МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение

высшего профессионального образования

«Донской государственный технический университет»

ДГТУ

Кафедра «ПОВТ и АС»

УТВЕРЖДАЮ

Зав.каф._____ Нейдорф Р.А.

"_____"_____2011г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к преддипломной практике на тему:

Распределенная файловая система

Практикант Лубягов Николай Александрович_____ Группа __ВИ–51_____

Специальность __230105 Программное обеспечение вычислительной техники и автоматизированных систем_____

Руководитель работы _____ / Клубков Иван Михайлович_____/

Нормоконтролер _____ / Котельникова Ирина Владимировна /

Ростов-на-Дону

2011г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение

высшего профессионального образования

«Донской государственный технический университет»

ДГТУ

Кафедра «ПОВТ и АС»

УТВЕРЖДАЮ

Зав.каф._____ Нейдорф Р.А.

"_____"_____2011г.

ЗАДАНИЕ

на преддипломную практику

Практикант Лубягов Николай Александрович_____ Группа__ВИ-51____

Тема _Распределенная Файловая Система_____

Исходные данные для преддипломной практики: _Параллельная кластерная файловая система для высокопроизводительных вычислительных систем [Электронный ресурс] -– http://www.swsys.ru/index.php?page=article&id=375___

Руководитель работы _____ /Клубков Иван Михайлович_____/

Задание принял к исполнению _____ /Лубягов Николай Александрович/

РЕФЕРАТ

Отчет содержит: листов - 153, рисунков - 24, приложений – 4.

Ключевые слова: ФАЙЛОВАЯ СИСТЕМА, ПРОТОКОЛ, ОБМЕН ФАЙЛАМИ, КЛИЕНТ, СЕРВЕР, ПАРСЕР, XML, КЛАСТЕР, ИНФОРМАЦИЯ.

В данном отчете дипломной работы «Распределенная файловая система» рассматривается:

- современные распределенные файловые систем;
- современные инструменты построения масштабируемых файловых систем;
- разработка схемы построения распределенной файловой системы;
- разработка протокола обмена данными файловой системы;
- разработка протокола обмена запросами на операции с файлами.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сетевая файловая система является программным продуктом, позволяющим производить монтирование каталогов в сети и трактовать удаленные файлы как локальные. Идея сетевых файловых систем далеко ненова. Это значит, что первые реализации файловых систем, внешних по отношению к физическим ЭВМ, появились в начале 80х - это файловые системы для VAX-VMS кластеров. Эту идею развивали многие компании, в результате чего мы по сути имеем несколько фундаментальных концептов разделяемых файловых систем, не говоря уже о количестве конкретных реализаций данных концептов. Фундаментальные концепты включают сетевые файловые системы (NFS), параллельные файловые системы (Lustre) и симметричные блочно-разделяемые файловые системы (GFS).

Распределенная сеть машин, может обеспечить более агрегированную производительность вычислительного оборудования по сравнению с большой ЭВМ. Сетевая обработка данных вообще более эффективна, нежели централизованная обработка.[1]

1 ОБЗОР СЕТЕВЫХ ФАЙЛОВЫХ СИСТЕМ

1.1 Текущее состояние вопроса

Уже прошли те времена, когда сайтов было мало и пользователи были согласны ждать, пока восстановят функционирование ненадежного сайта. В Интернете при столь высокой конкуренции даже пятиминутный сбой в работе сервиса вызывает отток пользователей к конкуренту. Поэтому каждый, кто хочет построить устойчивый сервис, должен начать с ее основ – отказоустойчивой файловой системы. Кластерные файловые системы еще не достаточно приспособлены для использования на крупных предприятиях: обычно процесс их развертывания и поддержания в работающем состоянии не так уж прост. Но зато они отлично масштабируются и достаточно дешевы, ведь для них достаточно самого простого серверного оборудования и свободных операционных систем и программного обеспечения.

1.2 Обзор существующих аналогов

Существует достаточно большое число файловых систем работающих по сети и организующих доступ к файлам на удаленных компьютерах. Среди таких файловых систем:

- SSHFS — предоставляет доступ к удалённой ФС через SSH;
- GmailFS — файловая система, которая хранит данные как почту в Gmail;
- WikipediaFS — просмотр и редактирование статей Википедии так, как будто они являются файлами;
- CurlFtpFS — предоставляет доступ к удалённой ФС через libcurl;
- LftpFS — предоставляет доступ к удалённой ФС через lftp;
- Andrew File System (AFS) — Сетевая кластерная файловая система;

- GlusterFS — высокопроизводительная кластерная ФС;
- GoogleFS — кластерная система;
- Lustre — кластерная файловая система.

Наиболее близкими файловыми системами к поставленной задаче являются AFS, GlusterFS и Lustre.

1.2.1 Google File System

Кластерная система оптимизированная для работы с большими блоками данных по 64 Мб (chunk), а также обладающая повышенной защитой от сбоев. Вся информация копируется и хранится в трёх (или более) местах одновременно, при этом система способна очень быстро находить реплицированные копии, если какая-то машина вышла из строя. Задачи автоматического восстановления после сбоя решаются с помощью программ, созданных по модели MapReduce. Является коммерческой тайной компании Google. Несовместима с POSIX и создавалась Google для своих внутренних потребностей[2].

1.2.2 SSHFS (Secure SHell FileSystem)

SSHFS (Secure SHell FileSystem) это файловая система для Linux (и других операционных систем, для которых существует реализация FUSE (Filesystem in Userspace), например Mac OS X), используемая для удаленного управления файлами по протоколу SSH (точнее, его расширению SFTP) таким образом, как будто они находятся на локальном компьютере. Администратор может настроить ограниченный аккаунт на сервере для обеспечения большей безопасности и пользователь сможет видеть только выделенную ему область в системе. [3]

1.2.3 GmailFS

Google бесплатно позволяет создавать почтовые ящики размером 6Гб. Достаточно интересным является использование GMail в качестве хранилища файлов. GmailFS позволяет пользователю смонтировать удаленную файловую систему, на которой он может использовать до 6 Гб дискового пространства.[4]

1.2.4 WikipediaFS

Это монтируемая Linux виртуальная файловая система позволяющая читать и редактировать статьи из Википедии и других сайтов основанных на Mediawiki так, как будто это обычные файлы. Что позволяет просматривать и редактировать статьи, используя обычный текстовый редактор. Текстовые редакторы, как правило, более удобны, чем просто формы браузера, когда речь идет о редактировании больших текстов, и они, как правило, включают такие полезные функции, как подсветка синтаксиса Mediawiki и проверка орфографии[5].

1.2.5 CurlFtpFS

Это файловая система для доступа к FTP хостам основанная на FUSE и libcurl, в отличии от других FTP файловых систем она имеет следующие функции: поддержка SSLv3 и TLSv1 шифрования, подключение через HTTP прокси, автоматическое переподключение в случае разрыва соединения, преобразовывает абсолютные ссылки в точку возврата на FTP-файловой системе[6].

1.2.6 LftpFS

Сетевая файловая система с доступом только для чтения, интеллектуально кэширующая зеркала(mirrors) сайтов. Используется для

создания копий Linux репозиториев (repositories). Она основана на FUSE и LFTP клиенте и поддерживает FTP, HTTP, FISH, SFTP, HTTPS, FTPS протоколы, позволяет работать с ней через прокси[7].

1.2.7 Andrew File System (AFS)

Продукт AFS представляет собой распределенную файловую систему, изначально разработанную в университете Карнеги-Меллона, она поддерживается и развивается как продукт корпорации Transarc (в настоящее время IBM Pittsburgh Labs). Она предлагает архитектуру клиент-сервер для обмена файлами, распределенного доступа содержимого только для чтения, распространения содержимого, предоставляя возможности независимости, масштабируемости, безопасности и свободной миграции. AFS доступна для широкого диапазона операционных систем, включая UNIX, Linux, MacOS X и Microsoft Windows. Она имеет несколько преимуществ по сравнению с традиционными сетевыми файловыми системами, в частности в области безопасности и масштабируемости. Секция AFS может поддерживать более двадцати пяти тысяч клиентов. AFS использует Kerberos (компьютерный сетевой протокол аутентификации, позволяющий отдельным пользователям общаться через незащищённые сети для безопасной идентификации. Так же является набор бесплатного ПО от Массачусетского Технологического Института (Massachusetts Institute of Technology (MIT)), разработавшего этот протокол. Его организация направлена в первую очередь на клиент-серверную модель и обеспечивает взаимную аутентификацию - оба пользователя через сервер подтверждают личности друг друга. Сообщения, отправляемые через протокол Kerberos, защищены от прослушивания и атак.) для аутентификации и осуществляет списки контроля доступа на директории для пользователей и групп. Каждый клиент кэширует файлы на локальную файловую систему, что

увеличивает скорость при последующих запросах на тот же файл. Это также позволяет иметь ограниченный доступа к файловой системы в случае сбоя сервера или коммерческих сетей[8].

IBM выделил из исходного продукта AFS, и сделал копию исходного кода для сообщества разработчиков и технического обслуживания. Она получила название Open AFS[9].

1.2.8 Lustre

Lustre представляет собой кластерную файловую систему, основными особенностями которой являются превосходные надежность и масштабируемость. Производительность также более чем высока — скорость передачи данных может достигать сотен гигабит в секунду, а теоретический максимум доступного дискового пространства измеряется петабайтами. Эта файловая система может использоваться как на скромных рабочих группах из нескольких компьютеров, так и на огромных кластерах, насчитывающих десятки тысяч машин[10].

Помимо этого поддерживаются все возможности, которые должна иметь любая уважающая себя кластерная файловая система:

- поддержка широкого ассортимента типов высокоскоростных сетевых соединений;
- надежная система «замков» для обеспечения параллельного доступа к файлам;
- возможность автоматического самовосстановления в случае падения любого из узлов;
- распределенное управление файловыми объектами для предоставления масштабируемого доступа к файлам.

Изначально архитектура этой файловой системы была разработана просто

в рамках исследовательского проекта Петера Браама в 1999, но он решил не останавливаться на достигнутом и основал Cluster File Systems, Inc., в которой уже и велась основная разработка самой файловой системы. Первый релиз Lustre 1.0 был выпущен в 2003 году. Спустя четыре года компания была приобретена Sun Microsystems в октябре 2007 года, но это лишь способствовало дальнейшему развитию проекта. Программное обеспечение, входящее в состав проекта, выпускается под лицензией GPL, что также сыграло немаловажную роль в его жизни.

Архитектура

Каждый компьютер, входящий состав кластера Lustre, выполняет свою четко определенную функцию:

- MDS — сервер метаданных предназначен для хранения всей служебной информации о системе: названия файлов, директорий, прав доступа и так далее. Достаточно наличие одного такого сервера в системе, но для обеспечения надежности на случай каких-либо сбоев, обычно его дублируют. Возможно использование внешнего хранилища данных (MDT), которое может быть общим для двух дублирующих друг друга MDS;
- OSS — компьютеры для хранения самих данных. Каждый из них работает с 2-8 OST, в их роли могут выступать практически любые средства хранения данных, начиная от просто жестких дисков или RAID массивов внутри OSS, заканчивая внешними системами хранения данных enterprise-класса. Сумма дискового пространства всех OST и является размером доступного дискового пространства всей файловой системы Lustre;
- клиент — компьютеры, непосредственно использующие файловую

систему. Им предоставляется полный параллельный доступ, полностью соответствующий стандарту POSIX.

Один и тот же компьютер теоретически может совмещать в себе несколько функций, но в большинстве случаев это нецелесообразно (за исключением совмещения клиентов с OST и, возможно, случаев, когда количество узлов кластера очень мало). Написанное выше наглядно можно представить схемой архитектуры системы (рис. 1.1).

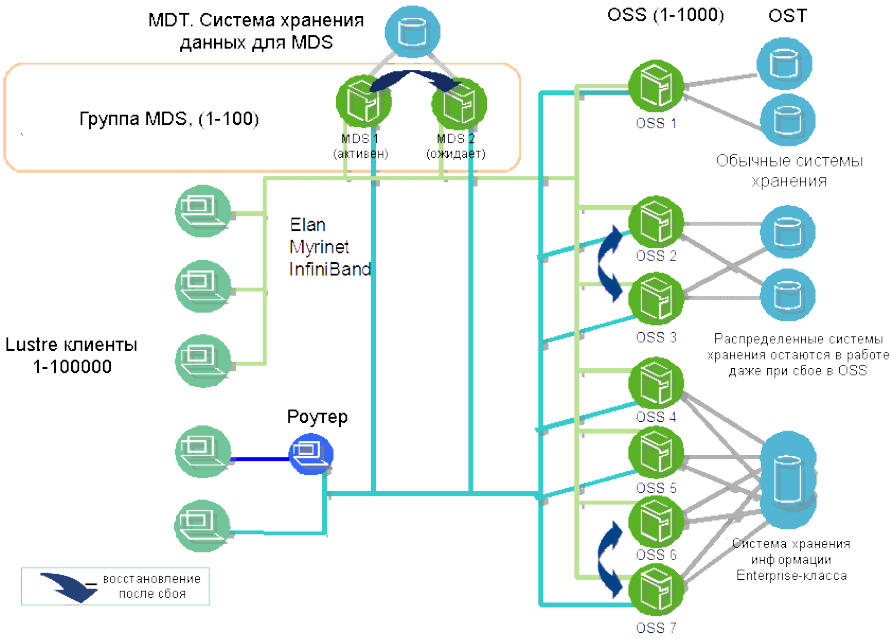

Рисунок 1.1 – Схема архитектуры системы Lustre

Помимо этого для функционирования системы необходим еще один компонент, по большому счету не являющийся ее частью — MGS. Его роль заключается в предоставлении конфигурационной информации всем компонентам одной или нескольким файловым системам Lustre. Он также нуждается в отдельном хранилище данных, но теоретически он может быть совмещен с одним из компонентов файловой системы.

Функционирование

Основным толчком для выполнения каких-либо действий в рамках всей файловой системы обычно является запрос с одного из клиентов. Программное обеспечение для клиентов представляет по сути интерфейс между виртуальной файловой системой Linux и серверами Lustre. Каждому типу серверов соответствует своя часть клиентского ПО: MDC, OSC, MGT. В отличии от Hadoop и GFS файловая система Lustre должна быть примонтирована к локальной системе клиентов для полноценного их функционирования.

Для осуществления коммуникации между клиентами и серверами используется собственный API, известный как LNET. Он поддерживает множество сетевых протоколов с помощью NAL.

В системе отсутствуют незаменимые компоненты, это является залогом отказоустойчивости системы. В случае возникновения каких-либо неполадок или сбоев в работе оборудования, работу вышедших из строя компонентов системы перехватят другие ее компоненты, что сделает сбой незаметным для пользователей системы. Это достигается за счет дублирования серверов, выполняющих одинаковые функции, а также наличие налаженных алгоритмов действий, направленных на автоматическое восстановление полноценного функционирования системы в случае возникновения чрезвычайных ситуаций. Но этого конечно же не достаточно для абсолютной надежности системы, в дополнение должна быть предоставлена как минимум система бесперебойного питания для всех компонентов кластера на случай проблем с электроэнергией в датацентре (для России более чем актуально).

В списке дополнительных возможностей, предоставляемых файловой системой, можно назвать возможность выделения квот на дисковое пространство для каждого пользователя системы, аутентификацию пользователей с помощью механизма Kerberos, повышение физической

пропускной способности сетевого соединения путем агрегирования физических сетевых соединений в одно логическое виртуальное сетевое соединение (достаточно интересная возможность, способная при выполнении определенных условий существенно повлиять на быстродействие системы). Помимо этого предоставляется целый ряд возможностей по созданию резервных копий данных на уровне файловой системы в целом, отдельных устройств или же файлов.

Эта файловая система нашла свое применение во множестве крупнейших кластеров и суперкомпьютеров по всему миру, но это не мешает ей с тем же успехом демонстрировать и на кластерах существенно меньшего масштаба. Около половины из самых производительных суперкомпьютеров во всем мире используют Lustre в качестве файловой системы. Помимо этого многие компании предоставляют ее в качестве основы для Linux кластеров (например HP StorageWorks SFS, Cray XT3, Cray XD1).

1.2.9 GlusterFS

Кластерная файловая система, способная хранить до нескольких пета-байт информации. Она позволяет соединять в одну большую параллельную сетевую файловую систему несколько хранилищ данных, посредством Infiniband RDMA или TCP / IP соединения. GlusterFS основана на создании стека дисковых пространств пользователей без ущерба для производительности.[11]

Особенности

GlusterFS представляет собой кластерную файловую систему, способную масштабироваться для хранения до нескольких петабайта данных. Как и многие другие кластерные файловые системы, GlusterFS агрегирует дисковое пространство большого количества машин в одну общую параллельную сетевую файловую систему через Infiniband RDMA или TCP/IP соединение. Обычно в

качестве аппаратной основы для этой файловой системы используется недорогое серверное оборудование, в полной мере реализуя принцип программного построения стабильности при использовании на ненадежном оборудовании[12]. Список основных ее особенностей по большей части мало чем отличается от других кластерных файловых систем:

- ФС состоит из клиентской и серверной частей. Клиентская часть позволяет монтировать файловую систему, а серверная — glusterfsd — экспортировать в нее локальное дисковое пространство;
- масштабируемость близка к O (1);
- широкий спектр возможностей за счет использования модульной архитектуры;
- имеется возможность восстановления файлов и директорий из файловой системы даже без ее инициализации;
- отсутствие централизованного сервера метаданных, что делает ее более устойчивой к потенциальным сбоям;
- расширяемый интерфейс выполнения задач, с поддержкой загрузки модулей в зависимости от особенностей выполнения пользователями операций по работе с данными;
- расширяющий функциональность механизм трансляторов;
- поддержка Infiniband RDMA и TCP/IP;
- возможность автоматического восстановления в случае сбоев;
- полностью реализована на уровне приложений, что упрощает ее поддержание в рабочем состоянии, портирование и дальнейшую разработку.

Но некоторые моменты все же заслуживают отдельного внимания.

Совместимость

Как уже упоминалось, файловая система реализована полностью на уровне пользовательских приложений, что делает возможным ее монтирование без каких-либо дополнительных патчей в ядре операционной системы, единственное требование к нему: поддержка FUSE. Серверная часть GlusterFS может функционировать на любой POSIX-совместимой операционной системе и протестирована на Linux, FreeBSD, OpenSolaris, в отличии от клиентской части, которая может работать только в Linux.

Модули

В виде модулей реализованы различные варианты выполнения основополагающих операций: передачи данных и балансировки нагрузки в рамках кластера. Транспортные модули обеспечивают передачу данных по различным типам соединений:

- TCP/IP;
- infiniband-verbs;
- infiniband-SDP.

Балансировка нагрузки может выполняться по следующим алгоритмам:

- ALU — использует целый ряд факторов, включающий объем свободного локального дискового пространства, активность операций чтения и записи, количество одновременно открытых файлов, скорость физического вращения дисков. Значимость, придаваемая каждому из показателей, может достаточно гибко настраиваться;
- RR — по очереди размещает файлы последовательно на каждом узле, после чего начинает процесс заново, образуя своеобразный цикл. Этот метод эффективен если файлы имеют примерно одинаковый размер, а узлы кластера — одинаковый размер экспортированного локального

дискового пространства;

- Random — распределяют файлы случайным образом;
- NUFA — приоритет отдается созданию файлов локально, а не на других узлах кластера;
- Switch — располагает файлы по определенным указанным особенностям имен файлов, по аналогии со switch (filename) в программировании, обычно в качестве критерия распределения файлов имеет смысл использовать их расширение.

Трансляторы

Они представляют собой очень мощный механизм для расширения возможностей GlusterFS, сама идея трансляторов бала позаимствована у GNU/Hurd и заключается она в загрузке бинарных библиотек (.so) в процессе работы системы в зависимости от использованных настроек и использовании их в виде своеобразной цепочки обработчиков при работе с файлами как на серверной, так и на клиентской стороне. В GlusterFS практически все дополнительные возможности реализованы именно виде трансляторов, начиная от дополнений, увеличивающих производительность, заканчивая средствами отладки. Вкратце перечислю основные из них:

- AFR — автоматическая репликация файлов;
- Stripe — разбивает файлы на блоки фиксированного размера;
- Unify — объединяет несколько узлов кластера в один большой виртуальный узел, один узел выделяется для обеспечения внутреннего namespace. Директории создаются на всех узлах, составляющих unify, а каждый файл — лишь на одном (если не используется AFR);
- Trace — предоставляют информацию для отладки в виде дополнительных записей в лог;

- Filter — фильтрация файлов на основании их имен и/или атрибутов;
- Posix-locks — обеспечивает POSIX блокировку записей независимую от используемой системы хранения;
- Trash — предоставляет функциональность сопоставимую с libtrash (или «корзиной» — если так понятнее);
- Fixed-id — обеспечивает доступ только для пользователей с определенными UID и GUID;
- Posix — соединяет GlusterFS с низлежащей локальной файловой системой;
- rot-13 — транслятор обеспечивает возможность шифрования и дешифрования данных по примитивному одноименному алгоритму.

Благодаря возможности работы через Infiniband (высокоскоростная коммутируемая последовательная шина, применяющаяся как для внутренних (внутрисистемных), так и для межсистемных соединений[13]) производительность передачи данных также достаточно высока — она может достигать десятков гигабит в секунду. Обработка сбоев в отдельных узлах также осуществляется достаточно эффективно, так как может быть автоматизирована. Из потенциальных недостатков можно назвать некоторое количество редко проявляющих себя багов в коде, а также достаточно большой размер заголовков в используемом протоколе (несколько сотен байт). В целом эта система вполне работоспособна и полноценно выдерживает конкуренцию со стороны своих opensource «коллег».

1.2.10 Hadoop и HDFS

Hadoop представляет собой платформу для построения приложений, способных обрабатывать огромные объемы данных[14]. Система основывается на распределенном подходе к вычислениям и хранению информации, основными ее особенностями являются:

- масштабируемость: с помощью Hadoop возможно надежное хранение и обработка огромных объемов данных, которые могут измеряться петабайтами;
- экономичность: информация и вычисления распределяются по кластеру, построенному на самом обыкновенном оборудовании. Такой кластер может состоять из тысяч узлов;
- эффективность: распределение данных позволяет выполнять их обработку параллельно на множестве компьютеров, что существенно ускоряет этот процесс;
- надежность: при хранении данных возможно предоставление избыточности, благодаря хранению нескольких копий. Такой подход позволяет гарантировать отсутствие потерь информации в случае сбоев в работе системы;
- кроссплатформенность: так как основным языком программирования, используемым в этой системе является Java, развернуть ее можно на базе любой операционной системы, имеющей JVM.

HDFS

В основе всей системы лежит распределенная файловая система под незамысловатым названием Hadoop Distributed File System. Представляет она собой вполне стандартную распределенную файловую систему, но все же она обладает рядом особенностей:

- устойчивость к сбоям, разработчики рассматривали сбои в оборудовании скорее как норму, чем как исключение;
- приспособленность к развертке на самом обыкновенном ненадежном оборудовании;
- предоставление высокоскоростного потокового доступа ко всем данным;
- настроена для работы с большими файлами и наборами файлов;
- простая модель работы с данными: один раз записали — много раз прочли;
- следование принципу: переместить вычисления проще, чем переместить данные.

Архитектура HDFS

Проще всего ее демонстрирует схема, позаимствованная с официального сайта проекта и переведенная на русский язык (рис. 1.2).

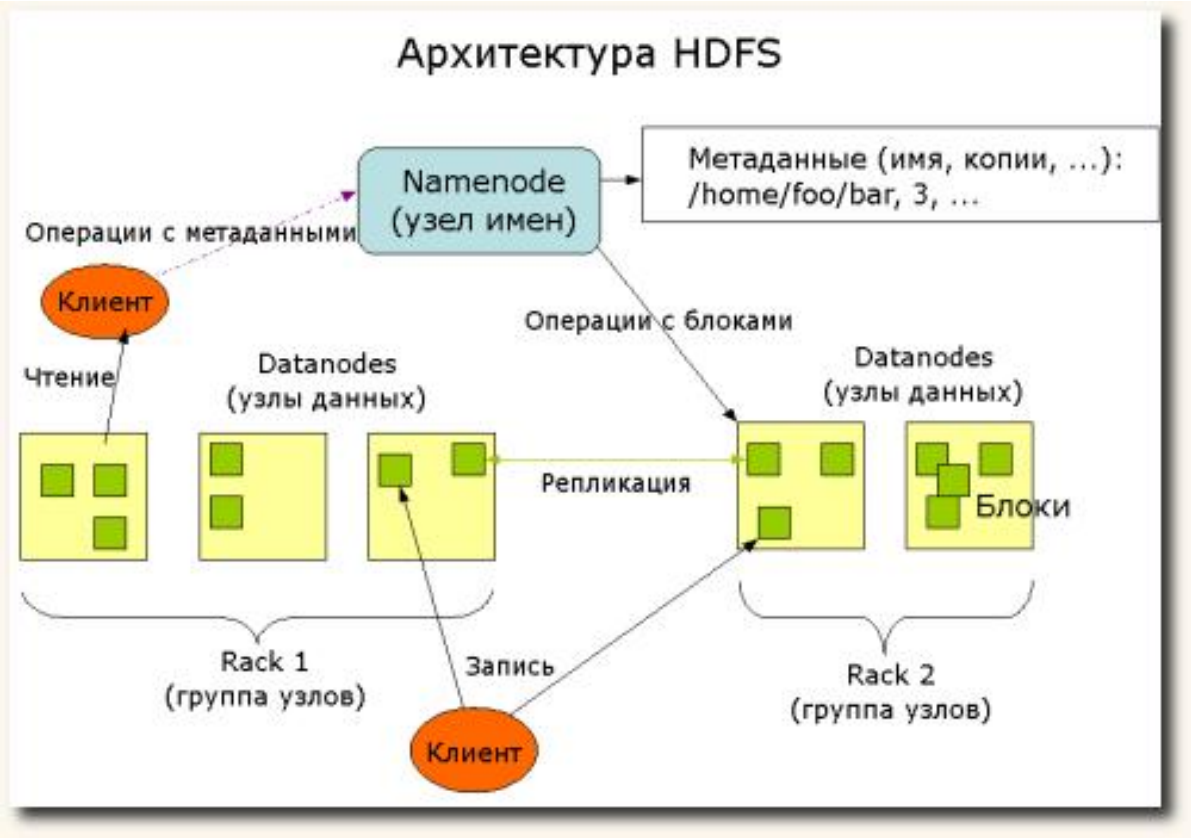

Рисунок 1.2 – Схема архитектуры HDFS

Компоненты HDFS:

- namenode — этот компонент системы осуществляет всю работу с метаданными. Он должен быть запущен только на одном компьютере в кластере. Именно он управляет размещением информации и доступом ко всем данным, расположенным на ресурсах кластера. Сами данные проходят с остальных машин кластера к клиенту мимо него;
- datanode — на всех остальных компьютерах системы работает именно этот компонент. Он располагает сами блоки данных в локальной файловой системе для последующей передачи или обработки их по запросу клиента. Группы узлов данных принято называть Rack, они используются, например, в схемах репликации данных;
- клиент — приложение или пользователь, работающий с файловой системой. В его роли может выступать практически что угодно.

Пространство имен HDFS имеет классическую иерархическую структуру: пользователи и приложения имеют возможность создавать директории и файлы. Файлы хранятся в виде блоков данных произвольной (но одинаковой, за исключением последнего; по-умолчанию 64 mb) длины, размещенных на Datanode'ax. Для обеспечения отказоустойчивости блоки хранятся в нескольких экземплярах на разных узлах, имеется возможность настройки количества копий и алгоритма их распределения по системе. Удаление файлов происходит не сразу, а через какое-то время после соответствующего запроса, так как после получения запроса файл перемещается в директорию /trash и хранится там определенный период времени на случай, если пользователь или приложение передумают о своем решении. В этом случае информацию можно будет восстановить, в противном случае — физически удалить.

Для обнаружения возникновения каких-либо неисправностей, Datanode периодически отправляют Namenode'у сигналы о своей работоспособности. При прекращении получения таких сигналов от одного из узлов, Namenode помечает его как «мертвый» и прекращает какое-либо с ним взаимодействие, до возвращения его работоспособности. Данные, хранившиеся на «умершем» узле реплицируются дополнительный раз из оставшихся «в живых» копий, и система продолжает свое функционирование, как ни в чем не бывало.

Все коммуникации между компонентами файловой системы проходят по специальным протоколам, основывающимся на стандартном TCP/IP. Клиенты работают с Namenode с помощью так называемого ClientProtocol, а передача данных происходит по DatanodeProtocol, оба они обернуты в Remote Procedure Call (RPC).

Система предоставляет несколько интерфейсов, среди которых командная оболочка DFSShell, набор ПО для администрирования DFSAdmin, а также простой, но эффективный веб-интерфейс. Помимо этого существуют

несколько API для языков программирования: Java API, C pipeline, WebDAV и так далее.

MapReduce

Помимо файловой системы, Hadoop включает в себя framework для проведения масштабных вычислений, обрабатывающих огромные объемы данных. Каждое такое вычисление называется Job (задание) и состоит оно, как видно из названия, из двух этапов:

- map — целью этого этапа является представление произвольных данных (на практике чаще всего просто пары ключ-значение) в виде промежуточных пар ключ-значение. Результаты сортируются и групируются по ключу и передаются на следующий этап;
- reduce — полученные после map значения используются для финального вычисления требуемых данных. Практически любые данные могут быть получены таким образом, все зависит от требований и функционала приложения.

Задания выполняются, подобно файловой системе, на всех машинах в кластере (чаще всего одних и тех же). Одна из них выполняет роль управления работой остальных — JobTracker, остальные же ее беспрекословно слушаются — TaskTracker. В задачи JobTracker'a входит составление расписания выполняемых работ, наблюдение за ходом выполнения, и перераспределение в случае возникновения сбоев.

В общем случае каждое приложение, работающее с этим framework'ом, предоставляет методы для осуществления этапов map и reduce, а также указывает расположения входных и выходных данных. После получения этих данных JobTracker распределяет задание между остальными машинами и предоставляет клиенту полную информацию о ходе работ.

Помимо основных вычислений могут выполняться вспомогательные

процессы, такие как составление отчетов о ходе работы, кэширование, сортировка и так далее.

1.3 Выбор библиотеки для реализации файловой системы

1.3.1 FUSE

Filesystem in Userspace (FUSE) (Файловая система в пользовательском пространстве) — это модуль для ядер Unix-подобных ОС, с открытым исходным кодом и относящийся к свободному программному обеспечению. Модуль распространяется под лицензиями GNU GPL и GNU LGPL. Он позволяет пользователям без привилегий создавать их собственные файловые системы без необходимости переписывать код ядра. Это достигается за счёт запуска кода файловой системы в пространстве пользователя, в то время как модуль FUSE только предоставляет «мост» для актуальных интерфейсов ядра. FUSE была официально включена (слита) в главное дерево кода Linux в версии 2.6.14[15]. С помощью FUSE можно разработать файловую систему в пространстве пользователя без знания внутреннего устройства файловой системы или изучения программирования модулей ядра. FUSE особенно полезна для написания виртуальных файловых систем. В отличие от традиционных файловых систем, которые по существу сохраняют информацию для восстановления данных с диска, виртуальные файловые системы не хранят данные непосредственно. Они действуют как представление, трансляция (перевод) существующей файловой системы или устройства хранения. В принципе, любой ресурс, доступный для использования FUSE, может быть экспортирован в файловую систему. Сама система FUSE была частью проекта A Virtual Filesystem (AVFS), но потом AVFS выделился в собственный проект на SourceForge.net [8].

FUSE позволяет разрабатывать полностью функциональную файловую

систему, которая имеет простую библиотеку API, может использоваться непривилегированными пользователями и обеспечивает защищенную реализацию. К тому-же FUSE обладает доказанной стабильностью.

