

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.04 Программная инженерия

Система управления версиями с шифрованием

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
И.В. Рудаков
(И.О.Фамилия)
«24» сентября 2021г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине Операционные системы
Студент группы ИУ7-76Б, ИУ7и-76Б
Мередова Айджахан, Нгуен Фыок Санг
(Фамилия, имя, отчество)
Тема курсовой работы Система управления версиями с шифрованием
Направленность КР (учебная, исследовательская, практическая, производственная, др.)
учебная
Источник тематики (кафедра, предприятие, НИР) кафедра
График выполнения работы: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание Разработать систему управления версиями с шифрованием, а также разноуровневым доступом (гость/пользователь) и, собственным протоколом на прикладном уровне с поддержкой базового функционала аналогичный протоколу FTP (скачивание, изменение, отправка, назначение прав доступа к файлам и папкам)

Оформление курсовой работы

Расчетно-пояснительная записка на 20-30 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
Расчетно-пояснительная записка должна содержать постановку, введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

Дата выдачи задания «24» сентября 2021г.

Руководитель курсовой работы

Студент

Студент

	<u>Н.О. Рогозин</u>
(Подпись, дата)	(И.О.Фамилия)
	<u>А. Мередова</u>
(Подпись, дата)	(И.О.Фамилия)
	<u>Ф.С. Нгуен</u>
(Подпись, дата)	(И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

Введение	4
1. Аналитическая часть	5
1.1. Описание предметной области	5
1.2. Предварительные требования к информационной системе	5
1.3. Составление диаграммы классов	5
1.4. Разработка схемы базы данных	5
2. Конструкторская часть	7
2.1. Техническое задание	7
2.1.1. Протокола	7
2.1.2. Основания для разработки	8
2.1.3. Требования к программе	8
2.1.4. RSA - шифрование	9
2.1.5. Стадии и этапы разработки	10
2.1.6. Порядок контроля	10
2.2. Архитектура программного обеспечения	10
2.2.1. Передача и приём файлов	11
2.3. Коммиты	13
2.4. Откат	13
2.5. Загрузить	13
2.6. Выводы	13
3. Технологическая часть	14
3.1. Используемые средства разработки	14
3.1.1. Язык программирования	14
3.1.2. Среда разработки	14
3.2. Основные результаты тестирования	15
3.3. Команда bash: diff	15
3.4. Команда bash: patch	15
3.5. SQLite3	15
4. Экспериментальная часть	17
4.1. Условия эксперимента	17
4.2. Интерфейс программы	17
4.3. Результат работы программы	18
4.4. Выводы	21
Заключение	22

Введение

Актуальность – из-за постоянного роста пользователей интернета и компьютеров, в последние годы создание программных продуктов стало все более разнообразным, сложным и расширенным. Комплексы разрабатываются с учётом потребностей на несколько дней вперёд. Таким подходом со временем начали создавать различные приложения, которые должны были помочь программистам разрабатывать продукты больших объёмов лучше, быстрее и проще. Одним из типов таких вспомогательных приложений, отвечающих этим требованиям, является «контроль версий».

Цель работы – решить проблемы предметной области, такие как:

- 1) проблема надёжности хранения данных;
- 2) возможность возврата к более ранней версии;
- 3) упрощение командной разработки.

Возможности системы:

- регистрация / вход пользователей в систему;
- загрузка / выгрузка / удаление файлов из системы;
- вести логи работы с разными сообщениями;
- возврат к старым версиям.

1. Аналитическая часть

В этой части, представим техническое описание проекта. В подразделе приведём примеры существующих решений в нашей области.

1.1. Описание предметной области

Системы контроля версий:

- обеспечить авторизацию пользователя;
- вести журналирование событий;
- показывать статус программы.

1.2. Предварительные требования к информационной системе

На основе анализа предметной области можно сформулировать следующие предварительные требования к информационной системе:

Функциональные требования:

Список действующих лиц:

- Администратор – добавляет новых пользователей и управляет правами каждого из пользователей, редактирует файлы (добавляет, обновляет, удаляет)
- Пользователь – скачивает и смотрит доступные документы.

