САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра Информатики

Группа 21.Б07-мм

Афанасьев Алексей Юрьевич

Разработка файловой системы для присоединенных сетевых хранилищ с многопользовательским доступом

Отчёт по учебной практике

Научный руководитель: к.ф.-м.н. доц. каф. информатики, Ловягин Н.Ю.

Санкт-Петербург 2024 г.

Содержание

Введение

Для научно-исследовательской работы была выбрана тема: «Разработка файловой системы для присоединенных сетевых хранилищ с многопользовательским доступом». Целью разработки - вынести шифрование передаваемых и хранимых данных с маломощного сетевого хранилища на более мощные клиентские компьютеры. Решение должно привести к централизованному хранению данных и управлению доступом к ним. Существуют различные необходимые механизмы, которые играют большое значение, и разработка которых может представлять отдельные темы для выступления. Конечная система требует реализовать механизмы безопасности, управления правами пользователей и надёжную работу при параллельном доступе к файлам.

В данной работе было исследовано, как можно создать систему, которая поддерживает многопользовательский доступ к файлам. Осуществлялась разработка клиент-серверного приложения, где несколько клиентов могут одновременно запрашивать файлы у сервера. Для этого использовался язык программирования C для Linux-систем, что позволит достичь высокой производительности и надежности. Это необходимый этап, на основе которого, можно развивать в дальнейшем различные возможности системы.

Сетевые хранилища файлов широко используются для того, чтобы разные пользователи могли работать с общими данными. Такие системы позволяют удобно хранить и передавать файлы, делая доступ к ним быстрым и централизованным. Важной задачей здесь является поддержка одновременной работы множества пользователей, которые могут подключаться к одному хранилищу через сеть.

Цели и задачи

Цель данной работы — разработать систему для организации многопользовательского доступа к файлам через сеть. Система должна быть реализована на основе клиент-серверной архитектуры, поддерживать одновременные подключения нескольких пользователей и обеспечивать надёжную передачу файлов. Для достижения цели были поставлены следующие задачи:

1. Разработать простое клиент-серверное приложение.
2. Сопроводить возможные ошибки при работе с сетью сервера и клиента проверками на корректность.
3. Реализовать модуль в сервере для логирования своих действий в специальный файл.
4. Организовать возможность сервера находить и отправлять файлы из специальной директории.
5. Реализовать передачу данных файла от сервера клиенту.

Обзор

Обзор существующих решений

При ознакомлении с статьями для поиска существующих работ, относящихся к данной теме, было выделено несколько, которые оценивают актуальность работы.

NAS-CFS (IEEE, 2003) [1] NAS-CFS предложена как файловая система для сетевых хранилищ с поддержкой безопасности. Основная задача — предоставить доступ к файлам через сеть с базовыми мерами защиты данных. Однако работа уже устарела, а разработка не имеет признаков поддержки и выхода на рынок. В то же время она актуальна для анализа в качестве начального подхода к задаче организации централизованного файлового хранилища.

Преимущества:

- Интеграция с сетевым хранилищем.
- Поддержка многопользовательского доступа.

Недостатки:

- Устаревшая архитектура и технология.
- Ограниченные возможности защиты данных.

KryptoNAS (Springer, 2007) KryptoNAS [2] — это гибридное программно-аппаратное решение для обеспечения защищённой передачи данных в системах NAS. Оно сосредоточено на применении аппаратных средств шифрования для защиты данных на этапе хранения и передачи. Однако данное решение ориентировано на использование специфического оборудования, что ограничивает его гибкость для внедрения в средах, где нужна чисто программная реализация.

Преимущества:

- Высокий уровень безопасности благодаря использованию аппаратного шифрования.

Недостатки:

- Зависимость от аппаратного обеспечения.
- Ограничения для программно-ориентированных систем.

Анализ существующих решений показывает, что большинство доступных разработок либо устарели (NAS-CFS), либо зависят от специфического оборудования (KryptoNAS). Это подтверждает актуальность задачи разработки современной программной системы для многопользовательского доступа к файлам с защитой данных, реализованной на программном уровне без привязки к аппаратуре.

Выбор языка программирования C для Linux-систем

Язык программирования C обеспечивает низкоуровневый доступ к системным ресурсам, что важно для сетевых приложений, работающих с файлами и сетью. Этот язык даёт возможность использовать системные вызовы Linux для управления сетевыми соединениями и файлами. Язык C остаётся одним из самых популярных языков в системном программировании. Большое количество готовых решений и сообщество разработчиков делает его удобным выбором для разработки и поддержки сетевых приложений в Linux.