При помощи FUSE вы можете разработать файловую систему в виде исполняемых двоичных файлов, связанных с библиотеками FUSE; другими словами, эта интегрированная файловая система не требует от вас изучения внутреннего устройства файловой системы или программирования модулей ядра[16].

Что касается файловых систем, то файловая система в пространстве пользователя не является чем-то новым. Приведем несколько примеров коммерческих и академических реализаций таких систем:

- LUFS — гибридная файловая система в пространстве пользователя, поддерживающая неопределенное число файловых систем прозрачно для любого приложения. Она состоит из модуля ядра и демона, работающего в пространстве пользователя. По существу, он делегирует большинство VFS-вызовов специализированному демону, который их обрабатывает;
- UserFS позволяет пользовательским процессам быть смонтированными как нормальные файловые системы. Этот экспериментальный прототип предоставляет программу ftpfs, которая организует анонимный FTP с интерфейсом файловой системы;
- Ufo Project — глобальная файловая система для Solaris, которая позволяет пользователям работать с удаленными файлами так, как будто они являются локальными;
- OpenAFS — это версия с открытыми исходными кодами файловой системы Andrew FileSystem;
- CIFS — Common Internet FileSystem.

Так же с помощью FUSE разработаны уже рассмотренные GlusterFS, GmailFS, CurlFtpFS, SSHFS, WikipediaFS, LftpFS. В отличие от этих коммерческих и академических примеров FUSE переносит возможности этих проектов файловых систем в Linux. Поскольку FUSE использует исполняемые файлы (вместо, например, разделяемых объектов, используемых в LUFS), облегчается отладка и разработка. FUSE работает с обеими версиями ядра (2.4.x и 2.6.x) и теперь поддерживает Java™-связывание, т.е. вы не ограничены программированием файловой системы в C и C++

1.4 Вывод

В следствии исследования предметной области были выявлены положительные, и отрицательные особенности существующих файловых систем, которые можно учесть в разрабатываемой системе. Их можно представить в виде таблицы (табл. 1.1).

Название ПС		Особенность
AFS	-	предлагает архитектуру клиент-сервер для обмена файлами, распределенного доступа содержимого только для чтения
	+	Каждый клиент кэширует файлы на локальную файловую систему, что увеличивает скорость при последующих запросах на тот же файл. Это также позволяет иметь ограниченный доступа к файловой системы в случае сбоя сервера или коммерческих сетей
Gluster	+	ФС состоит из клиентской и серверной частей. Клиентская часть позволяет монтировать файловую систему, а серверная — glusterfsd — экспортировать в нее локальное дисковое пространство
	+	широкий спектр возможностей за счет использования модульной архитектуры

	+	отсутствие централизованного сервера метаданных, что делает ее более устойчивой к потенциальным сбоям
	+	расширяемый интерфейс выполнения задач, с поддержкой загрузки модулей в зависимости от особенностей выполнения пользователями операций по работе с данными
	+	полностью реализована на уровне приложений, что упрощает ее поддержание в рабочем состоянии, портирование и дальнейшую разработку.
	+	Серверная часть GlusterFS может функционировать на любой POSIX-совместимой операционной системе и протестирована на Linux, FreeBSD, OpenSolaris, в отличии от клиентской части, которая может работать только в Linux.
	+	Балансировка нагрузки может выполнятся по нескольким алгоритмам, реализуемым, в виде отдельных модулей.
	-	достаточно большой размер заголовков в используемом протоколе (несколько сотен байт).
Hadoop	+	при хранении данных возможно предоставление избыточности, благодаря хранению нескольких копий. Такой подход позволяет гарантировать отсутствие потерь информации в случае сбоев в работе системы;
	+	Для обнаружения возникновения каких-либо неисправностей, Datanode периодически отправляют Namenode'у сигналы о своей работоспособности. При прекращении получения таких сигналов от одного из узлов, Namenode помечает его как «мертвый» и прекращает какое-либо с ним взаимодействие, до возвращения его работоспособности

Таблица 1.1 – Особенности существующих файловых систем

С учетом особенностей, можно сделать следующие выводы по построению

ф.с. Архитектура программы реализующей файловую систему, должна является клиент-серверной, без наличия сервера транзакций, и метаданных, как в GlusterFS в отличии от Lustre и HDFS, это упростит ее функционирование, хотя может вызвать проблемы с конфигурированием нескольких клиентов. Файловая система должна кэшировать файлы на стороне клиента, благодаря чему повышается надежность системы, даже при сбое в сети она продолжит частично функционировать, что так-же упростит функции сохранения данных на другие компьютеры при выключении или выходе из строя компьютера на который производится запись, подобно AFS. Идея трансляторов и модулей, позволит достичь более структурированный код, и даст возможность расширения файловой системы по мере необходимости без модификации самого ядра системы, таким образом как это сделано в GlusterFS. Использовать минимально возможный размер заголовка в пакетах при передаче содержимого файлов, для того чтобы свести к минимуму объем передаваемых по сети данных на заголовках пакетов, в отличии от GlusterFS. Реализовать несколько алгоритмов выбора компьютера на который производится запись как отдельные модули, реализовать обработку запросов от клиентов отдельными модулями, в том числе аутентификацию и файловые операции, подобно GlusterFS. В протоколе должны быть ping пакеты подобно системе HDFS, которые будут отправляться к клиентам от серверов, и если клиент не получает информацию, в течении некоторого промежутка времени, то сервер будет считаться «умершим» и удален из списка серверов внутри клиента. Реализовать два отдельных независимых протокола для данных, которые передают содержимое файлов и для метаданных, как в Hadoop и HDFS. Организовать дублирование информации при записи файлов, по возможности на несколько серверов.

1.5 Постановка задачи

Реализовать распределенную файловую систему, создающую единое дисковое пространство для компьютеров локальной сети. При осуществлении записи в единое пространство наиболее подходящий для размещения файлов компьютер должен выбираться системой автоматически. Предусмотреть возможность динамического подключения и отключения узлов сети, при этом дисковое пространство должно быть доступно постоянно, кроме файлов, размещенных на отключенных в данный момент узлах сети.

Для этого необходимо:

- создать протокол обмена данными между клиентом и сервером;
- разработать алгоритмы надежного обмена данными, устойчивого к аппаратным сбоям, и сбоям в сети;
- создать программу клиент, монтирующего файловую систему;
- создать программу сервер, реализующего операции с файлами по запросам клиента.

При решении задачи будем применять:

- TCP/IP для создания соединения между клиентом и сервером;
- Протокол обмена данными на основе XML для связи клиента и сервера;
- Модульную архитектуру, благодаря которой программы будут подгружать модули расширений, клиента и сервера динамически.

Протокол обмена данными должен обеспечивать взаимодействие клиента и сервера. Он должен организовывать возможность отправки запросов операций с файлами, и получения ответов о результатах операций, на эти запросы.

Разрабатываемые алгоритмы обмена данными должны обеспечивать взаимодействие клиентов с группой серверов, и взаимодействие серверов с

несколькими клиентами одновременно. Так же должна быть организованна возможность динамического подключения и отключения серверов.

Клиентское приложение должно монтировать файловую систему в указанную директорию, организовывать подключение к серверам, отправляя на сервера запросы об операциях над файлами производимых в точке монтирования.

Серверное приложение должно взаимодействовать с клиентами и выполнять запрошенные клиентами операции над файлами в общедоступных директориях.

Сервер и клиент должны реализовывать модульную архитектуру, благодаря которой, при запуске программ будет просматриваться папка, с находящимися в ней динамически скомпонованными библиотеками — модулями, и подгружать их по мере необходимости. Модули должны расширять возможности протокола, организуя возможности взаимодействия клиента и сервера.

В соответствии с поставленными задачами произвести алгоритмическое проектирование и программное конструирование программ реализующих распределенную файловую систему. Выполнить тестирование работоспособности приложений, подтверждающие корректность работы программы.

В качестве инструментов реализации поставленной задачи разрешено использовать любые технологии, языки программирования и среды разработки.

2 АЛГОРИТМИЧЕСКОЕ КОНСТРУИРОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

При построении файловой системы можно выделить две независимых программы, сервер и клиент. Сервер создает соединение и ожидает подключения клиентов, для работы по сети можно использовать соединение по TCP-IP. Клиент подключившись к серверу посылает команды для работы с файлами, сервер выполняет, те или иные действия и возвращает ответы клиенту. Клиент должен одновременно подключаться к группе серверов. Со стороны клиента должен стоять FUSE модуль который позволяет монтировать файловую систему на клиентскую машину, серверная же сторона просто выполняет действия над файлами используя системные вызовы ядра UNIX. В качестве протокола обмена можно разработать протокол на основе XML (eXtensible Markup Language) и обрабатывать его XML парсером по технологии SAX (Simple API for XML). Формат XML позволяет создавать легко расширяемый интуитивно понятный протокол разметки, таким образом при внесении модификаций в новых версиях файловой системы можно добиться, того что старые версии клиентов будут работать с новыми серверами и наоборот без внесения значительных изменений в код программ. Однако XML не позволяет передавать внутри себя поток данных таких как файлы. Поскольку протокол является текстовым, он не может обеспечить передачу символов, которые являются частью его разметки. Для решения данной проблемы обычно использую кодировку формата MIME (Multipurpose Internet Mail Extensions — многоцелевые расширения почты интернета) — стандарт, описывающий передачу различных типов данных по электронной почте, а также, спецификация для кодирования информации и форматирования сообщений таким образом, чтобы их можно было пересылать по Интернету. Однако данный формат вызывает значительный прирост к пересылаемому трафику от сервера к клиенту, что в результате может повлечь значительное замедление функционирования системы. Для решения

поставленной задачи можно использовать второй двоичный протокол, через который будет производиться передача содержимого файлов. В таком случае клиент и сервер буде иметь два постоянно открытых соединения. Через первое соединение будет производиться передача запросов на файловые операции (чтение содержимого директорий, переименование удаление, получение и изменение атрибутов файлов и прочие операции), а по второму протоколу будет производиться непосредственно сама передача данных содержимого файлов.

2.1 Требования к разрабатываемой ФС

При построении файловой системы следует учесть следующие требования:

- в случае ошибки нехватки дискового пространства, данные должны быть переписаны на другую машину, и произведена до-запись. Если подобных машин не найдено, тогда вызывать ошибку;
- в случае отключения компьютера в момент записи должен быть выбран новый компьютер и данные перезаписаны на него, таким образом программа на стороне клиента должна кэшировать данные и производить их запись на выбранный компьютер параллельно;
- программа должна функционировать в операционной системе UNIX и ее разновидностях;
- разрабатываемое приложение должно состоять из двух программ: клиента который подключается к набору серверов и выполняет операции над файлами, и сервера, который реализует файловую систему. Клиент подключается к серверу и производит операции над файлами. Любой клиент может подключаться к одному или более серверов одновременно, любые сервера могут быть подключены к нескольким клиентам;

- файловая система должна поддерживать расширяемые модули для сервера и клиента, благодаря которым будут осуществляться операции с файлами, выбор компьютеров для записи фалов, аутентификация и прочие действия;
- для реализации передачи данных разработать два собственных протокола обмена информацией между сервером и клиентом, один для передачи метаданных, второй для передачи содержимого файлов, благодаря которым будут производиться все поставленные задачи;
- внутри протокола должны быть пакеты поддержания соединения для проверки работоспособности сервера.

2.2 Структура разрабатываемой файловой системы

В результате анализа полученной информации можно выделить основные составляющие части разрабатываемой программы и проиллюстрировать взаимодействие с другими компонентами системы (рис. 2.1).

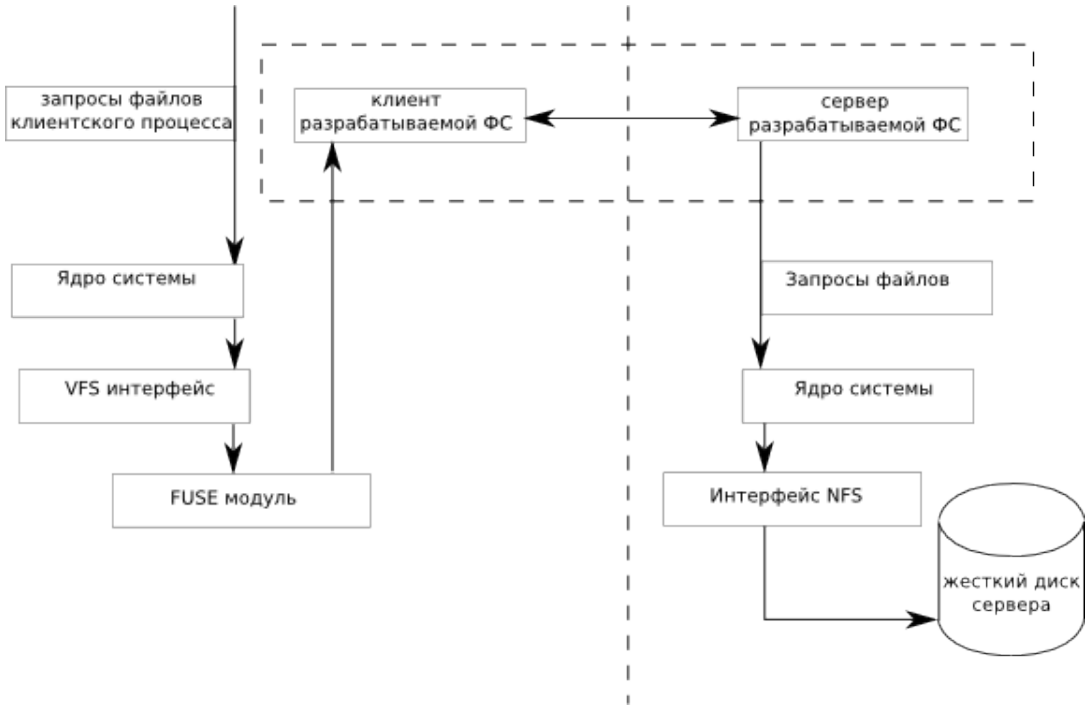

Рисунок 2.1 – Схема взаимодействие разрабатываемой программы с компонентами системы

Здесь пунктиром выделена та часть которую требуется разработать. Связь соединяющая сервер и клиент является связью многие к многим, т.е. один клиент может подключаться к нескольким серверам и один сервер может быть подключен к нескольким клиентам одновременно.

Структуру создаваемой программы, тогда можно описать следующим образом, (рис. 2.2).

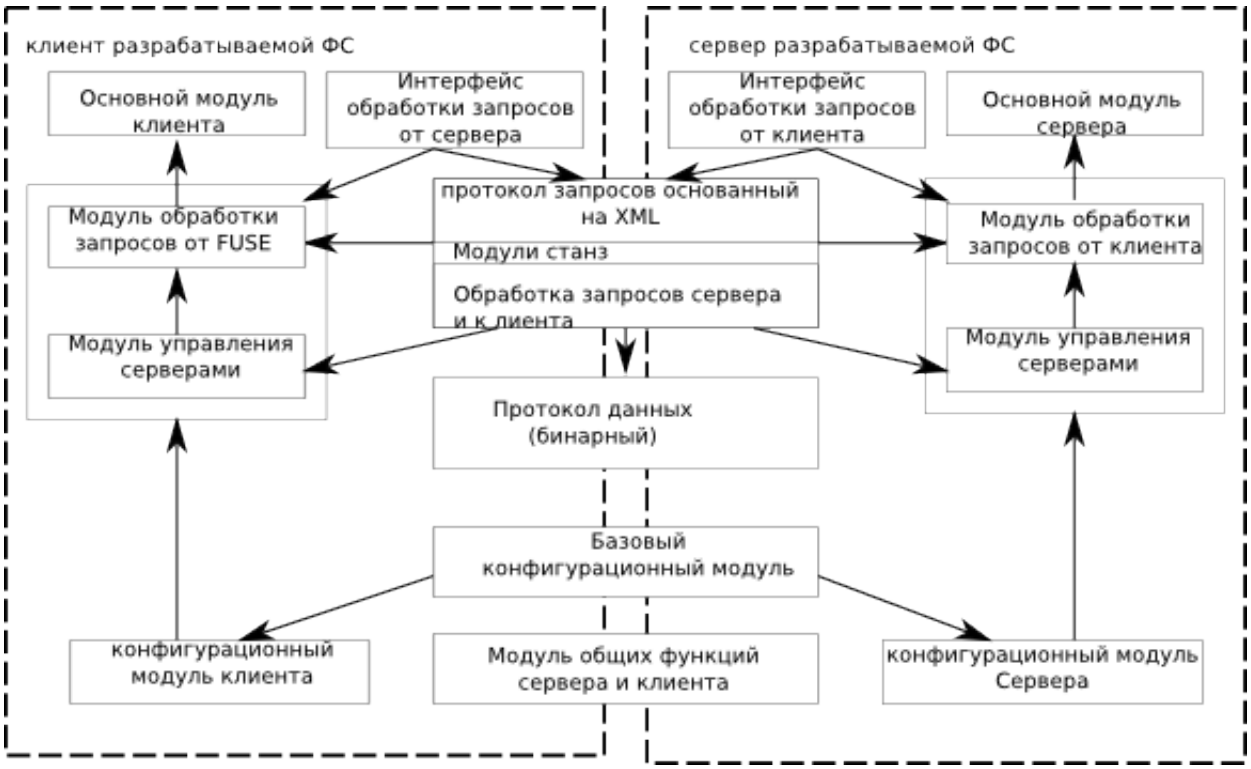

Рисунок 2.2 – Структурная схема разрабатываемой программы

Таким образом можно выделить около шестнадцати взаимосвязанных модулей.

Общие модули для сервера и клиента:

- протокол запросов основанный на XML — описывает сам протокол работы клиента и сервера;
- модуль станз — в нем описываются станзы которыми обменивается сервер и клиент;
- обработка запросов сервера и клиента — описывает протокол, который выполняет запросы на прием и передачу станз, данных

файлов, чтения содержимого и отправка содержимого файлов, объединяя в себе бинарный протокол;

- бинарный протокол — модуль описывает сам протокол через который будет производиться передача данных, константы идентификаторы пакетов, структуры передаваемых данных и методы с помощью которых будет передаваться информация в бинарном протоколе. А так же интерфейс событий приходящих от бинарного протокола;
- модуль общих функций сервера и клиента — содержат общие глобальные функции и типы данных, которые используются как сервером так и клиентом;
- базовый конфигурационный модуль — в нем описаны свойственные как для клиента так и для сервера данные из конфигурационного файла.

Модули клиента:

- основной модуль — содержит метод main;
- модуль обработки запросов FUSE — Содержит саму виртуальную файловую систему, в которой находятся файлы хранящиеся на серверах;
- модуль управления серверами — содержит список экземпляров серверов, в нем описаны сами сервера, содержаться экземпляры их протоколов, через которые к ним выполняются запросы;
- конфигурационный модуль клиента — в нем находится производный от базового конфигурационного модуля модуль, в нем описывается все свойственные для клиента данные из конфигурационного файла;
- интерфейс обработки запросов от сервера — содержит интерфейс в котором описаны методы вызываемые при тех или иных событиях происходящих на сервере и возврата от него команд.

Модули сервера:

- основной модуль — содержит метод main;
- модуль обработки запросов клиента — содержит методы выполняющиеся при происхождении событий на стороне клиента, запросов на чтение файлов и каталогов;
- модуль управления клиентами — содержит список экземпляров клиентов, в нем описаны сами клиенты, содержаться экземпляры их протоколов. От них сервер получает запросы на те или иные действия выполняет какие либо события связанные с реальной файловой системой;
- конфигурационный модуль сервера — в нем находится производный от базового конфигурационного модуля модуль, в нем описывается все свойственные для сервера данные из конфигурационного файла;
- интерфейс обработки запросов от клиента — содержит интерфейс в котором описаны методы вызываемые при тех или иных событиях происходящих на стороне клиента и возврата от него команд.

2.3 Основные алгоритмы разрабатываемой ФС

В разрабатываемых приложениях можно выделить основные алгоритмы, для взаимодействия сервера и клиента. Часть из алгоритмов уникальна для клиента и сервера, часть является общей.

2.3.1 Алгоритмы уникальные для сервера

Сервер принимает подключения от клиентов и их обрабатывает в отдельных потоках, для этого сначала сервер инициализирует начальные данные, загружает конфигурационный файл, создает сокетные подключения, для протокола на основе XML и для бинарного протокола. Затем сервер ожидает,

когда к нему подключится новый клиент, после чего будет получен дескриптор сокета, этого клиента, через который необходимо производить обмен данными клиента и сервера. На следующем этапе сервер создается новый поток управления, внутри, которого происходит основной цикл взаимодействия клиента и сервера, до тех пор пока клиент не отключится, а основной поток программы переходит в ожидание подключения нового клиента. В случае если сервер получил сигнал завершения приложения, SIGINT, производится обработка сигнала, в таком случае программа производит освобождение всех мьютексов ожидающих как по бинарному протоколу, так по протоколу на основе XML данных, отправляет клиенту запрос на отключение сервера, и выходит из всех потоков обработки клиентов, освобождает выделенные ресурсы, сокетные соединения, и конфигурационный файл, затем программа-сервер завершает свою работу.

Общий алгоритм подключения клиентов к серверу можно описать следующей блок-схемой (рис. 2.3):

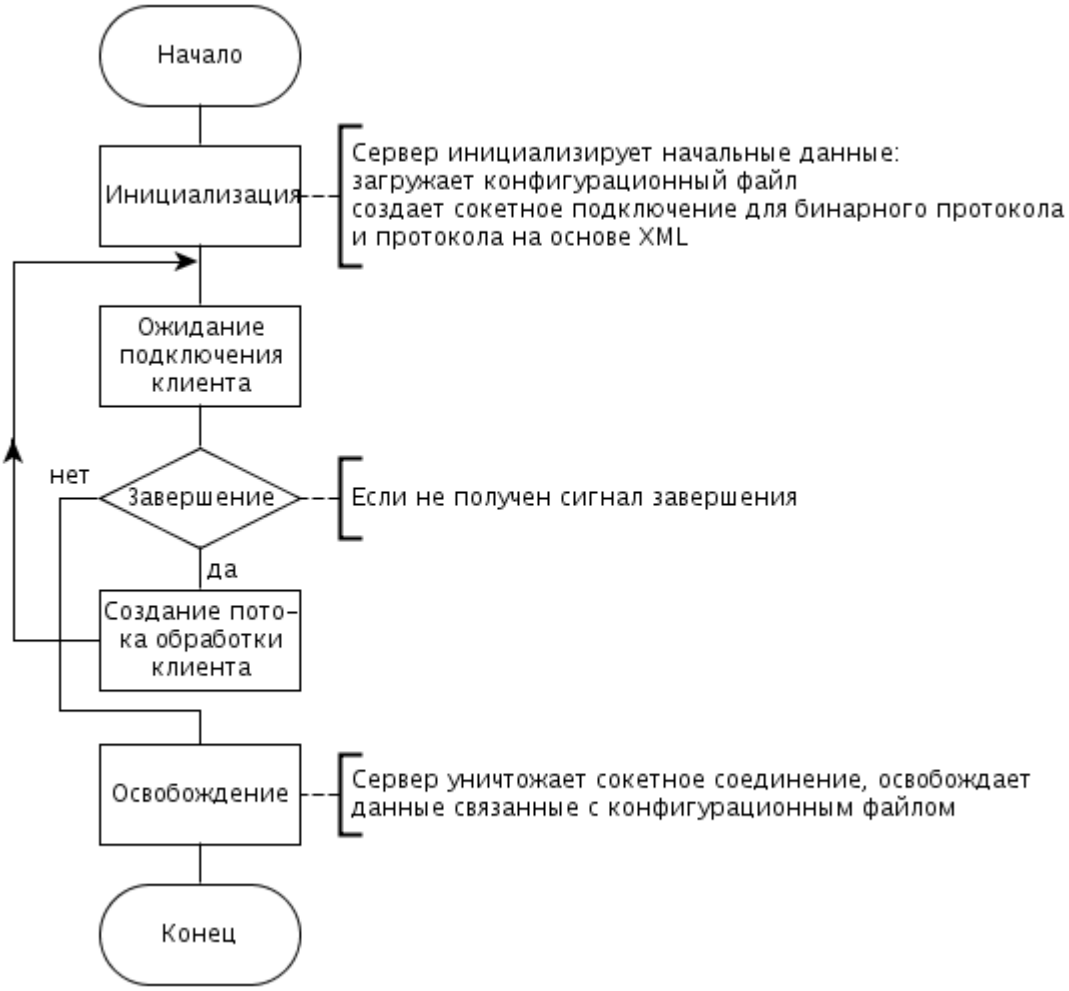

Рисунок 2.3 – Алгоритм организации подключений клиентов к серверу

Внутри потока обработки клиента можно выделить алгоритм который будет выполняться параллельно для каждого клиента. В качестве входных данных в него передаются идентификаторы сокетных подключений к клиенту по бинарному протоколу и по протоколу на основе XML. Затем производится нициация данных парсера на основе XML протокола и бинарного протокола. Создание XML парсера, который будет производить разбор XML потока. Регистрация обработчиков событий на начала и конец тега, для потокового разброра, а так-же на произвольные данные внутри тегов. Затем для подключенного клиента производится регистрация модулей, модули будут обрабатывать запросы приходящие от клиента, и должны зарегистрировать обработчики. При регистрации модулей вычисляется новый ID

зарегистрированных модулей, он будет передан в функцию регистраций модулей, и в функцию удаления регистраций модулей. На следущем этапе создается поток обработки бинарного протокола, а затем вход в цикл разбора протокола на основе XML, этот цикл будет выполнятся до тех пор пока либо не будет получен закрывающийся тег от сервера </stream:stream> либо до завершения работы программы сервера. После завершения работы с клиентом производится удаление регистраций модулей, и закрытие сокетного подключения. Описанный алгоритм представлен на рисунке (рис. 2.4):

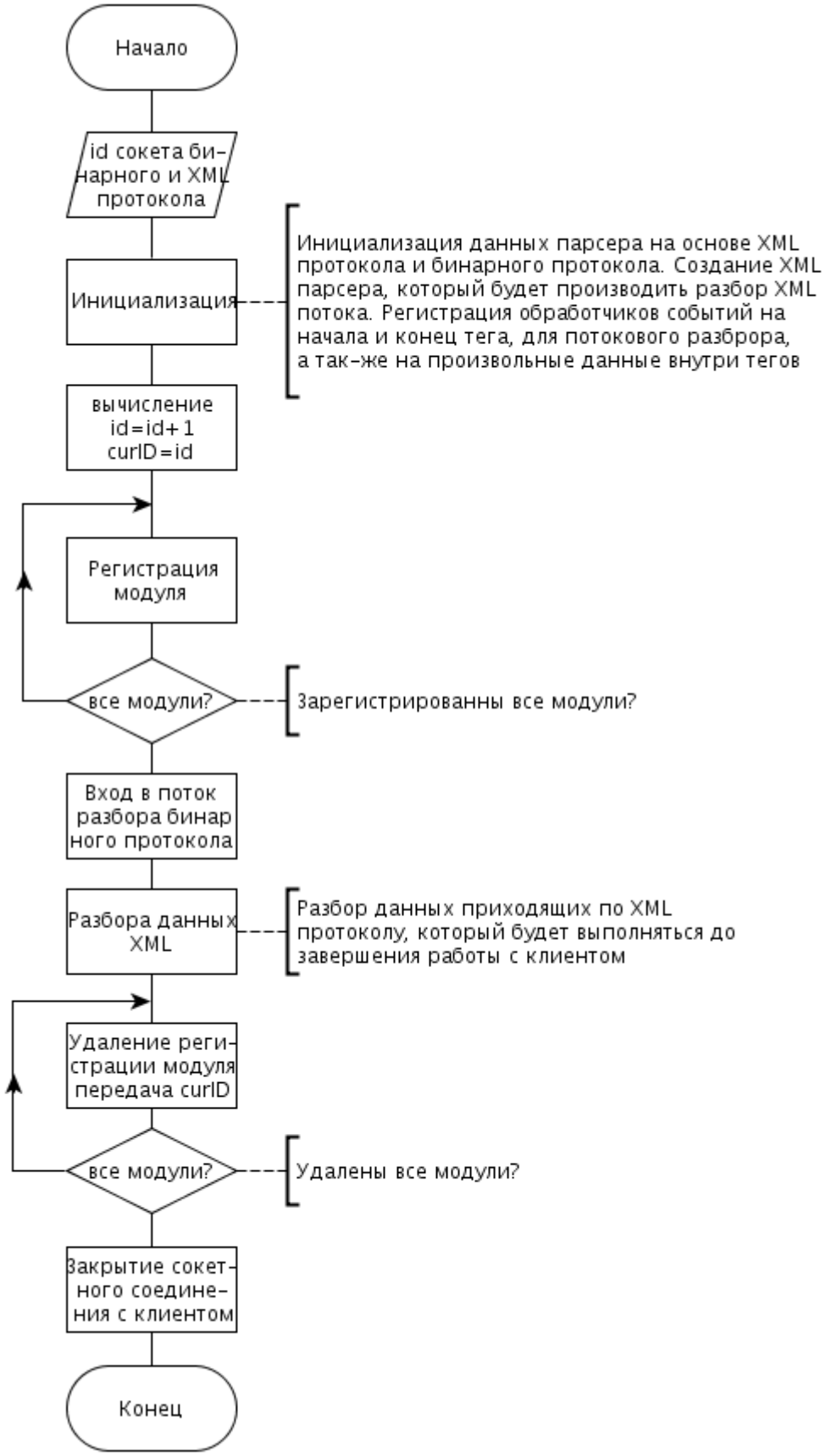

Рисунок 2.4 – Алгоритм цикла обработки клиента

Регистрацию модулей для протоколов, можно выделить в отдельный алгоритм. Каждый модуль должен зарегистрировать обработчики станз приходящих из клиента, которые он будет обрабатывать. Входными параметрами данного алгоритма являются указатель на объект протокола на основе XML, указатель на объект бинарного протокола, указатель на конфигурационный файл сервера, и идентификатор регистрируемых модулей. Модуль создает необходимые обработчики станз, и регистрирует их в объекте протокола на основе XML. Сохраняет указатели на протоколы на основе XML и бинарного протокола во внутренние структуры, при этом идентификатор регистрируемых модулей, может быть использован как ключ в этой структуре, для того чтобы различать клиентские подключения.

Ниже представлена блок-схема алгоритма регистрации модуля (рис. 2.5):

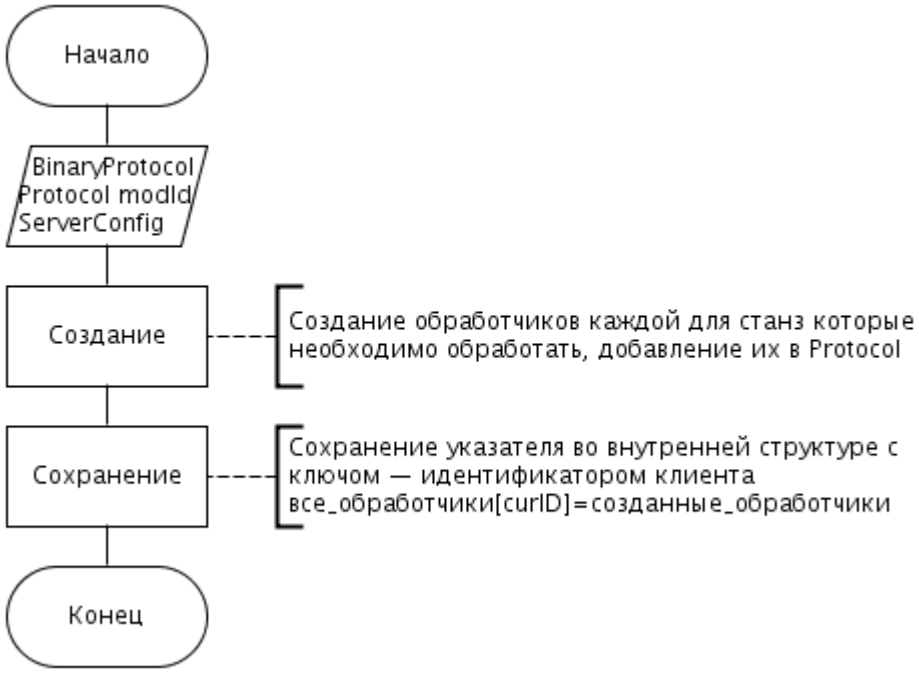

Рисунок 2.5 – Алгоритм регистрации модулей сервера

Алгоритм удаления модуля выполняется после отключения клиента, в качестве входных данных ему передается идентификатор подключенных

модулей, modId, он используется для поиска нужного указателя на класс протокол клиента, и на структуру которая содержит обработчики станз. Далее производится удаление регистраций обработчиков из объекта протокола на основе XML и уничтожение объектов обработчиков стан.