1.3. Составление диаграммы классов

Для представления используемых объектов и для конкретизации их параметров и выполняемых функций мы построим диаграмму классов, в которой каждый из них определенным образом взаимодействует друг с другом. При входе в систему клиент может попасть на конкретные классы, а на другие попасть только после взаимодействия с первыми. В связи с этим была спроектирована предварительная диаграмма классов, которая в ходе разработки и тестирования программного обеспечения может дополняться или изменяться.

1.4. Разработка схемы базы данных

В базе данных хранятся только данные о пользователях:

- username
- password

2. Конструкторская часть

2.1. Техническое задание

2.1.1. Протокола

ТСР – это протокол транспортного уровня с установлением соединения, предназначенного для надёжной передачи данных по сети. **ТСР** обеспечивает безопасное соединение для передачи данных, то есть данные проверяются на наличие ошибок и это гарантирует, что пакеты отправляются строго по порядку, предотвращая перегрузку сети и т.п. проблем. Так как, в разрабатываемом проекте нужно обеспечить максимально возможную безопасность при передаче данных т.е. минимизировать их потери, то будет использоваться протокол **ТСР**.

Таким образом, приложение будет иметь следующие возможности:

- соединение между сервером/клиентом будет осуществляться с использованием методов «Трёхстороннего установления связи» примитивами **accept()**, **listen()** и **connect()**;

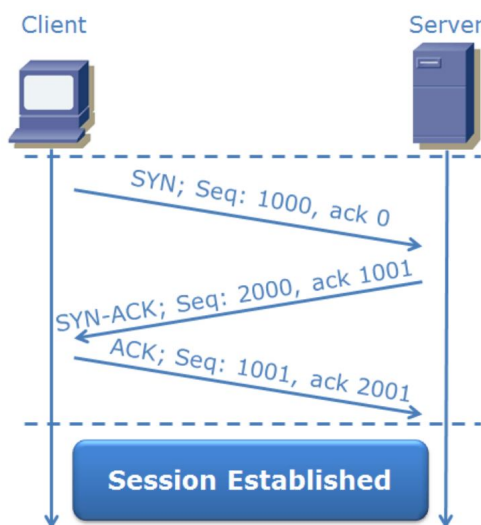


Рис. 1. Трёхсторонний метод установления связи.

- для адресации, на сетевом уровне будет использоваться протокол IP \Rightarrow каждый узел в сети будет идентифицирован 32-битной серией, и на транспортном уровне будут использоваться порты, использующие метод однозначной идентификации процессов, запущенных на сетевой системе;
- будет реализована передача информации через сокеты, абстракцию к файловым дескрипторам в программировании ANSI-C, к которым могут быть прикреплены адреса IP + порт;

- байтовые последовательности будут передаваться между сокетами с использованием примитивов read() и write();

2.1.2. Основания для разработки

Документ разработан на основании курсового проекта по компьютерным сетям.

2.1.3. Требования к программе

Пользовательский интерфейс должен быть разработан как терминальное приложение.

Требования к системе:

- 1) Будет реализован протокол на уровне приложения (прикладной), который предназначен, чтобы облегчить выполнение команд: на добавление, скачивание файлов и аналогично, по типу FTP добавление шифрования на все передаваемые данные.
- 2) В самом процессе шифрования, будет использован метод шифрования RSA, так как мы не можем жертвовать скоростью передачи в рамках курсового проекта, т.к. в комбинации с протоколом передачи three-pass, мы итак теряем скорость. Но мы можем пожертвовать скоростью передачи данных относительно передачи, работая соотносительно «лёгкими» данными (маленький размер файлов).

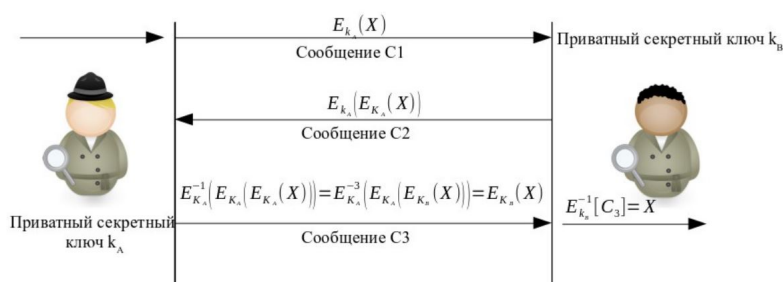


Рис. 2. "three-pass" протокол.

