативный формат изображений для веб-разработки. Он предлагает эффективное сжатие изображений как с потерями, так и без потерь качества, основываясь на алгоритмах сжатия, адаптированных из видеокодека VP8. WebP использует контейнер RIFF (Resource Interchange File Format), который позволяет легко интегрировать этот формат в существующие системы и приложения.

Основные характеристики и преимущества формата WebP включают:

- Эффективное сжатие: WebP обеспечивает значительно лучшее сжатие изображений по сравнению с традиционными форматами, такими как JPEG и PNG, сохраняя при этом высокое качество изображений.
- Поддержка прозрачности: Формат поддерживает альфа-каналы, что позволяет создавать изображения с прозрачностью.
- Открытый стандарт: WebP является открытым форматом, что означает, что он доступен для использования всеми разработчиками и организациями без лицензионных ограничений.
- Широкая поддержка: несмотря на то, что формат относительно новый, поддержка WebP постепенно распространяется среди основных браузеров и операционных систем. Например, Google Chrome, Opera и Firefox поддерживают отображение изображений в формате WebP, а Android начиная с версии 4.0 поддерживает чтение и запись таких изображений.
- Однако, несмотря на все преимущества, важно учитывать, что некоторые старые браузеры и приложения могут не поддерживать формат WebP, что требует от разработчиков предоставления запасных вариантов изображений в более устоявшихся форматах для обеспечения совместимости.

2.7 Флаги запуска

- cert-crt – путь к файлу vss.crt (по умолчанию "certificates/vss.crt")
- cert-key – путь к файлу vss.key (default "certificates/vss.key")
- url – адрес сервера
- listen-port – порт для топологии (по умолчанию "3311")
- log-enable – включает логи
- log-level – уровень логирования (error, info, verbose, debug) (по умолчанию "info")
- log-path – путь к файлу логов (default "logs/log_2024-05-14T07:06:16.txt")
- platform - платформа ('windows', 'linux', 'darwin', …) (по умолчанию "windows")
- storage – включает роль storage
- router – включает роль router
- runner – включает роль runner
- token – токен безопасности
- username – имя пользователя (по умолчанию "admin")
- password – пароль для аутентификации (по умолчанию "admin")

3 Реализация

3.1 Конфигурация

Роль конфигурации в программе играет критически важную функцию в определении её поведения, функциональности и взаимодействия с пользователем. Конфигурация позволяет настраивать различные аспекты программы, включая права доступа, параметры работы, интерфейс пользователя и многое другое.

3.1.1 StorageRole

Содержит поля:

- Enable – флаг включения роли storage.

3.1.2 RunnerRole

Содержит поля:

- Enable – флаг включения роли runner.
- Platform – имя платформы ОС.

3.1.3 RouterRole

Содержит поля:

- Enable – флаг включения роли router.

3.1.4 Roles

Содержит поля:

- Storage – объект StorageRole.
- Runner – объект RunnerRole.
- Router – объект RouterRole.

3.1.5 Log

Содержит поля:

- Enable – флаг включения логирования.
- Path – путь к файлу.
- Level – уровень логирования.

3.1.6 User

Содержит поля:

- Username – имя пользователя.
- Password – пароль.
- Token – токен аутентификации.

3.1.7 FiltersSettings

Содержит поля:

- BlackList – чёрный список.
- WhiteList – белый список.
- CurrentList – текущий выбранный список.

3.1.8 ReplicationSettings

Содержит поля:

- SrcHostname – имя хоста источника.
- DstHostname – имя целевого хоста.
- SrcPath – путь к файлу источника.
- DstPath – путь к целевому файлу.
- Cron – крон строка.
- CronName – название периода.

3.1.9 Settings

Содержит поля:

Filters – настройки фильтрации.

Replication – настройки репликации.

3.1.10 Config

Содержит поля:

- Url – адрес сервера.
- Hostname – имя хоста.
- ListenPort – порт сервера для определения топологии
- Roles – настройки ролей.
- Log – настройки логирования.
- User - настройки пользователя.
- Certificate – сертификат для безопасного общения инстансов.
- RootCAs – зашифрованный сертификат.
- Settings – настройки.

3.2 Роли

3.2.1 Storage

Содержит поля:

- Config – конфигурация инстанса.

3.2.2 Runner

Содержит поля:

- Config – конфигурация инстанса.
- streamSessions – таблица сессий стрима, ключ – id процесса.

3.2.3 Router

Содержит поля:

- Config – конфигурация инстанса.
- Storages – список инстансов с ролью storage.
- Runners – список инстансов с ролью runner.
- Hostnames – таблица инстансов, ключ – имя хоста.
- ReplicationTasks – список задач репликации.

3.3 Сообщения

3.3.1 Оповещение (Notify)

Содержит поля:

- Type – имя роли.
- Url – адрес хоста.
- Hostname – имя хоста.
- Token – токен безопасности.
- Platform – название платформы.
- Timestamp – метка времени в наносекундах.

Используется для оповещения роутера о появлении инстанса приложения в сети.

3.3.2 Топология (Topology)

Содержит поля:

- Storages – массив Notify структур, описывающих приложения с ролью storage.
- Runners – массив Notify структур, описывающих приложения с ролью runner.

Используется для хранения информации о кластере, поиска данных в запросах и отображения информации о топологии на основной странице.

3.3.3 Информация о файле (FileInfo)

Содержит поля:

- ModTime – время последней модификации.
- Size – размер.
- Url – адрес хранилища.
- Platform – название платформы.
- Hostname – имя хоста.

Используется для предоставления информации о файловой системе.

3.3.4 Директория файловой системы (FilesystemDirectory)

Содержит поля:

- Directories – таблица ключ-значение, описывающих директории, где ключ – имя директории, а значение FileInfo структура.
- Files – таблица ключ-значений, описывающих файлы, где ключ – имя файла, а значение массив FileInfo структур, т.к. на разных хостах в одной директории могут храниться файлы с одинаковым именем.

Используется для предоставления информации о файловой системе.

3.3.5 Запрос аутентификации клиента (ClientAuthRequest)

Содержит поля:

- Username – имя пользователя.
- Password – пароль пользователя.

Используется для аутентификации пользователя и ролей приложения.

3.3.6 Запрос вставки (InsertRequest)

Содержит поля:

- Type – тип файла.
- Path – путь до директории.
- Name – имя файла.

Используется для добавления файла или директории.

3.3.7 Запрос переименования (RenameRequest)

Содержит поля:

- Path – путь до директории.
- OldName – старое имя файла.
- NewName – новое имя файла.

Используется для переименования файлов и директорий.

3.3.8 Запрос копирования/перемещения (CopyRequest/MoveRequest)

Содержит поля:

- SrcPath – путь к файлу на хосте источнике.
- DstPath – путь к файлу на целевом хосте.
- SrcUrl – адрес хоста источника.

Используется для копирования или перемещения файлов и директорий.

3.3.9 Запрос открытия/запуска файла (OpenRequest)

Содержит поля:

- Platform – платформа ОС.
- Path – путь к файлу.
- SrcUrl – адрес хоста источника.
- Hostname – имя хоста источника.

Используется для старта запуска выбранного файла.

3.3.10 Ответ в строке состояния (StatusBarResponse)

Содержит поля:

- Status – статус (успех/провал).
- Text – комментарий.

Используется для передачи статуса выполнения запросов.

3.3.11 Ответ открытия/запуска файла (OpenResponse)

Содержит поля:

- Pid – идентификатор процесса запустившего файл.
- RunnerUrl – адрес хоста, на котором был запущен файл.
- ClientUrl – адрес хоста клиента, который отправил запрос.
- Error – ошибка открытия.
- StatusBar – статут выполнения запроса.

Используется для ответа ролей runner и router, и дальнейшего получения информации и о открытом/запущенном файле.

3.3.12 Координаты и дельта координат (Coords/CoordsDelta)

Содержит поля:

- X – абсолютное горизонтальное значение относительно нуля координат или изменение по сравнению с предыдущим значением.
- Y – абсолютное вертикальное значение относительно нуля координат или изменение по сравнению с предыдущим значением.

Используется для передачи координат событий мыши.

3.3.13 Событие мыши (MouseEvent)

Содержит поля:

- Type – тип события (нажата или отпущена левая/правая клавиша, прокрутка)
- Coords – абсолютные координаты курсора мыши в момент события.
- Scroll – изменения координат при прокрутке.

Используется для передачи событий нажатия кнопки мыши или прокрутки.

3.3.14 Событие клавиатуры (KeyboardEvent)

Содержит поля:

- Type – тип нажатия клавиши (нажата или отпущена).
- Keycode – код клавиши.

Используется для передачи события нажатия клавиши на клавиатуре.

3.4 Сервер

Сервер – основа приложения, он занимается аутентификацией пользователя и других инстансов приложения и перенаправляет запросы в соответствующие обработчики.

Алгоритмы аутентификации и обработки запросов приведены на рисунках 3.2.1 и 3.2.2 соответственно.