Алгоритм удаления модуля можно представить следующим образом (рис 2.6):

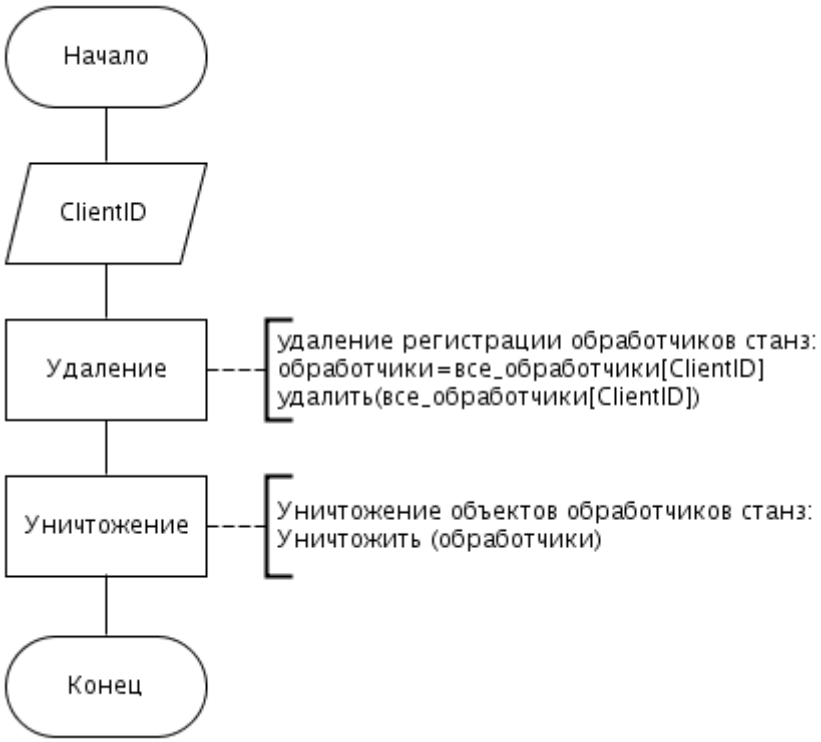

Рисунок 2.6 – Алгоритм удаления модулей сервера

2.3.2 Алгоритмы уникальные для клиента

Клиент должен подключаться к нескольким серверам, постоянно поддерживая с ними соединение, выполняя действия над файлам на их дисках. Список серверов клиент должен взять из конфигурационного файла, и подключиться к каждому, запустить поток обработки, зарегистрировать обработчики операций над файлами в структуре fuse_operations и войти в цикл FUSE, который завершиться только тогда, когда файловая система будет

отмонтирована.

На начальном этапе работы клиента, он получает необходимые данные из конфигурационного файла. Для этого создается экземпляр объекта конфигурационного файла, затем из него клиент получает адреса серверов, порты для подключения по бинарному протоколу и протоколу на основе XML, для каждого сервера. В цикле, для каждого сервера создается экземпляр объектов бинарного, и протокола на основе XML, и производится подключение, если оно прошло успешно то создаются потоки для обработки бинарого протокола и протокола на основе XML. Если подключиться не получилось то в таком случае, адрес сервера, и порты заносятся в структуру, которая сохраняет данные для не подключенных серверов, эта структура будет опрошена в будующем и алгоритм подключения будет выполняться еще еще раз, для этих серверов периодически. После того как клиент будет подключен ко всем доступным серверам, Создаются потоки обработки бинарного протокола и протокола на основе XML они точно такие-же как для сервера. Единственная разница протоколов сервера и клиента состоит в том что клиент подключается к серверу, а сервер ожидает сокетное подключение от клиента.

На следующем этапе производится регистрация модулей расширений клиента она, состоит в заполнении структуры обработчиков функций, для операций над файловой системой. В модули расширений передается указатели на объект конфигурационного файла, а так-же указателей на список, состоящий из указателей на объекты бинарного протокола и протокола на основе XML, для каждого сервера к которому подключен клиент. Модули, при вызове функций обработки действий над ФС, отсылают запросы на сервера, и получают от них ответы.

Данный алгоритм можно представить следующим образом (рис. 2.7):

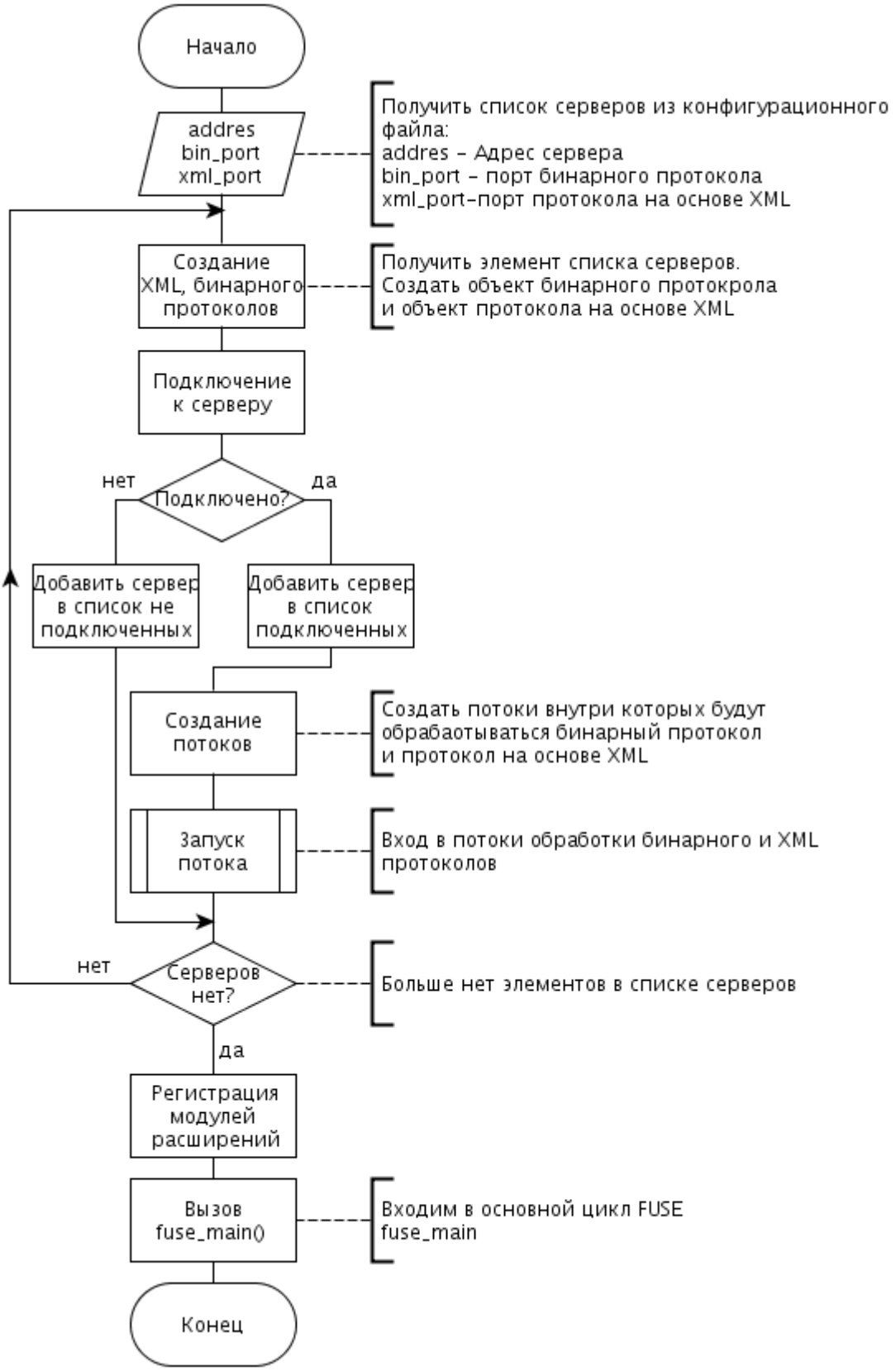

Рисунок 2.7 – Алгоритм подключения клиента к серверам

2.3.3 Общие алгоритмы для клиента и сервера

Цикл разбора данных приходящих по протоколу на основе XML, должен быть организован внутри модуля разбора XML, данный модуль использован как в серверной так и клиентской части.

Сначала производится создание парсера XML который будет разбирать приходящие с противоположной стороны данные. Парсеру передаются указатели на функции которые будут вызываться при получении начала, конца, или произвольных данных внутри тега. Затем через уже созданные сокетные подключения принимаются данные, каждая порция полученных данных передается парсеру, который их анализирует. В случае если объем полученных данных равен нулю, это означает что либо произошла ошибка соединения, либо клиент отключился, либо в результате анализа данных парсером произошла ошибка и он вернул отрицательный результат, тогда происходит уничтожение парсера и уничтожение инициализированных для разбора данных, .мьютексов и прочих ресурсов. Алгоритм разбора потокового XML можно представить следующей блок-схемой (рис. 2.8):

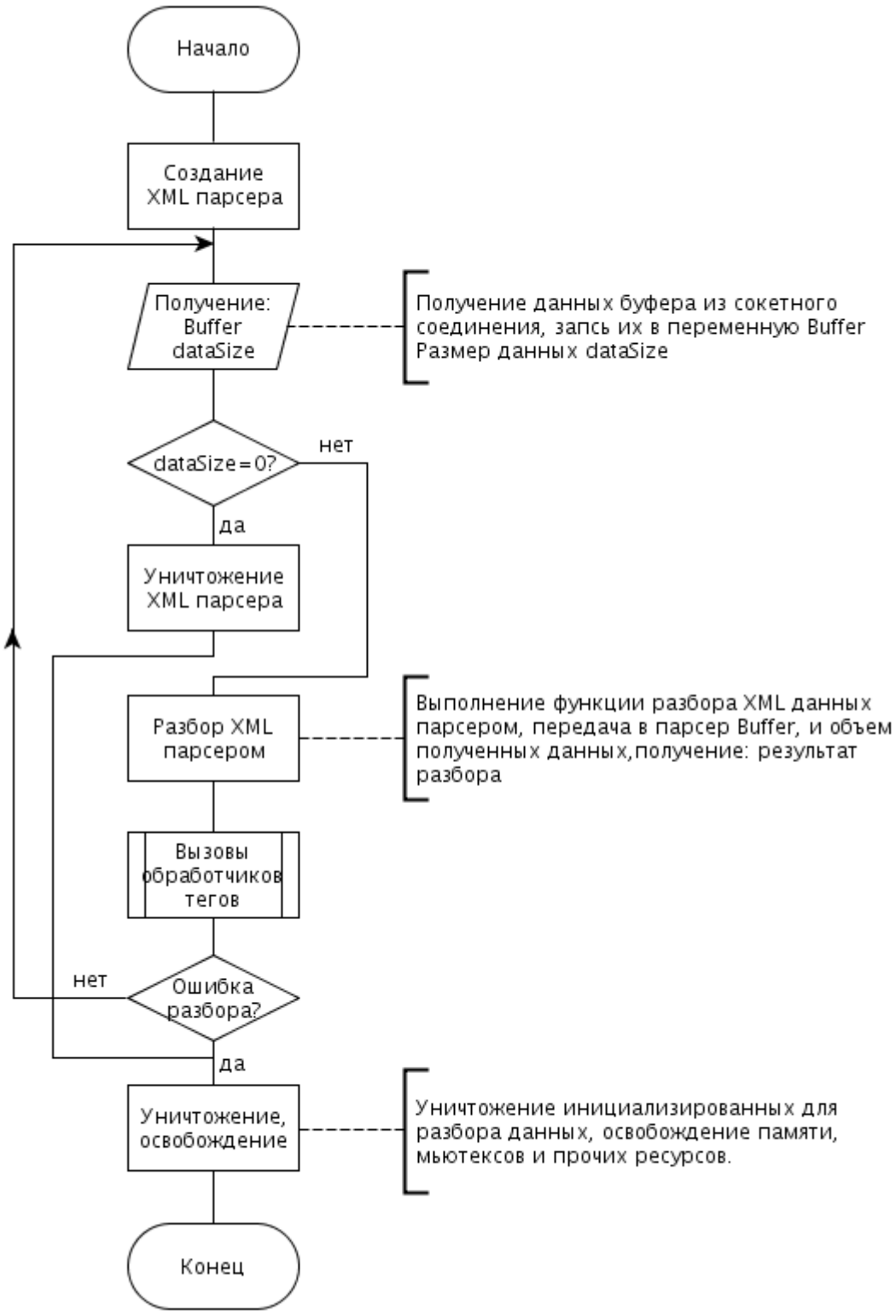

Рисунок 2.8 – Алгоритм разбора потокового XML

Алгоритм, будет вызывать обработчики тегов XML протокола, посредством внешнего потокового парсера. Анализатор будет находить начало

тега, внутри потока данных вида:

<имя_тега атрибут1= «значение1» атрибут2= «значение2» атрибутN= «значениеN»>

И вызывать обработчик начала тега. Затем повторять операцию для всех вложенных тегов, и входить внутрь дерева, после чего, должны быть вызваны обработчики произвольных данных внутри тега, если они есть:

<имя_тега атрибуты>

произвольные текстовые данные

</имя тега>

Данные и имя тега будут переданы обработчик.

Еще один обработчик, производит обработку окончания тега. Каждый тег должен закрыться. Ему передается имя тега. В нем будет получен один целый тег, если число вызовов обработчиков начала и конца тега различается на один. В таком случае будет получена готовая станза, которая будет передана в подходящие обработчики станз. Если разность равна нулю, то это означает что обработана самая верхняя в дереве нода и нужно завершать обработку. В общем случае поток XML можно представить следующим образом:

<stream:stream>

Верхний уровень, он будет создан при подключении клиентов и сервере один раз.

<имя_тега атрибуты>

Первый уровень, на нем формируется целая единичная команда к серверу.

<имя_тега атрибуты>

Более низкие уровни описывающие, данные которые требуется передать в обработчики станз.

</имя_тега>

</имя тега>

</stream:stream>

Таким образом можно выделить три взаимосвязанных алгоритма.

Алгоритм обработки начала тега, реализован внутри функции обработчика начала тега, вызываемой парсером. Входными параметрами данного алгоритма являются указатель на объект протокола, атрибуты тега, и имя полученного тега. Сначала текущий уровень увеличивается на единицу создается новый лист дерева, с атрибутами attrs и и именем name. В том случае если это первый лист дерева, то есть текущаяя нода в объекте XML не оперделена, то в объекте протокола присваивается в качестве ссылки на верхнюю ноду — созданная нода. Текущая нода в объекте протокола присваивается вновь созданной. Данный алгоритм можно представить следующей блок-схемой(рис 2.9):

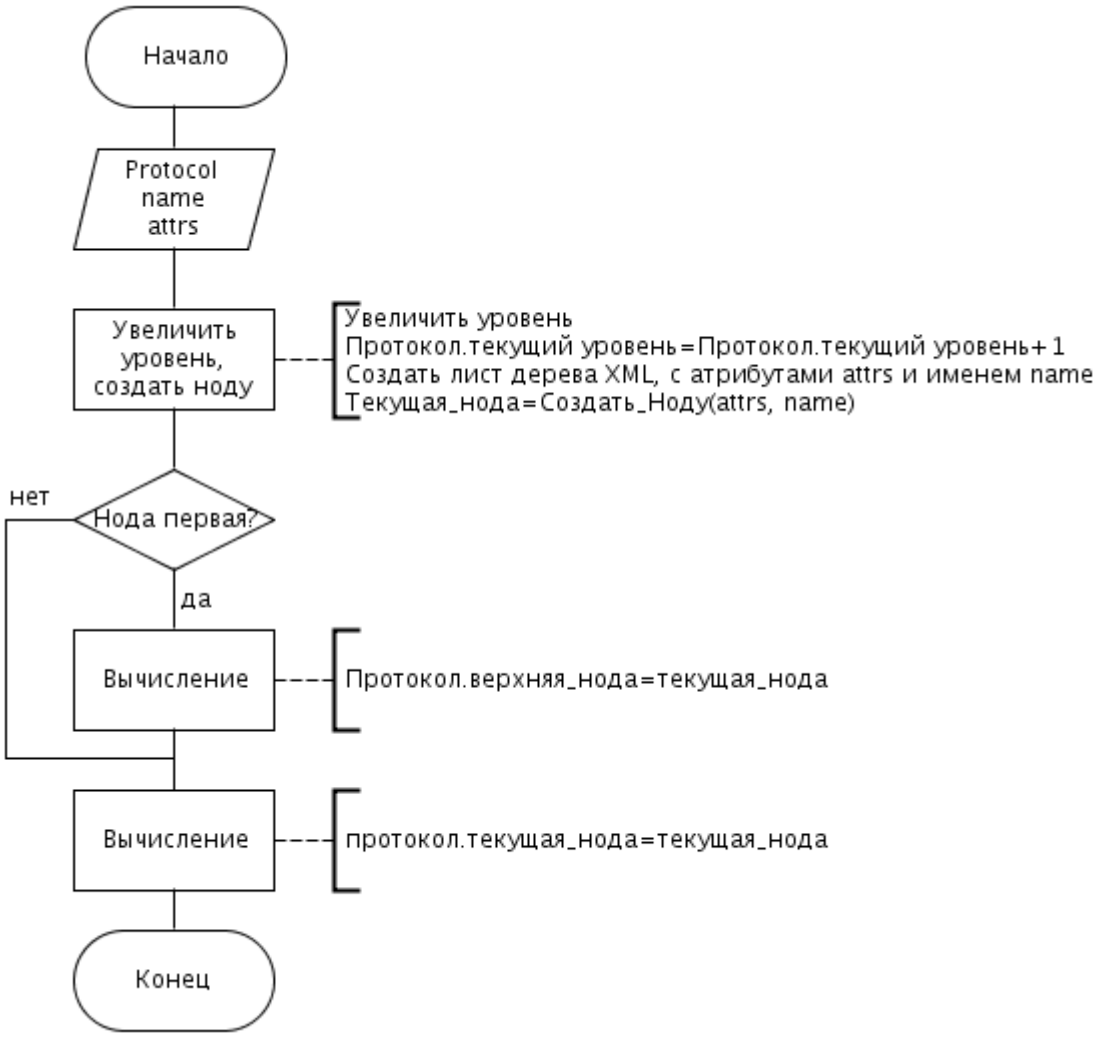

Рисунок 2.9 – Алгоритм обработки начала тега

Алгоритма обработки текстовых данных внутри тега, присваивает текстовые данные текущего тега, в текущий лист дерева. Блок-схема, алгоритма представлена далее (рис. 2.10):

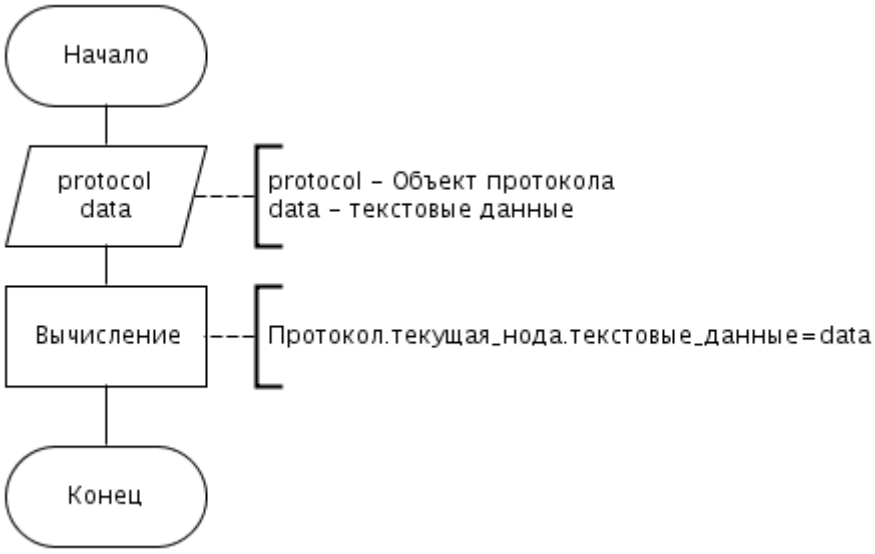

Рисунок 2.10 – Алгоритм обработки текстовых данных внутри тега

Алгоритм обработки конца тега, в качестве входных параметров принимает объект протокола на основе XML, из него извлекаются обработчики станз, а так-же текущая нода. Алгоритм выбирает первый обработчик, сравнивает подходит ли он для данной станзы, т.е. совпадает ли имя тега, его пространство имен (атрибут xmlns, являющийся уникальным для каждого типа станз) с указанными в обработчике. Если нода подходит, то он будет вызван, в новом потоке, а в качестве параметра ему будет передана копия текущей станзы. За тем будет выбран следующий обработчик, если он есть, и алгоритм повторится для него. После того как все обработчики сработают, будет уничтожена текущая станза а в качестве текущей выбран ее родитель.

Данный алгоритм представлен в блок-схеме ниже (рис. 2.11):

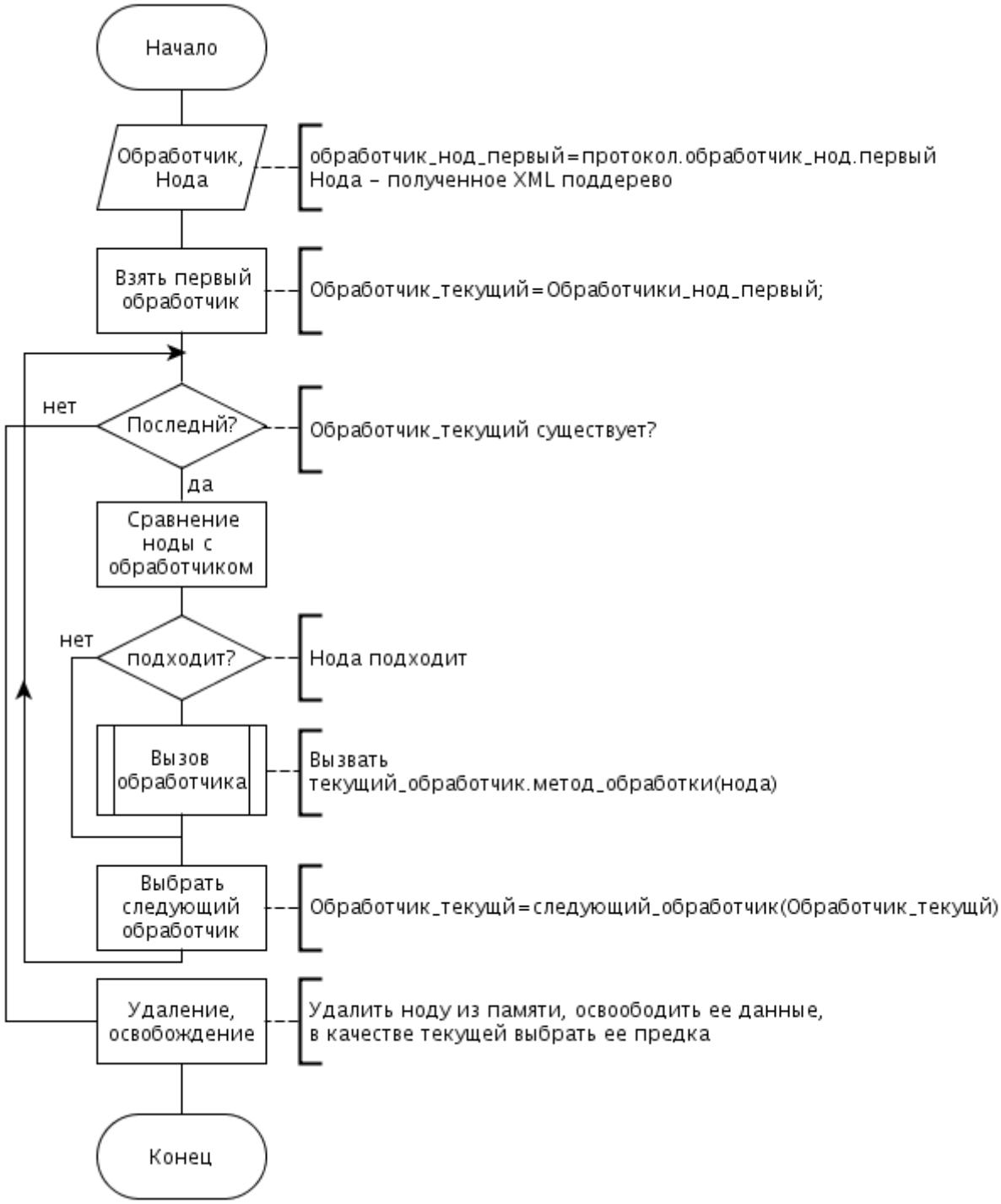

Рисунок 2.11 – Алгоритм обработки конца тега

Внутри метода обработки станз, станза будет преобразована в нужный формат, и из нее будут извлечены данные. Обработчик может отправить результат и дождаться ответа от противоположной стороны, для этого используется, атрибут id. Он должен содержаться у каждой сформированной

станзы, клиент или сервер отправляет станзу, а противоположная сторона отвечает другой станзой, у которой id совпадает с отправленной, эта станза выбирается из потока и возвращается как ответ.

2.4 Конструирование протоколов обмена

2.4.1 Протокол управления файлами

Для построения протокола можно выбрать следующие конструкции языка XML, по аналогии с уже существующим и работающим в сфере IM систем протоколом JABBER[17].

Установка соединения

После установления соединения первым начинает общение клиент отправляя серверу версию поддерживаемого протокола XML, после чего открывается тег strem, внутри которого будут находится все передаваемые клиентом запросы.

```
<?xml version="1.0"?>
<stream:stream xmlns:stream="http://etherx.jabber.org/streams" version="1.0"
xmlns="mydfs:client" to="доменное имя сервера" xml:lang="en"
xmlns:xml="http://www.w3.org/XML/1998/namespace" >
```

Затем, сервер возвращает свою версию и stream, и те возможности, которые он поддерживает, возможности планируется расширять с новыми версиям, к ним относятся метод аутентификации и поддержка сжатия и шифрования в файловой системе, которая пока не планируется.

```
<?xml version='1.0'?>
<stream:stream xmlns='mydfs:client' xmlns:stream='http://etherx.jabber.org/streams'
id='3722602580' from='доменное имя сервера' version='1.0' xml:lang='ru'>
<stream:features>
```

</stream:features>

В случае появления ошибок в ходе подключения, обнаружившая ошибку сторона должна отправить станзу следующего формата:

<stream:error>

<xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>

</stream:error>

Например сервер обнаружил ошибку разбора.

Клиент:

<message xml:lang='en'>

<body>Ошибка XML не закрыт тег body!

</message>

Сервер:

<stream:error>

<xml-not-well-formed xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>

</stream:error>

Сервер:

</stream:stream>

Аутентификация

После чего производится аутентификация клиента, если она требуется, и начинается обмен сообщениями между сервером и клиентом. Закрытие соединения производится закрытием тега stream затем противоположная сторона, должна отправить свой закрывающийся тег stream и разорвать соединение.

Клиент:

</stream:stream>

Сервер:

</stream:stream>

На данный момент тип аутентификации поддерживаемый сервером будет иметь только одно значение:

<mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

<mechanism>NONE</mechanism>

</mechanisms>

После чего клиент выбирает механизм аутентификациии отправлеет его серверу:

<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl" mechanism="NONE" />

При неудачном идентифицировании клиента возвращается станза следующего вида:

<failure xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

<not-authorized/>

</failure>

Обмен данными между сервером и клиентом

После аутентификации и установки соединения начинается обмен данными между сервером и клиентом. К этим данным относятся все операции с файлами в файловой системе. Они могут повторятся и выполняться сколько угодно раз и идти в любой последовательности. Формат запросов имеет следующий вид:

<имя_тега id='уникальный идентификатор' from='ip адрес или отправителя' to='ip адрес получателя' type='тип запроса'/>

Тип запроса может принимать следующие значения:

- set при установке каких-либо параметров, например записи данных в файл или создания директории;
- get получении каких-либо данных у противолежащей стороны,

например запрос списка файлов внутри директории;

- result при возврате ответов на set или get.

Запрос содержимого директории

Клиент запрашивает у сервера содержимое директории, для этого отправляется тег следующего содержания:

Запрос:

```
<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>
    <readdir xmlns='readdir-data' dir="путь к директории"/>
</iq>
```

Ответ:

```
<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
    <readdir xmlns='readdir-data' dir="путь к директории">
        <file filename='имя файла 1'/>
        <file filename='имя файла 2'/>
        <file filename='имя файла ..'/>
        <file filename='имя файла N'/>
    </readdir>
</iq>
```

Запрос атрибутов файлов

После запроса содержимого директории клиент запрашивает у сервера содержимое атрибутов и типов файлов, для этого отправляется тег следующего содержания.

Запрос:

```
<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>
    <getattr xmlns='getattr-data' dir="путь к файлу или папке"/>
```

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< getattr xmlns='getattr-data'/>

<mode>XXXX</mode>

<nlink>XXXX</nlink>

</getattr>

</iq>

- mode — атрибуты файла
- nlink — число ссылок на файлы

Открытие файла

Запрос:

<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>

<open xmlns='open-data' dir="путь к файлу или папке">

<mode>XXXX</mode>

<desc>XXXX</desc>

</open>

</iq>

- mode режим открытия файла принимает числовые значения например O_RDONLY;
- desc дискриптор файла.

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< read xmlns='open-data' result='XXXX'/>

</iq>

- result содержит в себе ответ об ошибке или успешном открытии. 0

успешное открытие.

Чтение файла

Запрос:

```
<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>
    <read xmlns='read-data' type='get' dir="путь к файлу или папке">
        <offset>XXXXX</offset>
        <size>XXXXX</size>
    </read>
</iq>
```

Ответ:

```
<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
    < read xmlns='read-data' type='result' result='XXXX'>
        <patckets_id>XXXX</patckets_id>
        <size>XXXXX</size>
    </read>
</iq>
```

- result принимает значение 0 в случае успеха, иначе номер ошибки;
- offset — смещение с которого будет производиться чтение;
- size в запросе: размер сколько требуется прочитать, в ответе, сколько реально будет прочитано;
- packets_id принимает номер потока, через который по бинарному протоколу будут передаваться данные.

В случае успеха клиент отправляет сообщение подтверждающее отправку, если оно получено, то начинается чтение по бинарному протоколу, иначе по истечении тайм аута файл должен быть закрыт.

```
<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
```

< read xmlns='read-data' type='sublmit' id='идентификатор потока'/>
</iq>

Запись в файл

Запрос:

<iq id='идентификатор' xml:lang='ru-RU' type='get' to='ip сервера'>
<write xmlns='write-data' type='get' dir="путь к файлу или папке">
<offset>XXXXX</offset>
<size>XXXXX</size>
</write>
</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
< write xmlns='write-data' type='result' result='XXXX'>
<patckets_id>XXXX</patckets_id>
</write>
</iq>

- result принимает значение 0 в случае успеха, иначе номер ошибки;
- offset — смещение с которого будет производиться чтение;
- size — размер сколько требуется записать;
- packets_id — номер потока через который по бинарному протоколу будут передаваться данные.

В случае успеха клиент отправляет сообщение подтверждающее отправку, если оно получено, то начинается чтение по бинарному протоколу, иначе по истечении тайм аута файл должен быть закрыт.