Таким образом, язык C был выбран благодаря своей эффективности, низкоуровневому управлению системными ресурсами и сетевыми интерфейсами, а также широкой поддержке в Linux-системах, что делает его лучшим выбором для создания высокопроизводительных многопользовательских сетевых приложений.

Ход работы

Постановка задачи

Задача работы состоит в разработке системы для организации многопользовательского доступа к файлам в сетевом хранилище с обеспечением безопасной передачи данных. Необходимо реализовать клиент-серверное приложение, которое позволяет нескольким пользователям одновременно запрашивать файлы с сервера.

Реализация клиент-серверного приложения

Клиент-серверная архитектура предполагает модель взаимодействия, где сервер предоставляет ресурсы или услуги, а клиенты обращаются к ним. Для этого используются сокеты (network sockets) — точки соединения для передачи данных между устройствами по сети. В Linux система сокетов основана на API, где сокет создается с помощью системных вызовов, таких как socket(), bind(), listen(), и accept() на стороне сервера, и connect() на стороне клиента. Через API программы могут создавать, изменять и управлять процессами, работать с файлами, памятью и сетевыми ресурсами.

Создано приложение, где сервер прослушивает порт, принимает подключения клиентов и обрабатывает их запросы. Клиенты могут отправлять запросы на получение файлов. На скрине продемонстрировано, как клиент устанавливает соединение с сервером и получает запрашиваемый файл.

```
alec@debian:~$ ./server
alec@debian:~$ ./client
Connecting to: 127.0.0.1
Enter the name of the file you want to request: first_file.txt
Server response endalec@debian:~$
```

Обработка ошибок сети

Работа с сетевыми соединениями может сопровождаться ошибками, такими как недоступность сервера, ошибки соединения, неправильные данные или потеря соединения. Поэтому важно обрабатывать ошибки на каждом этапе взаимодействия с сетью. Это достигается путём проверки возвращаемых значений системных вызовов и использования специальных методов обработки. Реализованы проверки на ошибки в каждом системном вызове. Например, проверка возвращаемого значения от функций.

Логирование действий сервера

Логирование — важный компонент серверных приложений, который позволяет отслеживать работу сервера, записывать его активность и выявлять возможные проблемы. Логи включают информацию о подключениях клиентов, запрашиваемых файлах, ошибках и других действиях.

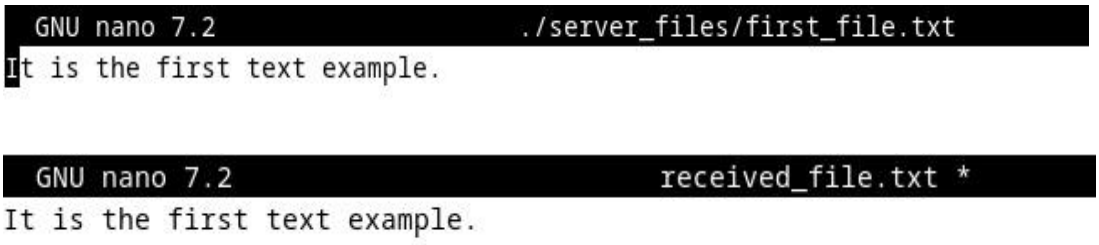
Добавлен модуль логирования, который записывает события в отдельный файл. Логи включают информацию о времени подключения клиента, имени запрашиваемого файла и статусе выполнения запроса (успех/ошибка).

Поиск файлов на сервере

В файловой системе сервера определён каталог (/home/alec/server_files/), в котором хранятся файлы, доступные для передачи. Сервер должен уметь искать файл в этой директории по запросу клиента. Сервер обрабатывает запрос от клиента, проверяет наличие запрашиваемого файла в специальной директории. Реализован механизм поиска с использованием функций работы с файлами (fopen(), fread()). Если файл найден, он открывается для чтения и подготовки к передаче клиенту. Важно, что сервер ищет каталог по полному пути, так как это позволяет избавиться от необходимости следить за тем, чтобы сервер запускался из той же директории. Таким образом, давая ему некоторую независимость.