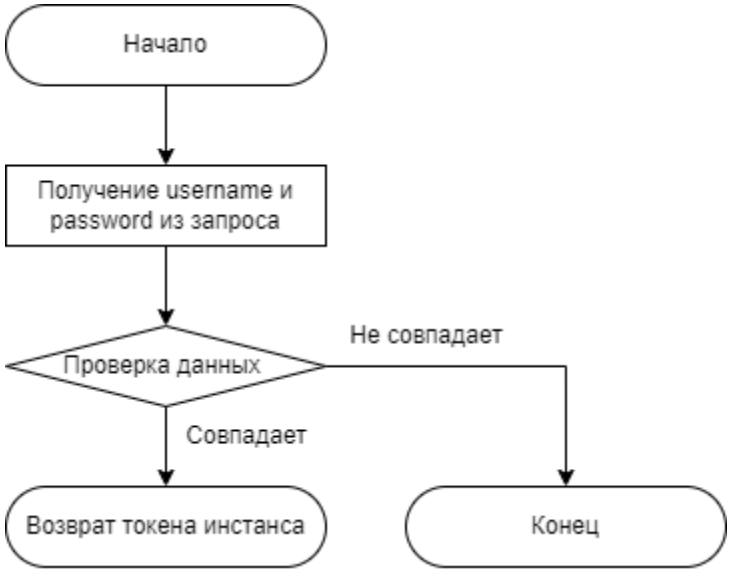

Рис. 3.4.1 – Алгоритм аутентификации

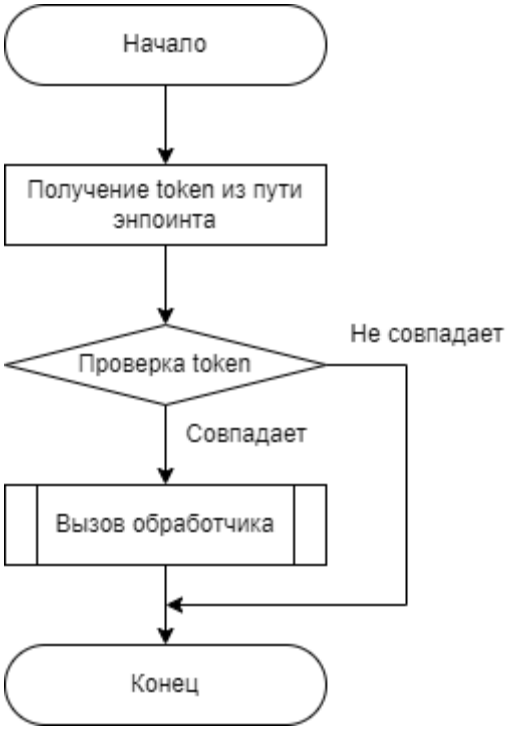

Рис. 3.4.1 – Алгоритм обработки запросов

3.5 Роль Storage

3.5.1 Уведомление роутера

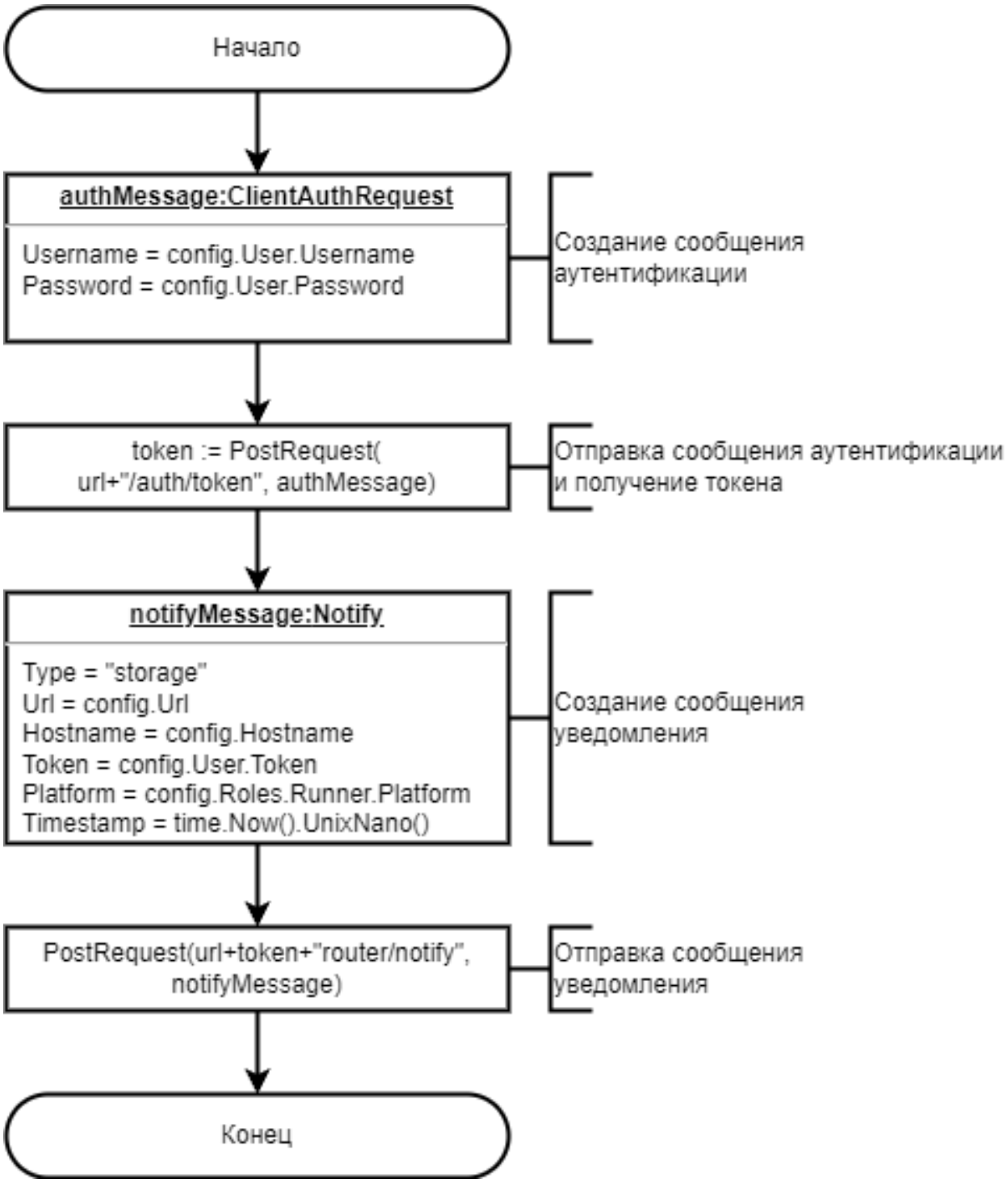

Рис. 3.5.1.1 – Алгоритм уведомления роутера

3.5.2 Сбор данных о файловой системе

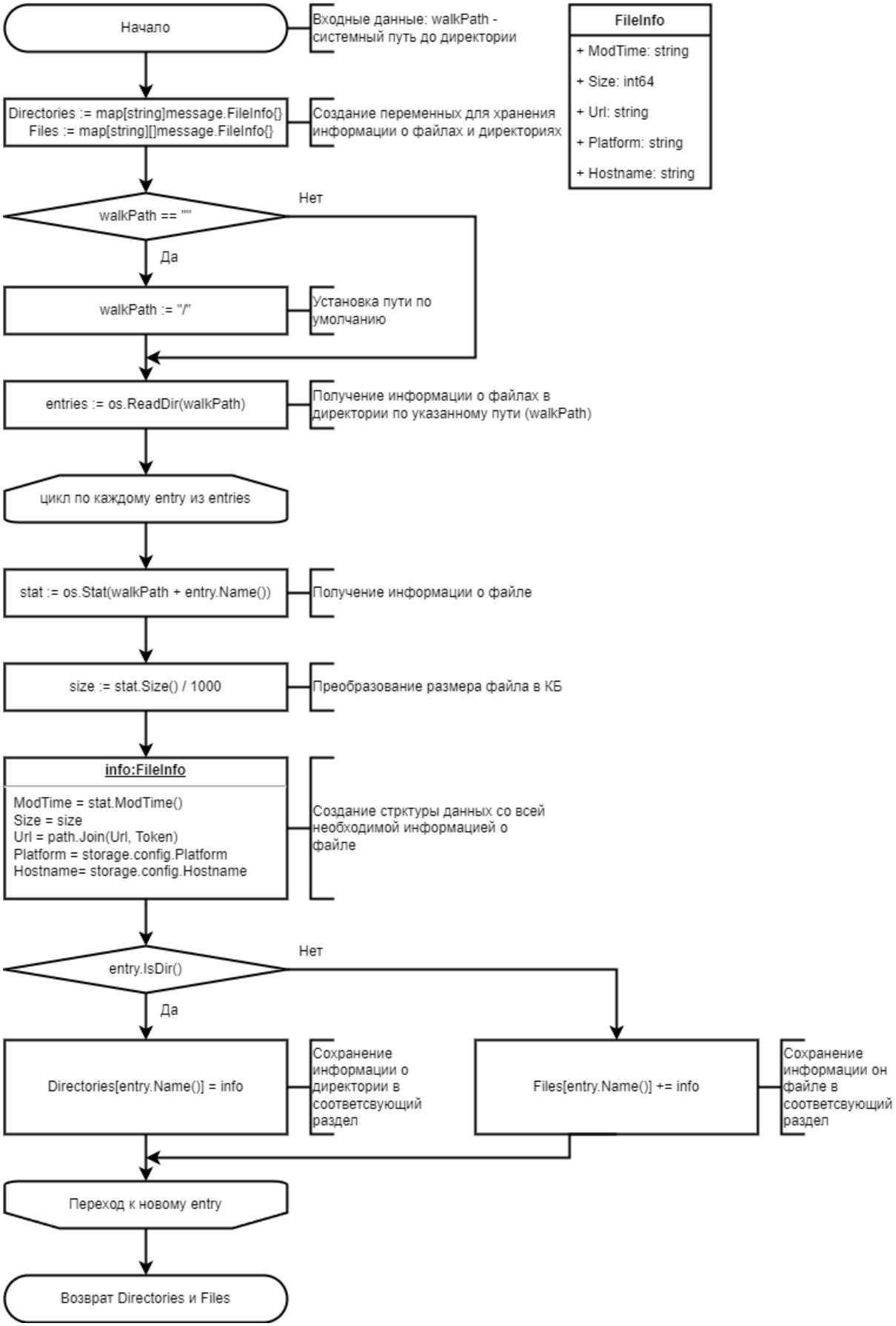

Рис. 3.5.2.1 – Алгоритм сбора данных о файловой системе

3.5.3 Обработчик добавления файла

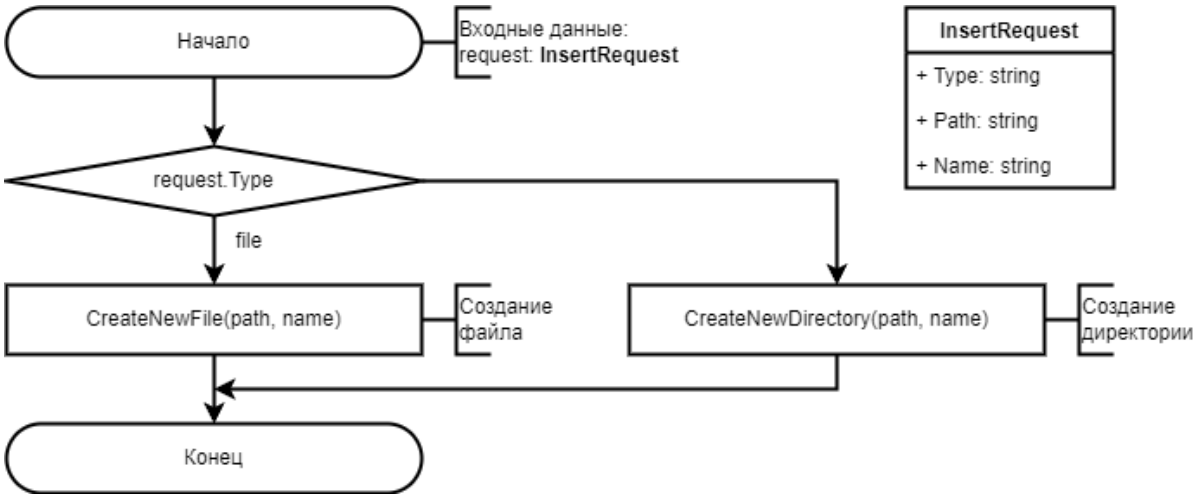

Рис. 3.5.3.1 – Алгоритм добавления файла

3.5.4 Обработчик удаления файла

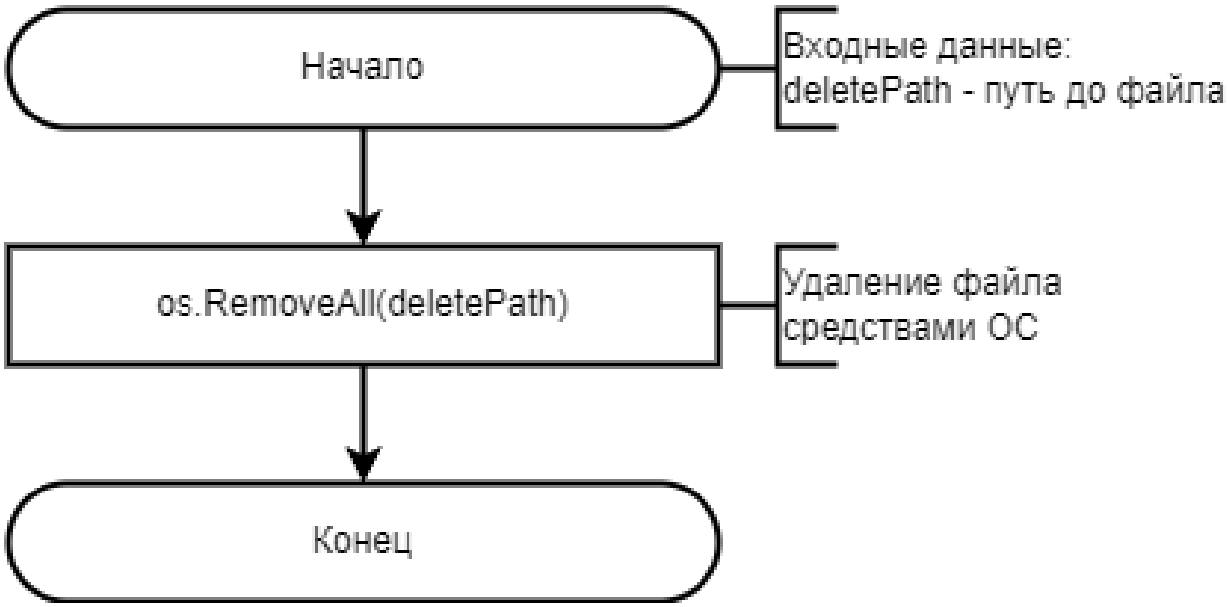

Рис. 3.5.4.1 – Алгоритм удаления файла

3.5.5 Обработчик получения файла

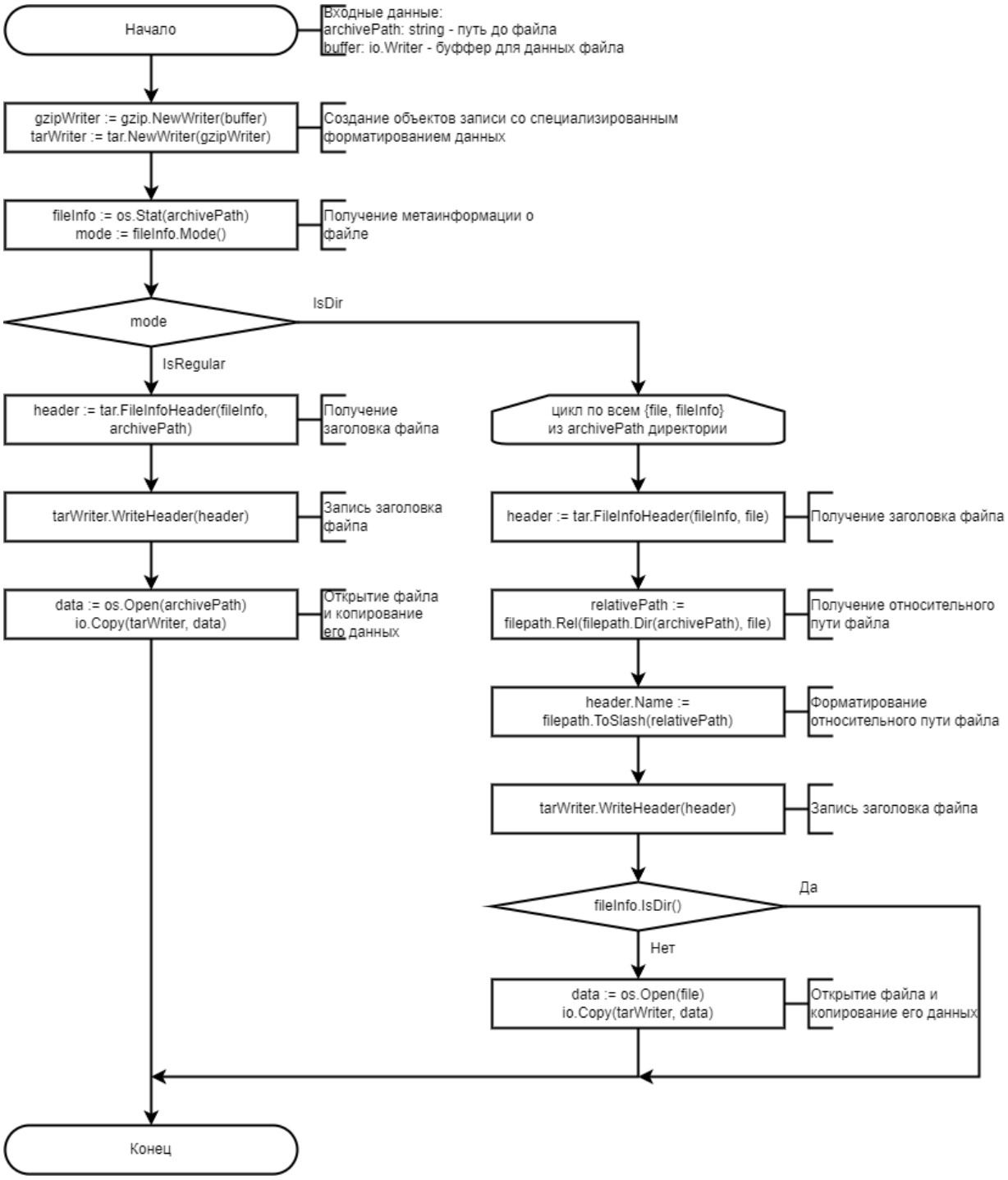

Рис. 3.5.5.1 – Алгоритм получения файла

3.5.6 Обработчик копирования файла

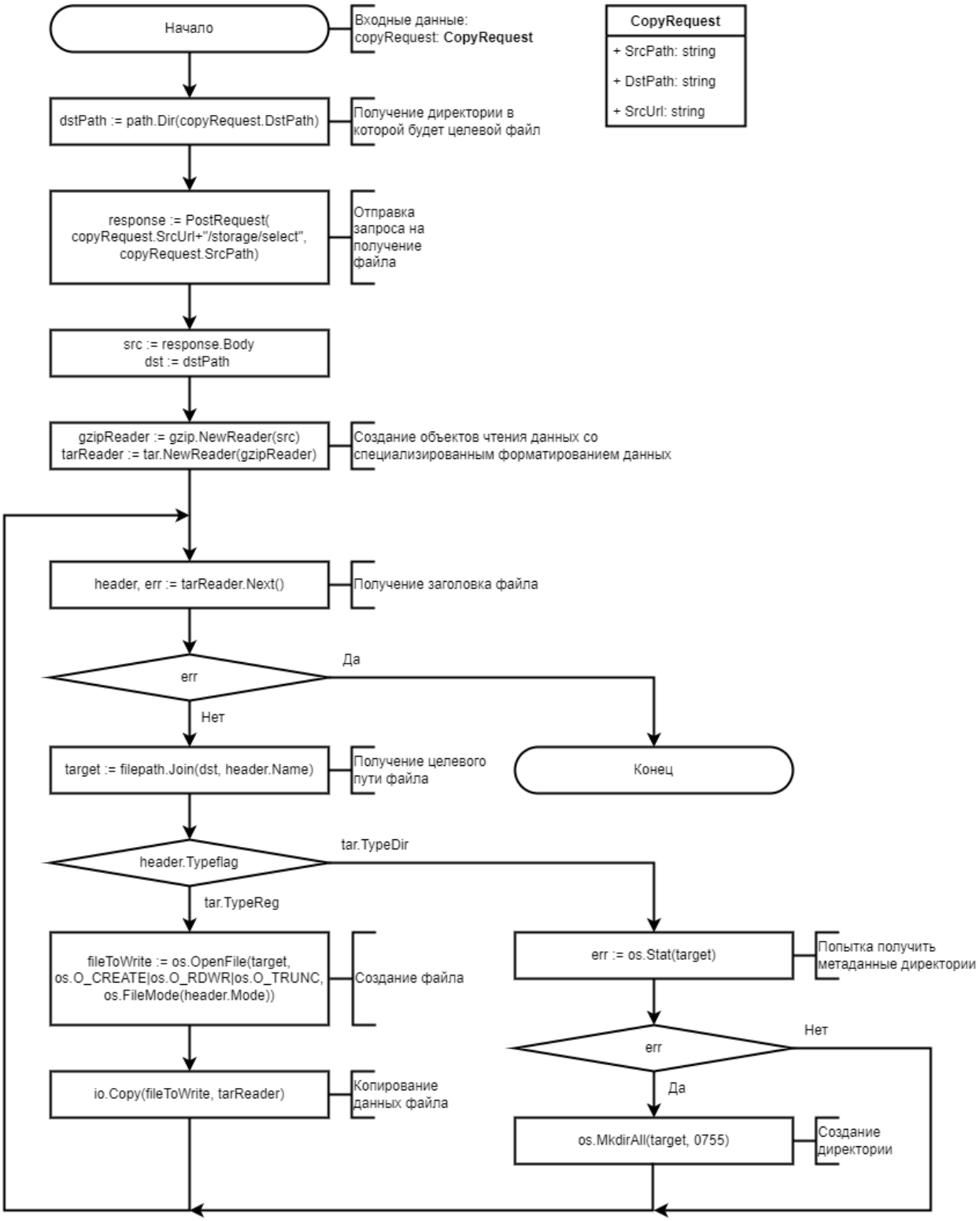

Рис. 3.5.6.1 – Алгоритм копирования файла

3.5.7 Обработчик перемещения файла

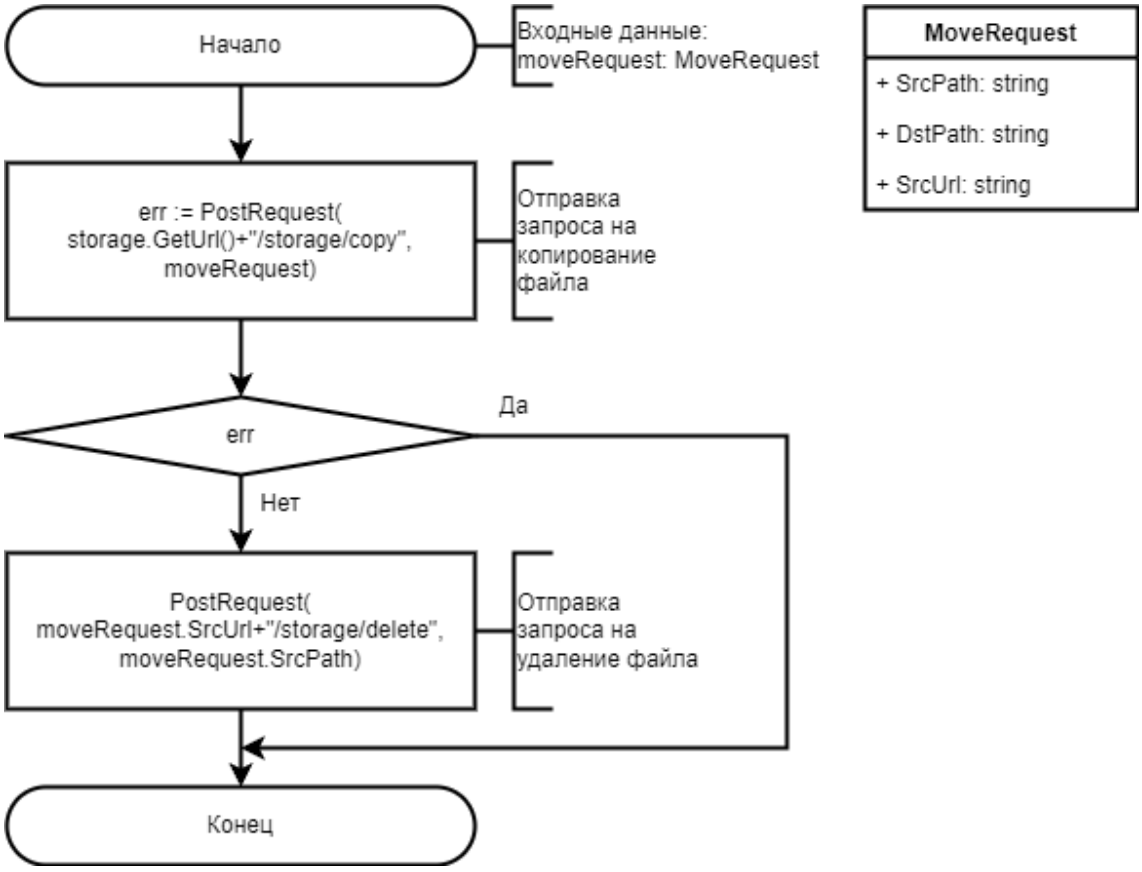

Рис. 3.5.7.1 – Алгоритм перемещения файла

3.5.8 Обработчик переименования файла

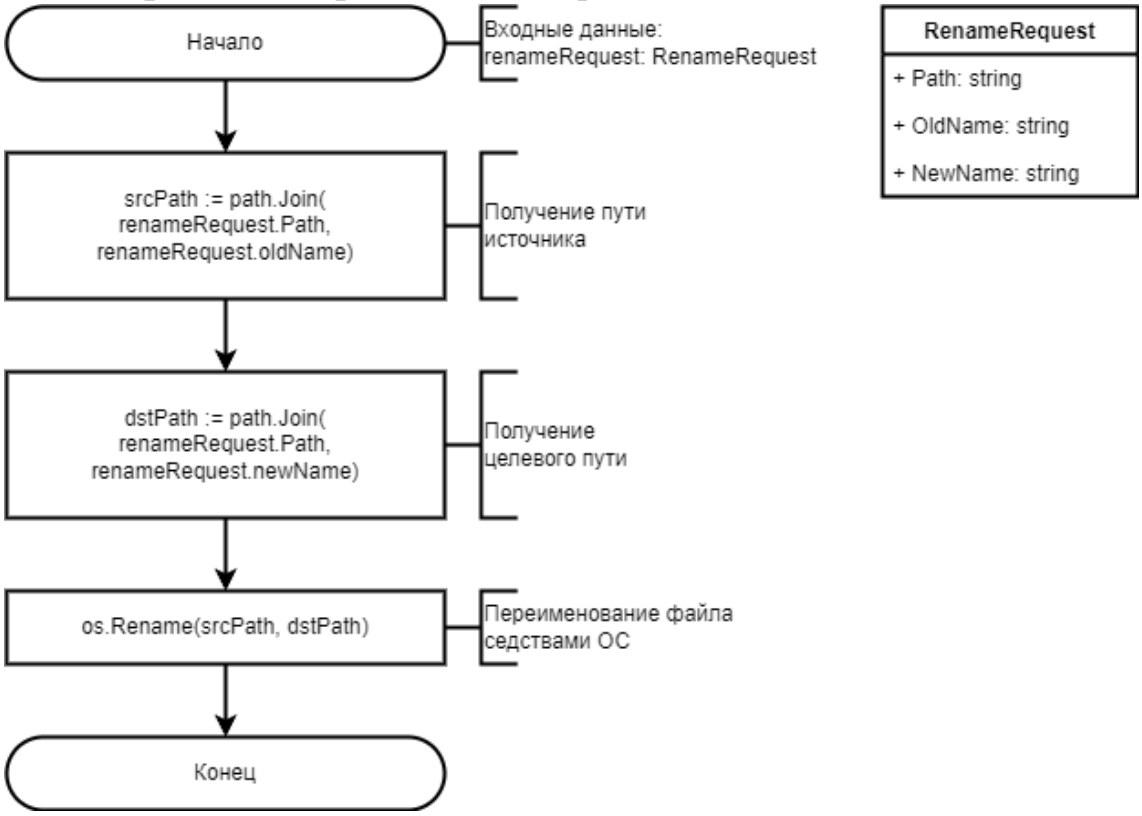