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
< write xmlns='write-data' type='sublmit' id='идентификатор пакетов'/>

</iq>

Закрытие файла

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>
< release xmlns='release-data' dir='путь к файлу' desc='XXXX' >
</iq>

- desc дескриптор файла.

Переименование фала

Запрос:

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>
< rename xmlns='rename-data' dirfrom='путь к файлу на данный момент'
dirto='новый путь к файлу' >
</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>
< rename xmlns='rename-data' result='ответ'>
</iq>

- result номер ошибки операции.

Удаление файла

Запрос:

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>
< unlink xmlns='unlink-data' dir='путь к файлу на данный момент' >
</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< unlink xmlns='unlink-data' result='ответ'>

</iq>

- result номер ошибки операции.

Удаление директории

Запрос:

<iq from='ip сервера' to='ip клиента' type='set' id='идентификатор'>

< rmdir xmlns='rmdir-data' dir='путь к файлу на данный момент' >

</iq>

Ответ:

<iq from='ip сервера' to='ip клиента' type='result' id='идентификатор'>

< rmdir xmlns='unlink-data' result='ответ'>

</iq>

- result номер ошибки операции.

2.4.2 Бинарный протокол, обмена содержимым файлов

Бинарный протокол используется для чтения и записи файлов. Он содержит заголовок и тело. В заголовке находятся информация о содержимом. В теле само содержимое файла. Нужно сделать учет того, что будут новые поля, таким образом имеем следующую структуру:

- ID заголовка 16Bit, беззнаковое целое;
- размер заголовка 16Bit, беззнаковое целое;
- номер заголовка в последовательности 32Bit, беззнаковое целое;
- размер тела 32Bit, беззнаковое целое;
- тело.

Данные файлов передаются блоками, заранее заданного размера.

2.5 Построение конфигурационного файла

В результате анализа полученной информации можно использовать следующую структуру конфигурационного файла.

2.5.1 Конфигурационный файл сервера

```
conf{
* сеция глобальных данных как сервера так и клиента
  global_data{
    extension_dir="<путь к папке с расширениями>";
    module_dir="<путь к папке с модулями>";
  }
* секция данных индивидуальных для сервера
  server_data{
    server_ip=<IP на котором ожидается подключение клиентов>;
    bibary_protocol_ip=<предпочтительный IP для бинарного протокола>;
    server_port=<Порт для подключения конечных клиентов>;
    shared_dir="<путь к папке с общедоступными данными>"
  }
* секция в которой будут храниться данные расширений и модулей
  module_data{
    <название модуля>{
      <название параметра>=значение параметра;
    }
  }
}
```

2.5.2 Конфигурационный файл клиента

```
conf{
* сеция глобальных данных как сервера так и клиента
  global_data{
    extension_dir="<путь к папке с расширениями>";
    module_dir="<путь к папке с модулями>";
  }
* секция данных индивидуальных для клиента
  client_data{
* секции серверов к которым подключается данный клиент.
    servers{
      server{
        server_ip=<"IP адрес сервера">;
        server_port=<"порт сервера">;
      }
    }
  }
* секция в которой будут храниться данные расширений и модулей
  module_data{
    <название модуля>{
      <название параметра>=значение параметра;
    }
  }
}
```

2.6 Вывод

В результате алгоритмического конструирования, решено выбрать в

качестве базового протокола для обмена метаданными протокол XML и расширить его таким образом, чтобы можно было реализовать функции файловой системы. Разработаны принципы обмена данными между сервером и клиентом, протокол обмена метаданными и построены конструкции XML. Описан бинарный протокол для передачи содержимого файлов. Разработана структура создаваемой системы, и связи между программными модулями. Так же разработана структура конфигурационного файла для сервера и клиента. Разработаны основные алгоритмы реализации взаимодействия клиентов и серверов между собой.

3 ПРОГРАММНОЕ КОНСТРУИРОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ

3.1 Выбор XML парсера

Так как решено использование формата представления данных XML в качестве базового формата для обмена запросами операций между сервером и клиентом, то необходимо выбрать подходящий XML анализатор. Для этого рассмотрены следующие анализаторы:

3.1.1 LibXML2

Libxml2 это анализатор XML на языке C, разработанный для проекта Gnome (но используется вне платформы Gnome), это бесплатное программное обеспечение доступно в соответствии с лицензией MIT.

Libxml2 это очень легко переносимая библиотека, ее можно собрать и использовать без особого труда, на большом числе вариантов операционных систем. К ним относятся: Linux, Unix, Windows, CygWin, MacOS, MacOS X, RISC Os, OS/2, VMS, QNX, MVS, VxWorks, и.т.д. Библиотека используется в модуле Jabber клиента Pidgin[18].

3.1.2 QtXML

Модуль QtXml обеспечивает работу с потоками чтения и записи XML документов и реализацию их в форме SAX и DOM. Однако, требует наличие библиотеки QT для создания приложений использующих его.

3.1.3 Expat

Expat это библиотека для синтаксического анализа XML, написанные на языке C. Это потоко-ориентированный анализатор, в котором приложение регистрирует обработчики событий для элементов, которые анализатор может найти в XML документе (например начало тега). Данный анализатор

поддерживается во многих языках, таких как Erlang, Ocaml, Objective-C, Python, Simkin, Ruby и др. С применением данного анализатора написаны такие проекты как Ejabberd - jabber сервер, и xmpppy - библиотека протокола XMPP для языка Python[19].

3.1.4 Решение

Наиболее подходящим для данной задачи является анализатор Expat, он успешно применяется в таком крупном проекте как сервер Ejabberd компании ProcessOne, для анализа потокового протокола XMPP системы Jabber. Протокол разрабатываемой файловой системы схож по своей структуре с протоколом xmpp, и применение данного анализатора вполне оправдано.

3.2 Выбор библиотеки для построения конфигурационного файла

Для сохранения конфигурации программы требуется создать конфигурационные файлы. Для этой цели целесообразно применить библиотеку XFlib[20]. XF (eXchange Format) - это универсальный, легкий и переносимый формат представления данных в текстовом виде, который просто воспринимается человеком и обрабатывается программами. XF значительно более гибок и лаконичен, чем XML. Он полностью поддерживает стандарт Unicode и может применяться для создания приложений на различных языках.

3.3 Модули разрабатываемых приложений, и иерархия классов

В результате проектирования исходная задача была разбита на несколько подзадач. В результате программного конструирования было реализовано 47 классов и структур данных. Для каждой подзадачи можно представить свою независимую иерархию классов. Общую схему функционирования сервера можно представить следующим образом(рис. 3.1):

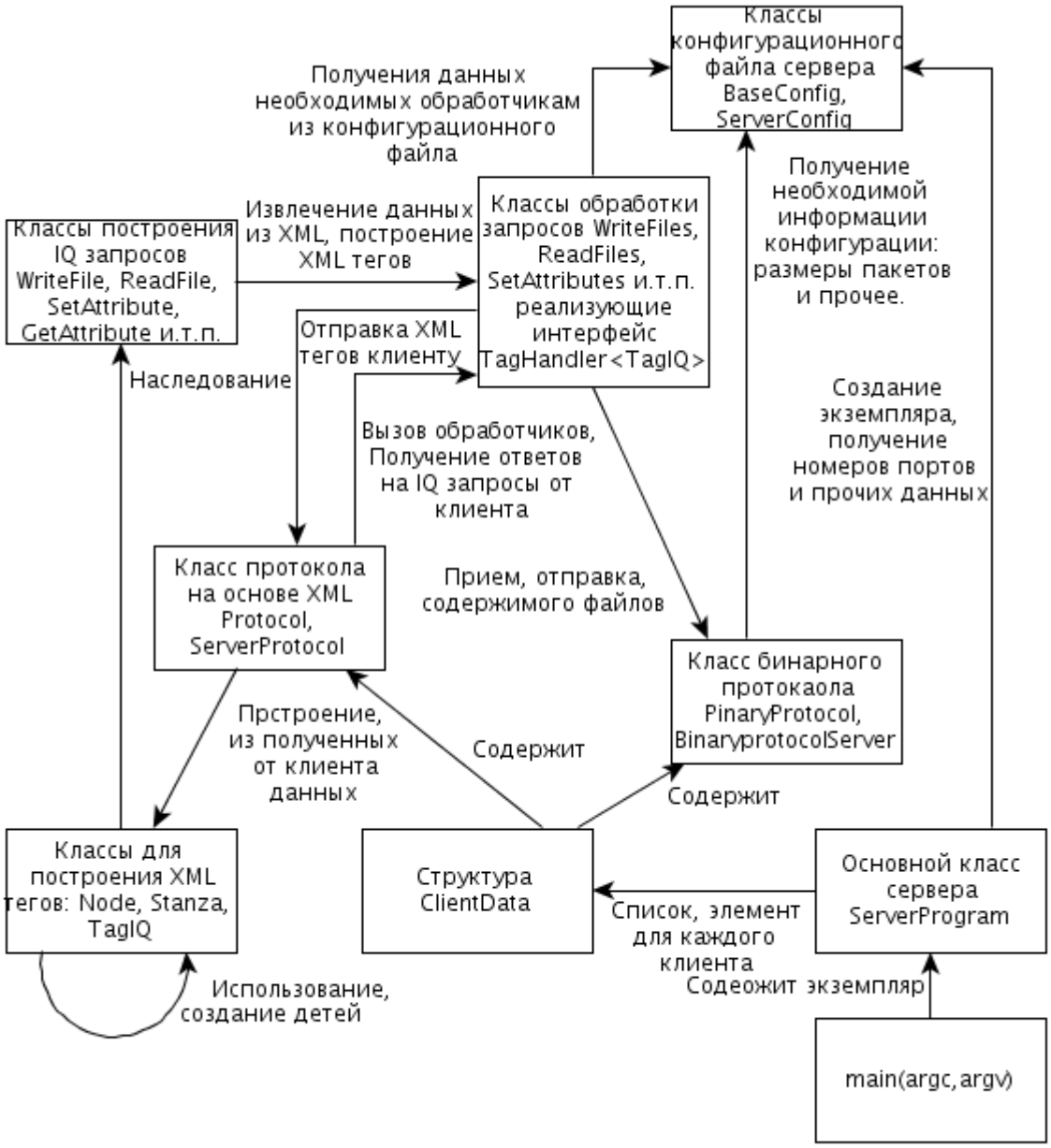

Рисунок 3.1 – Схема взаимодействия классов сервера

Классы построения IQ запросов, и классы обработчики запросов, относятся к модулю расширения сервера LibFileOperations, и могут быть добавлены в последствии, в зависимости от требуемых нужд.

Схема взаимодействия классов клиента, представлена на рисунке ниже (рис. 3.2):

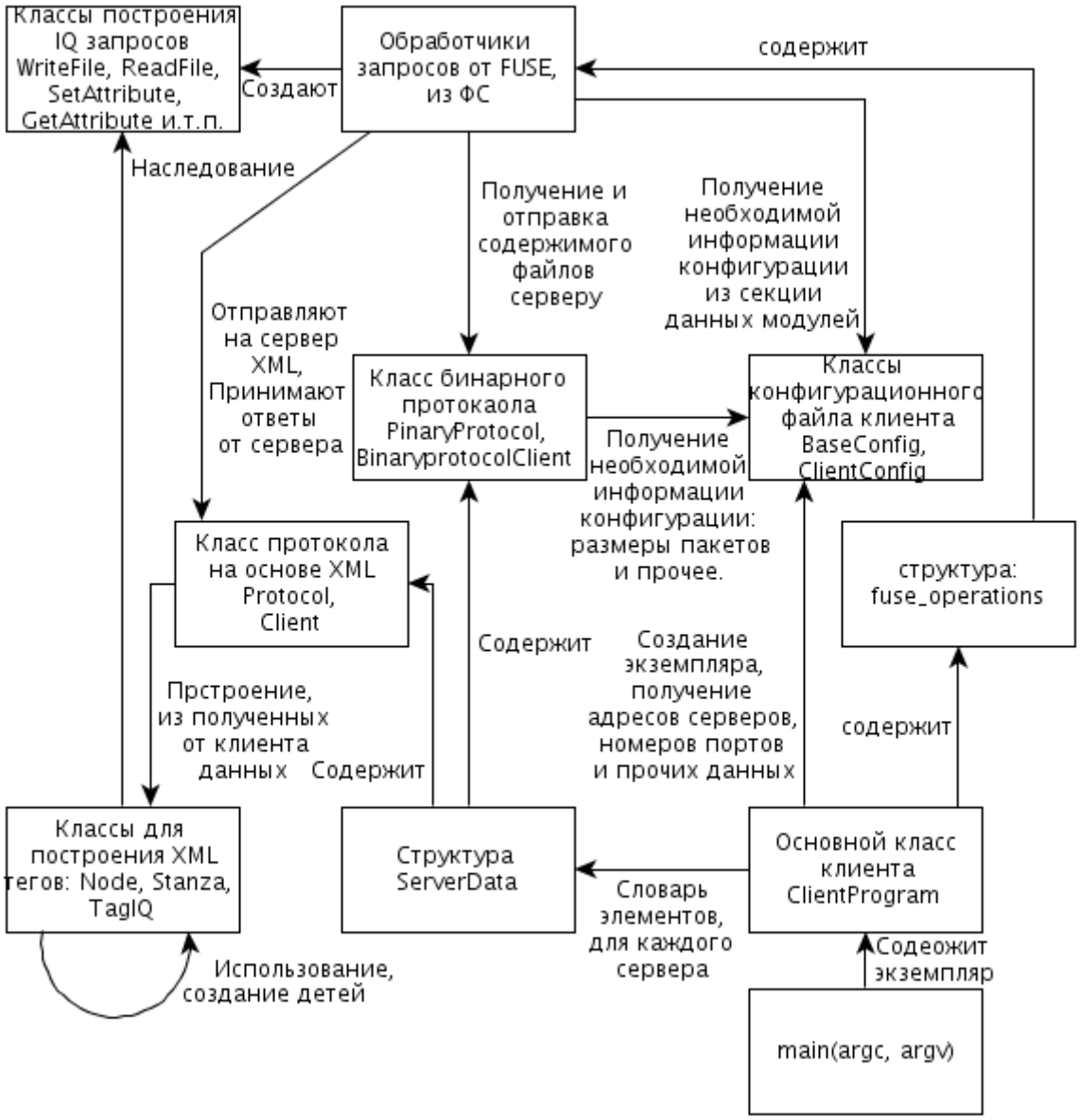

Рисунок 3.2 – Схема взаимодействия классов клиента

Здесь обработчики запросов от FUSE и классы IQ запросов, являются частью модуля расширения клиента, который может быть расширен, для требуемых нужд.

3.3.1 Классы формирования XML тегов

Протокол на основе XML, состоит из 18 классов, необходимых для построения XML тегов, и извлечения из них данных. Они представлены ниже (рис 3.1)

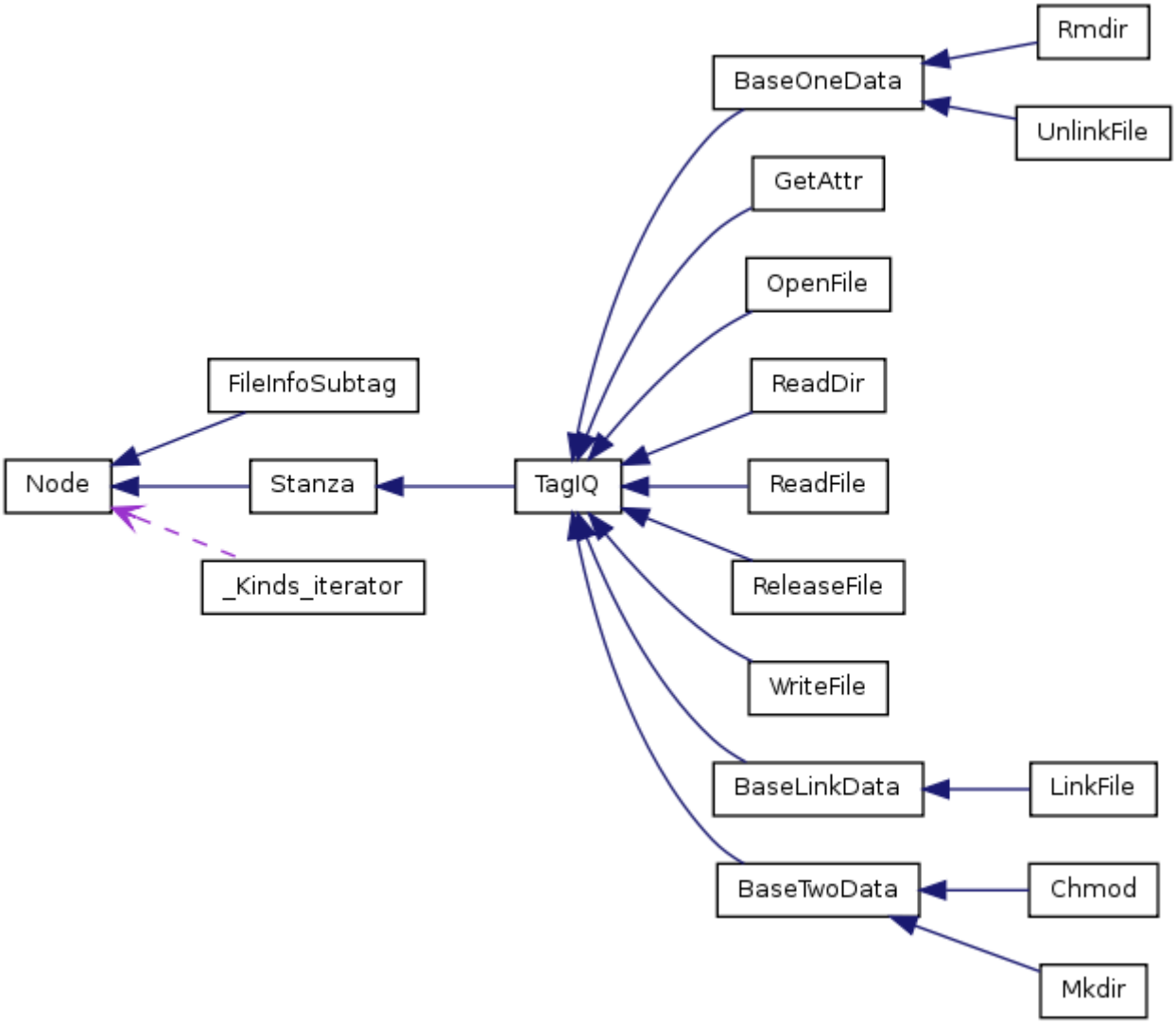

Рисунок 3.1 – Схема наследования классов реализующих построение XML

Они реализуют следующие подзадачи:

- Node - описывает единичный элемент XML дерева;
- _Kinds_iterator — описывает интерфейс для перебора элементов XML дерева;
- Stanza — описывает полностью сформированный тег, который может быть отправлен по сети, в нем есть атрибуты, такие как идентификатор отсылающей стороны, идентификатор принимающей, уникальный идентификатор станзы;
- TagIQ — реализует протокол обмена в формате «запрос-ответ» клиент либо сервер могут друг у друга запросить определенные действия,

например «открыть файл» в ответ на что сервер должен проанализировать возможность открытия и вернуть результат.

Следующие классы реализуют сам обмен данными:

- GetAttr — реализует тег, который формирует запрос на получение атрибутов файла. В нем реализованы методы задания имени файла и его атрибутов, получения их из XML формата;
- OpenFile — реализует тег открытия файла, он формирует запрос на открытие файла сервером, и ответ о результате открытия;
- ReadDir — реализует тег запроса содержимого директории, формирует запрос и возвращает ответ со списком файлов и каталогов;
- ReadFile — формирует запрос на чтение файла, из клиента в определенной позиции, возвращает идентификатор пакетов необходимых для отправки данных по бинарному протоколу, объем данных которые можно прочесть;
- ReleaseFile — формирует запрос на освобождение файла. Когда на клиентской программе происходит закрывание файла она отправляет тег и сервер освобождает связанные с ним ресурсы, возвращая подтверждение;
- WriteFile — формирует запрос на запись файла, в запросе передается позиция для записи, объем данных для записи и имя файла, а так-же возвращается, объем сколько данных удалось записать, и идентификатор пакета для передачи через бинарный протокол.

В результате анализа классов было выявлено что часть из них передают всего один или два или три параметра, число или строку и возвращают только результат. Так например для создания директории требуется передать путь к создаваемой папке и режим (права доступа), и вернуть ответ- целое число свидетельствующее об успешном завершении т.е. 0, либо об ошибке, значение —

errno, в то время как для изменения прав на файл потребуется передать те же данные. Имя файла, права доступа и вернуть ответ об успехе операции. По этой причине был выделен класс BaseOneData который реализует, общие действия, общие для таких классов. Классы BaseLinkData, BaseTwoData, BaseOneData выведены из одного кода, через макросы. От них наследованны следующие классы:

- RmDir — описывает тег с именем файла, необходимым для удаления директории;
- UnlinkFile — описывает тег с именем файла для уничтожения жесткой ссылки на файл (удаления файла);
- LinkFile — описывает тег с именеами двух файлов и атрибутами, для создания жесткой ссылки на файл;
- Chmod — описывает тег, с именем файла и режимом, для изменения режима доступа к файлу;
- Mkdir — формирует тег с именем и режимом доступа директории для создания.

Так же было выявлено, что атрибуты файла, реализуемые структурой fuse_file_info передаются в некоторых случаях, как например при открытии файла, при чтении и при удалении файла, и данный тег был вынесен в класс FileInfoSubtag, который описывает под элемент запроса.

3.3.2 Классы обработки станз протокола на основе XML

Для обработки запросов с серверной стороны была организована следующая иерархия классов, она содержит 12 классов и один интерфейс (рис.3.2):

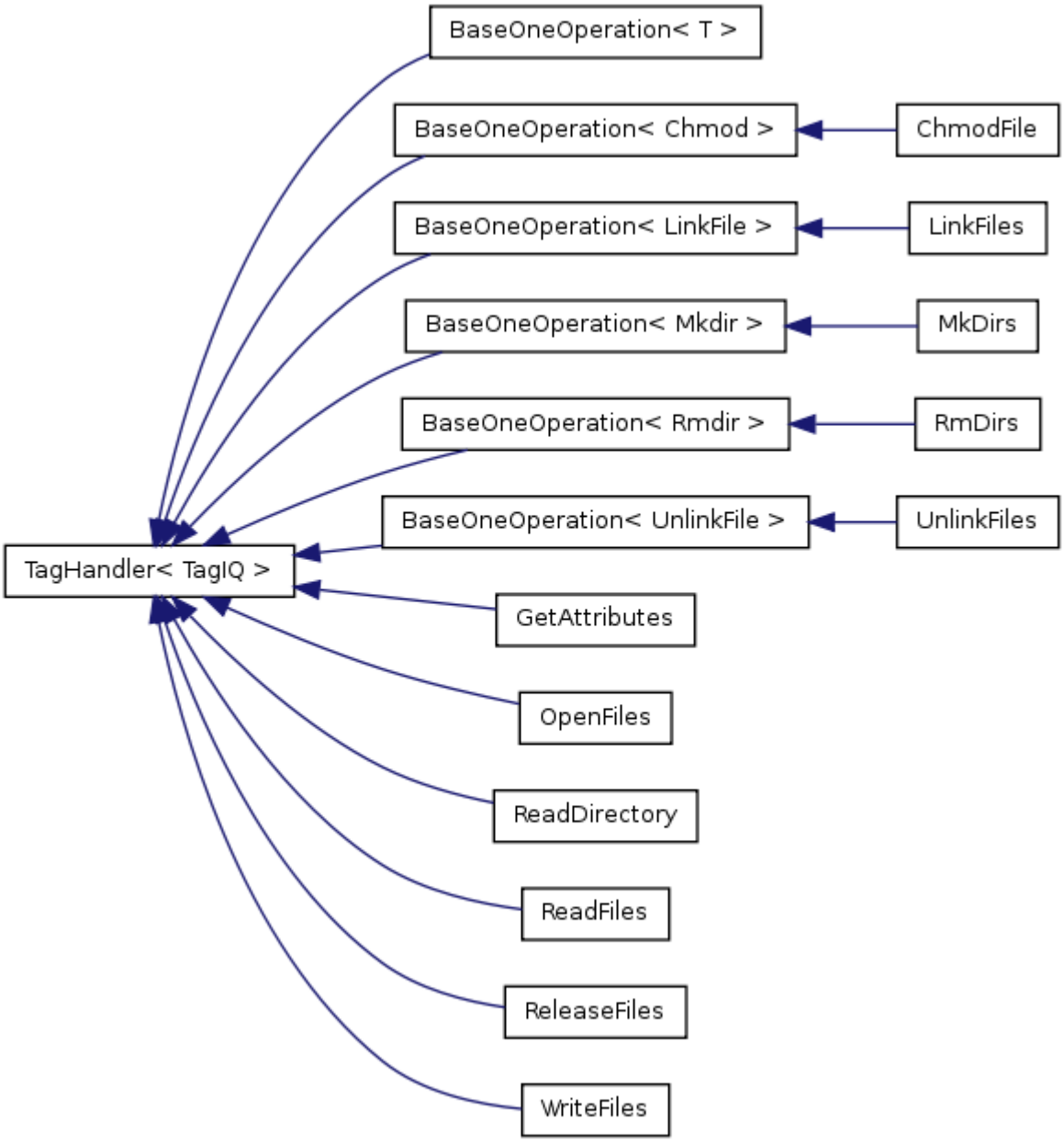

Рисунок 3.2 – Схема наследования классов обработки IQ запросов.

Здесь интерфейс TagHandler регистрируется внутри класса Protocol для обработки запросов приходящих с противоположной стороны. Каждая пришедшая подходящая станза, зарегистрированная для обработки будет вызывать метод handleTag(T &tag), внутри объекта который зарегистрирован для ее обработки. Остальные классы образуют библиотеку операций над файлами зарегистрированную в сервере. Классы наследованные от BaseOneOperaion реализуют обработку с танз, которые сходны друг с другом,

только выполняют действия над файловой системой. Они реализуют всего один абстрактный метод virtual int fileFunc(T* filetag); который производит операцию над фс, и возвращает результат в виде целого числа, содержащего значение -errno либо 0 в случае успеха. Классы выполняют следующие действия:

- ChmodFile — обработка станзы реализуемой классом Chmod, изменение атрибутов файла;
- LinkFiles — обработка станзы реализуемой классом LinkFile, создание жестких ссылок;
- MkDirs — обработка станзы реализуемой классом Mkdir, создание директорий;
- RmDirs — обработка станзы реализуемой классом Rmdir, удаление директорий;
- UnlinkFiles — обработка станзы реализуемой классом UnlinkFile, удаление жестких ссылок;
- GetAttributes — обработка станзы реализуемой классом GetAttr, получение атрибутов файла;
- OpenFiles — обработка станзы реализуемой классом OpenFile, открытие файлов;
- ReadDirectory — обработка станзы реализуемой классом ReadDir, чтение содержимого директорий.
- ReadFiles — обработка станзы реализуемой классом ReadFile, чтение содержимого файлов, и передача его по бинарному протоколу.
- ReleaseFiles — обработка станзы реализуемой классом ReleaseFile, освобождение ресурсов открытых файлов, проверка возможности открытия;
- WriteFiles — обработка станзы реализуемой классом WriteFile, Запись файлов на диск, прием данных по бинарному протоколу.

3.3.3 Классы протокола на основе XML

Для передачи и приема сформированных XML данных по сети и вызова обработчиков, была реализована следующая иерархия классов(рис. 3.3):

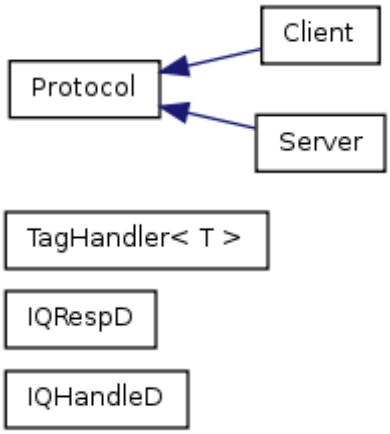

Рисунок 3.3 – Классы и структуры данных протокола на основе XML.

Тут реализовано 4 класса и 2 структуры данных.

- Protocol - базовый класс для сервера и клиента. В нем реализованы методы для резистрации обработчиков IQ запросов, и обработчиков запросов произвольных тегов. Он инициирует начальный тег, запускает отдельный поток для обработки данных, и организует всю обработку XML тегов. Наиболее важными методами в классе являются:

  void addTagHandler(TagHandler<Node>::PTagHandler handler, string name="", string xmlns="");

  void addIQHandler(TagHandler<TagIQ>::PTagHandler handler, string xmlns="", string name=QUERY_TAG);

  void removeIQHandler(TagHandler<TagIQ>::PTagHandler handler);

  Они принимают в качестве параметра, класс реализующий TagHandler<T> и вызывают обработчик, xmlns — пространство имен XML тега, который нужно обрабатывать, name — имя тега;
- TagHandler<T> интерфейс реализующий обработку тегов, все обработчики должны его реализовывать;

- IQRespD и IQHandleD — внутренние структуры, реализующие обработку тегов;
- Client — реализует подключение к серверу, кдасс содержит метод

  void connect_server(string address, u_int16_t port);

  Он организует подклчение к серверу, address это адрес сервера в сети, а port это порт на котором сервер ожидает подключение клиента.

3.3.4 Классы бинарного протокола

Бинарный протокол состоит из трех классов, и трех структур данных(рис. 3.4):

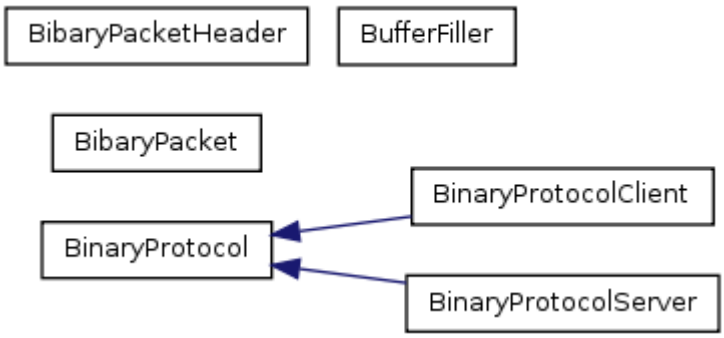

Рисунок 3.4 – Классы и структуры данных бинарного протокола.

Данные классы используются для реализации, чтения и записи файлов. В них реализованы следующие подзадачи:

- BinaryProtocol - Описывает сам протокол обмена между клиентом и сервером в оба направления. Он содержит общие как для клиента так и для серверной части методы, для передачи и приема данных через пакеты. В нем реализованы методы отправки и получения данных. Основными внешними методами для него являются:

  void setFillBuffer(void* buffer,size_t size,unsigned long int pktID);

  Он ставит в ожидание заполнения данными некоторого буфера

  buffer — адрес памяти куда загружаются принимаемые данные

  size — размер, сколько данных необходимо принять

pktID — идентификатор пакетов из, которые будут заполняться в буфер.

void waiteFillBuffer(unsigned int pktID);

Останавливает вызывающий поток, до того момента пока не заполнится буфер. pktID — идентификатор пакета, данные которого заполняются в буффер.

void setSendBufferData(const void* buffer,size_t size,unsigned long int pktID);

Отправляет данные.

buffer — буфер с данными для отправки;

size — размер отправляемых данных;

pktID — идентификатор пакетов, в которых будут отправлятся данные.

startTread(int skt_id); - метод который запускает поток передачи данных.

skt_id — идентификатор сокета, через который будут передаваться данные.

void disconnect(){} - Разрыв соединения, и закрытие сокета;

- BinaryProtocolClient — описывает метод необходимый для подключения к серверу void connect_server(string address, unsigned short port); где, address — адрес сервера в сети либо доменное имя, а port — порт на котором сервер ожидает подключение клиента, через который будем подключаться;
- BinaryProtocolServer — организует ожидание подключения клиента;
- BinaryPacket — Описывает содержимое одного пакета данных, состоящего из заголовка и тела с самими данными;
- BinaryPackeHeader — заголовок пакета передаваемого по сети.