- 3) На минимальной основе приложения должны быть определены следующие услуги:
 - вход в систему - разрешает операцию входа пользователя
 - `create_user <user> <password> <rights>` - разрешает создание пользователя (только у администраторов)
 - `download<fileName>` - разрешает работу файла в репозитории

- `adfd<fileName>` - разрешает операцию добавления файла в репозиторий. (история, чтобы её видели только администраторы)
- `commit <сообщение>` - загрузит изменения, внесённые в текущий репозиторий (только у администраторов)
- `revert <versionNumber>` - позволяет вернуться к предыдущей версии приложения, изменяющее содержимое каталога `currentFiles`, из которого некоторые пользователи могут скачивать файлы (они есть только у администраторов)
- `uploadedFiles` – позволяет просматривать имена отслеживаемых файлов. приложение (только у администраторов)
- `currentFiles` – позволяет просматривать имена файлов в репозитории
- `workFiles` - позволяет просматривать имена файлов, находящихся в папке работа с клиентами
- `status` – разрешает просмотр состояний
- `showDiff <versionNumber> <fileName>` – разрешает другой просмотр сделано во время фиксации в конкретный файл.
- `showHistory` - показывает историю коммитов с сообщениями.

2.1.4. RSA - шифрование

RSA – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Алгоритм создания открытого и секретного ключей RSA-ключи генерируются следующим образом:

- 1) выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое)
- 2) вычисляется их произведение $n = p * q$, которое называется модулем;
- 3) вычисляется значение функции Эйлера от числа n :

$$\phi(n) = (p-1) * (q - 1)$$
- 4) выбирается целое число e ($1 < e < \phi(n)$), взаимно простое со значением функции $\phi(n)$; число e называется открытой экспонентой;

- 5) вычисляется число d , мультипликативно обратное к числу e по модулю $\phi(n)$, то есть число удовлетворяющее сравнению: $d * e = 1 \pmod{\phi(n)}$; число d называется секретной экспонентой; обычно оно вычисляется при помощи расширенного алгоритма Евклида;
- 6) пара (e, n) публикуется в качестве открытого ключа RSA;
- 7) пара (d, n) играет роль закрытого ключа RSA и держится в секрете.

2.1.5. Стадии и этапы разработки

- 1) Разработка архитектуры программы.
- 2) Анализ технологий разработки программного обеспечения
- 3) Выбор среды и языка разработки серверной части и инструментов
- 4) Реализация программного обеспечения
- 5) Отладка и тестирование программы

2.1.6. Порядок контроля

Проверка системы осуществляется в 3 этапа:

- 1) Проверка логической структуры исходного кода.
- 2) Тестирование базовых функций приложения
- 3) Правка багов

2.2. Архитектура программного обеспечения

Клиент-сервер – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Физически клиент и сервер – это программное обеспечение. Обычно они взаимодействуют через компьютерную сеть посредством сетевых протоколов и находятся на разных вычислительных машинах, но могут выполняться также и на одной машине. Программы-сервера, ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или сервисных функций (например, работа с

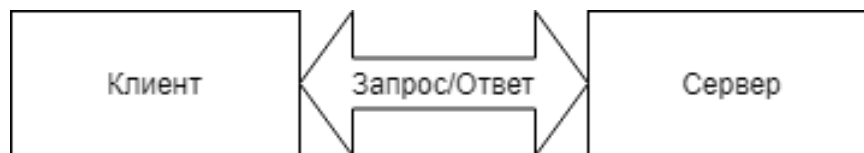


Рис. 3. Клиент-серверная модель

электронной почтой, общение посредством систем мгновенного обмена сообщениями, просмотр web-страниц во всемирной паутине).

Чтобы использовать все ресурсы, доступные в вычислительной системе, а также для более быстрого и эффективного применения необходимо использование одной из главных парадигм клиент/сервер.

2.2.1. Передача и приём файлов

Основная идея заключается в том, чтобы выделять для каждого клиента один `fork()`, и таким образом, имея возможность обслуживать сразу нескольких клиентов. Для реализации данного подхода, необходимо создать новый сокет для сервера, через который будет осуществляться связь между `fork()` процессом и клиентом. Первоначальный сокет будет сохранён и будет использоваться родительским процессом/основной поток делит на разных клиентов, разные `fork()`.