Рис. 3.5.8.1 – Алгоритм переименования файла

3.6 Роль Runner

3.6.1 Уведомление роутера

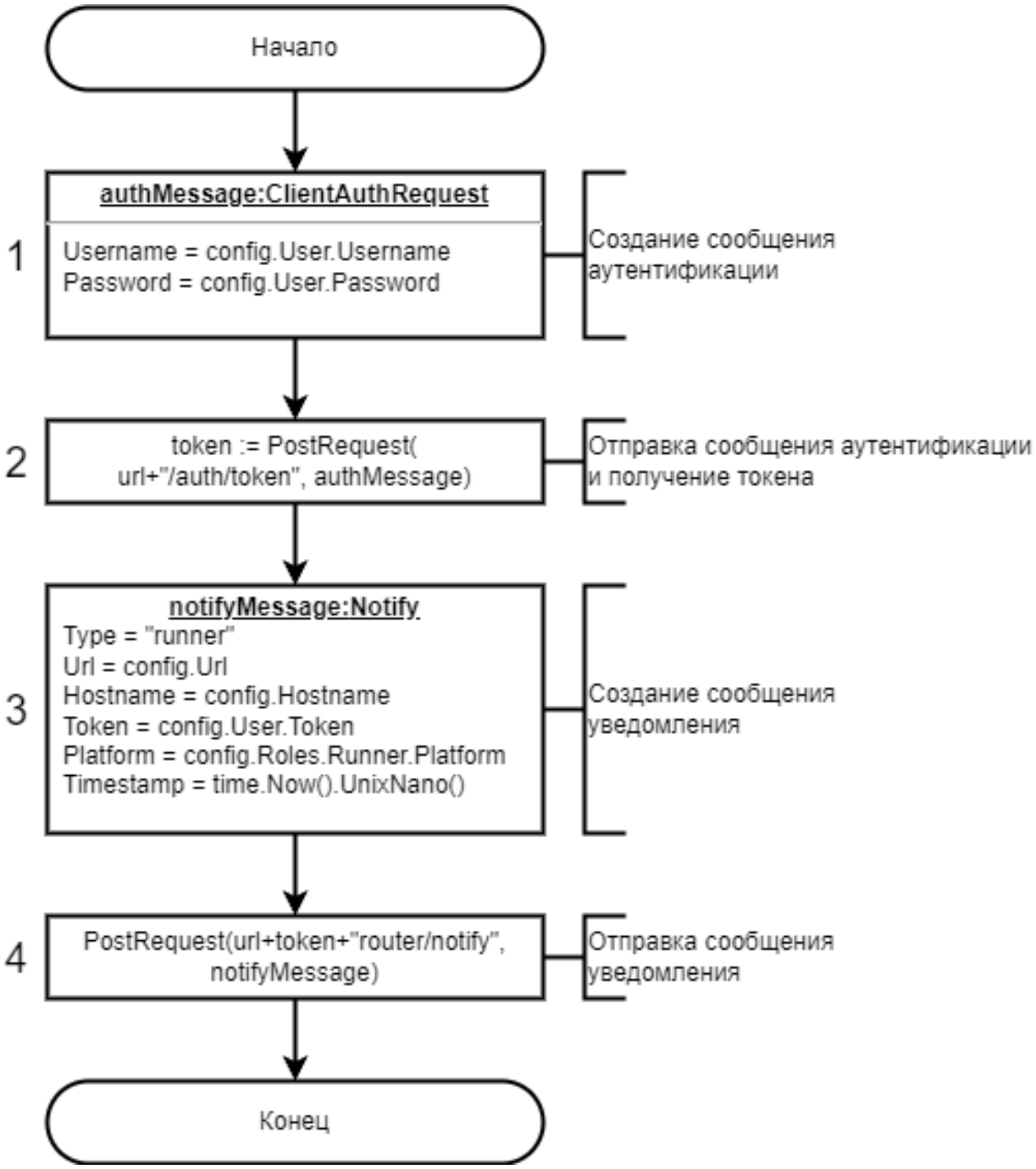

Рис. 3.6.1.1 – Алгоритм уведомления роутера

3.6.2 Обработчик открытия/запуска файлов

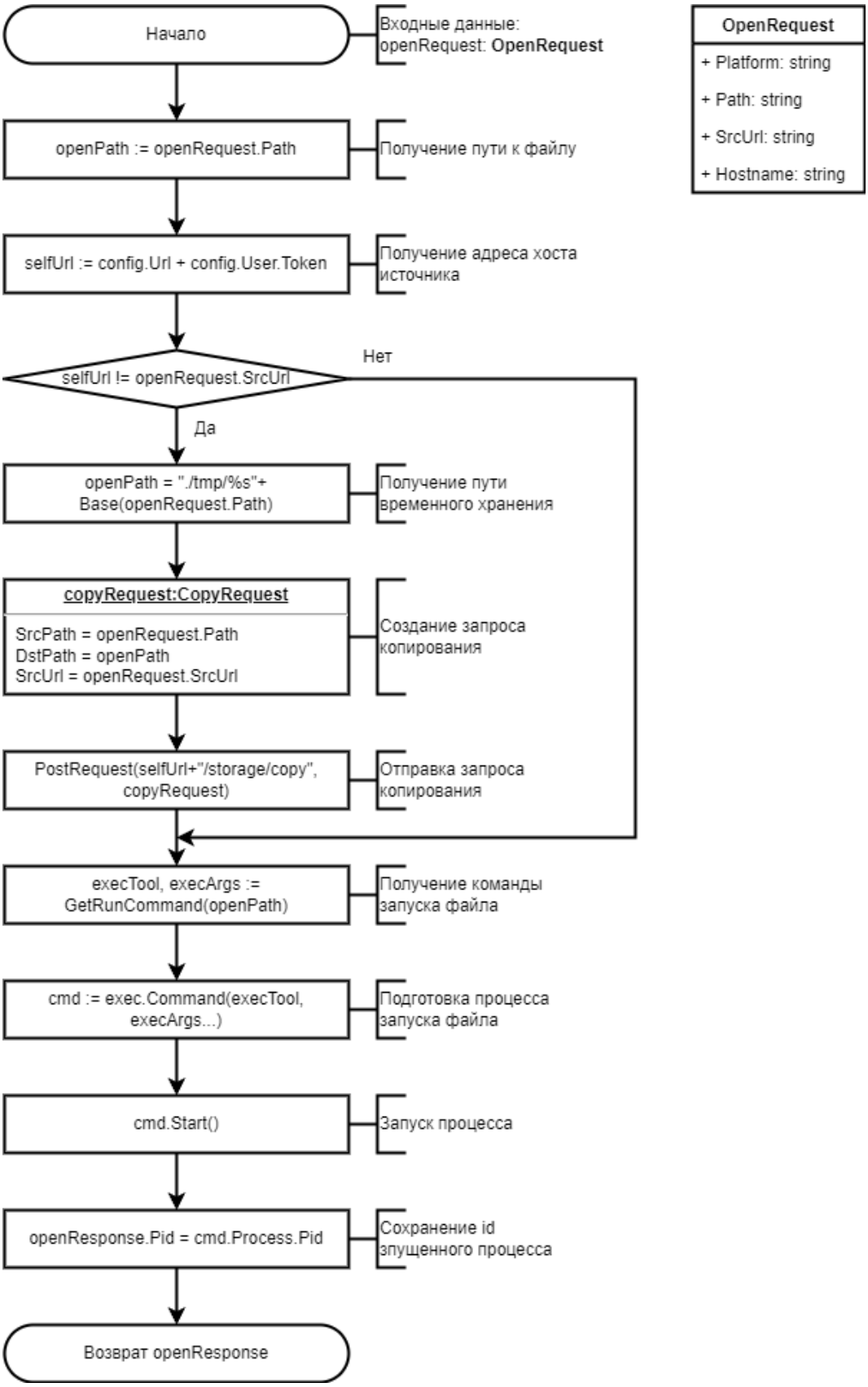

Рис. 3.6.2.1 – Алгоритм открытия и запуска файлов

3.6.3 Обработчик получения изображения приложения

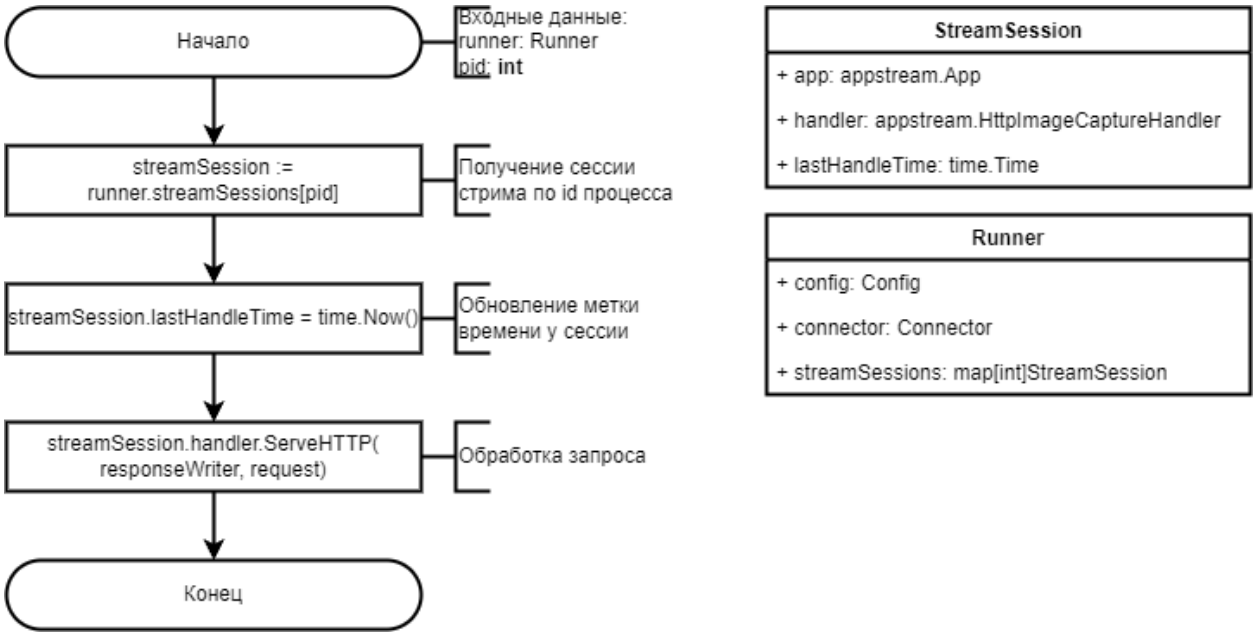

Рис. 3.6.3.1 – Алгоритм получения изображения приложения

3.6.4 Обработчик события клавиатуры

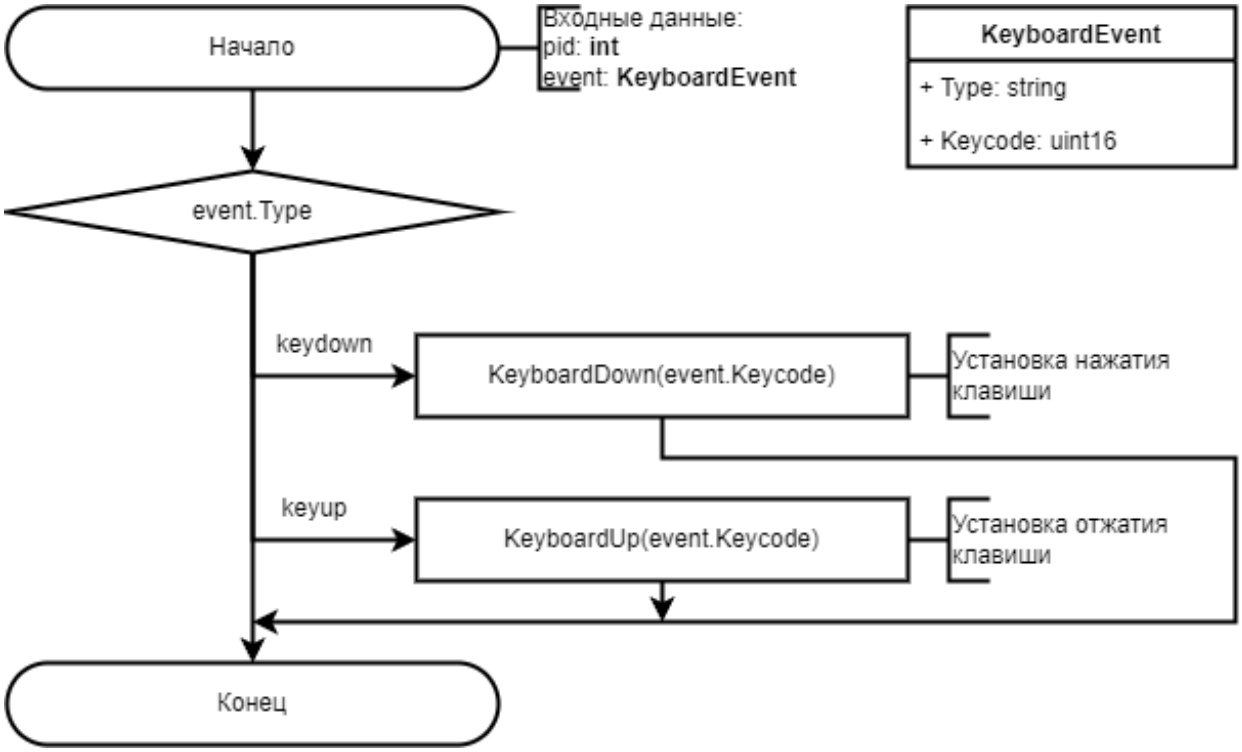

Рис. 3.6.4.1 – Алгоритм обработки события клавиатуры

3.6.5 Обработчик события мыши

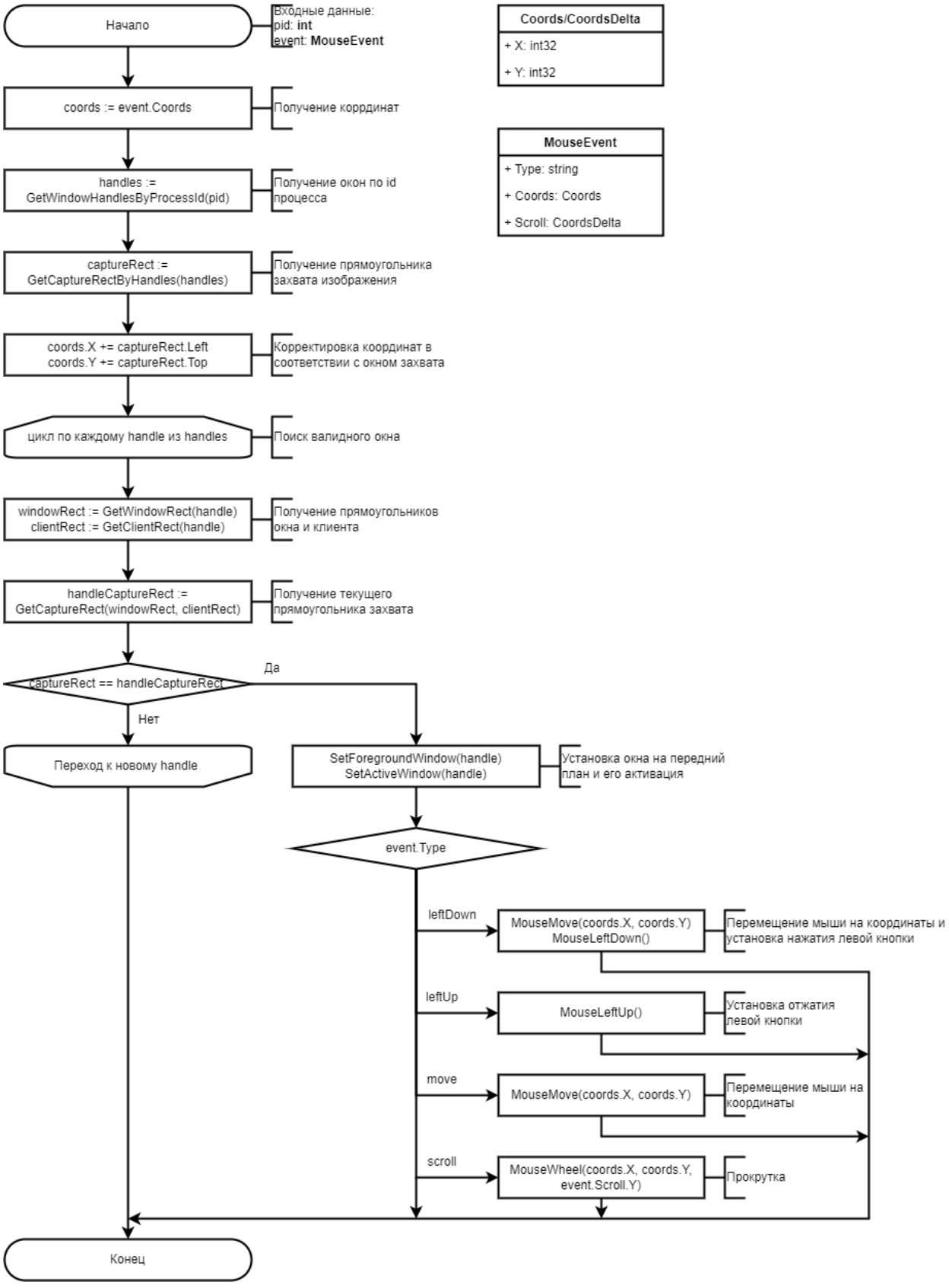

Рис. 3.6.5.1 – Алгоритм обработки события мыши

3.6.6 Обработчик создания прямого стрима приложения

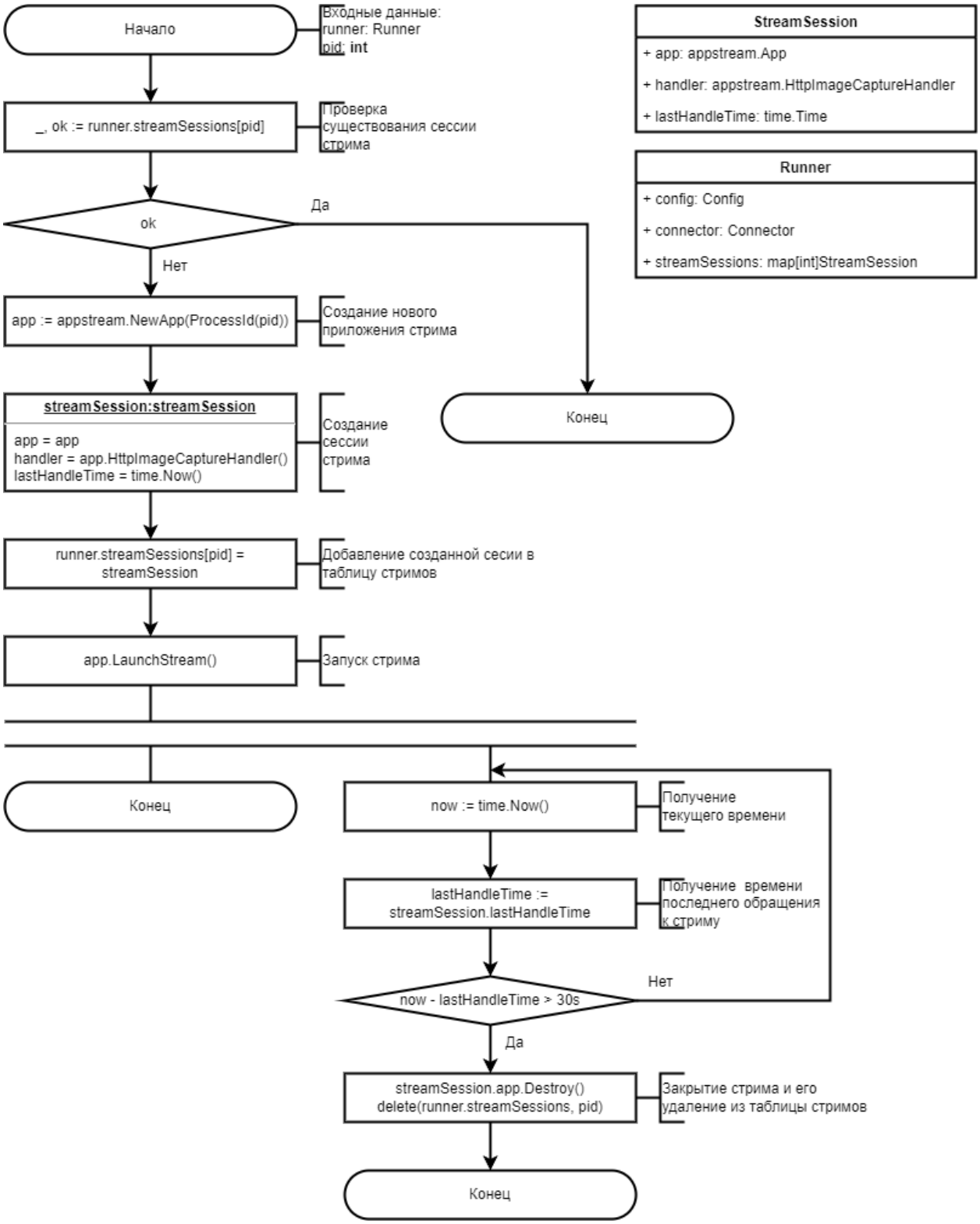

Рис. 3.6.6.1 – Алгоритм создания прямого стрима приложения

3.6.7 Обработчик создания стрима приложения

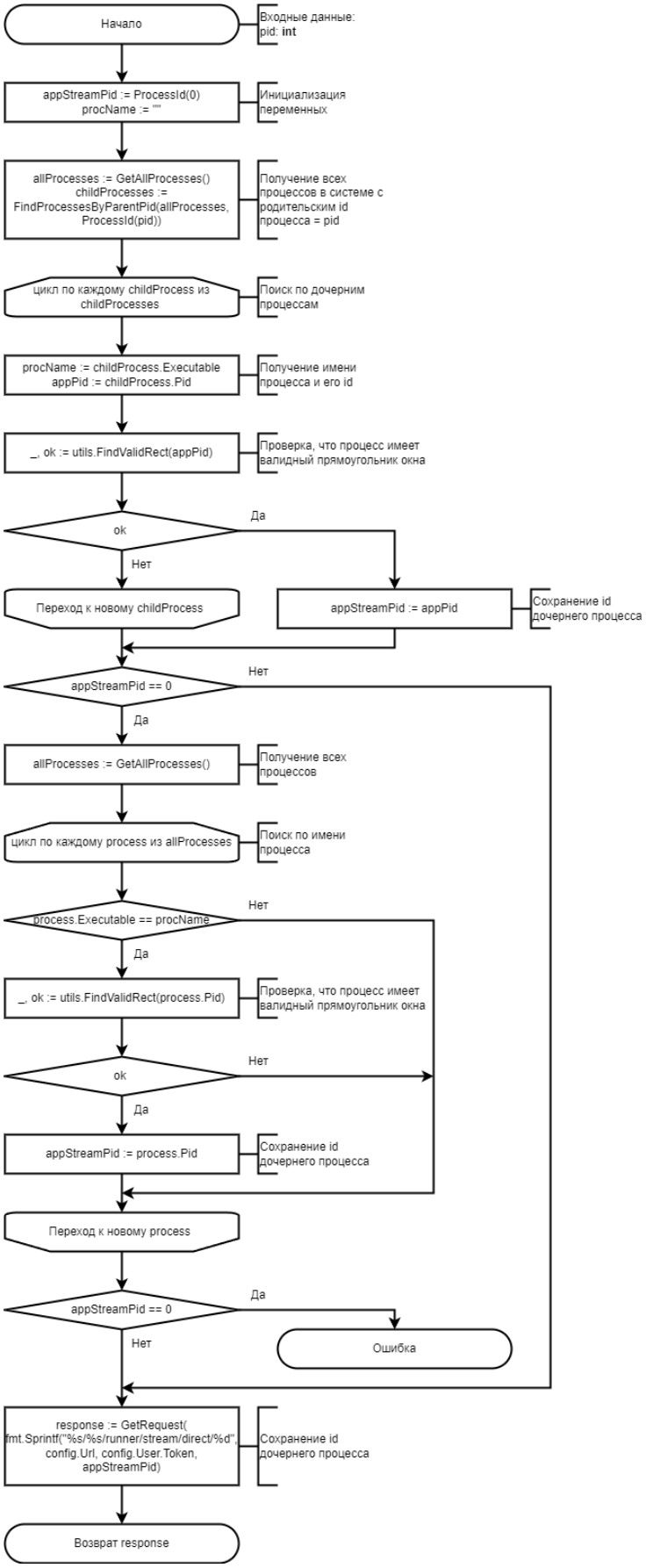

Рис. 3.6.7.1 – Алгоритм создания стрима приложения

3.6.8 Стрим приложения

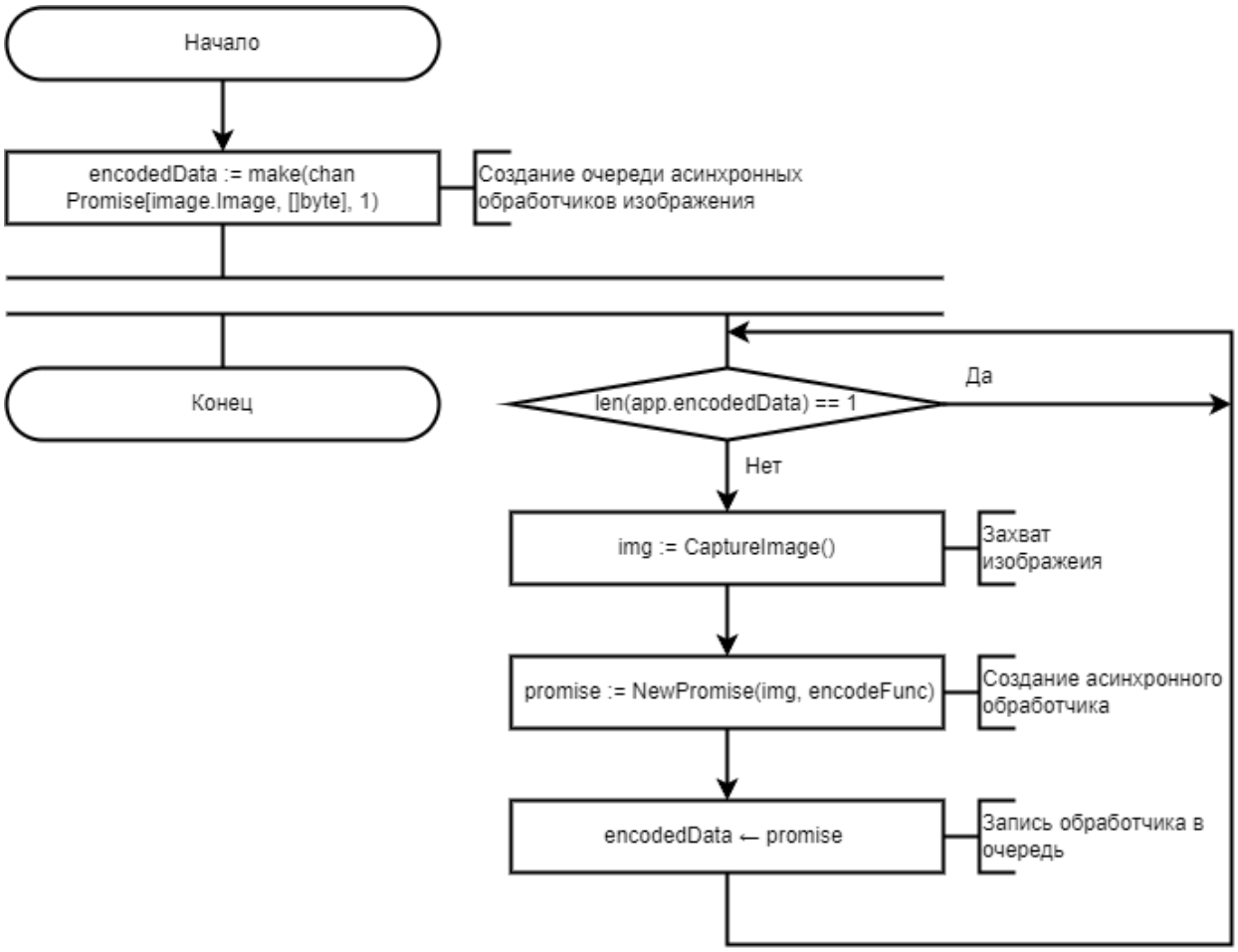

Рис. 3.6.8.1 – Алгоритм внутренней работы стрима приложения