- BufferFiller — структура используемая для организации ожидания заполнения буфера.

Данные через бинарный протокол передаются внутри структур BibaryPacketHeader и BibaryPacket.

Заголовок пакета бинарного протокола, структура BibaryPacketHeader описана следующим образом:

```
#pragma pack(push,1)
struct BibaryPacketHeader{
    unsigned long int pktID;
    Идентификатор передаваемого пакета.
    unsigned short headSize;
    Размер заголовка.
    size_t bodySize;
    Размер тела.
};
#pragma pack(pop)
    Структура описывающая бинарный пакет:
struct BibaryPacket{
    BibaryPacketHeader header;
    void * body;
};
```

Экземпляр класса BinaryProtocolServer создается в сервере для каждого подключенного к серверу клиента, а в каждом подключенном к серверу клиенте создается экземпляр BinaryProtocolClient, для каждого сервера. Оно передается во все модули внутри расширений сервера и клиента соответственно.

3.3.5 Классы чтения конфигурационного файла

Конфигурационный файл, построен с использованием библиотеки Xflib. Базовый класс, открывает файл и читает данные совпадающие с конфигурационным фалом сервера и клиента, инициализирует необходимые поля. Клиентский, и серверный класс наследуются от него, и читают свои части. Базовый класс реализован в модуле BaseConfigFile.h, серверный ServerConfigFile.h, а клиентский в ClientConfigfile.h

Схема наследования классов чтения конфигурационного файла представлена следующим образом (рис. 3.7):

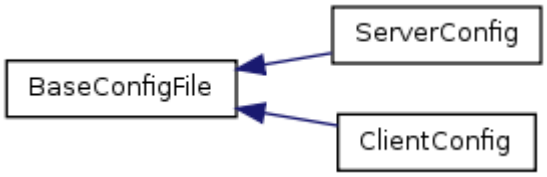

Рисунок 3.7 – Схема наследования классов чтения конфигурационного файла.

Базовый класс определен следующим образом:

```
class BaseConfigFile{
protected:
    xfMap *conf;
    xfNode *root;
   xfNode *moduledata_node;
public:
    long int getAttrInt(xfNode* node, long int defval=-1);
    string getAttrString(xfNode* node, string defval="");
    long int xfCharToInt(xfChar* str);
    string xfCharToStr(xfChar* str);
     long int getModuleAttributeInt(wstring modulename,wstring attr,long int defval
```

);

Получение атрибута модуля, modulename — имя модуля, это секция в конфигурационном файле где модуль должен сохранять свои данные, attr — имя атрибута который необходимо получить, defval — значение по умолчанию, оно будет передано функцией если атрибут отсутствует.

string getModuleAttributeString(wstring modulename,wstring attr, string defval); Аналогичен getModuleAttributeInt только возвращает атрибут- строку в не число.

xfNode* getMofuleAttribute(wstring modulename, wstring attr);

modulename — имя модуля, attr — имя атрибута, возвращает ноду на указанный атрибут.

BaseConfigFile(){};

BaseConfigFile(char* filename);

Конструктор, в качестве параметра принимает имя конфигурационного файла.

char* getModulesDir();

Возвращает директорию с модулями.

char* getExtendsDir();

получаем директорию с расширениями

xfNode* getModuleDataNode() {return moduledata_node;};

Возвращает ноду, которая указыват на секцию где все модули сохраняют свои данные.

virtual ~BaseConfigFile();

Деструктор закрывает конфигурационный файл

};

Класс конфигурационного файла сервера

Этот класс определен в модуле ServerConfigFile.h он содержит

возможности получения уникальных для сервера данных из конфигурационного файла.

```
class ServerConfig : public BaseConfigFile
{
protected:
    xfNode* server_node;
    string shareddir;
    unsigned short serverPort;
    unsigned short serverBinPort;
    string serverIP;
public:
    string getIP(){return serverIP;}
```

IP адрес через который сервер будет ожидать подключение коиентов

```
    unsigned short getServerPort(){return serverPort;}
```

получение порта через который сервер будет ожидать подключения к протоклу на основе XML

```
    unsigned short getServerBinPort(){return serverBinPort;}
```

получение порта через который сервер будет ожидать подключения по бинарному протоколу

```
   string getSharedDir(){return shareddir;};
```

Директория в которой хранятся файлы доступные через данный сервер ФС

```
    ServerConfig();
    virtual ~ServerConfig();
};
```

Класс конфигурационного файла клиента

Этот класс определен в модуле ClientConfigFile.h он содержит

возможности получения уникальных для клиента данных из конфигурационного файла.

В нутри данного заголовочного файла так-же определена структура:

```
struct ServersConfStruct{
   unsigned int bin_port;
   unsigned int xml_port;
   string server_address;
};
```

Она содержит порт бинарного протокола и протокало на основе XML а так-же адрес каждого сервера, к которому должен быть подключен клиент.

Сам класс конфигурационного файла клиента описан следующим образом:

```
class ClientConfigFile: public BaseConfigFile {
protected:
   xfNode* client_node;
   xfNode* servers_node;
public:
   map<unsigned int,ServersConfStruct>  getServers();
```

Метод возвращает словарь структур ServersConfStruct, ключом является целое число, соответствующее порядковому номеру записи в конфигурационном файле. В структуре содержатся порты и адреса серверов к которым должен быть подключен клиент.

```
   ClientConfigFile();
   virtual ~ClientConfigFile();
private:
};
```

3.3.6 Основные классы сервера

Группа класс и структура которые организует основную часть сервера, не имеет наследования(рис. 3.5).

Рисунок 3.5 – Классы организующие основную часть сервера.

Класс ServerProgram является основным в серверной части программы. Он содержит конструктор внутри которого выполняется основной цикл программы. Он содержит поле ServerConfig* sc; Выполняет конструктор конфигурационного файла, считывая из него номера портов для бинарного протокола и протокола на основе XML на котором расположен сервер. В методе int ServerProgram::createSocket(uint16_t port) он создает новое сокетное соединение, через которое он будет обмениваться с клиентами данными. Затем запускает цикл, ожидания подключения клиента. Для этого используется метод waiteSocketConnection(i32SocketFD);. После того как клиент подключился по двум протоколам создается поток, который выполняет обработку данного клиента, внутри функции void* client_thread_func(void* data). Параметр data содержит структуру clientData с двумя идентификаторами сокетов. После чего Создаются экземпляры классов бинарного протокола BinaryProtocol и класса Server, которые выполняют обработку запросов от клента. Затем регистрируются обработчики модулей registerModule(Server* protocol, ServerConfig* sc, BinaryProtocol* bp) здесь передается класс бинарного протокла, и протокола на основе XML. Функция возвращает уникальный идентификатор зарегистрированного клиента модуля. Он используется внутри модулей. Данный идентификатор будет передан при отключении клиента, для того чтобы в модуле можно было удалить данные относящиеся именно к этому клиенту. Затем, выполняются методы которые запускают обработку клиента

bs.startTread(i32ConnectFDb); Инициирует поток внутри бинарного протокола, для обработки данных от пользователя, serv.parserCreate(i32ConnectFD); создает такой-же поток для обработки данных по протоколу XML serv.startParsing(); - запускает обработку тегов поступающих на сервер через протокол на основе XML, этот метод завершится только тогда когда клиент завершит свое соединение. На этом этапе вызовется функция unregisterModule(modid); внутри модуля, которая освободит данные связанные с модулем, и убрать обработчики тегов в сервере.

3.3.7 Классы расширений сервера

Интерфейсную часть модулей расширений сервера представляют две функции:

unsigned int registerModule(Server* protocol, ServerConfig* sc, BinaryProtocol* bp);

void unregisterModule(unsigned int mid);

В на данный момент все действия над файлами реализованы в модуле LibFileOperation, он выполняет базовые операции с файлами, такие как открытие, чтение, запись, просмотр содержимого каталога, создание и удаление ссылок на файлы, открытие файла и другие. Модуль содержит программный модуль main.cpp реализующий данные функции.

Структура moduleStruct:

```
struct moduleStruct{
   ReadDirectory* rd;
   GetAttributes* ga;
   OpenFiles* of;
   ReleaseFiles* rlf;
   ReadFiles* rf;
```

```
    WriteFiles* wf;
    ChmodFile* cm;
    UnlinkFiles* ul;
    LinkFiles* lf;
    RmDirs* rmd;
    MkDirs* md;
};
```

Содержит классы обработчиков действий над файлами. Словать static map<unsigned int,moduleStruct> modules; содержит набор этих структур, для каждого подключенного клиента. Ключом словаря является уникальный идентификатор клиента, который генерируется в функции registerModule, и возвращается в основную программу.

3.3.8 Основной модуль клиента.

Основной модуль клиента, реализуется следующими классами и структурами (рис. 3.7)

Рисунок 3.6 – Классы и структуры основной части клиента.

Класс ClientProgramm реализует основной цикл работы клиента. Полями данного класса являются:

- ClientConfigFile cf — экземпляр класса конфигурационного файла клиента.
- ServerDataMap serversData — словарь, ключем его является номер сервера в конфигурационном файле, а значением элемент структуры ServerData.

Структура ServerData описана следующим образом:

```
struct ServerData{
```

```
    Client* cl;
    BinaryProtocolClient* clbin;
};
```

Она содержит указатели на экземпляр класса Client и BinaryProtocolClient. Клиент одновременно подключается к нескольким серверам, и сохраняет указатели на созданные экземпляры классов в словаре.

В конструкторе класса ClientProgram, получается список серверов из конфигурационного файла к которым надо подключиться. Затем производится перебор серверов в цикле. Для каждого из них создаются экземпляры классов Client и BinaryProtocolClient:

```
ServerData sd;
sd.cl = new Client();
sd.clbin = new BinaryProtocolClient(&cf);
```

Потом происходит подключение клиента к серверу, по бинарному протоколу и по протоколу на основе XML:

```
sd.cl->connect_server(it->second.server_address,it->second.xml_port);
sd.clbin->connect_server(it->second.server_address,it->second.bin_port);
```

После этого регистрируются модули, которые будут обрабатывать поток данных приходящих из FUSE. Каждый модуль должен содержать функцию:

```
void registerModule(ServerDataMap* servsData, fuse_operations* clientFsoperations,ClientConfigFile* cf);
```

В нее передается словарь serversData, содержащий объекты протоколов, указатель на структуру fuse_operation — в него модули заполняют обработчики запросов файловой системы, и cf — конфигурационный файл клиента.

После регистрации модулей, запускается основной цикл FUSE вызовом метода fuse_main(argc, argv, &clientFsoperations, NULL);

3.3.9 Модуль расширения клиента ClientFSOperation

Расширяемый модуль клиента ClientFSOperation представлен следующими классами и структурами данных:

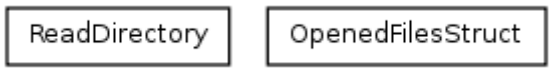

Рисунок 3.6 – Классы и структуры модуля расширения клиента.

Основную часть модуля составляет класс ReadDirectory. Он реализован в виде Singleton класса, имеет приватный конструктор и метод getInstance который возвращает указатель на экземпляр класса. При регистрации модуля, в функции registerModule метод getInstance вызывается в первый раз. Он имеет следующий интерфейс:

static ReadDirectory* getInstance(ServerDataMap* servsData=NULL, fuse_operations* clientFsoperations=NULL,ClientConfigFile* cf=NULL);

Ему передаются те-же параметры что и в функцию registerModule. В случае если экземпляра еще нет он будет создан и вызван конструктор, иначе будет возвращен указатель на уже существующий экземпляр класса. В конструкторе класса присваиваются указатели на функции которые будут выполнять операции над файлами, в виртуальной ФС, в структуру clientFsoperations.

Для выполнения операций над фс, реализованны следующие функции:

static int hello_write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fi) – запись данных в файл;

static int hello_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fi) – чтение данных из файла;

int get_f_attr(Protocol* proto, const char* path,struct stat& attr) – получение атрибутов файла;

static int hello_release(const char *path, struct fuse_file_info *fi) – освобождение файла;

static int hello_open2(const char *path, struct fuse_file_info *fi,mode_t mode=0) – Т.к. функция открытия файла и системный вызов creat похожи по своему назначению, то создана одна общая для них функция;

static int hello_open(const char *path, struct fuse_file_info *fi) – открытие файла;

int hello_create (const char *path, mode_t mode, struct fuse_file_info *fi) – создание файла, системный вызов creat над виртуальной ФС;

static int hello_readdir(const char *path, void *buf, fuse_fill_dir_t filler,off_t offset, struct fuse_file_info *fi) – чтение содержимого директории;

static int hello_getattr(const char *path, struct stat *stbuf) – получение атрибутов файла.

Часть операций очень похожи при работе над ФС, на сервер уходят одни и те-же данные. На серверной части они выделены как BsaeOneData, на клиентской стороне они реализованы шаблонной функцией OneDataFunc, и перегруженной функцией OneDataFunc_ заполняющей атрибуты тегов:

inline void OneDataFunc_(LinkFile* bd,string p1, string p2)

inline void OneDataFunc_(UnlinkFile* bd,string p1,mode_t p2)

inline void OneDataFunc_(Rmdir* bd,string p1,mode_t p2)

inline void OneDataFunc_(Chmod* bd,string p1, mode_t p2)

inline void OneDataFunc_(Mkdir* bd,string p1, mode_t p2)

Шаблонная функция создает тег типа X и задет параметры, в зависимости от типа Y, вызывая перегруженную функцию OneDataFunc_.

template <class X, class Y=int>

int OneDataFunc(const char *path,string resXMLNS,Y mode=Y())

Функция OneDataFunc используются в следующих функциях:

int hello_mkdir (const char *path, mode_t mode) – создание директории;

int hello_chmod (const char *path, mode_t mode) – изменение режима

доступа;

int hello_rmdir (const char *path) – удаление директорий;

int hello_unlink (const char *path) – уничтожение жестких ссылок на файлы;

int hello_link (const char *path_from, const char *path_to) – создание жестких ссылок на файлы.

3.4 Выводы

В процессе программного конструирования был выбран в качестве XML анализатора Expat. Разработана структура программного кода, схемы взаимодействия и наследования классов, сервера и клиента, а так-же динамически подгружаемых модулей. Для построения конфигурационных файлов решено использовать библиотеку XFLib, как наиболее простой и легко читаемый и анализируемый формат представления данных.

4 ТЕСТИРОВАНИЕ РАСПРЕДЕЛЕННОЙ ФАЙЛОВОЙ СИСТЕМЫ

Разработанное программное средство обладает всеми заявленными характеристиками, о чем свидетельствует «Акт сдачи/приемки» приложения Ж.

Для подтверждения корректности работы программы было применено два персональных компьютера с операционной системой Debian GNU Linux 6.0 Squeeze. Соединенных в локальную сеть. На одном из компьютеров было запущено два серверных приложения, с разными директориям общедоступных в файловой системы данных. Для этого были созданы два конфигурационных файла:

Конфигурационный файл первого сервера имеет вид:

```
conf{
* сеция глобальных данных как сервера так и клиента
   global_data{
      extension_dir="extensions";
      module_dir="modules";
  }
* секция данных индивидуальных для сервера
  server_data{
    server_ip="127.0.0.1";
    server_port=1100;
    server_bin_port=1101;
    shared_dir="/home/nick/FSROOT/1/"
  }
* секция в которой будут храниться данные расширений и модулей

  module_data{
}
```

```
  }
}
```

Конфигурационный файл второго сервера выглядел следующим образом:

```
conf{
* сеция глобальных данных как сервера так и клиента
   global_data{
      extension_dir="extensions";
      module_dir="modules";
  }
* секция данных индивидуальных для сервера
  server_data{
    server_ip="127.0.0.1";
*     bibary_protocol_ip="127.0.0.1";
    server_port=1100;
    server_bin_port=1101;
    shared_dir="/home/nick/FSROOT/1/"
  }
* секция в которой будут храниться данные расширений и модулей
  module_data{
  }
}
```

Программа клиент была запущена на другом ПК, конфигурационный файл клиента имел следующий вид:

```
conf{
* сеция глобальных данных как сервера так и клиента
   global_data{
      extension_dir="extensions";
```

```
    module_dir="modules";
  }
* секция данных индивидуальных для сервера
  client_data{
    servers{
       server_A{
          address="10.83.220.6";
          xml_port=1100;
          bin_port=1101;
       }
       server_B{
          address="10.83.220.6";
          xml_port=1102;
          bin_port=1103;
       }
    }
  }
* секция в которой будут храниться данные расширений и модулей
  module_data{
    lib_file_opertion {
       max_packet_size=40960;
    }
  }
}
```

Запуск программы сервера осуществлялся командами ./serverfsp а клиента ./clientfsp -d <точка монтирования> , опция -d использована для входа в режим отладки fuse.

Для тестирования приложений был разработан тестовый пример. Для начала теста необходимо создать файл, с произвольными данными, над которыми будут производится операции ФС. Для генерации такого файла используется команда:

dd if=/dev/urandom of=./random_file ibs=1048576 count=200

В результате ее выполнения будет создан файл объемом в 200 мегабайт.

Файл копируется в папку, в которую примонтирована ФС с помощью команды:

cp ./random_file ./testd

При этом копирование должно происходить одновременно на два сервера. Затем, производится тест на чтение файла, и сравнение его побайтно:

cmp ./random_file ./testd/random_file

Если файловая система работает верно, то программа завершиться без вывода какого либо сообщения, если файлы отличаются то ФС работает не корректно. В результате данной операции сообщений получено небыло, что свидетельствует о корректности работы ФС.

Следующий этап тестирования, создание директорий:

cd testd

mkdir testdir

В результате должна быть создана новая директория, это проверяется коммандой:

ls -l

Ее вывод:

-rw-r--r-- 1 nick nick 209715200 Окт 25 13:42 random_file

drwxr-xr-x 2 nick nick 4096 Окт 25 13:52 testdir

Т. е. функции чтения директории и создания папки, а так-же получения атрибутов файла работают верно.

Тест перемещение фала, выполняется командой mv:

mv ./random_file ./testdir/

Команда не должна затратить много времени и выполнится не более чем за одну секунду, т.к. не приходит перемещения данных.

проверка успешности операции проводится командой ls

#ls

вывод

testdir

что свидетельствует о том что файла в исходной папке нет

#ls ./testdir/

ее вывод:

random_file

Файл находится в папке назначения, это доказывает корректность работы функции перемещения и переименования файла.

Тест на изменение режима доступа к файлам, производится командами:

ls -l

вывод

-rw-rw-rw- 2 nick nick 209715200 Окт 25 13:42 random_file

что свидетельствует о том что файл доступен как для чтения так и для записи

chmod -r ./random_file

ls -l

вывод

--w--w--w- 2 nick nick 209715200 Окт 25 13:42 random_file

теперь файл невозможно читать, пробуем прочесть его:

#cat ./random_file

Вывод

cat: ./random_file: Отказано в доступе

что подтверждает что атрибуты на чтение работают верно

chmod 0777 ./random_file

ls -l

вывод

-rwxrwxrwx 2 nick nick 209715200 Окт 25 13:42 random_file

Теперь файл имеет режим доступа для чтения, записи и выполнения, для группы, владельца, и других пользователей, что подтверждает корректность работы режимов доступа Ф.С.

Для проверки скорости чтения с файловой системы, может использоваться коммана rsync:

rsync --progress -v ./testdir/random_file ../random_file2

Скорость передачи данных в один поток составила 5МБайт/сек.

Тест на создание жеских ссылок выполняется командой

link ./testdir/random_file ./random_file1

ls

Вывод команды

random_file1

что свидетельствует о успешном создании жесткой ссылки.

Проверка удаления файла производится командой

rm ./restdir/random_file

Убедится что файл действительно удален можно с помощью команды

ls ./restdir/

ее вывод будет пустым.

Тест был пройден успешно, все действия были выполнены корректно, что доказывает работоспособность программ.

Поскольку данное приложение должно выполняться длительное время, на

серверной и клиентской стороне, которое может занимать от нескольких часов до нескольких месяцев, важным критерием являются утечки памяти связанные с приложением. Для тестирования приложения на утечки памяти и ошибки синхронизации процессов-потоков, возможных при обращении к одному и тому-же участку памяти из разных процессов, а так-же ошибок взаимоблокировок процессов был применен отладчик valgrind, с модулем memcheck и модулем helgrind. Для этого запуск приложений производился следующей командами:
valgrind --tool=helgrind ./serverfsp и valgrind --tool=helgrind ./clientfsp -d <точка монтирования>
для проверки памяти и командами:
valgrind --tool=memcheck ./serverfsp и valgrind --tool=memcheck ./clientfsp -d <точка монтирования>
для проверки взаимодействия процессов соответственно, затем тестирование производилось повторно. В процессе работы программы внутри valgrind, ошибок взаимодействия процессов обнаружено не было. Были обнаружены не значительные однократные утечки памяти внутри библиотеки xflib, которые не влияют на дальнейший ход выполнения программы после ее инициализации.

В целом программа работает корректно и выполняет заявленные функции.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ РАСПРЕДЕЛЕННОЙ ФАЙЛОВОЙ СИСТЕМЫ

5.1 Резюме

Предлагается разработка программного продукта (ПП) предназначенного для монтирования каталогов удаленных ЭВМ в компьютерной сети и использования файлов в них как локальные.

Использование данного ПП возможно в разных организациях, где необходимо организовать единое дисковое пространство между ЭВМ.

Целью намеченного бизнеса является получение прибыли за счет предложению рынку конкурентоспособного продукта.

Для разработки и создания ПП необходимы капитальные вложения составляют 20232 руб., капитальные вложения будут погашены через 12 месяцев.

Критический объем продаж программного продукта составляет 38 копий. Запас финансовой прочности, за этот же период, составляет 21 копии ПП, коэффициент финансовой прочности равен 35%.

Цель данного бизнес-плана – выработка стратегических решений путём рассмотрения данного бизнеса с позиции маркетингового синтеза.

5.2 Характеристика ПП

Программный продукт позволяет монтировать каталоги на удаленных ЭВМ создавая единое дисковое пространство для нескольких объединенных в сеть компьютеров. Каждый компьютер может записать в это дисковое пространство некоторые данные, и использовать их совместно с другими компьютерами в сети, как будто это локальные файлы.

Разрабатываемый программный продукт может применяться для

эффективной организации дискового пространства вычислительных машин и повышения надежности хранимых данных в сети, путем дублирования их на несколько ЭВМ. ПП должен реализовывать следующие функции:

- позволяет монтировать каталоги на удаленных ЭВМ как локальные создавая единое дисковое пространство для нескольких объединенных в сеть компьютеров;
- в случае ошибки нехватки дискового пространства, данные должны быть переписаны на другую машину, и произведена до-запись. Если подобных машин не найдено, тогда вызывать ошибку;
- в случае отключения компьютера в момент записи должен быть выбран новый компьютер и данные перезаписаны на него, таким образом программа на стороне клиента должна кэшировать данные и производить их запись на выбранный компьютер параллельно;
- программа должна функционировать в операционной системе UNIX и ее разновидностях;
- разрабатываемое приложение должно состоять из двух программ: клиента который подключается к набору серверов и выполняет операции над файлами, и сервера, который реализует файловую систему. Клиент подключается к серверу и производит операции над файлами. Любой клиент может подключаться к одному или более серверов одновременно, любые сервера могут быть подключены к нескольким клиентам;
- файловая система должна поддерживать расширяемые модули для сервера и клиента, благодаря которым будут осуществляться операции с файлами, выбор компьютеров для записи фалов, аутентификация и прочие действия;

Охарактеризуем рассматриваемый проект с позиции маркетинга по

параметрам замысел, реальное исполнение, область применения, преимущества у пользователя:

- Замысел. Организация единого дискового пространства;
- Реальное исполнение. Распределенная файловая система;
- Область применения. ПП может применяться в различных коммерческих и не коммерческих организациях;
- Преимущества у пользователя. Использование стандартного интерфейса ФС для работы с данными на удаленных ЭВМ, повышенная надежность целостности данных, распределение сетевого трафика при обращении к данным на удаленных ЭВМ позволяющее уменьшить нагрузку на ЛВС, использовать данные совместно с другими пользователями, использовать свои данныме на других ЭВМ в ЛВС.
- Конкуренты. Andrew File System, Lustre, Gluster FS, HDFS, Google FS.

5.3 Исследование и анализ рынка

В качестве основных потребителей рассматриваются коммерческие и некоммерческие организации Ростова-на-Дону. Процесс сегментирования представлен в ниже приведенной таблице.

Сегменты рынка	Планируемый объем продаж по годам					
	2011г	2012г		2013г		Всего
	2 полугодие	1 полугодие	2 полугодие	1 полугодие	2 полугодие	
Интернет провайдеры	5	7	9	11	12	44
Хостинг компании	3	4	6	7	10	30
Коммерческие организации	2	3	6	8	11	30

Итого	10	14	21	26	33	104

Таблица №5.1 «Ориентировочная сегментация потенциальных потребителей»

5.4 Производственный план

5.4.1 Расчёт единовременных затрат

В структуре единовременных затрат ($З_к$) выделяют капитальные затраты (К), включающие затраты на приобретение или дооборудование вычислительной техники (Квт), приобретении пакета прикладных программ (Кппп) и операционных систем (Кос):

$$З_к = K_{ВТ} + K_{ОС} + K_{ППП} \quad (5.1)$$

В состав затрат на приобретение ВТ, ЛВС, ППП, ОС включаются транспортные расходы и затраты на установку. Единовременные затраты приведены в таблице №__.

№	Наименование тех. средства или ПО	Тип или модель	Кол-во (шт.)	Поставщик	Стоимость одного изделия	Сумма (руб.)
1	Системный блок	Granda I3 2100	1	Solwin	12990	12990
2	Монитор	SAMSUNG B1940R BMB [TFT]	1	Solwin	5990	5990
3	Клавиатура	Genius KB110, Black [PS/2]	1	Solwin	203	203
4	Мышь	Genius Navigator 535 Laser 1600dpi	1	Solwin	690	690

		[USB]				
5	Операционная система	Debian Squeeze GNU/Linux	1	knoppix.ru	359	359
6	Прикладное ПО	Eclipse IDE	1	The Eclipse Foundation	0	0
7	Прикладное ПО	GNU Compiler Collection	1	Free Software Foundation, Inc.	0	0
Итого (руб.):						20232

Таблица №5.2 «Единовременные затраты»

Зк=20232

5.4.2 Расчёт текущих затрат на разработку ПП

Для расчета текущих затрат определим состав персонала, участвующего в разработке программного продукта. Расчет зарплаты персонала приведен в таблице №5.3

Категория персонала	Количество сотрудников	Оплата за 1 час (руб.)	Время, необходимое для разработки программного продукта (час)	Общая зарплата сотрудников данной категории (руб.)
Инженер-программист	1	110	288	31680
Итого (руб.)				31680

Таблица №5.3 «Расчет заработной платы персонала»

Оплата труда производится на основе повременной оплаты, исходя из часовой ставки ($C_ч$) и времени на разработку ПП ($T_{пр}$). В этом случае

заработная плата рассчитывается по формуле:

$$З_{пр} = \sum (С_ч) \cdot Т_{пр} \quad (5.2)$$

Потребность в капиталовложениях на разработку проекта представлена в таблице №5.4.

Наименование статей затрат	Формула для расчета	Сумма
1.Единовременные затраты (Зк)	$З_К = К + З_{ПК}$	20232
1.1. Затраты на приобретение ВТ (Квт)	$К_{ВТ}$	19873
1.2. Затраты на приобретениепакетов прикладных программ и операционных систем (Кппп)	$К_{ППП}$	359
2. Текущие затраты (С)	$С = З_{ПР} + Н_З + З_Н$	105811
2.1. Затраты на заработную плату (Зпр)	$З_{ПР} = \sum (С_Ч) * Т_{ПР}$	31680
2.2. Страховые взносы ($Н_з = 34\% от З_р$)	$Н_З = 34 от З_{ПР}$	10771
2.3. Накладные расходы (Зн)	$З_Н = 200 от З_{ПР}$	63360
Итого затрат (З)	$З = К + С$	125093

Таблица №5.4 «Потребность в капиталовложениях на разработку проекта. »

5.4.3 Определение цены ПП

Рассчитав текущие затраты на разработку ПП (С), можно определить себестоимость одной копии ПП по формуле:

$$С_1 = \frac{С}{N} + З_{тир} + З_{сер} \quad (5.3)$$

где N- объем продаж в натуральном выражении;

$З_{тир}$ – затраты на тиражирование в расчёте на одну копию;

$З_{сер}$ – затраты на сервисное обслуживание в расчете на одну копию.

К затратам на тиражирование необходимо отнести стоимость носителя ($С_н$) информации для программного продукта и расходы связанные с записью ($С_{зап}$) на электронный носитель.

$З_{тир}$ – затраты на тиражирование в расчете на одну копию, руб.

$$З_{тир} = З_{нос} + З_{пр \cdot тир} + СВ_{тир} + З_{н \cdot тир} \quad (5.4)$$

где $З_{нос}=60(руб.)$ – затраты на приобретение электронного носителя при производстве копии программы;

$З_{пр\cdot тир}$ – заработная плата программиста на производство одной копии, которая определяется по формуле:

$$З_{пр\cdot тир}=C_ч\cdot T_{прк} \quad (5.5)$$

где $C_ч=110$руб. – часовая тарифная ставка программиста;

$T_{прк}=0,2$ час. – время, необходимое оператору ЭВМ на тиражирование одной копии;

$З_{пр\cdot тир}=110\cdot 0,2=22\ руб.$

$СВ_{тир}=34\ от\ З_{пр\cdot тир}$ – социальные взносы, руб.;

$СВ_{тир}=0,34\cdot 22=7,48\ руб.$

$З_{н\cdot тир}$ – накладные расходы на производство одной копии:

$З_{н\cdot тир}=200-300\%\ от\ З_{пр\cdot тир}$

$З_{н\cdot тир}=2,0\cdot 22=44\ руб.$

$З_{тир}=60+22+7,48+44=133,48\ руб.$

К затратам на сервисное обслуживание относятся услуги по внесению изменений и дополнений по желанию клиента.

Затраты на оказание данной услуги рассчитываем следующим образом:

$$З_{сер}=З_{пр\cdot сер}+СВ_{сер}+З_{н\cdot сер} \quad (5.6)$$

где $З_{пр\cdot сер}$ – заработная плата сотрудника за сервисное обслуживание, руб, вычисляется по формуле:

$$З_{пр\cdot сер}=\sum C_ч\cdot T_{изм} \quad (5.7)$$

где $C_ч$ – часовая тарифная ставка программиста

$C_ч$ = 110 руб.

$T_{изм}$ – время, необходимое программисту на внесение изменений, час.;

$T_{изм}=2\ час.$

$З_{пр\cdot сер}=110\cdot 20=220\ руб.$

$СВ_{сер} = 0{,}34 \cdot 220 = 74{,}8\ руб.$ – социальные взносы;

$З_{н \cdot сер}$ – накладные расходы, руб.,

$З_{н \cdot сер} = 200\ \%\ от 220 = 440\ руб.$

$З_{сер} = 220 + 74{,}8 + 440 = 734{,}8\ руб.$

Таким образом стоимость одной копии:

$$C_1 = \frac{105811}{104} + 133{,}48 + 734{,}8 = 1885{,}7\ (руб.)$$

Определяем оптовую цену одной копии по следующей формуле:

$$Ц_{О1} = C_1 + П_1$$

где: C_1 – себестоимость одной копии, руб.;

$П_1$ – прибыль на одну копию, руб.;

Определяем прибыль:

$$П_1 = \frac{C_1 \cdot P}{100} \qquad (5.8)$$

где Р – процент предполагаемой рентабельности (Р=34%);

$$П_1 = \frac{1885{,}7 \cdot 34}{100} = 641{,}14\ (руб.)$$

$$Ц_{О1} = 1885{,}7 + 641{,}14 = 2526{,}83\ (руб.)$$

Цена продажи программного продукта:

$$Ц_{пр} = Ц_{О1} + НДС \qquad (5.9)$$

где НДС – налог на добавленную стоимость в соответствии действующей ставкой на данный вид продукции (18%),

$$Ц_{пр} = 2526{,}83 + \frac{2526{,}83}{100} \cdot 18 = 2981{,}66\ (руб.)$$

5.5 План маркетинговых действий

План маркетинга - это план мероприятий по достижению намечаемого объема продаж и получению максимальной прибыли путем удовлетворения рыночных потребностей. В данном разделе отражаются: товарная, ценовая, сбытовая политика и сервисное обслуживание.