Мы также можем использовать ту же парадигму для клиентов. Когда задерживаем запрос, чтобы он был обработан в дальнейшем сыном предыдущего процесса, но в данном случае это будет применимо только в случае загрузки, т.е. не подходит для нашей реализации.

Для передачи данных мы будем использовать схему FTP, т.е. у нас будет одна область управления, через которую мы будем анализировать различные команды, которые мы получим от клиента и часть данных, куда будем передавать файлы. Для этого, когда нужен перевод файла между двумя экземплярами, мы создадим новое соединение от сервера к клиенту, через который будет эффективно передавать данные.

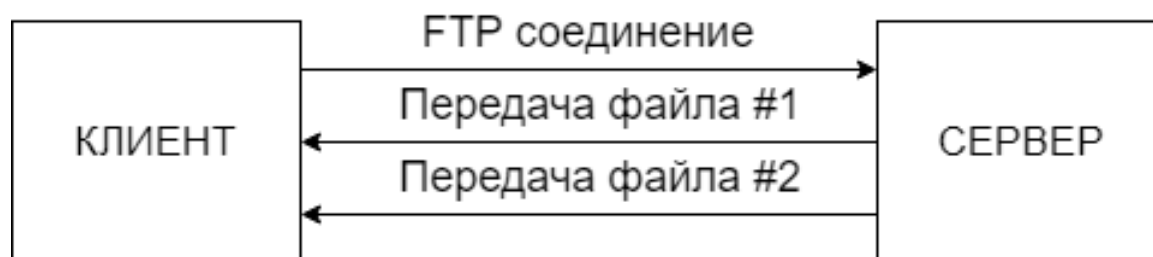


Рис. 4. Общая схема протокола FTP

Таким образом, каждый коммит будет иметь закрытый ключ, который

будет сгенерирован в момент его рождения и после будет представлять собой длину, с которой мы будем перемещать символ в таблице ascii (по шифру Цезаря)