3.6.9 Выбор обработчика изображения

В главе 0

Открытие файлов и работа с ними было выбрано 3 формата обработки изображений. Необходимо найти наиболее оптимальный из выбранных форматов.

Критерием отбора будет суммарное время обработки и передачи изображения.

Формат JPEG используется со стандартным сохранением качества в 75%.

Формат изображения	Время обработки, мс	Время загрузки, мс	Суммарное время, мс
PNG	10 – 20	60 – 70	70 – 90
JPEG	40 – 45	2 – 5	42 – 50
WebP	200 – 220	1 – 2	201 – 222

Таким образом можно сделать следующие выводы:

- Формат WebP хорошо подходит для уже преобразованных изображений, так как имеет очень малый размер и быстро декодируется браузером.
- PNG имеет малое время обработки, но из-за получившегося размера, время загрузки оказывается больше чем время подготовки JPEG формата. Удобен при использовании кэширования изображений.
- JPEG единственный формат с потерей качества изображения, но выигрывает по суммарному времени предоставляя около 20-23 кадров в секунду. Наиболее подходящий вариант для постоянно изменяющегося потока изображений.

3.6.10 Захват изображения на ОС Windows

Захват изображения на ОС Windows программным способом с использованием WinAPI включает в себя несколько подходов, среди которых наиболее распространенными являются использование функции BitBlt для прямого блочного копирования содержимого памяти и использование функции GetDIBits для получения изображения в формате DIB (Device Independent Bitmap).

BitBlt позволяет копировать содержимое одной области памяти в другую. Для захвата экрана, нужно получить дескриптор устройства (HDC) текущего окна или всего экрана, а затем использовать BitBlt для копирования содержимого этого HDC в буфер обмена или непосредственно в файл изображения.

GetDIBits используется для получения изображения в формате DIB, что позволяет сохранить изображение в файле с сохранением качества цвета и разрешения. Этот метод требует предварительного создания DIBSection, который будет использоваться как буфер для хранения изображения.

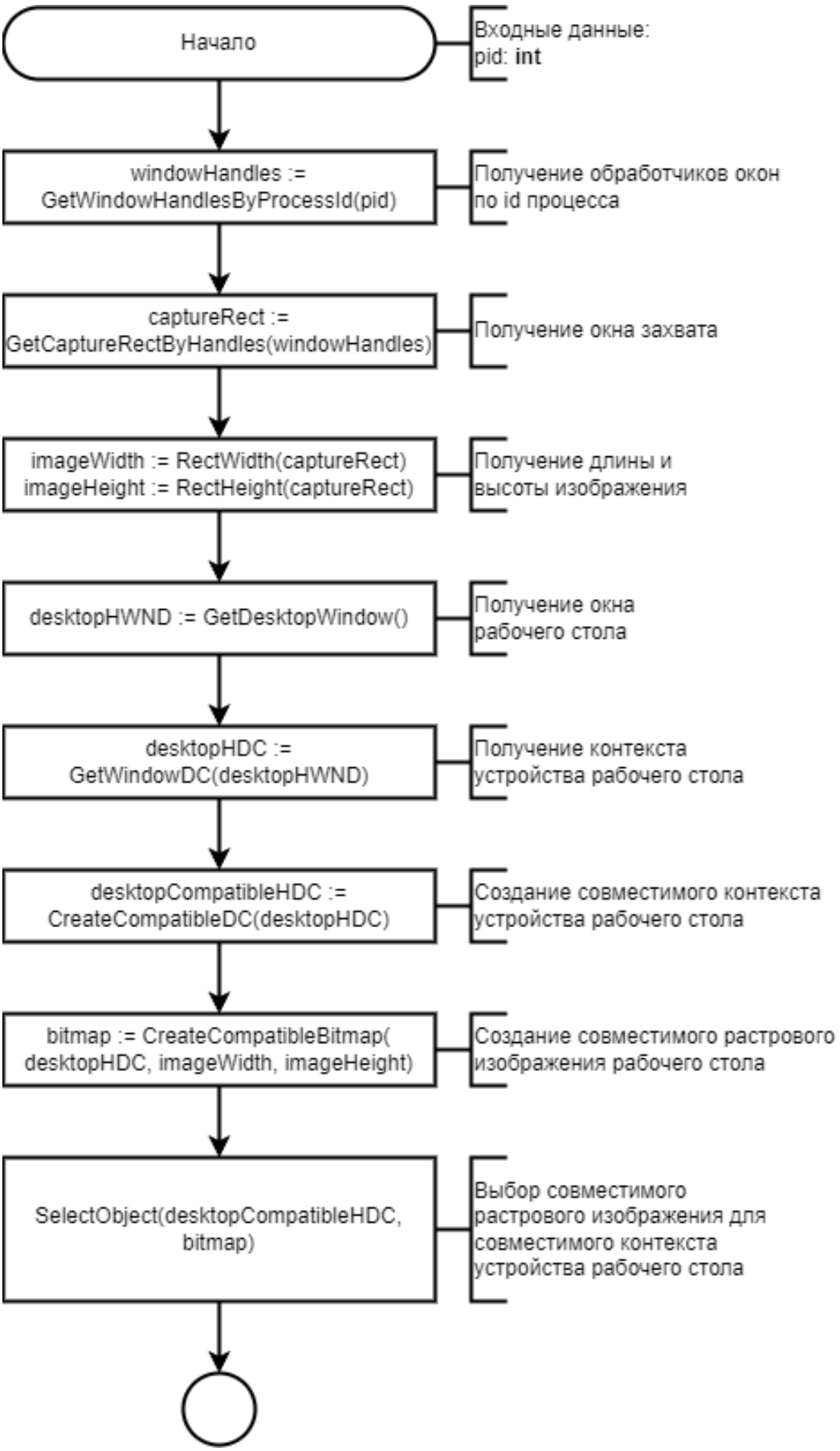

Рис. 3.6.10.1 – Алгоритм внутренней работы стрима приложения (ч.1)

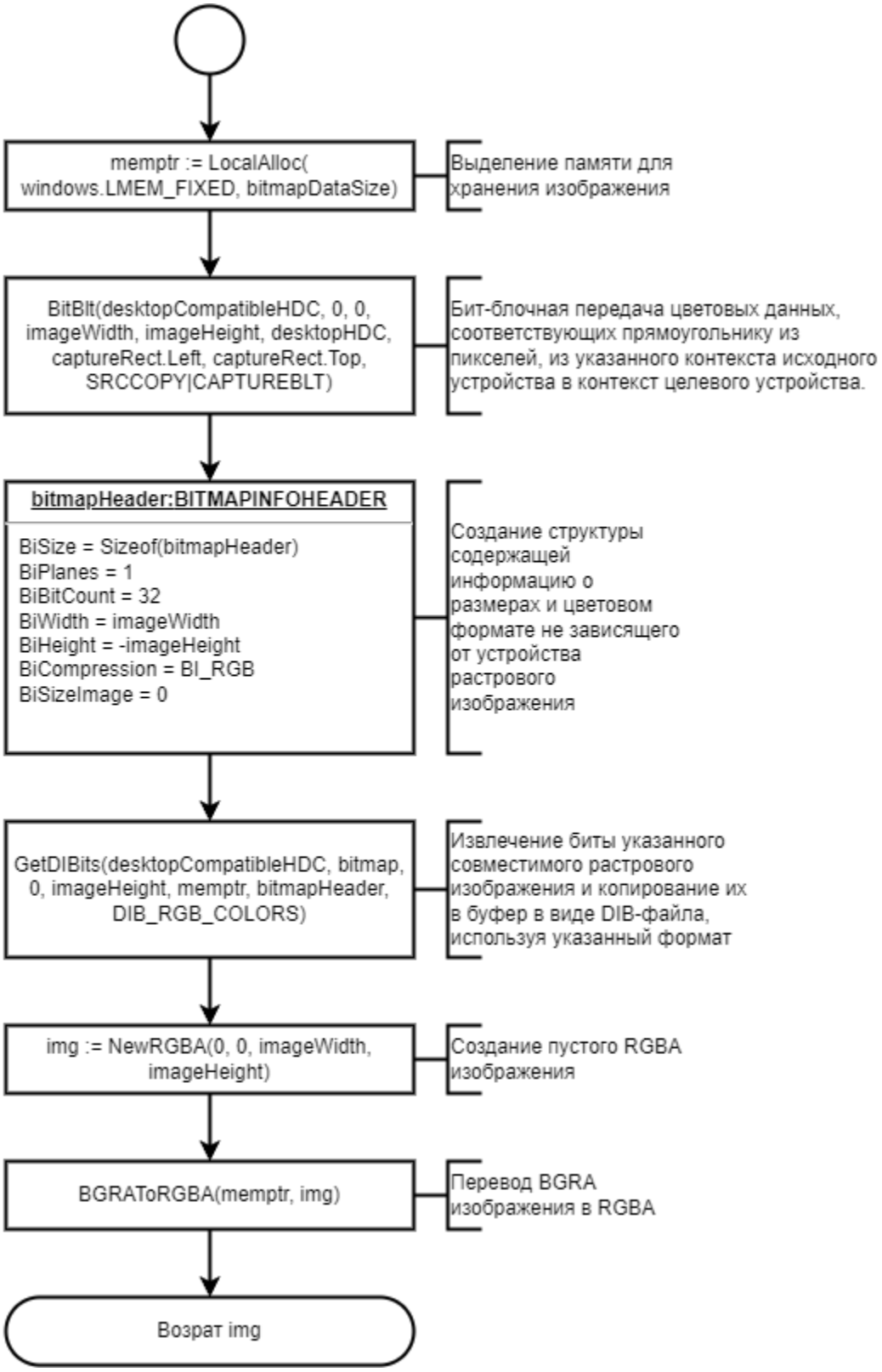

Рис. 3.6.10.2 – Алгоритм внутренней работы стрима приложения (ч.2)