Передача файлов от сервера клиенту

Передача данных через сеть предполагает поблочную отправку файла от сервера к клиенту. Для этого файл автоматически разбивается на блоки данных, которые передаются по сети до тех пор, пока весь файл не будет отправлен. Сервер открывает файл для чтения, считывает его содержимое порциями и передаёт клиенту с помощью функции send(). На стороне клиента данные принимаются через recv(), и сохраняются в локальный файл. Обработка передачи осуществляется в цикле до тех пор, пока весь файл не будет передан.



Продемонстрирован результат передачи данных от сервера из файла «first_file.txt» к клиенту в файл «received_file.txt».

Эти разделы показывают поэтапную реализацию системы и применение различных теоретических подходов для создания многопользовательского клиент-серверного приложения.

Тестирование

О возможностях приложения можно судить по проведенным тестам времени передачи файла. Здесь приведены результаты измерений, полученные значения среднего и дисперсии. Дисперсия показывает, насколько результаты измерений (в данном случае — времени передачи файла) отклоняются от среднего значения. Если дисперсия велика, это указывает на значительные колебания во времени передачи, что может свидетельствовать о нестабильной работе приложения, влиянии внешних факторов (сети, нагрузки на сервер) или неэффективности алгоритмов передачи данных.

```
alec@debian:~/server_files$ du -ah big_file.txt
2,8M    big_file.txt
```

```
alec@debian:~$ ./client
Connecting to: 127.0.0.1
Enter the name of the file you want to request: big_file.txt
Time taken to receive file: 20.253 seconds.
Server response endalec@debian:~$
```

```
alec@debian:~$ ./client
Connecting to: 127.0.0.1
Enter the name of the file you want to request: first_file.txt
Time taken to receive file: 20.297 seconds.
Server response endalec@debian:~$ sudo nano log.txt
[sudo] пароль для alec:
alec@debian:~$ ./client
Connecting to: 127.0.0.1
Enter the name of the file you want to request: first_file.txt
Time taken to receive file: 20.410 seconds.
Server response endalec@debian:~$
```

```
alec@debian:~$ sudo gcc timer.c -o timer -lm
alec@debian:~$ ./timer
Среднее время: 20.378 секунд
Дисперсия: 0.006932
```

Вывод

В ходе работы было успешно разработано клиент-серверное приложение для организации многопользовательского доступа к файлам. Основными задачами проекта были создание сетевого взаимодействия, поддержка одновременных подключений, корректная обработка ошибок и логирование, а также безопасная передача данных. Благодаря применению языка C и использования возможностей Linux API удалось реализовать решение, которое эффективно выполняет задачи, такие как передача файлов и многопоточная работа с клиентами. Приложение протестировано в условиях реальных подключений, что подтвердило его надёжность и работоспособность. При этом учтены размеры файла. Были проведены тесты как на мелком файле, так и на файле размером несколько мегабайт.

Таким образом, поставленные цели достигнуты: разработано стабильное клиент-серверное приложение, способное поддерживать многопользовательский доступ, с логированием и обработкой ошибок, что доказывает его пригодность для практического использования в сетевых хранилищах.

Заключение

В ходе работы в течение весеннего семестра были выполнены следующие задачи:

1. Реализован сервер, который может обрабатывать запросы на файлы от нескольких клиентов одновременно.
2. Реализован клиент, который отправляет запросы на получение файлов с сервера.
3. Добавлены проверки результатов системных вызовов с выводом сообщений об ошибках для правильной обработки проблем с соединением и передачей данных.
4. Реализован модуль, который записывает ключевые события сервера (подключение клиентов, запросы файлов, ошибки) в отдельный лог-файл.
5. Сервер находит запрашиваемые клиентом файлы в специальной директории. Использованы стандартные функции работы с файлами, чтобы обеспечить возможность чтения и передачи данных.
6. После успешного нахождения файла, сервер передаёт его содержимое клиенту, отправляя файл порциями, что обеспечивает надёжную передачу данных даже для больших файлов.

Список литературы

1. https://ieeexplore.ieee.org/abstract/document/1410761
2. https://link.springer.com/chapter/10.1007/978-3-8348-9363-5_22
3. https://github.com/Alec-afana/Project-5-sem
4. Куроуз Д.Ф., Росс К.В. Компьютерные сети. Нисходящий подход. - 6-е изд. - М.: Издательство "Э", 2016. - 912 с.
5. Роберт Лав Linux. Системное программирование. - 2-е изд. - СПб.: Питер, 2014. - 448 с.