Товарная политика предполагает постоянную модификацию ПП, учитывая требования пользователей, бесплатные обновления, сервисное обслуживание в течение всего жизненного цикла ПП, а также on-line консультации.

Ценовая политика предполагает установление цены, ориентируясь на цены мирового рынка, а также ее изменение в зависимости от области предпринимательской деятельности.

Сбытовая политика предполагает:

- создание и регулирование коммерческих связей через посредников, дилеров, агентов и пр;
- организация и участие в ярмарках, выставках;
- использование кредита в различных формах, продажа в рассрочку, лизинг;
- презентацию продукции специально для потенциальных потребителей.

Сервисное обслуживание предполагает предпродажный и послепродажный сервис.

Предпродажный сервис ориентирован на постоянное изучение и анализ требований потребителей с целью совершенствования качественных параметров предлагаемой продукции.

Послепродажный сервис предусматривает комплекс работ по обслуживанию (комплекс работ по установке ПП и обучению пользователя).

5.6 Потенциальные риски

В рыночных условиях этот раздел особенно важен, и от глубины его проработки в значительной степени зависит доверие потенциальных инвесторов, кредиторов и партнеров по бизнесу.

Следует учитывать следующие виды рисков: производственные, коммерческие, финансовые и риски, связанные с форс–мажорными

обстоятельствами.

Производственные риски связаны с различными нарушениями в производственном процессе:

- выход из строя комплектующих ПК;
- потеря информации из-за выхода из строя носителей информации;
- потеря информации из-за аварийного отключения электропитания;
- обрыв линии передачи информации между сервером и пользователями рабочих станций.

Производственные риски можно снизить с помощью дублирования информации, копирования информации на разные носители, создание архивов программных версий, использования средств защиты.

Коммерческие риски связаны с реализацией продукции на товарном рынке:

- уменьшение размеров и емкости рынков;
- снижение платежеспособного спроса;
- появление новых конкурентов;
- отсутствие рекламы;
- отсутствие консультационного центра.

Коммерческие риски можно снизить, если создать отдел маркетинга для рекламирования ПП, а также создание консультационного центра для поддержки пользователей.

Финансовые риски вызываются:

- инфляционными и девальвационными процессами;
- отплатой по безналичному расчету.

Финансовые риски можно снизить, если предусмотрены условия по реализации: объем реализации по безналичному расчету должен быть не менее 10000 руб.

Риски, связанные с форс-мажорными обстоятельствами - чрезвычайное событие, которое невозможно было предвидеть и предотвратить:

- стихийные бедствия.

Мерой по предотвращению рисков, связанных с форс-мажорными обстоятельствами, является работа предприятия с достаточным запасом финансовой прочности.

Для снижения общего влияния рисков предусмотрено коммерческое страхование: страхование имущества, транспортных перевозок, перестрахование и т.д.

5.7 Финансовый план

Финансовый план является заключительным разделом бизнес-плана и содержит обоснование экономической эффективности затрат, произведенных в связи с разработкой и реализацией ПП.

Предполагаемые доходы от продаж (Qпр) определяются по формуле:

$$Q_{ПР} = Ц_{О} \cdot N \qquad (5.10)$$

где Цо – оптовая цена ПП,(без НДС) руб.; N - объем продаж по периодам в соответствии с исследованиями рынка, шт.

Издержки производства (И) включают, кроме текущих затрат, расходы на тиражирование, сервисное обслуживание, маркетинг, рекламу и некоторые виды налогов (на имущество, местные налоги и т.п.):

$$И = C_{1} \cdot N \cdot 1{,}25 \qquad (5.11)$$

где С1 - себестоимость копии ПП. Расчеты производятся по годам и сводятся в таблице №5.4:

Показатели	2011	2012		2013	
	1 полугодие	1 полугодие	2 полугодие	1 полугодие	2 полугодие
Доходы от продаж (Qпр)	25268,32	35375,65	53063,47	65697,63	83385,45
Издержки производства (И)	23571,19	32999,67	49499,5	61285,1	77784,93
Прибыль от реализации	6411,364	8975,91	13463,87	16669,55	21157,5
Налог на прибыль 20%	1282,273	1795,182	2692,773	3333,909	4231,5
Чистая прибыль	5129,091	7180,728	10771,09	13335,64	16926
Справочно:планируемый объем продаж ПП(шт.)	10	14	21	26	33

Таблица №5.5 - Финансовый план

В данном случае разработчик принимает решение осуществить разработку ПП за счет собственных средств без привлечения коммерческого кредита, поэтому определим срок окупаемости ПП, который определяется сроком окупаемости дополнительных капитальных вложений, представленный в таблице № 5.5

	2011	2012		2013	
	2 полугодие	1 полугодие	2 полугодие	1 полугодие	2 полугодие
Сумма капитальных вложений	20232				
Ожидаемая чистая прибыль	5129,091	7180,728	10771,09	13335,64	16926
Коэффициент дисконтирования	0,97	0,889	0,889	0,7	0,7
Дисконтированная чистая прибыль	4975,219	6383,667	9575,501	9334,946	11848,2
Непогашенный остаток капитальных вложений, руб.	15256,78	8873,114			
Остаток чистой прибыли на конец периода, руб.			702,3867	10037,33	21885,53

Таблица №5.6: Срок окупаемости программного продукта

Расчеты показывают, что возврат капитальных вложений возможен во втором полугодии 2012 года. Во втором полугодии второго года выпуска возможна капитализация прибыли. Срок возврата капитальных вложений – 1 год.

5.8 Расчёт безубыточности

Под безубыточностью в разработанном бизнес-плане понимается объем продаж ПП в натуральном выражении, при котором возможно покрытие всех расходов без получения прибыли.

Расчет достижения безубыточности производится по следующей формуле, для определенного года:

$$Q_{кр} = \frac{ФЗ}{Ц_{пр} - ПЗ_1} \quad (5.12)$$

где:

- $Q_{КР}$ – критический объем продаж программного продукта;

- $ФЗ$ – сумма условно-постоянных (фиксированных) затрат;
- $ПЗ_1$ – сумма условно переменных затрат для одного программного продукта;
- $Ц_{ПП}$ – цена продажи одного программного продукта.

Произведем расчет безубыточности продукта для 2013 года:

$$ФЗ = З_Н + З_М + Н \quad ; \qquad (5.13)$$

- $З_М$ – затраты на маркетинг и коммерческие расходы ($З_М = 2\% от Q_{пр}$ соответствующего периода);
- $Н$ – налоги. Налог на имущество НИ определяется в процентах от стоимости ВТ и ЛВС по действующей ставке (Ни = 2% от стоимости ВТ и ЛВС).

$$Q_{ПР} = 2526{,}83 \cdot 59 = 149083 (руб.)$$

$$З_М = \frac{149083}{100} \cdot 2 = 2981{,}66$$

$$Н = \frac{19873}{100} \cdot 2 = 397{,}46$$

$$ФЗ = 2981{,}66 + 63360 + 397{,}46 = 66739{,}12 (руб.)$$

$$ПЗ_{пер} = И_{пер} - ФЗ \qquad (5.14)$$

$И = C_1 \cdot N \cdot 1{,}25$, где N количество проданных копий за период, 59 копий

$$И_{пер} = 59 \cdot 1885{,}69 \cdot 1{,}25 = 139070$$

$$ПЗ = 139070 - 66739{,}12 = 72330{,}9 (руб.)$$

$$ПЗ_1 = \frac{72330{,}9}{59} = 1225{,}94 (руб.)$$

$$Q_{кр} = \frac{ФЗ}{Ц_{пр} - ПЗ_1} \qquad (5.15)$$

$$Q_{кр} = \frac{66739{,}12}{2981{,}66 - 1225{,}94} = 38{,}01 \approx 38 шт$$

После расчета критического объема продаж строится график безубыточности. (Рисунок 5.1):

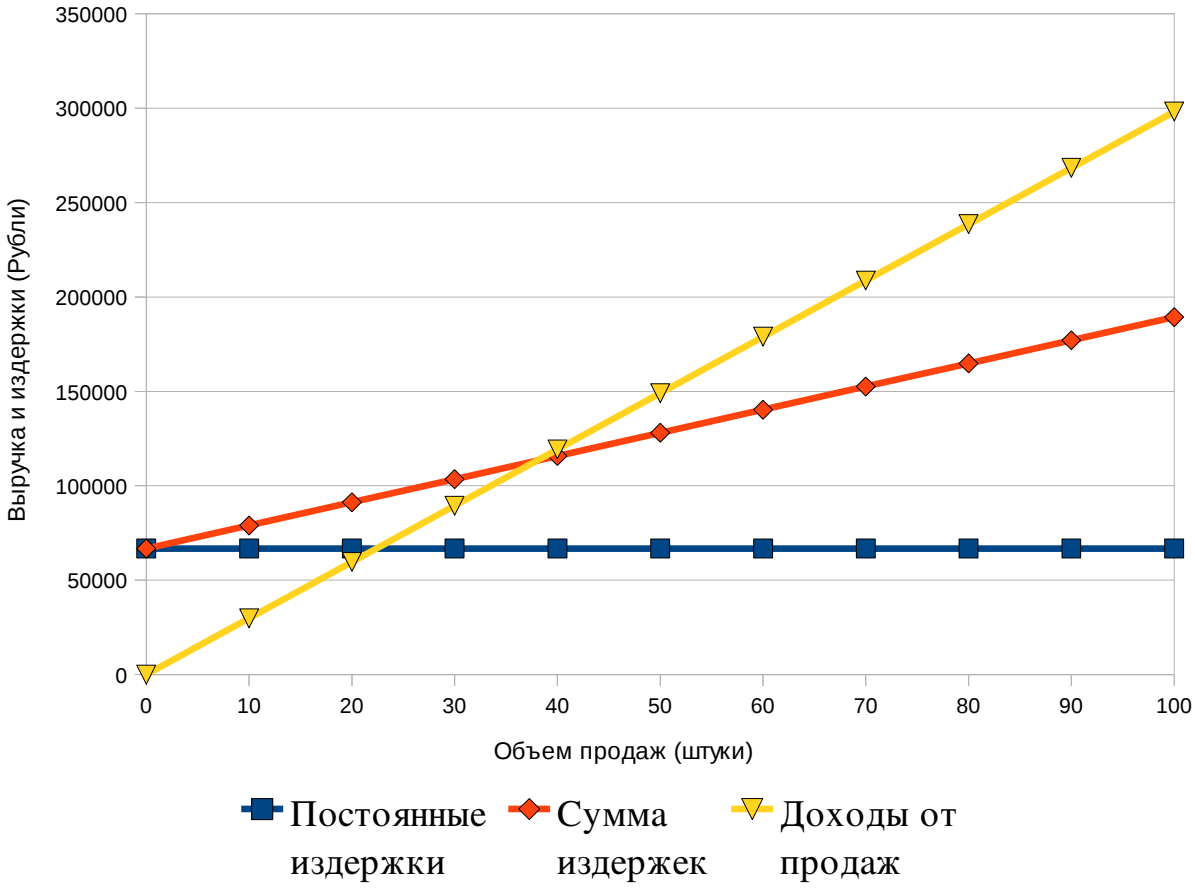

Рисунок 5.1 – График безубыточности за 2013год

Влево от точки безубыточности лежит область убытков, вправо – область прибыли.

Запас финансовой прочности (по расчёту за последний год производства) определяется по следующей формуле:

$$ЗФП = Q_{пр} - Q_{кр} = 59 - 38 = 21 \text{шт} \quad (5.16)$$

Коэффициент запаса финансовой прочности $K_{ЗФП}$ определяется отношением величины запаса финансовой прочности к объему продаж. Он характеризует степень финансовой устойчивости, рекомендуемая нижняя граница 30% к объему продаж.

$$K_{ЗФП} = \frac{ЗФП}{Q_{пр}} \cdot 100 = \frac{21}{59} \cdot 100 \approx 35\,\% \quad (5.17)$$

Таким образом, можно констатировать, что разработка, при данном

объеме продаж, имеет достаточный запас финансовой прочности и является финансово устойчивой.

6 БЕЗОПАСНОСТЬ И ЭКОЛОГИЧНОСТЬ ПРОЕКТА

6.1 Введение

Данный дипломный проект представляет собой разработку распределенной файловой системы. Предмет темы проекта может использоваться везде, где требуется организовать единую файловую систему для нескольких компьютеров (подобие сетевого диска в Windows). При этом, цель создания подобной системы может быть любой - повышение надежности сохранности файлов, совместного использования несколькими пользователями одних и тех же фалов, и т.п. Таким образом, пользователем данной системы может быть любая организация, где есть сервера (условно, сервером может быть как та машина за которой работает пользователь, так и какая-то выделенная машина в серверной) и где есть пользователи (клиенты) которые работают с этими данными, на серверах. Соответственно, раздел «Безопасность и экологичность проекта» посвящен оценке опасных и вредных факторов при работе на персональном компьютере.

6.2 Опасные и вредные факторы при работе на ЭВМ

Согласно ГОСТ 12.2.007.0-75 изделия, создающие электромагнитные поля, имеют защитные элементы (экраны, поглотители) для ограничения воздействия этих полей в рабочей зоне до допустимых уровней.

Электрическая схема ЭВМ исключает возможность его самопроизвольного включения и отключения. Конструкция исключает возможность неправильного присоединения ее сочленяемых токоведущих частей.

Согласно ГОСТ 25861-83 маркировка выключателей и других органов управления и регулирования помещается в непосредственной близости от них.

Меры защиты от поражения электрическим током и защитное заземление

выполняются в соответствии с требованиями ГОСТ 25861-83 и ГОСТ 12.1.030-81.

Изоляция, применяемая в ЭВМ, служащая для защиты от поражения электрическим током, выполняется одним из следующих способов:

- применением сплошного или слоистого изоляционного материала, толщина и пути утечки по поверхности которого обусловлены видом обеспечиваемой защиты;
- применением воздушных зазоров;
- совместным применением вышеуказанных способов.

Электрическая и механическая прочность, а также теплостойкость изоляции соответствует требованиям, предъявляемым к ЭВМ. По ГОСТ 21552-84 электрическое сопротивление изоляции средств вычислительной техники между разобщенными токоведущими цепями, а также между токоведущими цепями и корпусом в зависимости от климатических условий эксплуатации должно быть не менее 20 МОм, при наибольшем значении рабочей температуры - 5 МОм, при наибольшем значении относительной влажности - не более 1 МОм (при рабочих напряжениях от 100 до 500 В).

Допустимые уровни звукового давления в активных полосах частот, уровни звука, создаваемые ПЭВМ на рабочем месте оператора, удовлетворяют требованиям ГОСТ 12.1.003-83 с эквивалентным уровнем звука не более 50 дБ по шкале А.

При этом эквивалентные уровни звука для таких видов трудовой деятельности, как работа по выработке концепций, новых программ, творчество, не должны превышать 40 дБА.

В соответствии с требованиями по безопасности к видеомониторам по ГОСТ 21552:

- ЭЛТ видеомонитора является взрывобезопасной без дополнительной

защиты. Применение взрывоопасных ЭЛТ запрещено.

- Мощность дозы рентгеновского излучения в любой точке пространства на расстоянии 5 см от экрана видеомонитора не превышает 0.03 мкР/с при 40-часовой рабочей неделе.
- Корректированный уровень звуковой мощности шума - по ГОСТ 26329, при этом допустимый уровень шума на рабочем месте оператора не превышает 50 дБА.
- Плотность потока ультрафиолетового излучения не превышает 10 Вт/м.
- Эксплуатационная документация содержит указания по безопасным приемам работ при техническом обслуживании видеомониторов и требования к обслуживающему персоналу.

Общее время работы на ЭВМ не должно превышать четырех часов в день. Продолжительность непрерывной работы не более 30 минут с перерывами не менее 15 минут.

6.3 Искусственное освещение

Искусственное освещение широко используют в производственных помещениях, на открытых площадках, проездах и многих других объектах, поэтому его правильное проектирование и расчет имеют очень важное значение.

При проектировании осветительной установки решают ряд технических задач: выбирают тип источника света, систему освещения, норму освещенности, тип светильников, способы освещения, схему размещения светильников, рассчитывают освещенность в заданных точках, уточняют после этого размещении число светильников, определяют единичную мощность светильников и ламп.

Выбор ламп освещения зависит, прежде всего, от требований к качеству

светопередачи. При этом предпочтение отдается газоразрядным источникам, так как при их свете облегчается работа глаз, наблюдается меньшее утомление организма в целом, повышается работоспособность человека и производительность труда. Используем газоразрядные лампы т.к. необходимо обеспечить тонкое различение цветов.

Выбор системы освещения обусловливается технологическими или архитектурными соображениями с учетом требований норм. В гигиеническом отношении более желательна система общего освещения, так как она создает благоприятное распределение яркостей в поле зрения.

При повышенных требованиях к освещенности отдельных рабочих мест используют комбинированную систему освещения, при которой помимо общего освещения применяют местное. Использование в производственных условиях только местного освещения категорически запрещено.

Согласно выбранным источникам света и системе освещения, определяется значение нормированной освещенности. Для этого используют отраслевые нормы или СНиП П-4-79.

6.3.1 Расчет искусственного освещения

Выбираем светильники ОД с газоразрядными лампами. Светильники расположены параллельными рядами.

Тип проводки – закрытая в строительных конструкциях под штукатуркой, провода – АППВ, выключатель нормального исполнения.

Минимальная освещенность от комбинированного освещения:

Emin.0 =200 лк.

Система освещения – комбинированная: общее равномерное плюс местное.

Потребляемая освещенность при комбинированном освещении

газоразрядными лампами от светильников общего освещения:

Ео.к =200 лк,

от местного:

Ем.к =150 лк.

Необходимый коэффициент запаса (по выделяемой пыли):

Кз =1,6.

Расстояние между светильниками по длине $L_{св.д}$. определяется по формуле:

$$L_{СВ.Д.} = \gamma \cdot h_{СВ}, \quad (6.1)$$

где γ - коэффициент, учитывающий расстояние между светильниками и высоту подвески светильников, равный 1,6;

$h_{СВ}$ - высота подвески светильников, равная 3 м.

$$L_{СВ.Д.} = 1{,}6 \cdot 3 = 4{,}8 \text{ м}$$

Расстояние между светильниками по ширине $L_{СВ.Ш}$ примем равным длине светильника плюс 0,05 м, тогда $L_{СВ.Ш}$ =1,2 м.

Расстояние от стены до первого ряда светильников определяется по формуле:

$$L_1 = 0{,}3 \cdot L_{СВ.Д.}, \quad (6.2)$$

$L_1 = 0{,}3 \cdot 4{,}8 = 1{,}44$ м

Расстояние между крайними рядами по ширине помещения определятся по формуле:

$$L_2 = b - 2 \cdot L_1, \quad (6.3)$$

где b – ширина помещения, равная 3 м,

$L_2 = 3 - 2 \cdot 1{,}44 = 0{,}12$ м

Число рядов, которое можно расположить между крайними рядами по ширине помещения определяется по формуле:

$$N_{СВ.Ш.} = \frac{L_2}{L_{СВ.Ш.}} - 1, \quad (6.4)$$

$$N_{СВ.Ш.} = \frac{0{,}12}{1{,}2} - 1 = -0.9 \approx 0$$

Общее число рядов светильников по ширине определяется по формуле:

$$N_{СВ.Ш.О} = N_{СВ.Ш.} + 2, \quad (6.5)$$

$$N_{СВ.Ш.О} = 0 + 2 = 2$$

Расстояние между крайними рядами светильников по длине помещения определяется по формуле:

$$L_3 = a - 2 \cdot L_1, \quad (6.6)$$

где а – длина помещения, равная 4 м

$L_3 = 4 - 2 \cdot 1{,}44 = 1{,}12$ м

Число светильников, которое можно расположить между крайними рядами по длине, определяется по формуле:

$$N_{СВ.Д.} = \frac{L_3}{L_{СВ.Д.}} - 1, \quad (6.7)$$

$$N_{СВ.Д.} = \frac{1{,}12}{4{,}8} - 1 = -0{,}766 \approx 0$$

Общее число рядов светильников по длине определяется по формуле:

$$N_{СВ.Д.О} = N_{СВ.Д.} + 2, \quad (6.8)$$

$$N_{СВ.Д.О} = 0 + 2 = 2$$

Общее число рядов светильников, которые необходимо установить по

длине и ширине определяется по формуле:

$$N_{СВ.ОБЩ.} = N_{СВ.Ш.О.} \cdot N_{СВ.Д.О.}, \quad (6.9)$$

$$N_{СВ.ОБЩ.} = 2 \cdot 2 = 4$$

Коэффициенты отражения от стен ($\rho_{СТ.}$) и потолков ($\rho_{ПОТ.}$) выбираем исходя из окраски стен и потолков равными: $\rho_{СТ.} = 50$ %, $\rho_{ПОТ.} = 70$ %.

Коэффициент z, учитывающий равномерность освещения в зависимости от типа светильников и коэффициента γ, выбираем равным 1,13.

Площадь пола ($S_П$), освещаемого помещения определяется по формуле:

$$S_П = a \cdot b, \quad (6.10)$$

$$S_П = 3 \cdot 4 = 12 \text{ м}^2.$$

Определяем показатель помещения по формуле:

$$\varphi = \frac{a \cdot b}{(a+b) \cdot h_{СВ.}}, \quad (6.11)$$

$$\varphi = \frac{12}{(3+4) \cdot 3} = 0{,}57$$

Коэффициент использования светового потока равен: $\eta_И = 0{,}53$.

Требующийся световой поток одной лампы определяется по формуле:

$$F_{Л.РАСЧ.} = \frac{E_{О.К.} \cdot K_З \cdot z \cdot S_П}{N_{СВ.ОБЩ.} \cdot \eta_И}, \quad (6.12)$$

$$F_{Л.РАСЧ.} = \frac{200 \cdot 1{,}6 \cdot 1{,}13 \cdot 12}{4 \cdot 0{,}53} = 2046{,}79 \text{ лм}$$

По напряжению в сети UC и световому потоку одной лампы:

$F_{Л.РАСЧ.} = 2046$ лм

по справочным таблицам (ГОСТ 2239-70) определяем необходимую мощность электролампы:

ЛД40-4 WЛ = 40 Вт.

В каждом светильнике имеется две лампы ЛД40-4 со световым потоком $F_{Л.ТАБЛ.}$ = 2340 лм каждая.

Действительная освещенность определяется по формуле:

$$E_{ДЕЙСТВ.} = \frac{F_{Л.ТАБЛ.} \cdot N_{СВ.ОБЩ.} \cdot \eta_{И}}{K_З \cdot z \cdot S_П}, \quad (6.13)$$

$$E_{ДЕЙСТВ.} = \frac{2340 \cdot 4 \cdot 0{,}6}{1{,}6 \cdot 1{,}13 \cdot 12} = 259 \text{ лк.}$$

Площадь комнаты 12 м^2. Светильники располагаются в один ряд по две лампы в каждом. Расположение ряда на потолке выбирают таким образом, чтобы освещенность рабочего места была достаточной для работы.

6.4 Влияние электромагнитных полей, создаваемых ЭВМ, на человека

Каждое устройство, которое производит или потребляет электроэнергию, создает электромагнитное излучение. Это излучение концентрируется вокруг устройства в виде электромагнитного поля. Видеотерминалы излучают электромагнитные волны в очень широком диапазоне. В радиодиапазоне они продуцируются катодной трубкой; основной же источник – горизонтальные и вертикальные отклоняющие катушки, которые обеспечивают сканирование электронного луча по экрану в диапазоне 15…35 кГц. На расстоянии 50 см от экрана напряжённость электрического поля имеет значение от меньших единицы до 10 В/м, а магнитная индукция – от 10–8 до 10–7 Тл. Видеотерминалы излучают также переменные электрические и магнитные поля с частотой 50 или 60 Гц и их гармоники.

Однако наблюдения за людьми показали неблагоприятное для здоровья действие низкочастотных электромагнитных полей частотой 50…60 Гц: ночью у большинства испытуемых повышался в крови уровень мелатонина – гормона шишковидной железы, или эпифиза. Эпифиз выполняет роль основного

“ритмоводителя” функций организма: чувствительные клетки сетчатки, воспринимающие свет, передают информацию о его интенсивности и качестве по нервным путям в эпифиз, специфические клетки которого чутко реагируют на свет и обеспечивают регуляцию синтеза мелатонина (свет “угнетает” синтез мелатонина, поэтому ночью его содержание в крови самое высокое, а утром и днём – минимальное). Нарушение этого ритма (например, вследствие систематического искусственного освещения человека ночью) может повлечь за собой серьёзные заболевания, в частности, образование опухоли. Особенный вред избыточная освещённость приобретает тогда, когда на организм действуют какие–либо канцерогенные факторы, например химические или радиационные.

На человека, работающего за компьютером, оказывают влияние не только разнообразные электромагнитные поля (в том числе изменяющиеся с частотой 50…60 Гц), но и интенсивный свет, который действует на глаза, а значит, и на эпифиз. Поэтому операторам желательно проводить за ними не более половины рабочего времени, воздерживаться от работы в вечернее, тем более в ночное, время. В ряде экспериментов было обнаружено, что электромагнитные поля с частотой 50…60 Гц, возникающие вокруг видеодисплеев, могут инициировать биологические сдвиги вплоть до нарушения синтеза ДНК в клетках животных.

6.5 Расчет защитного заземления ЭВМ

Заземление выполняет две основные электрозащитные функции. Первая заключается в создании необходимых условий для быстрого отключения и замыкания на землю. Вторая - уменьшение до требуемых пределов возможного напряжения прикосновения. (Под напряжением прикосновения понимается напряжение, приложенное непосредственно к телу человека на пути "руки - ноги").

Защитному заземлению подлежат металлические части ЭВМ, доступные

для прикосновения человека и не имеющие других видов защиты, обеспечивающих электробезопасностью.

В соответствии с ГОСТ 12.1.030-81 "Электробезопасность. Защитное заземление, зануление", защитное заземление должно обеспечивать защиту людей от поражения электрическим током при прикосновении к металлическим нетоковедущим частям, которые могут оказаться под напряжением в результате повреждения изоляции.

Защитное заземление выполняется преднамеренным электрическим соединением металлических частей электроустановок с "землей" или ее эквивалентом.

Материал, конструкция и размеры заземлителей, заземляющих и нулевых защитных проводников должны обеспечивать устойчивость к механическим, химическим и термическим воздействиям на весь период эксплуатации.

В электроустановках напряжением до 1000 В в сетях с изолированной нейтралью или изолированными выводами однофазного источника питания электроэнергией сопротивление заземляющего устройства в стационарных сетях должно быть не более 4 Ом.

Целью данного расчета является определение основных параметров заземления (число, размеры и размещение одиночных заземлителей и заземляющих проводников).

При этом расчет производится для размещения заземлителя в однородной земле.

Выбираем с учетом всех требований значение сопротивления защитного заземления Rдоп = 4 Ом (Rдоп должно быть равно 4 Ом для электроустановок напряжением до 1000 В в сети с изолированной нейтралью.)

Согласно справочной литературе задаемся удельным сопротивлением грунта $\rho = 30$ Ом·м (чернозем).

Выбираем групповой контурный заземлитель (уголок) длиной 3м.

Определяем сопротивление одиночного заземлителя по формуле:

$$RC = 0,159 \frac{\rho_i}{l_i}\left(\ln\frac{2l_i}{d_i} + \frac{1}{2}\ln\frac{4t + l_i}{4t - l_i}\right), \quad (6.14)$$

где: l - длина заземления, равная 3 м;

d - диаметр заземления, равный 0,1 м;

t – глубина заложения, t = 2 м.

$$RC = 0,159\frac{40}{3}\left(\ln\frac{2\cdot 3}{0,1} + \frac{1}{2}\ln\frac{4\cdot 2+3}{4\cdot 2-3}\right) = 10 \text{ Ом}$$

Определяем приближенное количество заземлителей $n^{/}$ по формуле:

$$n^{/} = \frac{RC}{R_{ДОП}\cdot N_S}, \quad (6.15)$$

где N_S – коэффициент использования, равный 0,8

$$n^{/} = \frac{10}{4\cdot 0,8} = 4$$

Располагая полученное число заземлителей на плане объекта, получим длину полосы связи l_i = 3,5 м.

Из справочной литературы определяем значения: коэффициента использования вертикальных стержневых заземлителей без учета влияния полосы связи NS = 0,83; коэффициента использования горизонтального полосового заземлителя, соединяющего вертикальные стержневые заземлители N_P = 0,89.

Определяем сопротивление растекания тока полосы связи по формуле:

$$RN = 0,366\frac{\rho_i}{l_i}\lg\frac{2l_i^{\,2}}{b_i h_i}, \quad (6.16)$$

где: b_i – расстояние от поверхности земли до середины вертикального заземлителя, равное 0,5 м;

h_i - расстояние от поверхности земли до оси горизонтальной полосы,

равное 0,52 м.

$$RN = 0{,}366 \frac{40}{3{,}5} \lg \frac{2 \cdot 3{,}5^2}{0{,}5 \cdot 0{,}52} = 8{,}3 \text{ Ом}$$

Определяем уточненное число заземлителей по формуле:

$$n = \frac{RN \cdot RC - R_{доп.} \cdot N_P \cdot RC}{R_{доп.} \cdot N_S \cdot RC}, \quad (6.17)$$

$$n = \frac{10 \cdot 8.3 - 4 \cdot 0{,}89 \cdot 10}{4 \cdot 0{,}83 \cdot 10} = 2 \text{ шт.}$$

Определяем сопротивление группового заземлителя по формуле:

$$R_{общ} = \frac{RN \cdot RC}{N_P \cdot RC + n \cdot N_S \cdot RN}, \quad (6.18)$$

$$R_{общ} = \frac{8{,}3 \cdot 10}{0{,}89 \cdot 10 + 2 \cdot 0{,}83 \cdot 8{,}3} = 3.7 \text{ Ом}$$

Так как $R_{ОБЩ.} \leq R_{ДОП.}$, 3.7 Ом ≤ 4 Ом, то задача завершена.

6.6 Микроклимат

Параметры микроклимата могут меняться в широких пределах, в то время как необходимым условием жизнедеятельности человека является поддержание постоянства температуры тела благодаря терморегуляции, т.е. способности организма регулировать отдачу тепла в окружающую среду. Принцип нормирования микроклимата – создание оптимальных условий для теплообмена тела человека с окружающей средой.

Вычислительная техника является источником существенных тепловыделений, что может привести к повышению температуры и снижению относительной влажности в помещении. В помещениях, где установлены компьютеры, соблюдаются определенные параметры микроклимата.

Таблица 6.1 - Параметры микроклимата для помещений, где установлены компьютеры

Период года	Параметр микроклимата	Величина
Холодный	Температура воздуха в помещении Относительная влажность Скорость движения воздуха	22…24°С 40…60% до 0,1м/с
Теплый	Температура воздуха в помещении Относительная влажность Скорость движения воздуха	23…25°С 40…60% 0,1…0,2м/с

Таблица 6.2 - Нормы подачи свежего воздуха в помещения, где расположены компьютеры

Характеристика помещения	Объемный расход подаваемого в помещение свежего воздуха, м3 /на одного человека в час
Объем до 20м3 на человека 20…40м3 на человека Более 40м3 на человека	Не менее 30 Не менее 20 Естественная вентиляция

Для обеспечения комфортных условий используются как организационные методы (рациональная организация проведения работ в зависимости от времени года и суток, чередование труда и отдыха), так и технические средства (вентиляция, кондиционирование воздуха, отопительная система).