3.7 Роль Router

3.7.1 Сбор данных о файловой системе

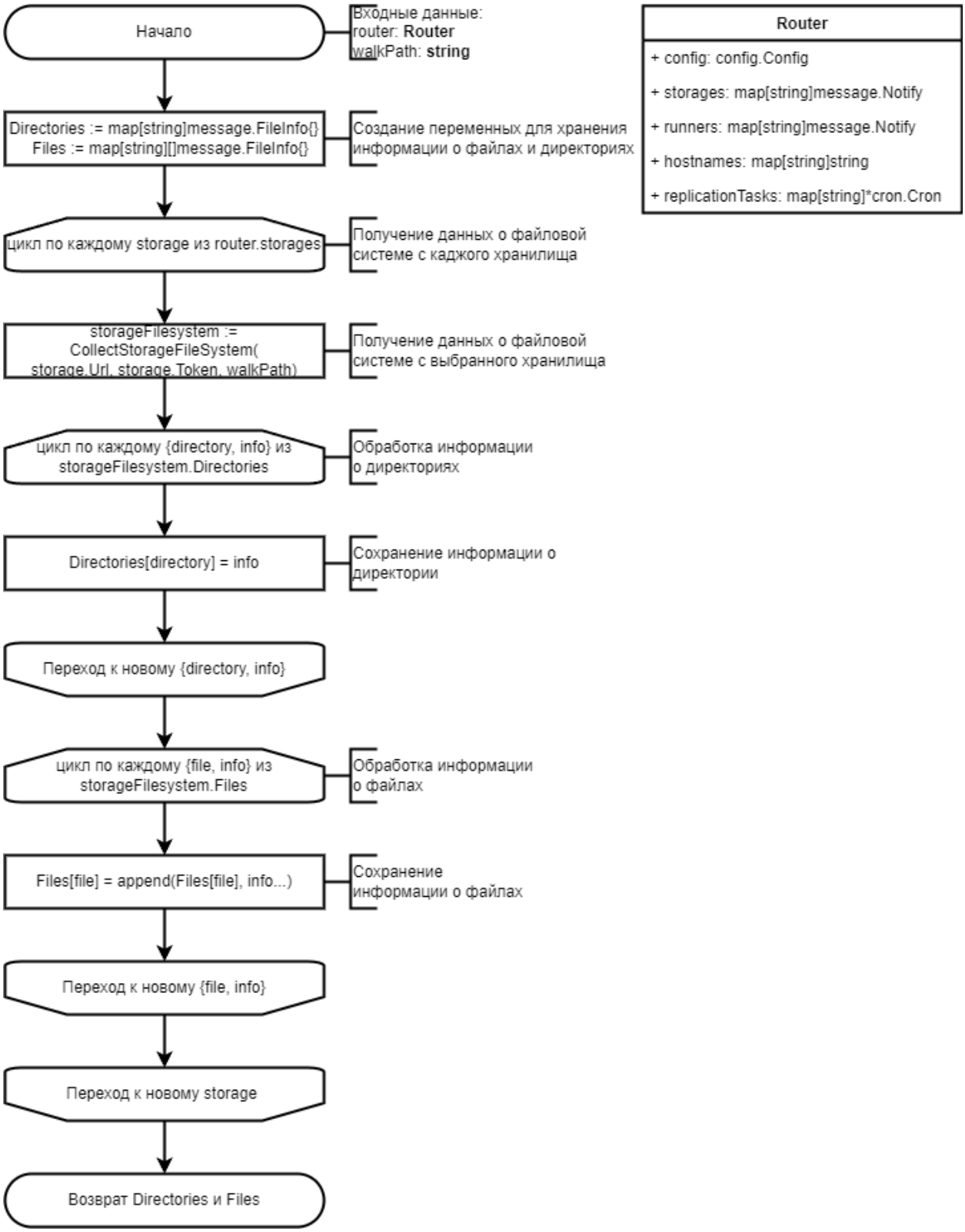

Рис. 3.7.1.1 – Алгоритм сбора данных о файловой системе

3.7.2 Отслеживание устаревших инстансов

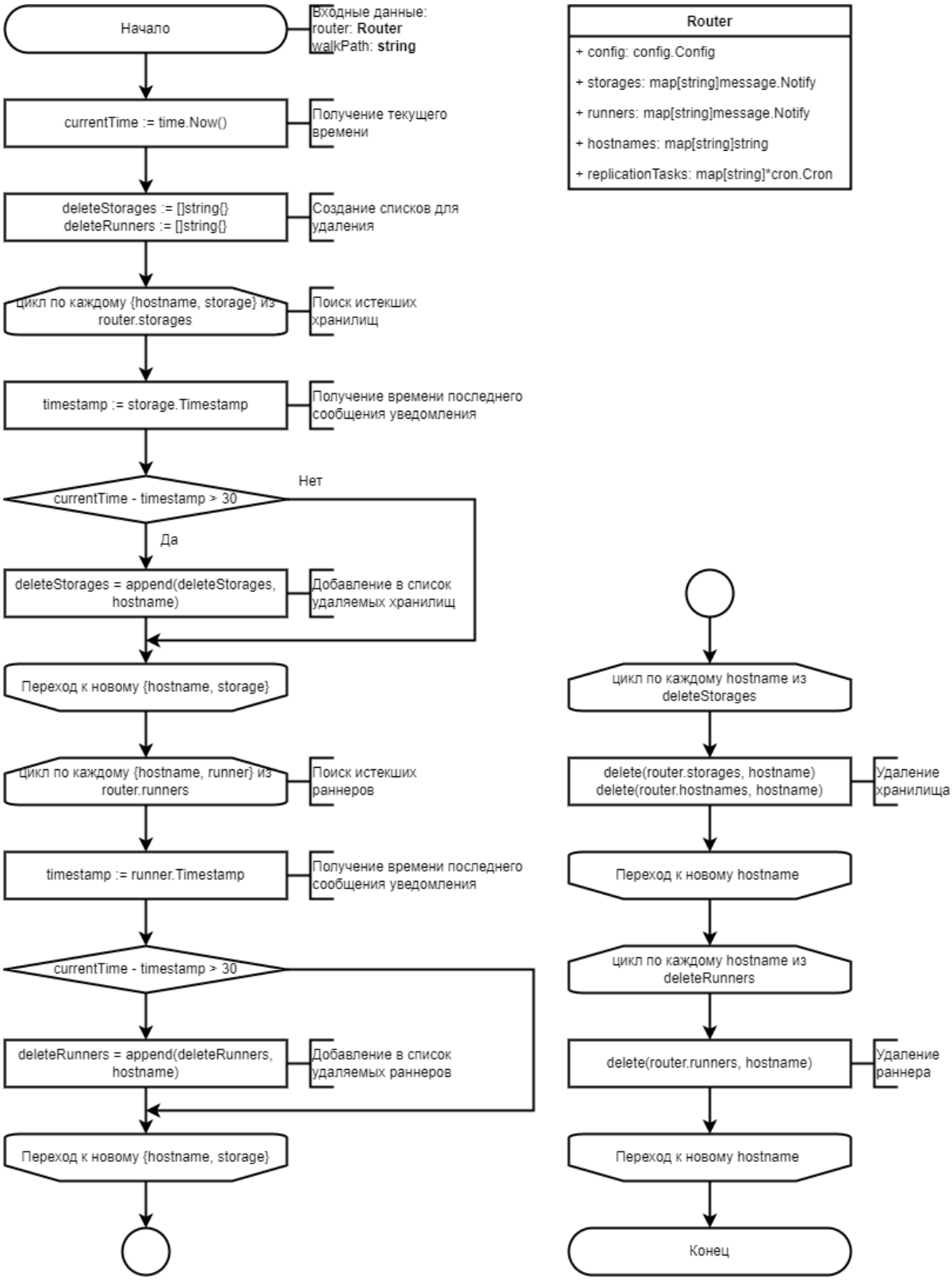

Рис. 3.7.2.1 – Алгоритм отслеживания устаревших инстансов

3.7.3 Репликация

3.7.3.1 Добавление задач репликации

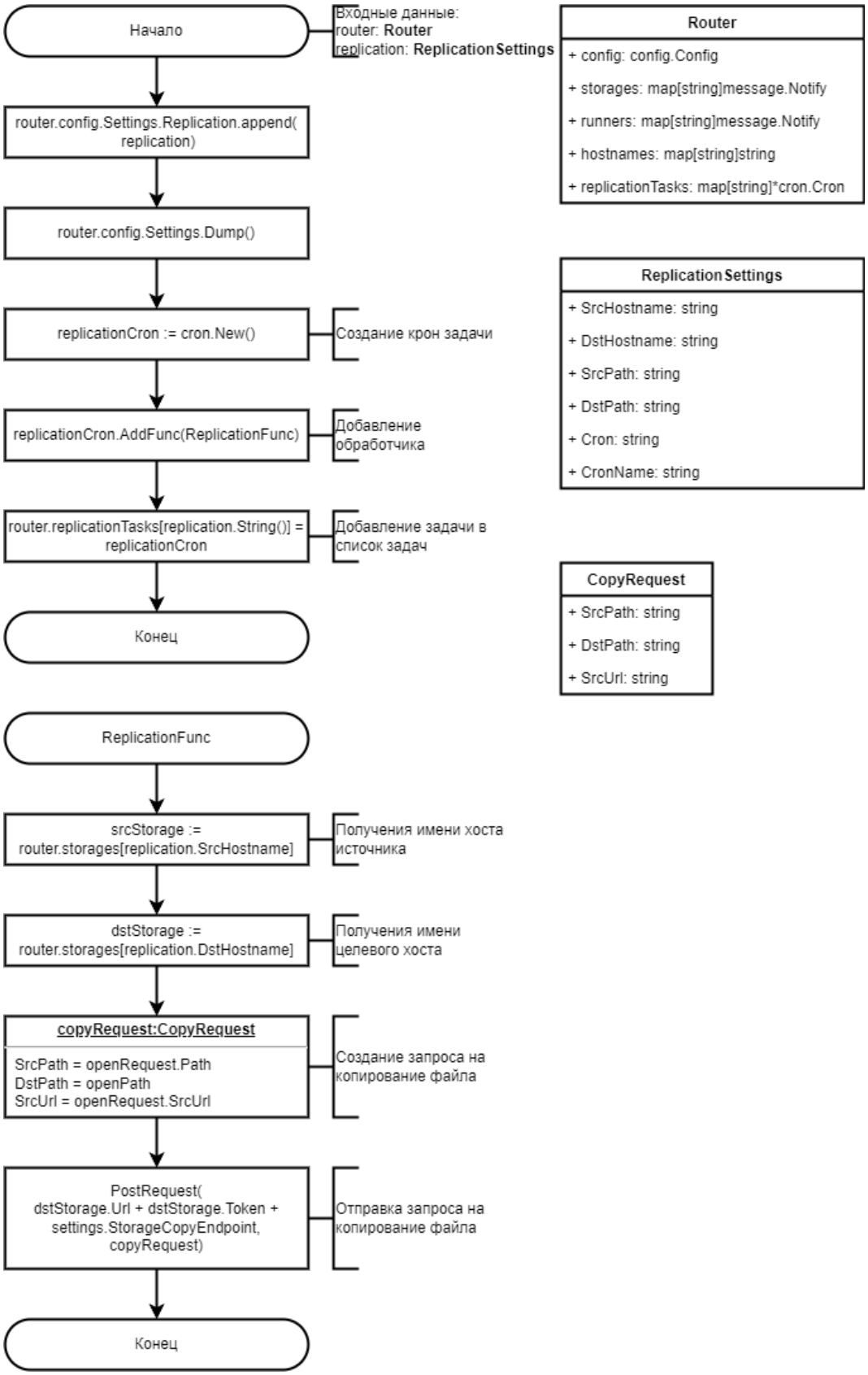

Рис. 3.7.3.1 – Алгоритм добавления задач репликации

3.7.3.2 Удаление задач репликации

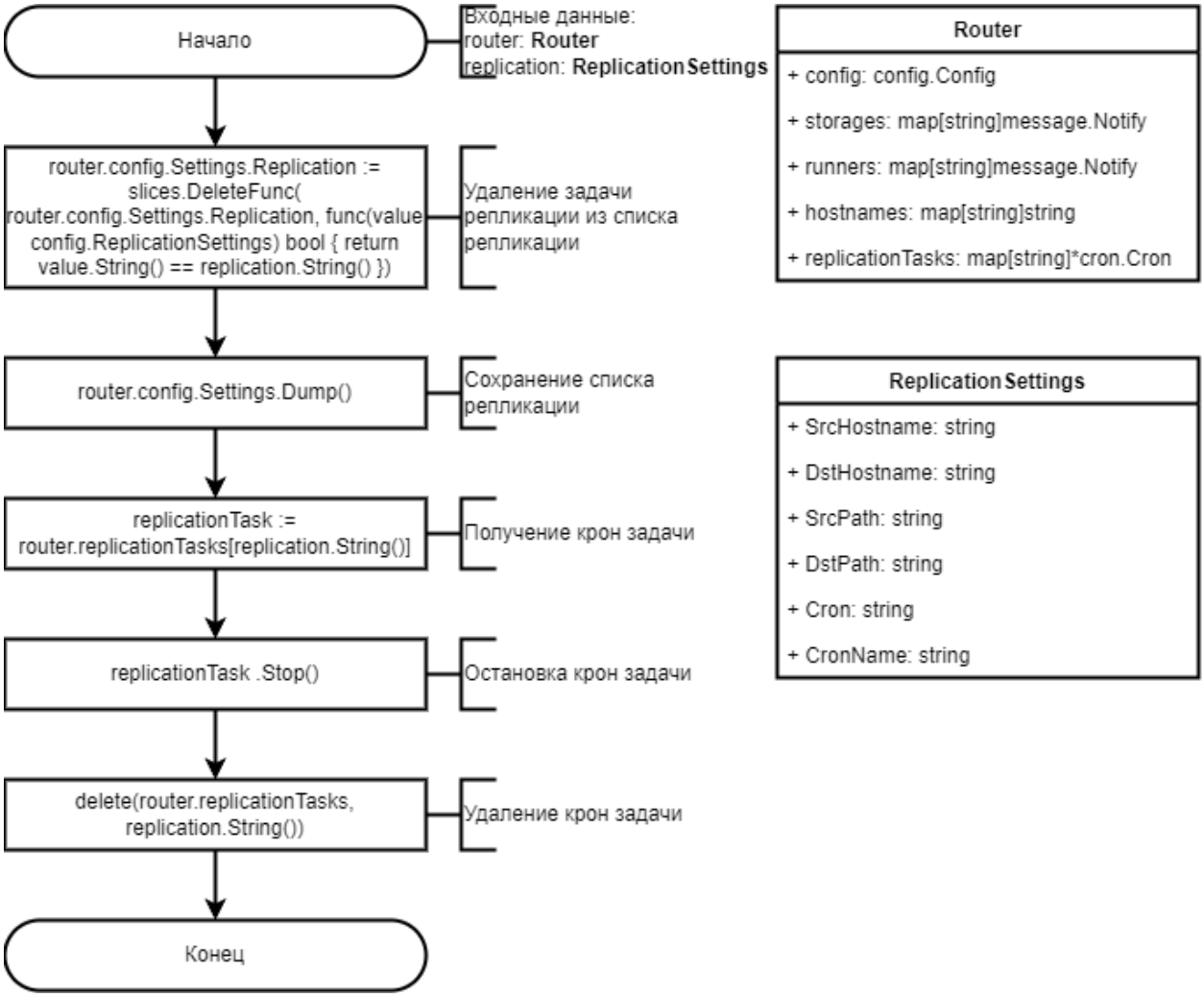

Рис. 3.7.3.2 – Алгоритм удаления задач репликации

3.7.4 Фильтрация файлов

3.7.4.1 Добавление фильтра

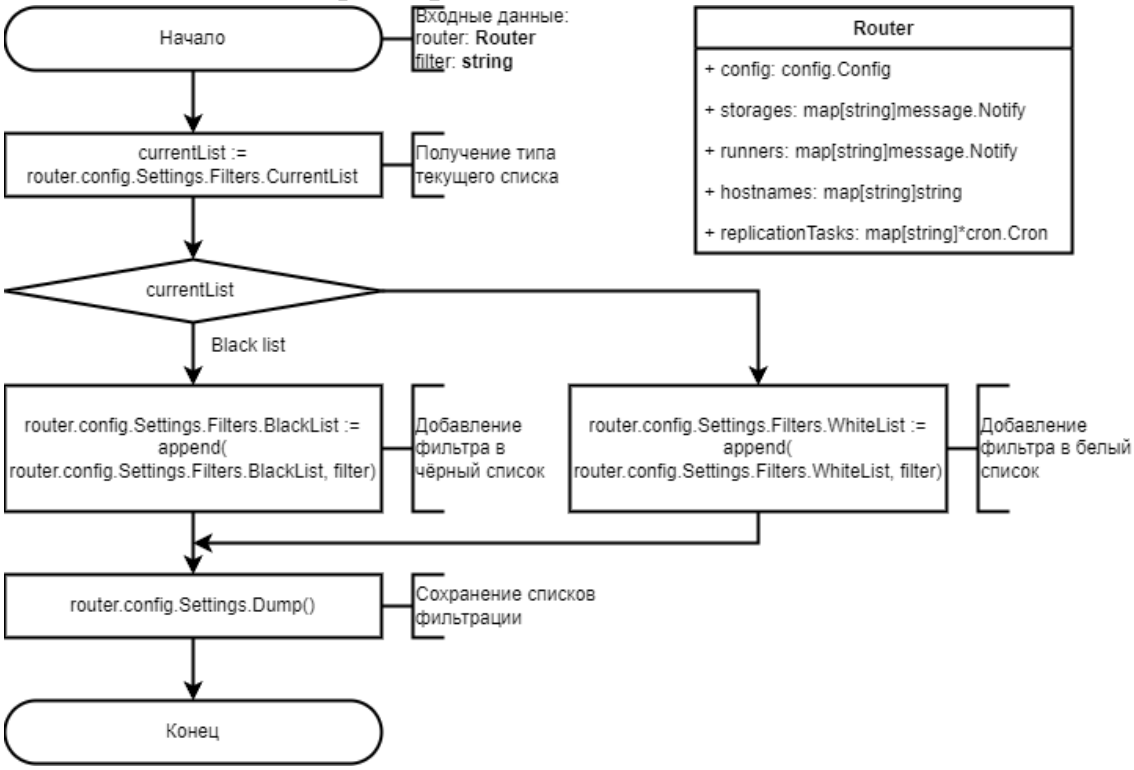

Рис. 3.7.4.1 – Алгоритм добавления фильтра

3.7.4.2 Удаление фильтра

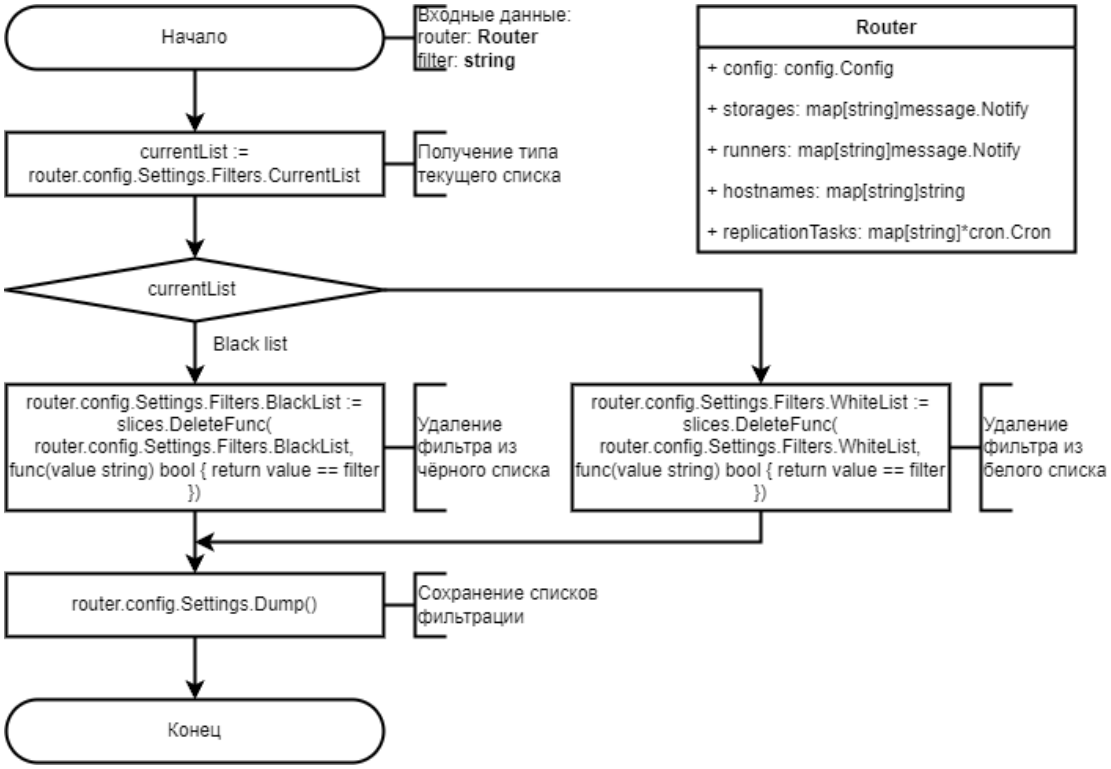

Рис. 3.7.4.2 – Алгоритм удаления фильтра

3.7.4.3 Изменения типа фильтрации

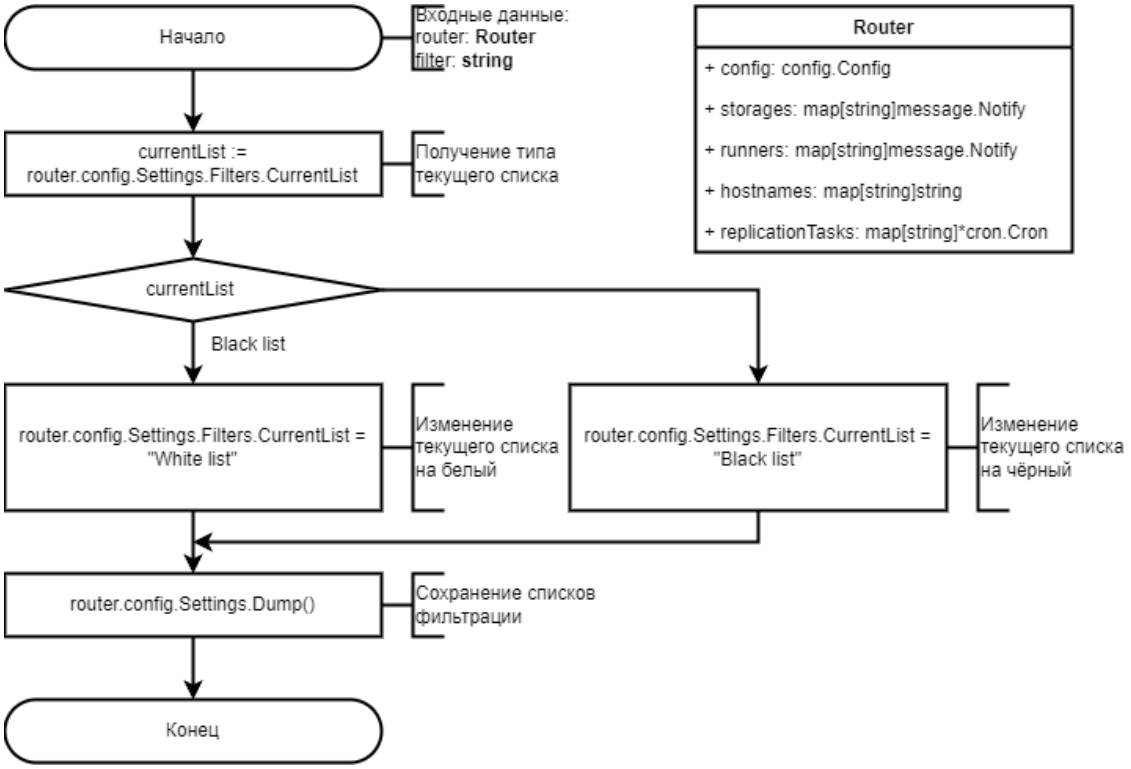

Рис. 3.7.4.3 – Алгоритм изменения типа фильтрации

3.7.5 Обработчик уведомлений

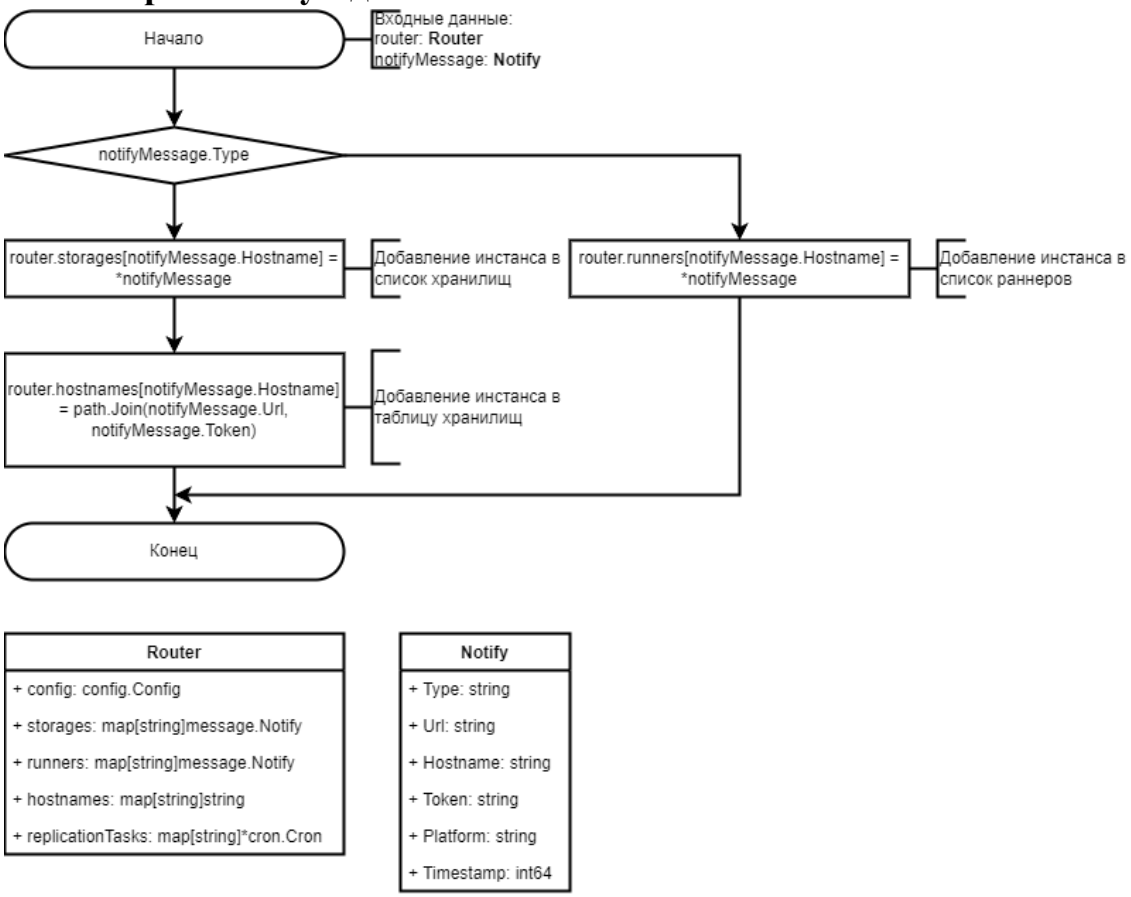

Рис. 3.7.5.1 – Алгоритм обработки уведомлений

3.7.6 Обработчик открытия/запуска файлов

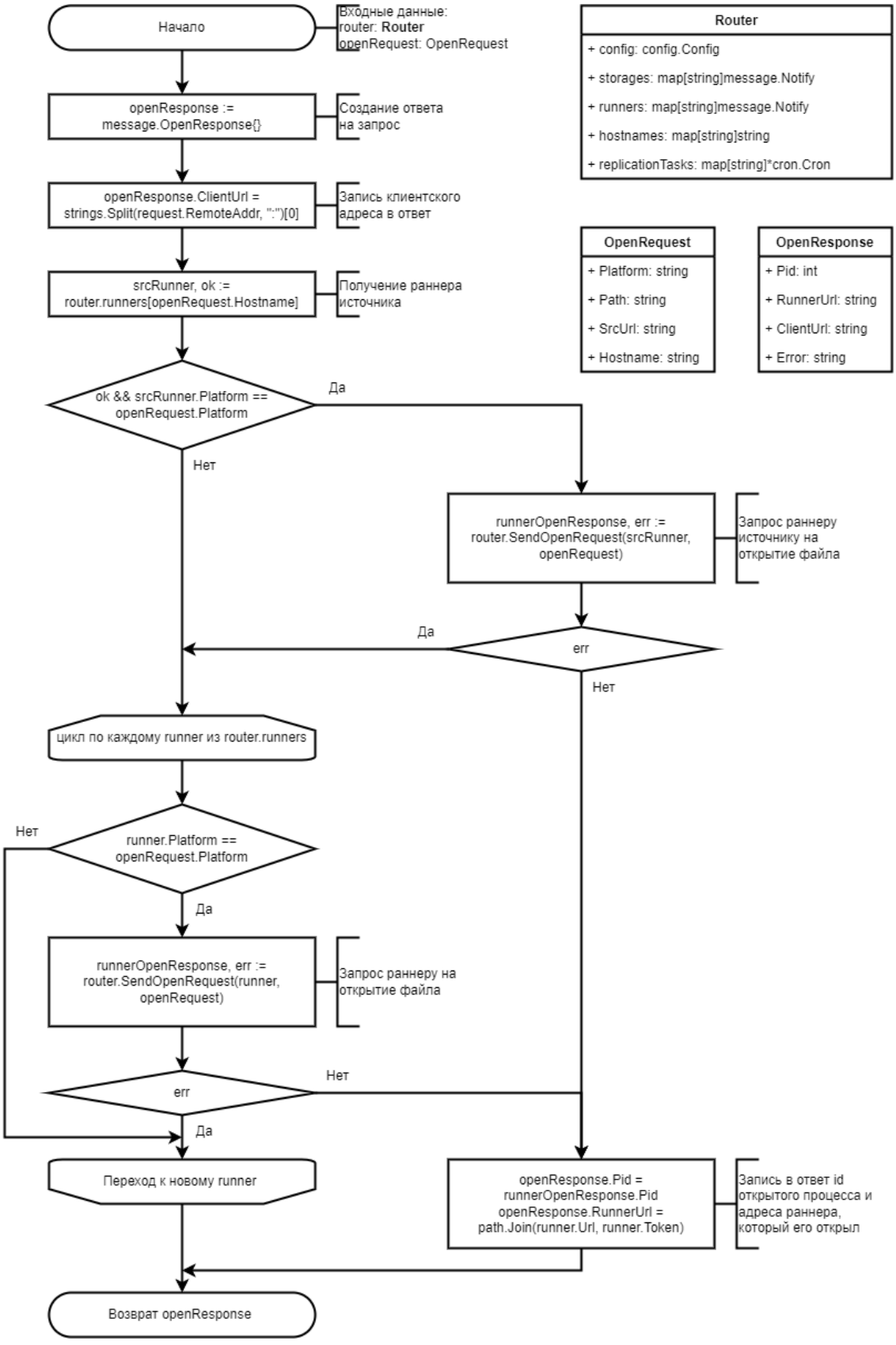

Рис. 3.7.6.1 – Алгоритм открытия файла

4 Методика тестирования

Для проведения тестирования функциональности работы с файловой системой, включая просмотр директорий, переход в новые директории, создание и удаление файлов/директорий, копирование и перемещение файлов/директорий, а также открытие файлов, можно применить следующую методику тестирования:

Общие рекомендации:

- Предварительное состояние: перед проведением тестов убедиться, что система находится в известном состоянии, например, после полной очистки или сброса.
- Повторяемость: повторять тесты для проверки стабильности системы.
- Документирование: документировать результаты тестирования и любые обнаруженные ошибки или проблемы.
- Использование автоматизированных инструментов: при возможности использовать автоматизированные инструменты для тестирования, чтобы увеличить покрытие тестирования и сократить время на тестирование.

Эта методика тестирования поможет обеспечить надежность и стабильность работы с файловой системой, выявляя потенциальные проблемы и ошибки на ранних этапах разработки.

4.1 Просмотр директории файловой системы

Цель: проверить возможность просмотра содержимого директории.

Тест-кейсы:

- Просмотр корневой директории.
- Просмотр пустой директории.
- Просмотр директории с несколькими файлами и поддиректориями.

4.2 Переход в новую директорию

Цель: проверить возможность перехода в новую директорию.

Тест-кейсы:

- Переход в существующую поддиректорию.
- Попытка перехода в несуществующую директорию.
- Переход в родительскую директорию.

4.3 Создание файла/директории

Цель: проверить возможность создания новых файлов и директорий.

Тест-кейсы:

- Создание нового файла в текущей директории.
- Создание новой поддиректории.
- Попытка создания файла с уже существующим именем.

4.4 Удаление файла/директории

Цель: проверить возможность удаления файлов и директорий.

Тест-кейсы:

- Удаление существующего файла.
- Удаление пустой директории.
- Попытка удаления директории с непустым содержимым.

4.5 Копирование файла/директории

Цель: проверить возможность копирования файлов и директорий.

Тест-кейсы:

- Копирование файла в другую директорию.
- Копирование директории вместе с ее содержимым.
- Попытка копирования файла с измененным именем.

4.6 Перемещение файла/директории

Цель: проверить возможность перемещения файлов и директорий.

Тест-кейсы:

- Перемещение файла в другую директорию.
- Перемещение директории вместе с ее содержимым.
- Попытка перемещения файла в директорию с именем, совпадающим с именем файла.

4.7 Открытие файла

Цель: проверить возможность открытия файлов для чтения или записи.

Тест-кейсы:

- Открытие существующего файла для чтения.
- Попытка открыть несуществующий файл.
- Открытие файла для записи в существующую директорию.
- Попытка открыть файл, который уже открыт в другой программе.

4.8 Результаты

Для тестирования используется фреймворк Pytest.

Pytest является полнофункциональным инструментом для тестирования на Python, который помогает писать лучше программы. Он обеспечивает простоту написания небольших, читаемых тестов и масштабируемость для поддержки сложного функционального тестирования приложений и библиотек.

Основные особенности и преимущества Pytest:

- Меньше шаблонного кода: Pytest позволяет сократить количество шаблонного кода, необходимого для написания тестов, что делает тесты более чистыми и легче читаемыми.
- Хорошая структура вывода: Pytest предоставляет подробный и понятный вывод результатов тестирования, что облегчает анализ результатов.
- Меньше изучения: Pytest имеет простой синтаксис и минималистичный подход, что снижает порог входа для новых пользователей.
- Управление состоянием и зависимостями: используется механизм фикстур (fixtures) для управления тестовыми зависимостями и состояниями, что упрощает написание тестов.
- Фильтрация тестов: Pytest позволяет легко фильтровать тесты по имени, тегам и другим критериям, что упрощает запуск конкретных наборов тестов.
- Параметризованное тестирование: поддержка параметризованного тестирования для уменьшения дублирования кода между тестами.
- Архитектура на основе плагинов: Pytest обладает гибкой архитектурой на основе плагинов, что расширяет его функциональность и позволяет интегрироваться с другими инструментами и фреймворками.

Далее приведены результаты успешности тестирования и распределение времён для каждого теста.

```
================================== test session starts ==================================
platform win32 -- Python 3.12.2, pytest-8.1.1, pluggy-1.4.0 -- D:\projects\my\virtual-storag
e-system\.win-venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: D:\projects\my\virtual-storage-system
collected 22 items

tests/app/test_server.py::test_status PASSED                                        [  4%]
tests/app/test_storage.py::test_fylesystem_root PASSED                              [  9%]
tests/app/test_storage.py::test_fylesystem_empty_dir PASSED                         [ 13%]
tests/app/test_storage.py::test_fylesystem_dir PASSED                               [ 18%]
tests/app/test_storage.py::test_chdir_exists PASSED                                 [ 22%]
tests/app/test_storage.py::test_chdir_not_exists PASSED                             [ 27%]
tests/app/test_storage.py::test_chdir_parent PASSED                                 [ 31%]
tests/app/test_storage.py::test_create_file PASSED                                  [ 36%]
tests/app/test_storage.py::test_create_dir PASSED                                   [ 40%]
tests/app/test_storage.py::test_create_exists PASSED                                [ 45%]
tests/app/test_storage.py::test_delete_file PASSED                                  [ 50%]
tests/app/test_storage.py::test_delete_dir_empty PASSED                             [ 54%]
tests/app/test_storage.py::test_delete_dir_not_empty PASSED                         [ 59%]
tests/app/test_storage.py::test_copy_file PASSED                                    [ 63%]
tests/app/test_storage.py::test_copy_dir PASSED                                     [ 68%]
tests/app/test_storage.py::test_copy_file_new_name PASSED                           [ 72%]
tests/app/test_storage.py::test_move_file PASSED                                    [ 77%]
tests/app/test_storage.py::test_move_dir PASSED                                     [ 81%]
tests/app/test_storage.py::test_move_dir_to_file PASSED                             [ 86%]
tests/app/test_storage.py::test_open_file PASSED                                    [ 90%]
tests/app/test_storage.py::test_open_not_exits_file PASSED                          [ 95%]
tests/app/test_storage.py::test_open_opened_file PASSED                             [100%]
```

Рис. 4.8.1 – Результаты успешности тестирования

```
================================== slowest durations ===================================
10.63s call     tests/app/test_storage.py::test_open_not_exits_file
7.18s call     tests/app/test_storage.py::test_copy_dir
4.79s call     tests/app/test_storage.py::test_copy_file
4.65s call     tests/app/test_storage.py::test_delete_dir_not_empty
3.64s call     tests/app/test_storage.py::test_open_file
3.35s call     tests/app/test_storage.py::test_copy_file_new_name
2.94s call     tests/app/test_storage.py::test_delete_file
2.07s call     tests/app/test_server.py::test_status
2.07s call     tests/app/test_storage.py::test_chdir_not_exists
2.07s call     tests/app/test_storage.py::test_fylesystem_root
2.07s call     tests/app/test_storage.py::test_chdir_parent
2.07s call     tests/app/test_storage.py::test_chdir_exists
2.07s setup    tests/app/test_server.py::test_status
2.07s call     tests/app/test_storage.py::test_fylesystem_empty_dir