6.7 Экологичность проекта

Используемые аппаратные средства соответствуют всем требованиям системы стандартов безопасности труда. В процессе эксплуатации отвечают требованиям экологической безопасности.

Люминесцентные лампы, использующиеся для искусственного освещения, являются ртутьсодержащими и относятся к 1 классу опасности.

Основным положением экологической безопасности является сохранность

целостности отработанных ртутьсодержащих ламп, так как ртуть содержащаяся в люминесцентных лампах, способна к активной воздушной, водной и физико-химической миграции. Это возможно в случае разгерметизации колбы.

Отработанные ртутьсодержащие лампы хранятся в заводской упаковке в металлическом шкафу, установленном в закрытом помещении.

В случае боя ртутьсодержащих ламп осколки собираются щеткой или скребком в металлический контейнер с плотно закрывающейся крышкой, заполненной раствором марганцовокислого калия. Место необходимо нейтрализовать раствором марганцовокислого калия и смыть водой. Контейнер и его внутренняя поверхность должны изготавливаться из материала неамальгирующего и неабсорбирующего ртуть (винипласт).

Освещение не должно создавать бликов на поверхности экрана, а освещенность поверхности экрана компьютера не должна быть более 300 лк.

Особое внимание при организации рабочих мест уделяется электромагнитной безопасности. Для того чтобы снизить электромагнитный фон, помещение должно быть удалено от посторонних источников электромагнитных полей, создаваемых мощными трансформаторами, электрическими распределительными щитами, кабелями электропитания с мощными энергопотребителями и пр.

6.8 Защита в ЧС

Требования к пожарной безопасности устанавливает ГОСТ 12.1.004-85 «Система стандартов безопасности труда. Пожарная безопасность. Общие требования».

Пожарная безопасность – состояние объекта, при котором исключается возможность пожара, а в случае его возникновения предотвращается воздействие на людей опасных факторов пожара и обеспечивается защита

материальных ценностей.

Пожарная безопасность обеспечивается системой предотвращения пожара и системой пожарной защиты. Во всех служебных помещениях обязательно должен быть "План эвакуации людей при пожаре", регламентирующий действия персонала в случае возникновения очага возгорания и указывающий места расположения пожарной техники.

Противопожарная защита – это комплекс технических и организационных мероприятий, направленных на обеспечение безопасности людей, на предотвращение пожара, ограничение его распространения, а также на создание условий для успешного тушения пожара.

Источниками зажигания в могут быть электронные схемы от ЭВМ, приборы, применяемые для технического обслуживания, устройства электропитания, кондиционирования воздуха, где в результате различных нарушений образуются перегретые элементы, электрические искры и дуги, способные вызвать загорания горючих материалов.

В современных ЭВМ очень высокая плотность размещения элементов электронных схем. В непосредственной близости друг от друга располагаются соединительные провода, кабели. При протекании по ним электрического тока выделяется значительное количество теплоты. При этом возможно оплавление изоляции. Для отвода избыточной теплоты от ЭВМ служат системы вентиляции и кондиционирования воздуха. При постоянном действии эти системы представляют собой дополнительную пожарную опасность.

К средствам тушения пожара, предназначенных для локализации небольших возгораний, относятся пожарные стволы, внутренние пожарные водопроводы, огнетушители, сухой песок, асбестовые одеяла и т.д.

В зданиях пожарные краны устанавливаются в коридорах, на площадках лестничных клеток и входов. Применение воды в машинных залах ЭВМ ввиду

опасности повреждения или полного выхода из строя дорогостоящего оборудования возможно в исключительных случаях, когда пожар принимает угрожающе крупные размеры.

Для тушения пожаров на начальных стадиях широко применяются огнетушители. По виду используемого огнетушащего вещества огнетушители подразделяются на следующие основные группы: пенные, газовые и углекислотные.

Пенные огнетушители, применяются для тушения горящих жидкостей, различных материалов, конструктивных элементов и оборудования, кроме электрооборудования, находящегося под напряжением.

Газовые огнетушители, применяются для тушения жидких и твердых веществ, а также электроустановок, находящихся под напряжением.

В помещениях применяются главным образом углекислотные огнетушители, достоинством которых является высокая эффективность тушения пожара, сохранность электронного оборудования, диэлектрические свойства углекислого газа, что позволяет использовать эти огнетушители даже в том случае, когда не удается обесточить электроустановку сразу.

6.9 Заключение о безопасности и экологичности проекта

Проведен анализ опасных и вредных факторов, воздействующих на оператора ЭВМ, работающего с разделенной файловой системой.

Одними из наиболее значимых факторов безопасности является:

- электробезопасноть;
- освещение;
- пожаробезопасность;

С целью обеспечения безопасных и комфортных условий труда оператора выполнены расчеты:

- защитного заземления;
- искусственного освещения;
- электромагнитного излучения;

В целом можно заключить, что рассмотренные условия труда оператора ПК, отвечают требованиям стандартов по обеспечению промышленной и экологической безопасности.

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы аналоги разрабатываемой программы, найдены их уникальные особенности, построена структура программы, разработаны используемые алгоритмы. В следствии исследования предметной области выявлены следующие положительные особенности файловых систем, которые решено использовать в разрабатываемой системе, к ним относятся:

- архитектура файловой системы является клиент-серверной, без наличия сервера транзакций, и метаданных, как в GlusterFS в отличии от Lustre и HDFS это упростит ее функционирование, хотя может вызвать проблемы с сконфигурированным нескольких клиентов;
- файловая система кэширует файлы на стороне клиента, благодаря чему повышается надежность системы. Даже при сбое в сети она продолжает частично функционировать, что так-же упростит функции сохранения данных на другие компьютеры при включении или выходе из строя компьютера на который производится запись подобно AFS;
- идея трансляторов и модулей, позволит достичь более структурированный код, и даст возможность расширения файловой системы по мере необходимости без модификации самого ядра системы, таким образом как это сделано в GlusterFS;
- использовать минимально возможный размер заголовка в пакетах при передаче содержимого файлов, для того, чтобы достичь минимальные потери в скорости на заголовках пакетов, в отличии от GlusterFS;
- реализовать несколько алгоритмов выбора компьютера, на который производится запись как отдельные модули, реализовать обработку запросов от клиентов отдельными модулями, в том числе аутентификацию и файловые операции, подобно GlusterFS;

- в протоколе должны быть ping пакеты подобно системе HDFS, которые будут отправляться к клиентам от серверов, и если клиент не получает информацию, в течении некоторого промежутка времени, то сервер будет считаться «умершим» и удален из списка серверов внутри клиента;
- реализовать два отдельных независимых протокола для данных, которые предают содержимое файлов и для метаданных, как в Hadoop и HDFS;
- в качестве библиотеки для построения файловой системы выбрать FUSE.

Так-же в процессе алгоритмического конструирования разработаны два протокола обмена данными между клиентом и сервером, которые использованы в разрабатываемой программе, они имеют следующие особенности:

- решено выбрать в качестве базового протокола для обмена метаданными протокол XML и расширить его таким образом чтобы можно было реализовать функции файловой системы. А в качестве XML анализатора выбрать Expat;
- разработан бинарный протокол для передачи содержимого файлов.

Для конфигурации программы были проанализированы библиотеки, позволяющие создавать файлы конфигурации и принято решение использовать библиотеку XFLib, как наиболее простой и легко читаемый и анализируемый формат представления данных. Коме того была разработана структура конфигурационного файла для сервера и клиента

В результате проделанной работы подготовлена необходимая информация для создания файловой системы, разработаны алгоритмы и структура модулей программы.

7 СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Обзор сетевых файловыз систем nfs, lustre и gfs (nfs lustre gfs) [Электронный ресурс] -– URL: http://www.opennet.ru/base/net/network_fs_review.txt.html

2. Google File System [Электронный ресурс] -– URL: http://ru.wikipedia.org/wiki/Google_File_System

3. SSHFS [Электронный ресурс] -– URL: http://ru.wikipedia.org/wiki/SSHFS

4. GmailFS - делаем хранилище файлов из почты Gmail [Электронный ресурс] -– URL: http://easylinux.ru/node/288

5. Сайт WikipediaFS [Электронный ресурс] -– URL: http://wikipediafs.sourceforge.net/

6. A FTP filesystem based on cURL and FUSE [Электронный ресурс] -– URL: http://curlftpfs.sourceforge.net/

7. Сайт проекта LTFP [Электронный ресурс] -– URL:http://lftpfs.sourceforge.net/

8. Разработка собственной файловой системы с помощью FUSE [Электронный ресурс] -– URL: http://www.ibm.com/developerworks/ru/library/l-fuse/index.html

9. Сайт проекта OpenAFS [Электронный ресурс] -– URL: http://www.openafs.org/

10. LUSTRE [Электронный ресурс] -– URL: http://www.insight-it.ru/net/scalability/lustre/

11. GlusterFS [Электронный ресурс] -– URL: http://www.gluster.com

12. GlusterFS [Электронный ресурс] -– URL: http://www.insight-it.ru/net/scalability/glusterfs/

13. InfiniBand [Электронный ресурс] -– URL:

http://ru.wikipedia.org/wiki/InfiniBand

14. Hadoop [Электронный ресурс] -– URL: http://www.insight-it.ru/net/scalability/hadoop/

15. FUSE [Электронный ресурс] -–URL: http://ru.wikipedia.org/wiki/Filesystem_in_Userspace

16. FUSE [Электронный ресурс] -– URL: http://fuse.sourceforge.net

17. XMPP [Электронный ресурс] -– URL: http://xmpp.org

18. LibXML2 [Электронный ресурс] -– URL: http://xmlsoft.org

19. Expat [Электронный ресурс] -– URL: http://expat.sourceforge.net

20. XF [Электронный ресурс] -– URL: http://xfhome.org

ПРИЛОЖЕНИЕ А ТЕХНИЧЕСКОЕ ЗАДАНИЕ

«СОГЛАСОВАНО»	«УТВЕРЖДЕНО»
Руководитель дип. проекта:	зав. кафедрой «ПОВТ и АС»
Клубков И.М. _____	Нейдорф Р.А. _____
« ___ » _____ 2011 г.	« ___ » _____ 2011 г.

П.А.1 Наименование

Наименование программного средства — «Распределенная файловая система».

П.А.2 Область применения

Разрабатываемое программное средство может применяться в сфере информационных технологий.

П.А.3 Основание для разработки

Разработка ведется на основании документа «Учебный план для студентов ВУЗа», факультета «Информатика и вычислительная техника» (ИиВТ) направления 230000 «Информатика и вычислительная техника» специальность 230105 «Программное обеспечение вычислительной техники и автоматизированных систем» Донского Государственного Технического Университета (ДГТУ), утвержденного министерством общего и профессионального образования 22.09.2002г., в соответствии с которым студенты, заканчивающие ВУЗ, должны предоставить к защите выпускную квалификационную работу, подтверждающую присвоение им квалификации «Инженер». Предметным основанием является задание на дипломную работу.

П.А.4 Назначение разработки

П.А.4.1 Функциональное назначение

Функциональное назначение разрабатываемой программы — создание общего дискового пространства группы компьютеров, объединенных в сеть.

П.А.4.2 Эксплуатационное назначение

ПО предназначено для эксплуатации на серверах и/или рабочих станциях, с целью образования единого дискового пространства.

П.А.5 Технические требования к программе или программному изделию

П.А.5.1 Требования к функциональным характеристикам

Программа должна организовывать единое дисковое пространство для групп серверов, которые позволят сохранять в него данные пользователей. Дисковое пространство должно поддерживать:

- создание файлов;
- запись данных в файлы;
- чтение файлов;
- изменение и чтение прав доступа на файлы;
- просмотр и создание директорий.

П.А.5.2 Требования к надежности

Надежность программного средства должна обеспечиваться:

- обработкой фатальных ошибок;
- обработкой всех исключительных ситуаций в процессе работы программы;
- обработкой ошибок связанных с недостаточностью дискового

пространства при записи на диск, и производить выбор нового компьютера;

- производить кэширование данных на стороне клиента для повышения надежности;
- обработка ошибок связанных с разрывом сетевого соединения в момент сеанса работы.

П.А.5.3 Условия эксплуатации

Для функционирования программы необходимо выполнять условия эксплуатации ЭВМ и носителей информации.

Климатические условия эксплуатации: условия, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

Требования к видам обслуживания: программа не требует проведения каких-либо видов обслуживания.

Требования к квалификации персонала: для конфигурирования и начальной установки программного продукта необходим специалист с квалификацией “инженер-программист”.

Для эксплуатации данной программы достаточно одного человека, получившего квалификацию оператора ПК или более высокую квалификацию.

П.А.5.4 Требования к составу и параметрам технических средств

Для функционирования программы необходимо выполнить системные требования:

- два или более ПК с UNIX совместимыми архитекторами (alpha, amd64, arm, armel, hppa, i386, ia64, mips, mipsel, powerpc, s390, sparc);
- объем ОЗУ не менее 32 МБ в зависимости от платформы;

- свободное место на жестком диске не менее 10 МБ для программы, а также некоторое количество места для кэша и распределенных данных;
- наличие сетевого интерфейса.

П.А.5.5 Требования к информационной и программной совместимости

Требования к операционной системе, наличию библиотек и языкам программирования:

- UNIX совместимая ОС (Linux с версией ядра 2.6, либо либо NetBSD, FreeBSD, OpenSolaris);
- наличие Expat — динамической библиотеки времени выполнения expat, библиотеки для работы с XML;
- наличие динамической библиотеки XFLib — анализатора формата XF;
- наличие библиотеки FUSE для клиентской части — интерфейс файловой системы в пространстве пользователя;
- компилятор C++.

П.А.5.6 Требования к программной документации

Документация должна включать:

- техническое задание (ГОСТ 19.201);
- руководство системного программиста (ГОСТ 19.503);
- руководство программиста (ГОСТ 19.504);
- руководство оператора (ГОСТ 19.505).

П.А.6 Технико-экономические показатели

К программе не предъявляется экономических требований.

П.А.7 Стадии и этапы разработки

Стадии и этапы разработки программного средства:

- сформулирована постановка задачи (с 01.02.2011 по 05.02.2011);
- изучение предметной области (с 06.02.1011 по 10.02.2011);
- разработка технического задания (с 11.02.11 по 14.02.11);
- согласование и утверждения технического задания (с 14.02.2011 по 15.02.2011);
- разработка методов решения поставленной задачи (с 15.02.2011 по 18.02.2011);
- алгоритмическое конструирование (с 15.02.2011 по 20.02.2011);
- разработка интерфейса программы (с 17.02.2011 по 18.02.2011);
- написание текста программных модулей (с 21.02.2011 по 07.03.2011);
- отладка модулей программы (с 06.03.2011 по 12.03.2011);
- разработка и испытание контрольных тестовых примеров (с 23.02.2011 по 5.03.2011);
- разработка руководств системного программиста, программиста, оператора (с 25.02.2011 по 7.03.2011);
- разработка пояснительной записки к данному программному продукту (с 22.02.2011 по 07.03.2011).

П.А.8 Порядок контроля и приемки

Порядок и контроль приёмки определяются заведующим кафедрой «ПОВТ и АС» и основан на демонстрации знаний технологии и умении создавать программные средства для различных предметных областей. Главным требованием к приемке является наличие правильно работающего программного модуля, иллюстрируемого тестовым примером, и отчета, представленного в

печатном виде.

Разработал

студент гр. ВИ-51 «___» _____ 2011 год

Лубягов Н.А. _____

ПРИЛОЖЕНИЕ Б РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА

П.Б.1 Общие сведения о программе

Программа «Распределенная файловая система» предназначена для создании общего дискового пространства для группы компьютеров, соединенных в сеть, каждый из которых имеет доступ к этому пространству. Программа обеспечивает:

- создание файлов;
- запись данных в файлы;
- чтение файлов;
- изменение и чтение прав доступа на файлы;
- просмотр и создание директорий.

Программа реализована на языке C++. Среда разработки – Eclipse,и компилятором G++ пакета GNU Compiler Collection.

Для выполнения программы необходимы следующие технические средства (минимальные требования):

- два или более ПК с UNIX совместимыми архитекторами (alpha, amd64, arm, armel, hppa, i386, ia64, mips, mipsel, powerpc, s390, sparc);
- объем ОЗУ не менее 32 МБ в зависимости от платформы;
- свободное место на жестком диске не менее 10 МБ для программы, а также некоторое количество места для кэша и распределенных данных;
- наличие сетевого интерфейса.

Для функционирования программы требуется:

- UNIX совместимая ОС (Linux с версией ядра 2.6, либо либо NetBSD, FreeBSD, OpenSolaris);
- наличие Expat — динамической библиотеки времени выполнения expat,

библиотеки для работы с XML;

- наличие динамической библиотеки XFLib — анализатора формата XF;
- наличие библиотеки FUSE для клиентской части — интерфейс файловой системы в пространстве пользователя;
- компилятор C++.

П.Б.2 Структура программы

Программа состроит из двадцати пяти модулей, в каждом из которых описан класс для решения конкретной подзадачи.

П.Б.3 Настройка программы

Для настройки приложений требуется редактирование конфигурационных файлов, сервера и клиента. Конфигурационные файлы реализованы с помощью библиотеки XFLib и соответствую формату XF. Состоят из двух секций, первая описывает настройки относящиеся как к серверу так и к клиенту, вторая описывает индивидуальные настройки каждого приложения. Общая секция имеет вид:

```
conf{
* сеция глобальных данных как сервера так и клиента
  global_data{
    extension_dir="<путь к папке с расширениями>";
    module_dir="<путь к папке с модулями>";
  }
}
```

В ней задаются пути для поиска модулей и расширений протокола. У каждого модуля могут быть свои параметры, параметры модулей храятся в секции module_data, которая имеет следующий формат:

```
conf{
  module_data{
     <название модуля>{
       <название параметра>=значение параметра;
     }
  }
}
```

П.Б.3.1 Конфигурирование сервера

Для конфигурирования сервера требуется отредактировать serverconfig.conf.

```
conf{
* секция данных индивидуальных для сервера
  server_data{
    server_ip=<IP на котором ожидается подключение клиентов>;
    bibary_protocol_ip=<предпочтительный IP для бинарного протокола>;
    server_port=<Порт для подключения конечных клиентов>;
    shared_dir="<путь к папке с общедоступными данными>"
  }
* секция в которой будут храниться данные расширений и модулей
}
```

П.Б.3.2 Кон фигурирование клиента

Для конфигурирования клиента требуется отредактировать файл clientconfig.conf

```
conf{
* сеция глобальных данных как сервера так и клиента
```

```
 global_data{
   extension_dir="<путь к папке с расширениями>";
   module_dir="<путь к папке с модулями>";
 }
* секция данных индивидуальных для клиента
 client_data{
* секции серверов к которым подключается данный клиент.
  servers{
   server{
     server_ip=<"IP адрес сервера">;
     server_port=<"порт сервера">;
   }
  }
 }
* секция в которой будут храниться данные расширений и модулей
 module_data{
   <название модуля>{
     <название параметра>=значение параметра;
   }
 }
}
```

На данный момент реализован только один расширяемый модуль, LibFileOperation. Его секция предсталена следующим образом:

```
lib_file_opertion {
      max_packet_size=40960;
}
```

max_packet_size — максимальный размер пакета, передаваемых по сети

данных через бинарный протокол при чтении и записи файла.

П.Б.4 Проверка программы

Проверка программного средства осуществлялась в процессе разработки при помощи тестов. Тестирование ПС в целом прошло успешно.

Результаты тестирования и экспериментов позволяют утверждать, что программное средство работает корректно.

П.Б.5 Сообщения системному программисту

Клиентское приложение может сообщать об ошибках, связанных с работой ФС, при этом они записываются в лог файл. А так-же при запуске в режиме отладки, возможно получение информации от модуля FUSE в консоль. В остальных случаях, приложение является демоном, и не возвращает никакой дополнительной информации. Серверное приложение выводит некоторую информацию о подключившихся клиентах и выполняемых операциях, в лог файл.

Все сообщения в лог файле содержат дату, время происхождения события и текст сообщения: Wed Nov 2 23:47:54 2011 запуск клиента.

Общие для клиента и сервера информационные сообщения:

- загрузка модуля: имя файла модуля — информирует о загрузке модуля расширения, с указанным именем файла;
- удаление модулей — информационное сообщения об удалении модулей при завершении работы программы;
- удаление ноды из-за не уникальности ее пространства имен: пространство имен ноды — сообщение о том, что внутри одного листа XML дерева, был добавлен второй потомок, с одинаковым пространством имен xmlns.

Общие для клиента и сервера сообщения об ошибках:

- ошибка открытия библиотеки модуля:код ошибки от dlerror — информирует об ошибке выполнения вызова dlopen, при загрузке модуля расширения;
- ошибка получения адреса функции registerModule модуля: код ошибки dlerror — ошибка выполнения системного вызова dlsym, при получении адреса функции регистрации модуля;
- ошибка получения адреса функции unregisterModule модуля: код ошибки dlerror — ошибка выполнения системного вызова dlsym, при получении адреса функции удаления регистрации модуля.
- ошибка, отсутствует мьютекс, ожидания пакета бинарного протокола — Отсутствует мьютекс в структуре, с указанным ID, при ожидании пакета по бинарному протоколу;
- ошибка: получен пакет, с не зарегистрированным ID :Идентификатор пакета — не критическая ошибка, свидетельствует о получении пакета с идентификатором, для которого в данный момент не зарегистрирован обработчик, в бинарном протоколе.
- получен не корректный XML: описание ошибки, на линии: номер строки в потоке XML — Ошибка свидетельствующая о том что получены не корректные данные по протоколу на основе XML, в результате данной ошибки противоположная сторона будет отключена.

Информационные сообщения клиента:

- запуск клиента — добавляется в лог при запуске клиентского приложения;
- сервер не достижим IP: адрес порт: порт — указанный в конфигурационном файле сервер, с данными адресом и портом на данный момент не достижим;

- добавление сервера id: идентификатор сервера — добавление нового сервера, в список обрабатываемых серверов.

Сообщения об ошибках клиента:

- ошибка: невозможно создать сокет — невозможно создать сокет для подключения к серверу;
- ошибка: адрес сервера не относится к категории корректных адресов: указанный адрес — указан не корректный адрес сервера в конфигурационном файле.
- ошибка: адрес сервера не содержит корректного IP-адреса:адрес сервера — получен не корректный IP адрес сервера.

Информационные сообщения сервера:

- запуск сервера — добавляется в лог при запуске серверного приложения;
- клиент подключился — сообщение о подключении нового клиента;
- клиент отключился — свидетельствует об отключении ранее подключенного клиента к серверу.

Сообщения об ошибках сервера:

- ошибка принятия — ошибка при выполнении функции accept сокета, сообщающая о невозможности принять клиента;
- ошибка при создании сокета — некорректно выполнен системный вызов socket, и не-был создан сокет сервером;
- ошибка связывания — некорректно выполнен системный вызов bind;
- ошибка прослушивания — некорректно выполнен системный вызов listen;
- ошибка открытия файла: путь к файлу Номер ошибки: значение errno — не критическая шибка, при открытии сервером файла, на запись или чтение, после ранее, успешно выполненной операции open клиентом,

возвращенная ОС;

- ошибка получения атрибутов файла: путь к файлу — не критическая ошибка, свидетельствующая о невозможности получить атрибуты файла запрошенного клиентом;
- ошибка fstat, номер: номер ошибки возвращенный ОС — ошибка, при чтении данных из файла, невозможно получить атрибуты файла;
- ошибка чтения — невозможно прочесть ранее открытый на чтение файл;
- нет ответа от клиента о подтверждении передачи содержимого — истечение времени ожидания подтверждении, ответа о передаче данных при записи файла.

ПРИЛОЖЕНИЕ В РУКОВОДСТВО ПРОГРАММИСТА

П.В.1 Назначение и условия применения программы

Назначение разрабатываемой программы заключается в создании общего дискового пространства для группы компьютеров, соединенных в сеть, каждый из которых имеет доступ к этому пространству. ПО предназначено для эксплуатации на серверах и/или рабочих станциях. Для работоспособности программного средства требуется два или более компьютеров, они могут являться как персональными компьютерами так и серверами. Взаимодействие с программой осуществляется приложениями работающими с файловой системой, которые выполняют операции над ФС, посредством стандартных системных вызовов.

П.В.2 Характеристика программы

Время выполнения файловых операций зависит от объемов передаваемых данных, скорости и загруженности сетевого соединения, частоты процессора сервера и клиента, скорости чтения и записи данных носителей на которых расположены общедоступные директории.

Оба приложения работают в режиме демона, и не имеют интерфейса.

Приложение «Распределенная файловая система» реализовано на языке программирования C++.

Для проверки работоспособности сконфигурированной и настроенной программы можно провести несколько операций над файлами, такими как переименование, запись, чтение, удаление файлов. Если в результате данных операций не будет выявлено ошибок, а записываемые данные будут совпадать с прочитанными, то программа работает верно.

П.В.3 Обращение к программе

Обращение к программе производится через какую либо командную оболочку. Целесообразно как серверную так и клиентскую части запускать через init.d либо rc.d скрипты, поместив скрипт запуска в директорию определенную операционной системой, и сконфигурировав их в соответствии с правилами ОС.

Серверная часть программы «Распределенная файловая система» может быть запущена вызовом исполняемого файла serverfsp, клиентское приложение можно запустить выполнив clientfsp <точка монтирования файловой системы>, так-же может быть передана опция -d для запуска клиента в режиме отладки. В таком случае приложение не будет отвязано от консоли и запущено как демон. Запуск приложений должен производится из директории где расположен конфигурационный файл.

П.В.4 Описание программного кода

П.В.4.1 Библиотека протоколов обмена данными ФС libfsprotocol.so

Структура BibaryPacketHeader описывает заголовок пакета бинарного протокола

unsigned long int pktID — Идентификатор пакета.

unsigned short headSize — размер заголовка пакета.

size_t bodySize — размер тела пакета.

Структура BibaryPacket — описывает пакет бинарного протокола:

BibaryPacketHeader header; — заголовок пакета.

void * body — данные, тело пакета.

Класс BinaryProtocol, описывает бинарный протокол обмена содержимым файлов

pthread_mutex_t isConnectedMutex — мьютекс для синхронизации доступа к isConnected.

bool isConnected — опердилят подключен ли в данный момент, если нет то функции пердачи/приема данных игнорируются.

pthread_mutex_t exit_mutex; — мьютекс ожидания завершения обмена данными в деструкторе класса.

pthread_mutex_t cs_mutex4 — мьютекс для защиты отправки пакета из двух потоков параллельно.

pthread_mutex_t cs_mutex3 — мютекс для синхронизации bufferFillers.

pthread_attr_t attr — атрибуты, для отвязки потока, обработки.

size_t max_pktsize — размер тела пакета, берется из конфигурационного файла.

BaseConfigFile* cf — указатель на объект конфигурационного файла.

int skt_id — дескриптор сокетного соединения обмена данными.

pthread_t binThread — поток внутри котором идет разбор приходящих данных.

void parse() — метод обработки разбора и приема данных.

map<unsigned long int,BufferFiller*> bufferFillers — Обьект с заполнителями, ключ — номер пакета, BufferFiller — объект содержащий указатель на буфер куда придут данные, и другие вспомогательные элементы.

bool getIsConnected() - ассинхронно безопасный доступ к isConnected.

void setIsConnected(bool v) — ассинхронно безопасная присвоение в isConnected.

BibaryPacket readPacket() - получение, текущего пакета из сокета.

ssize_t readToBuffer(void* buffer, size_t length) — чтение произвольных данных из сокета.

BinaryProtocol(BaseConfigFile* cf) — конструктор, cf указатель на объект конфига.

static unsigned long int getNextId() - ассинхронное вычисление нового идентификатора пакета.

void sendBuffer(const void* buffer, int length) — отправка буфера на через сокет.

void sendPacket(BibaryPacket packet) — отправка пакета, через сокет.

void setFillBuffer(void* buffer,size_t size,unsigned long int pktID) — добавление заполнителя для данных пришедших из пакетов с идентификатором pktID, размером size в буффер buffer.

size_t waiteFillBuffer(unsigned int pktID) — ожидание заполнения буффера пакетами с указанным идентификатором.

void setSendBufferData(const void* buffer,size_t size,unsigned long int pktID) — установка на отпарвку данных по бинарному протоколу, предварительно упакованных в паеты с идентифкатором pktID, из буффера buffer имеющего размер size.

void startTread(int skt_id) — запуск нового потока для обработки, из и приемм передачей данных из сокетного соединения skt_id.

Класс BinaryProtocolClient

Описывает уникальную часть бинарного протокола для клиента.

bool connect_server(string address, unsigned short port) — подключение клиента к серверу, где address это домен или адерс сервера, port номер порта TCP/IP.

BinaryProtocolClient(ClientConfigFile* cf) — конструктор, cf ссылка на конфигурационный файл.

Класс BinaryProtocolServer

Описывает серверную часть бинарного протокола

BinaryProtocolServer(BaseConfigFile* cf) — конструктор cf, указатель на объект конфигурационного файла.

Структура BufferFiller

Используется для заполнения данными буфера из пакетов.

unsigned int pktID — идентификатор пакета.

void* bufferS — буффер в который заполняются данные.

void* bufferC — текущая позиция в буфере куда заполнять дальше.

unsigned long int forRead — объем который все еще необходимо получить.

unsigned long int bufferSize — общйи размер буффера.

bool allReaded — если буффер заполнен, то true иначе false.

pthread_mutex_t mutex — мтютекс ддя allReaded.

pthread_cond_t cond_var — переменная состояния генерирует сигнал при заполнении буфера, при этом может быть выставлен allReaded, тогда поток ожидающий на waiteFillBuffer пробуждается.

Класс протокола на основе XML

Интерфейс TagHandler

Классы обработки запросов, должны реализовывать этот интерфейс, Шаблон специализирует тип обрабатываемых тегов, и может принимать

значение TagIQ для обработки IQ запросов, либо Stanza для обработки произвольных станз полученных из протокола.

virtual void handleTag(T &tag)=0 — метод, будет вызываться когда приходит указанный тег.

typedef TagHandler<T>* PTagHandler — тип указатель на экземпляр объекта, описан внутри интерфейса

typedef multimap<pair<string,string>, TagHandler<Node>::PTagHandler> TagHandelrsMapT — тип данных описывает обработчик одной стнзы, произвольного типа.

Структрура IQHandleD, описывает условия выполнения обработчиков IQ запросов, и определяет на них указатель:

string xmlns — пространство имен.

string subtagName — имя вложенного тега <iq> <тег/> </iq>.

TagHandler<TagIQ>::PTagHandler handler — указатель на экземпляр объекта реализующего интерфейс PtagHandler, для обработки IQ запросов.

typedef list<IQHandleD> IQHandelrsMapT — тип данных описывает список указателей на обработчики IQ запросов.

Класс Protocol, описывает протокол на основе XML

pthread_mutex_t cs_mutex — мьютекс для синхронизации параллельной отправки данных через сокет.

pthread_mutex_t cs_mutex1 — мьютекс синхронизации обращения и модификации req_id.

pthread_mutex_t cs_mutexNewNode — мьютекс для синхонизации добавления и удаления нод, в дерево при приеме данных.

pthread_mutex_t cs_mutexDEL — мьютекс синхронизации модификации и обращения к iqHandlers.

pthread_mutex_t mutDelEndNodRu — мьютекс синхронизации обращения к

num_run_threads.

unsigned int num_run_threads — число одновременно запущеных потоков, обработчиков станз.

pthread_cond_t cond_del_element — переменная состояния, сигнал срабатывает при модификации num_run_threads, с применением pthread_mutex_t.

pthread_attr_t attr — атрибуты потоков обработчиков IQ запросов, делают потоки отвязанными, благодаря чему сокращается время их запуска.

multimap<string,IQRespD> req_id — хранит ответы на IQ запросы, полученные с противоположной стороны.

char read_Buffer[BUFFSIZE] — read_Buffer буффер в который принимаются данные из сокета.

int skt_id — идентификатор сокета, через который производится передача данных.