2.05s call     tests/app/test_storage.py::test_fylesystem_dir
2.03s call     tests/app/test_storage.py::test_open_opened_file
1.97s call     tests/app/test_storage.py::test_move_dir
1.87s call     tests/app/test_storage.py::test_create_exists
1.84s call     tests/app/test_storage.py::test_move_file
1.82s call     tests/app/test_storage.py::test_delete_dir_empty
1.61s call     tests/app/test_storage.py::test_create_dir
1.48s call     tests/app/test_storage.py::test_move_dir_to_file
1.37s call     tests/app/test_storage.py::test_create_file
```

Рис. 4.8.2 – Распределение времени тестирования

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной дипломной работы была разработана персональная распределённая файловая система, направленная на обеспечение удобного и эффективного хранения и обработки информации в условиях современных сетевых сред. Разработанная система позволяет объединять ресурсы различных устройств хранения данных в единую пространство, обеспечивая при этом высокую доступность и надежность данных.

Основными достижениями работы стали разработка архитектуры системы, включающей в себя управления файлами, репликацию данных, работу с файлами в интерактивном режиме и асинхронную обработку запросов, а также реализация протоколов взаимодействия между компонентами системы. Это обеспечивают высокую степень автономии и гибкости системы, позволяя её адаптировать под различные условия эксплуатации.

В процессе разработки были учтены актуальные требования к безопасности данных, что позволило реализовать механизмы защиты от несанкционированного доступа и обеспечить целостность хранимых данных. Также была проведена работа по интеграции системы с существующими операционными системами и приложениями, что облегчает её внедрение в рабочие процессы пользователей.

Для дальнейшего развития системы можно рассмотреть следующие направления:

- Улучшение производительности: Исследование и внедрение новых алгоритмов для оптимизации обработки запросов и управления ресурсами, что позволит повысить скорость доступа к данным и общую производительность системы.
- Расширение возможностей репликации: разработка более продвинутых механизмов репликации данных для улучшения отказоустойчивости и обеспечения высокой доступности даже в условиях значительных изменений в сетевой инфраструктуре.
- Поддержка новых форматов файлов и протоколов.
- Автоматизация управления: разработка инструментов для автоматического мониторинга состояния системы, диагностики проблем и управления ресурсами, что упростит административное управление и повысит надежность системы.

Все задачи диссертационной работы выполнены в полном объеме, полученные результаты имеют научную и практическую значимость.

СПИСОК ЛИТЕРАТУРЫ

1. Документация по Go [Электронный ресурс] – URL: https://go.dev/doc/
2. Windows App Development [Электронный ресурс] – URL: https://learn.microsoft.com/en-us/windows/apps/
3. Документация по HTML [Электронный ресурс] – URL: https://developer.mozilla.org/ru/docs/Web/HTML
4. Документация по CSS [Электронный ресурс] – URL: https://developer.mozilla.org/ru/docs/Web/CSS/Reference
5. Документация по JS [Электронный ресурс] – URL: https://developer.mozilla.org/ru/docs/Web/JavaScript
6. RFC 1094 NFS: Network File System Protocol Specification [Электронный ресурс] – URL: https://datatracker.ietf.org/doc/html/rfc1094
7. Руководство по администрированию файловых систем ZFS Solaris [Электронный ресурс] – URL: https://docs.oracle.com/cd/E19253-01/820-0836/820-0836.pdf
8. The Google File System [Электронный ресурс] – URL: https://research.google/pubs/the-google-file-system/
9. Уайт, Том. Hadoop. Подробное руководство. — 2-е. — СПб.: Питер, 2013. — 672 с. — 1000 экз. — ISBN 978-5-496-00662-0.
10. Weil, Sage A. and Brandt, Scott A. and Miller, Ethan L. and Long, Darrell D. E. and Maltzahn, Carlos (2006). "Ceph: A Scalable, High-performance Distributed File System". Proceedings of the 7th Symposium on Operating Systems Design and Implementation. OSDI ’06. Seattle, Washington: USENIX Association. pp. 307—320.
11. Lustre Home [Электронный ресурс] – URL: https://www.lustre.org
12. Data replication strategies in wide-area distributed systems [Электронный ресурс] – URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5d849e722cb3a22b27086dd90962348e4fca7bcc
13. A Scalable Distributed File System [Электронный ресурс] – URL: https://dl.acm.org/doi/pdf/10.1145/268998.266694