XML_Parser xml_parser — структура, описывает XML парсер expat, который производит разбор XML.

pthread_mutex_t curDephMutex — мьютекс для синхронизации curDeph.

int curDeph — текущий уровень, дерева XML на котором находится разбераемая нода.

Node* topNode — указатель на верхний лист дерева.

Node* curNode — указатель на текущий лист дерева.

Node* curNodeSend — верхняя нода в которую добавляются отправляемые станзы.

void handlerNode(Node* curNodeL) — вызывает зарегистрированные обработчики нод, подходящие для curNodeL.

void handlNodes(Node* curNodeL, TagHandelrsMapT::iterator begin,TagHandelrsMapT::iterator end) — вызывает обрабротчики для curNodeL,

лежащие в диапазоне от iterator_begin до iterator_end.

void handlerIQ(Node* curNodeL) — вызов обработчиков для IQ заросов, в процессе вызова будет вызван метод manadgeIQResp.

void sendNodeStart(Node &stanza) — отправка начальной ноды, в процессе обмена данными.

void sendNodeEnd(Node &stanza) — отправка конечной ноды в процессе обмена данными.

void removeALLWaites() — особождение всех ожидающих ответа обработчиков.

TagHandelrsMapT tagHandlers — словарь обработчиков тегов.

IQHandelrsMapT iqHandlers — словаь обработчиков IQ запросов.

pthread_t nodeThread — поток в котором производится обработка станз.

void parserCreate(int sktID) — создание парсера, sktID дескриптор сокетного соединеия.

Node* getCurNodeSend() — возвращает текущую ноду для отправки.

void sendNode(Node &stanza) — отправ ляет ноду через сокетное соединение.

void addTagHandler(TagHandler<Node>::PTagHandler handler, string name="", string xmlns="") - добавляет новый обработчик произвольного тега, handler указатель на обработчик, name имя ноды xmlns пространство имен ноды, любой параметр может быть пустым, если пусты все параметры, обработчик вызовется для любой ноды.

void addIQHandler(TagHandler<TagIQ>::PTagHandler handler, string xmlns="", string name=QUERY_TAG) — добавляет в список обработчик IQ запросов, handler обрабоичк xmlns — пространство имен вложенного тега, name — имя тега, по умолчанию iq. В случае если какой-то параметр пуст, то выполняется для произвольного значения этого параметра.

void removeIQHandler(TagHandler<TagIQ>::PTagHandler handler) — удаления обработчика IQ запроса.

void startParsing() — вход в цикл разбора, будет продолжаться до отключения клиента.

virtual void startXMLTagNodeHandler(Node *node) — виртуальный метод может быть перекрыт в производных классах, для обработки начальной ноды XML потока полученной с противоположной стороны.

virtual void endXMLTagNodeHandler(Node *node) — виртуальный метод, может быть перекрыт в производных классах, для обработки конечной ноды XML потока полоученной с противоположной стороны.

void sendString(string data) - Отправка в сокет произвольного не XML содержимого, либо строк сформированных в виде XML.

void manadgeIQResp(TagIQ& st) — разблокировка мьютексов ожтидающих обработки переданного IQ запроса st.

TagIQ sendIQResp(TagIQ& st, string resultXMLNS, string resultSubTagName) — интерактивный обмен, в формате запрос ответ. Отправлет IQ запрос, ждет ответа, и после получения пробуждается и возвращает его.

Функции не являющиеся членами класса:

friend void startTag(void *userData,const XML_Char *name, const XML_Char **atts) — обработчик начала XML тега принятого из потока.

friend void endTag(void *userData,const XML_Char *name) — обработчик конца XML тега принятого из потока.

friend void characterData (void *userData,const XML_Char *s,int len) — обработчик произвольных текстовых данных внутри XML тега.

friend void* node_threadFunc(void* clnt) — Функция потока, внутри которой происходит обработка одной станзы.

Класс Server

Реализует серверную часть XML протокола

unsigned int clientID — порядковый номер клиента, который обрабатывает класс.

bool disconnectingSelf — если отключаемся сам, а не клиент примет значение true.

virtual void startXMLTagNodeHandler(Node* node) — перекрыт обработчик начала тега, создает и отправлет клиенту начальную ноду.

virtual void endXMLTagNodeHandler(Node* node) — перекрыт обработчик конечной ноды, в случае отключения по инециативе клиента, отправлет конечную ноду, и унечтожает текущую ноду для отправки.

unsigned int getClientID() - возвращает текущий идентификатор клиента.

void setClientId(unsigned int clntid) — устанавливает текущий идентификатор клиента.

void disconnectClient() - производит разрыв соединения с клиентом.

Класс Client

Реализует клиентскую часть протокола на основе XML

bool connect_server(string address, u_int16_t port) — подключение клиента к серверу, address — доменное имя, или IP адрес сервера, port TCP/IP порт через который устанавливается связь.

void createNodeSend() — создание и отправка начальной ноды.

virtual void startXMLTagNodeHandler(Node* node) — перекрыт обработчик начального тега, пришедшего от сервера.

virtual void endXMLTagNodeHandler(Node* node) — перекрыт обработчик конечного тега пришедшего по XML протоколу, отправляет конечную ноду на сервер.

Класс Node

Реализует построение XML дерева, для организации обмена по XML протоколу.

string XMLescape(string txt) — функция звменяет символы такие как < > & " на < > & " соответственно.

Члены класса:

pthread_mutex_t add_remove_mutex — мьютекс защищающти от асинхронного добавления и удаления узлов в дерево, в разных потоках.

string name — имя узла, описываемое классом.

map <string, string> attrs — атрибуты узла - ключь, значение.

Node* parent — указатель на родительский узел.

string xmlBody — текстовое содержимое узла.

list <Node*> kinds — дети узла, нижележащие листья.

void destroyKinds() - уничтожение всех потокмков узла, и удаление их из памяти.

virtual void copyKinds(const Node &basedNode) — копирование потомков из другого такого же узла, используется в конструкторе копирования и перегруженном операторе присваивания.

bool parentContainSelf() - возвращает true если родительский узел содержит данный, если это копия сформированной станзы второго уровня, то родительский не должен содержать себя, используется в случае задания отрибута xmlns, если true то при этом будет удален брат узла с таким-же пространством имен если он есть.

bool isCopy — возвращает true если копия, иначе false.

Public члены класса:

Node(string name, map <string, string> /*&*/attrs, Node* parent) - Конструктор, создает элемент ноды, parent-Указатель на родительскую ноду,

name- имя создаваемого листа в дереве, attrs-атрибуты ноды.

Node(string name, Node* parent=NULL,string xmlns="") - констрктор, создает элемент ноды string name — имя листа, parent — указатель на родительский элемент, xmlns — пространство имен, создаваемого элемента.

list <Node*>::iterator firstChild() — возвращает итератор на первый потомок в дереве.

list <Node*>::iterator endChild() — возвращает итератор на последний потомок в дереве.

string getName() — возвращает имя узла.

Node* getParent() - возвращает указатель на предка узла.

void setName(string n) — устанавливает новое имя узла.

void setNamespace(string xmlns) — устанавливает пространство имен ущла, эквивалентно вызову seAttribute(«xmlns», значение).

void setAttribute(string name, string value) — устанавливает значение атрибута узла, в строку.

void setAttribute(string name, long value) — перегруженная функция, устанавливает значение атрибута узла, в целое число.

string getAttribute(string name) — возвращает строку с атрибутом узла.

long getAttributeInt(string name) — возвращает значение атрибута, если он число.

string getNamesapace() - возвращает пространство имен узла

void removeKind(Node* node) — удаляет из узла потомок node, если он есть.

void deleteByXMLNS(string xmlns) — удалет из узла потомок с указанным xmlns.

string toXMLString() - формирует строку, представляющую собой xml ноду.

string toXMLStringStart() - возвращает строку содержащую заголовочный тег <имя тега атрибут1=значение1, атрибут2=значение2, атрибутN =значениеN>.

string toXMLStringEnd() - возвращает закрывающийся тег </имя тега>.

void setXMLBody(string body) — устанавливат текстовое содержимое внутри тега <имя_тега> текстовое содержимое </имя тега>.

void addXMLBody(string body) — добавляет к уже имеющимуся текстовому содержимому, новый текст, при этом данная операция может быть вызванна ассинхронно в разных потоках, что не приведет к потерям данных.

void setXMLBody(long body) — перегруженная функция, в случае если содержимое является числом.

string getXMLBody() — возвращает тестовое содержимое внутри тела тега.

long getXMLBodyInt() - возвращает содержимое тело тега, если оно является числом.

Node* getKind(string xmlns) — находит потомка, если он есть, с указанным xmlns, если потомка нет, возвращает NULL.

virtual void addChild(Node* child) - добавляет в узел потомка.

Node(const Node &baseNode) — конструктор копирования, создает копию переданной ноды.

_Kinds_iterator kindsNameBegin(string name) — возвращает итератор на первую ноду, среди дочерних листов, имеющую имя name.

_Kinds_iterator kindsNameEnd(string name) — возвращает итератор распоследнюю ноду, среди дочерних листов, имеющую имя name.

friend class Stanza; friend class TagIQ; — Классы, которые имеют доступ защищенному содержимому:

typedef _Kinds_iterator nameiterator — тип данных итератора для вложенных элементов.

Node &operator=(Node &obj) — оператор присваивания, создает копию переданной ноды.

Класс _Kinds_iterator

организует перебор нод, потомков:

list<Node*>::iterator curitem — текущая нода.

list<Node*>::iterator beginitem — первая нода в списке детей.

list<Node*>::iterator enditem — последняя нода в списке детей.

string name — тмя ноды, для которой перебираем.

public члены класса:

_Kinds_iterator &operator++() - перход к следующей ноде.

_Kinds_iterator operator++(int) — перфиксный оператор перехода к следующей ноде.

bool operator==(_Kinds_iterator &i1) — возвращает true если, элементы на которые указан итератор совпадают, т. е. curitem указывают на один адрес памяти.

bool operator!=(_Kinds_iterator &i1) — возвращает отрицания значения оператора ==.

Node* operator*() const — возвращает указатель на ноду, на которую указывает итератор.

Класс Stanza

Наследует класс Node, и содержит одну целеньком сформированную станзу, передаваемую через протокол, на основе XML.

Макроопределения используемые классом:

#define FROM "from" — атрибут от кого отправлена станза.

#define TO "to" — атрибут кому отправлена станза.

#define ID "id" — уникальный идентификатор станзы.

Глобальные опердиления:

static int curid=0 — идентификатор станзы, вычисляется при ее создании.

static pthread_mutex_t mutexID1 = PTHREAD_MUTEX_INITIALIZER — мьютекс для ассинхронного вычисления идентификатора станзы.

Public члены класса Stanza:

virtual ~Stanza() - конструктор по умолчанию, создает пустую станзу.

Stanza(string name, map <string, string> &attrs, Node* parent) — конструктор, name имя тега, attrs — атрибуты, parent — родитель.

Stanza(string name, Node* parent,string from="", string to="", string id="") - конструктор, name — имя тега, parent — родитель, from — от кого, to — кому, id — идентификатор станзы, если не задан будет генерироваться автоматически.

void setID(string id) — устанавливает атрибут идентификатора станзы в указанное значение.

void setFrom(string from) — устанавливает атрибут from в знаечение from.

void setTo(string to) — устанавливает атрибут to в значение to.

string getFrom() — возвращает значение атрибута from.

string getID() — возвращает значение идентификатора станзы.

string getTo() — возвращает значение атрибута to.

static string generateID() — генерирует новый идентификатор станзы, может вызываться если необходимо.

Класс TagIQ

Класс реализует построенеие IQ запросов, реализуемых внутри протокола на основе XML.

Макроопределения используемые классом

#define QUERY_TAG "query" — имя тега внутри iq запроса

класс TagIQ:

Public члены класса:

enum IQType{

Set=0,

Get,

Result,

Error,

Invalid

} — описывает тип IQ запроса, Set в случае установки каких либо параметров на противоположной стороне, Get — получение параметров от противоположной стороны Result — результат выполнения set или get запросов, Error — ошибка, в результате выполнения запроса, Invalid — запрос не корректен.

IQType getType() — возвращает тип запроса.

void setType(IQType type) — Устанавливает новый тип запроса.

virtual void addChild(Node* child) — добавляет в запрос ноду, перекрыт от класса Node.

long int getError(string& type) — получение кода ошибки, если информация об ошибке содержится в запросе, тег будет содержать потомка <error type=«тип ошибки» code= «код ошибки»/>.

void setError(long int errorcode,string type) — установка ошибки, type ее тип, и номер кода errorcode ошибки.

TagIQ(Node* parent=NULL,string from="", string to="", IQType type=Get, string id="",string queryXMLNS="") — конструктор принимает parent — указатель на родительский узел, from — от кого отправляется тег to — кому отправляется тег, type — тип тега, id — уникальный идентификатор запроса, если он не задан, будет генерироваться автоматически, queryXMLNS —

пространство имен вложенного тега, если он задан, то будет создан потомак <query xmlns= «значение queryXMLNS»/> .

TagIQ(const Node &basedNode) — конструктор копирования, либо преобразования типа станзы в указанный вид, принемает Node и формирует на ее основании TagIQ разбирая его атрибуты и потомков.

TagIQ &operator=(TagIQ &obj) — оператор присваивания, создает копию объекта.

Protected члены класса:

IQType subtype — содержит тип IQ запросы.

void setIQTypeF() - просматривает атрибуты type и выставляет тип IQ запроса, используется в конструкторе копирования и операторе присваивания.

virtual void copyKinds(const Node &basedNode) — переопределенный метод, копирования потомков узла.

Классы чтения конфигурационного файла

Класс BaseConfigFile, реализует базовые для клиента и сервера функции чтения конфигурационного файла.

Protected члены класса

xfMap *conf — карта представлена структурой типа xfMap, полученная в результате чтения конфигурационного файла.

xfNode *root — корневой элемент структуры.

xfNode *moduledata_node — лист дерева указывающий на данные модулей.

xfNode* global_node — указывает на лист с глобальными, одинаковыми данными как для сервера так и для клиента.

string moduleDir — содержит путь к папке с расширениями протокола.

Public члены класса:

long int getAttrInt(xfNode* node, long int defval=-1) — получение значения целого типа внутри листа дерева, на который указывает node, значение по умолчанию defval.

string getAttrString(xfNode* node, string defval="") — получение значения, узла node, значение по умолчанию defval.

long int xfCharToInt(xfChar* str) — преобразование типа xfChar в целое число.

string xfCharToStr(xfChar* str) — перобразование типа xfChar в строку типа string.

long int getModuleAttributeInt(wstring modulename,wstring attr,long int defval) — получение целочисленного атрибута модуля, modulename — имя модуля, attr — имя атрибута, defval — значение по умолчанию.

string getModuleAttributeString(wstring modulename,wstring attr, string defval) — получение строкового атрибута модуля, modulename — имя модуля, attr — имя атрибута, defval — значение по умолчанию.

xfNode* getMofuleAttribute(wstring modulename, wstring attr) — получение узла атрибута модуля, имя модуля modulename, attr — имя атрибута.

BaseConfigFile(char* filename) — конструктор принимает путь к конфигурационному файлу.

string getModulesDir() — возвращает директорию где должны находится модули расширения.

xfNode* getModuleDataNode() — возвращает узел, где находятся узлы конфигураций модулей.

Класс ServerConfig

Реализует серверную часть конфигурационного файла, производный от BaseConfigFile.

Protected члены класса:

xfNode* server_node — узел в котором находится настрока уникальная для сервера.

string shareddir — общедоступная директория, в которой находится содержимое ФС.

unsigned short serverPort — порт на котором сервер ждет подключения по протоколу на основе XML.

unsigned short serverBinPort — порт, через который сервер ждет подключения по бинарному протоколу.

string serverIP — IP адрес сервера,

Public члены класса:

unsigned short getServerPort() — получение порта через который сервер будет ожидать подключения.

unsigned short getServerBinPort() — получение порта через который сервер будет ожидать подключения по бинарному протоколу.

string getSharedDir() — получение директории в которой хранятся файлы доступные через данный сервер ФС.

ServerConfig() — конструктор, он вызовет конструктор суперкласса, с конфигурационным файлом «serverconfig.conf».

Класс Logger

Реализует запись логов в файл.

Наследуется от ofstream, и является сингтоном.

Private члены класса:

Logger() — конструктор, не позволяет создавать более одного экземпляра.

Public члены класса:

static Logger& getInstance() — получение объекта, экземпляра класса.

ostream &loghead(ostream &stream) — функция, не являющаяся членом класса, описывающая манипулятор, выводящий заголовок лога.

static Logger &logger=Logger::getInstance() - глобальная переменная logger, для доступа к экземпляру класса.

Модуль SdSync.h

Организует метод синхронизации для некоторых операций.

Структура sdSync состоит из:

pthread_cond_t condVar — переменная состояния, для организации блокировок по значениям переменных isDeleting и value.

pthread_mutex_t mutex_condV — мьтекс используемый для безопасной модификации переменных isDeleting и value.

unsigned int isDeleting — число потоков модифицирующих защищаемый объект.

unsigned int value — число потоков обращающийся к защищаемому объекту.

sdSync() — конструктор обнуляет isDeleting и value, инициализирует мьютексы и переменные состояния.

~sdSync() — деструктор, освобождает мьютексы и переменные состояния.

inline void sdSync_waite (sdSync* sds,bool isD) — блокировка, sds - объект синхронизации, isD состояние. isD принимает значение true при модификации защищаемого объекта, false — при обращении к защищаемому объекту. Если isD равно true, но предварительно была вызвана sdSync_waite, то процесс остановится, и будет ожидать пока не будет вызван sdSync_signal, столько же раз сколько был вызван sdSync_waite, с теми же параметрами. Если isD равно false то блокировка будет только в том случае если число вызовов sdSync_waite, со значением true, больше чем число вызовов sdSync_signal, со значением isD равным true.

inline void sdSync_signal(sdSync* sds, bool isD) — пробуждение потоков на

объекте синхронизации, параметры те-же, что и sdSync_waite.

Загрузка и выгрузка модулей

Класс, ModuleLoader организует загрузку и регистрацию модулей расширений, используется как серверным так и клиентским приложением.

Структура module_regstruct, описывает адреса функций вызываемые при регистрации и удалении регистрации модулей, параметры шаблона, T1 прототип функции регистрации, T2 прототип удаления регистрации модуля.

T1 register_module — указатель на функцию регистрации модуля.

T2 unregister_module — указатель на функцию удаления регистрации модуля.

void* dl_handle — указатель на манипулятор библиотеки модуля.

Класс ModuleLoader

Protected поля класса:

list <module_regstruct<T1,T2>> all_modules — список всех загруженных модулей.

BaseConfigFile* scM — указатель на конфигурационный файл

Public члены класса:

void findModules() — метод, производит поиск файлов в директории module_dir полученной из конфигурационного файла, загружает библиотеки.

void closeModules() — выгружает библиотеки, удаляет модули из структуры.

П.В.4.2 Клиентское приложение файловой системы

Файл main.cpp

int main(int argc, char* argv[]) — содержит функцию main. Она создает экземпляр класса ClientProgramm, и передает в него параметры и их число.

Файл ClientProgramm.cpp и ClientProgramm.h

static struct fuse_operations clientFsoperations — структура, содержит указатели на функции FUSE вызываемые при операциях над ФС.

typedef void (*registerModule)(ServerDataMap* servsData, fuse_operations* clientFsoperations, ClientConfigFile* cf,sdSync* syncObj) — тип данных, указатель на функцию вызываемую при регистрации модуля.

typedef void (*unregisterModule)(int) — тип данных, указатель на функцию вызываемую при удалении регистрации модуля.

Файл ServersData.h

Структура ServerData, описывает сервер к которому подключен клиент.

ServersConfStruct confStruct — параметры подключения к серверу.

Client* cl — указатель на экземпляр объекта клиента, соединенного с сервером через протокол на основе XML.

BinaryProtocolClient* clbin — указатель на экземпляр объекта клиента, соединенного с сервером по бинарному протоколу.

typedef map<unsigned int,ServerData> ServerDataMap — карта со списком серверов, к которым выполняется подключение.

typedef map<int,ServersConfStruct> NCServersDataMap — карта со списком серверов к которым не удалось подключиться.

Класс ClientProgramm:

Наследует класс ModuleLoader, применяя в качестве параметров шаблонов типы registerModule и unregisterModule. Класс реализует основную часть клиентского приложения:

Protected члены класса:

pthread_t clientThread — поток внутри которого происходит обработка ка подключения к серверу, экземпляр потока создается для каждого сервера.

pthread_t newServThread — поток внутри которого происходит периодический опрос серверов, если они работоспособны, то добавляться в список серверов.

ClientConfigFile cf — конфигурационный файл клиента.

ServerDataMap serversData — карта содержит подключенные сервра.

NCServersDataMap notConnectedServers — карта содежит не подключенные сервера.

sdSync sincObj — объект синхронизации, синхронизирует обращение к serversData.

pthread_mutex_t addServMutex — мьютекс, защищает от параллельного обращения к notConnectedServers.

Public члены класса:

void testAndAddServers() — вызывается внутри потока newServThread, опрашивает сервера, и добавлет в serversData, если подключиться к ним удалось.

ClientProgramm(int argc, char* argv[]) - конструктор класса, принимает в качестве парамета число и аргументы командной строки.

friend void* client_threadFunc(void* clnt) — функция, внутри который организован поток подключения к серверу, и выполнения разбора пришедших данных.

friend void* addNewServ_threadFunc(void* clnt) — функция поток,

периодически вызывает testAndAddServers().

П.В.4.3 Серверное приложение ФС

Модуль main.cpp

int main(int argc, char* argv[]) — содержит функцию main. Она создает экземпляр класса ServerProgram, и передает в него параметры и их число.

Модуль ServerProgram.cpp ServerProgram.h

typedef void (*registerModule) (unsigned int,Server* protocol, ServerConfig* sc, BinaryProtocol* bp) — тип данных, указатель на функцию, описывает функцию регистрации модуля.

typedef void (*unregisterModule)(unsigned int mid) — тип данных указатель на функцию, описывает функцию удаления регистрации модуля.

Структура clientData

ServerProgram* self — указатель на класс серверного приложения.

unsigned int i32ConnectFD — дескриптор сокета, через который производятся все опрации с клиентами, по протоколу на основе XML.

unsigned int i32ConnectFDb — дескриптор сокета, через который производятся все опрации с клиентами по бинарному протоколу.

Класс ServerProgram

Наследует класс ModuleLoader, специализирует шаблон типами данных registerModule и unregisterModule. Организует основную часть сервера.

Protected члены класса:

ServerConfig* sc — указатель на конфигурационный файл, из него будет получаться настройки сервера.

bool isExit — принимает значение true если получен сигнал SIGINT и

необходимо завершить работу приложения.

pthread_mutex_t connClientsMutex — мьютекс для защиты от параллельного доступа к conns_clnts.

map<int, Server*> conns_clnts — список указателей на экземпляры класса conns_clnts, для каждого подключенного клиента.

Public члены класса:

ServerProgram() — конструктор класса.

void sdCDesc() — если получен сигнал завершения, и нет больше клиентов подключенных к серверу, то освобождает и закрывает сокетные соединения.

int i32SocketFD — дескриптор сокета для клиентов, при подключении по протоколу на основе XML.

int i32SocketFDb — дескриптор сокета для клиентов подключенных по бинарному протоколу.

int createSocket(uint16_t port) — создает сокетный дескриптор, port — порт на котором ожидается подключение.

int waiteSocketConnection(int i32SocketFD) — ожидает подключения через сокетный дескриптор i32SocketFD, и возвращает сокетный дескриптор для обмена данными с клиентом.

friend void* client_thread_func(void* data) — функция выполняет обработку клиента, после его подключения.

friend void term_handler(int i) — функция обработчик сигнала, SIGINT, выполняет закрытие соединений с сервером, присваевает значение isExit=true, завершая работу сервера.

static ServerProgram* getInstance() — возвращает, указатель на экземпляр класса.

П.В.4.4 Модуль расширения сервер libFileoperaionServer.so

Модуль main.h, main.cpp

Организует основную часть модуля расширения сервера.

Структура moduleStruct, содержит обработчики iq запросов приходящих от клиента, каждый из них обрабатывает iq тег, со своим xmlns:

ReadDirectory* rd — чтение директорий, xmlns: readdir-data.

GetAttributes* ga — получение атрибутов, xmlns: getattr-data.

OpenFiles* of — открытие файла, xmlns: open-file.

ReleaseFiles* rlf — освобождение файла, xmlns: release-file.

ReadFiles* rf — чтение содержимого файла, xmlns:read-file.

WriteFiles* wf — запись файлов, xmlns: write-files.

ChmodFile* cm — изменение режима доступа к файлам xmlns: chmod.

UnlinkFiles* ul — удаление жестких ссылок на файл, xmlns: unlink.

LinkFiles* lf — создание жестких ссылок на файлы, xmlns: link.

RmDirs* rmd — удаление директорий, xmlns:rmdir.

MkDirs* md — создание директорий, xmlns: mkdir.

RenameFiles* rnf — переименование файлов, xmlns:rename.

TruncateFiles* trf — обрезание длинф файла, xmlns:truncate.

Каждый обработчик описан классом, который в конструкторе приниматет, указатель на класс бинарного протокола, указатель на класс протокола на основе XML, и конфигурационный фйл, и реализует интерфейс TagHandler. Далее приведено описание обработчика чтения данных файла:

Private члены класса:

Protocol* protocol — указатель на класс протокола на основе XML.

ServerConfig* sc — указатель на конфигурационный файл сервера.

BinaryProtocol* bp — указатель на бинарный протокол.

size_t max_readsize — максимальный объем данных который можно прочесть за один раз, получается из конфигурационного файла.

Public члены класса:

void handleTag(TagIQ &tag) — обработчик тега, принимает в качестве параметра, объект TagIQ, на основе которого строится объект ReadFile.

ReadFiles(Protocol &protocol,ServerConfig* sc,BinaryProtocol* bp) — конструктор protocol — экземпляр класса протокола на основе XML, sc — указатель на конфигурационный файл, bp — указатель на объект бинарного протокола.

Классы тегов операций над файлами

Классы GetAttr, OpenFile, ReadDir, ReadFile, ReleaseFile, WriteFile, наследуются от TagIQ, и расширяют его функциями, для задания и получения атрибутов, и добавления дочерних нод по необходимости. Класс BaseOneData, определяет классы имеющие один или два параметра, например имя файла и права доступа, при создании директорий или изменения прав на файлы. Таким образом от него наследуются классы: Chmod, Rmdir, UnlinkFile, Mkdir, RenameFile, TruncateFile, LinkFile. Они специализируют шаблон, соответствующим пространством имен, и формируют макроопределения для параметров функций получения и установки атрибутов.

П.В.4.5 Модуль расширения клиента clientfsoperations.so

Модуль main.cpp, main.h

Модуль описывает функции регистрации модуля расширения

extern "C" void registerModule(ServerDataMap* servsData, fuse_operations* clientFsoperations, ClientConfigFile* cf,sdSync* sincObj) — регистрация модуля расширения, servsData — словарь содержащий активные сервера, к которым

подключены, clientFsoperations — структура ее заполняют модули для реализации операций над ФС, cf — конфигурационный файл сервера, sincObj — объект для синхронизации данных, при обращении к servsData.

extern "C" void unregisterModule(int clntID) — удаление регистрации модулей, clntID — идентификатор подключенного клиента в структуре servsData.

Модуль ReadDirectory.cpp, ReadDirectory.h

Структура OpenedFilesStruct, описывает файлы которые были открыты клиентом

unsigned int opencount — сколько раз был открыт файл

int flags — флаги с которыми файл был открыт

list< pair<unsigned int, struct stat> >::iterator curserver — текущий сервер с которого производится чтение файла

list< pair<unsigned int, struct stat> > servers — список серверов, на которых был открыт файл.

Класс ReadDirectory

Private члены класса:

fuse_operations* clientFsoperations — структура FUSE в которой регистрируются обработчики ФС.

ClientConfigFile* cf — указатель на конфигурационный файл клиента.

private:

ReadDirectory(ServerDataMap& servsData, fuse_operations& clientFsoperations,ClientConfigFile& cf,sdSync* sincObj) — конструктор, т. к. класс является синглтоном, то он имеет приватный конструктор. ServsData — список серверов к которым подключен клиент, clientFsoperations — структура

полученная от fuse которая заполняется в конструкторе, cf — конфигурационный файл клиента, sincObj — объект синхронизации servsData.

Protected члены класса:

ServerDataMap* serversData — список серверов к которым подключен клиент.

sdSync* sincObj — объект синхронизации.

map<string,OpenedFilesStruct> ofiles — открытые файлы.

pthread_mutex_t openedfiles_mutex — мьютекс для защиты параллельного доступа к ofiles.

Public члены класса:

ServerDataMap* getServersData() — возвращает serverData.

static ReadDirectory* getInstance(ServerDataMap* servsData=NULL,

fuse_operations* clientFsoperations=NULL,ClientConfigFile* cf=NULL, sdSync* sincObj=NULL) — получение экземпляра объекта, параметры такие-же как у конструктора. Первый вызов должен содержать все параметры для создания объекта, последующие — возвращают уже существующий объект

void unregisterClient(int clntID) — удаление регистрации сервера, clntID — ключ в структуре servsData, метод вызывается из функции unregisterModule.

П.В.5 Выходные и входные данные

Входные и выходные данные программного средства – это информация о совершаемых операциях над файлами. Данная информация содержит пути к файлам их имена, дату и время модификации, создания, доступа к файлам, атрибуты, например права доступа, а так-же содержимое файлов. Она передается операционной системой в клиентское приложение, посредством модуля ядра fuse, и считывается либо записывается в общедоступной директории серверным приложением, посредством системных вызовов операций над файлами.

Для серверного приложения, входной и выходной информацией, так-же являются данные отправленные по сети клиентским приложением, а для клиентского — серверным.

П.В.6 Сообщения программисту

Сообщения для программиста, совпадают с сообщениями для системного программиста.

ПРИЛОЖЕНИЕ Г РУКОВОДСТВО ОПЕРАТОРА

П.Г.1 Назначение программы

Назначение разрабатываемой программы заключается в создании общего дискового пространства для группы компьютеров, соединенных в сеть, каждый из которых имеет доступ к этому пространству. Пользователь может производить любые операции над файлами, так как будто они находятся на на его компьютере.

П.Г.2 Условия выполнения программы

Для работоспособности программного средства требуется два или более компьютеров, они могут являться как персональными компьютерами так и серверами. Взаимодействие с программой осуществляется приложениями работающими с файловой системой, которые выполняют операции над ФС, посредством стандартных системных вызовов.

П.Г.3 Выполнение программы

Запуск программы производится через какую либо командную оболочку. Целесообразно как серверную так и клиентскую части запускать через init.d либо rc.d скрипты, поместив скрипт запуска в директорию определенную операционной системой, и сконфигурировав их в соответствии с правилами ОС.

Серверная часть программы «Распределенная файловая система» может быть запущена вызовом исполняемого файла serverfsp, клиентское приложение можно запустить выполнив clientfsp <точка монтирования файловой системы>, так-же может быть передана опция -d для запуска клиента в режиме отладки. В таком случае приложение не будет отвязано от консоля и запущено как демон. Запуск приложений должен производится из директории где расположен конфигурационный файл.

П.Г.4 Сообщения оператору

Клиентское приложение может сообщать об ошибках, связанных с работой ФС, при запуске в режиме отладки. В остальных случаях, приложение является демоном, и не возвращает никакой дополнительной информации. Серверное приложение выводит некоторую информацию о подключившихся клиентах и выполняемых операциях.

Так же клиентское приложение может выдавать номера ошибках работы с файловой системой, которые используются приложениям использующими ФС в том числе, для вывода сообщений оператору.