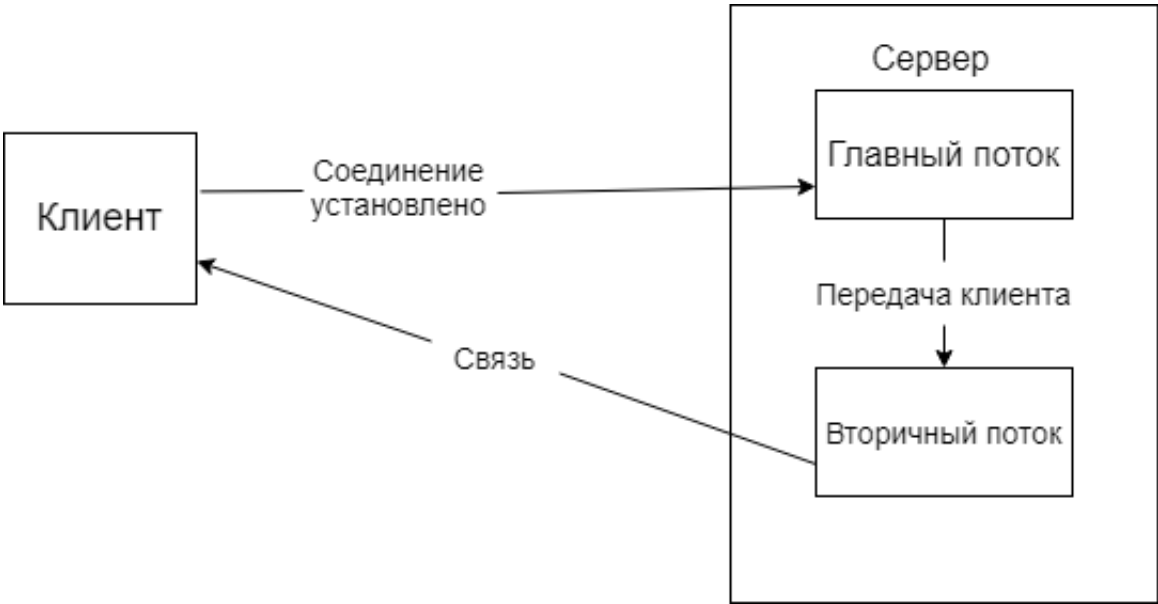


Рис. 5. Обслуживание клиентов после подключения

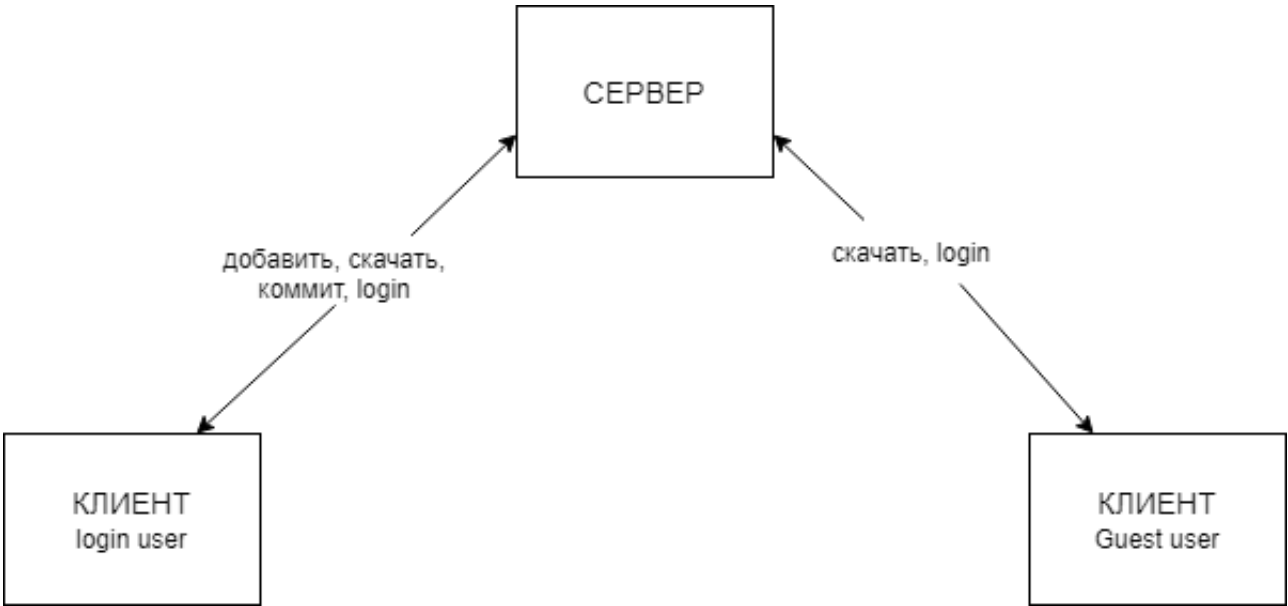


Рис. 6. Общая схема прав пользователей

2.3. Коммиты

Чтобы сохранить возможность перейти к началу, для отката к предыдущей версии от текущей (поскольку различия будут действительны только между версиями, фиксированными по коммитам), поэтому пользователь может сделать откат сразу после нового коммита. Затем клиент отправляет все свои файлы из каталога `./work`, в сервер, на хранилище, в каталог `./serverFiles/committedFiles`. Затем, пройдя по каталогу `./serverFiles/committedFiles`, выявляем различия (которые сохраняются в каталог по номеру версии) между файлами с тем же именем в каталоге `./serverFiles/currentFiles` и каталоге `./serverFiles/committedFiles` таким образом отличаясь от последней фиксации.

2.4. Откат

Для фиксации все файлы сохраняются в каталогах `./serverFiles/uploadedFiles` и `./serverFiles/currentFiles`, далее проверка идёт по каждому из них, чтобы зафиксировать версии (с первой, до последней) и для каждой будет применяться diff от текущей версии к каждому файлу в `./serverFiles/currentFiles`.

2.5. Загрузить

Для загрузки сначала проверяется существование указанного файла, и если ошибок нет (он существует и не повреждён), сервер получает оповещение, о запросе на выполнение операцию загрузки, создаёт новый сокет и начинает прослушивание, отправляя его в порт клиента (найденный с помощью `getsockname()`) на стороне клиента, чтобы определить, к какому порту подключиться для дальнейшей передачи информации. На данном этапе клиент подключается к найденному порту, а также начальному IP-адресу, по которому сервер принял его. На данный момент передача данных может быть произведена через это новое соединение без использования исходного соединения. Данная модель передачи файлов аналогична по реализации протоколу FTP.

2.6. Выводы

Для осуществления поставленных задач, необходимо реализовать клиент-серверную архитектуру. Со стороны сервера необходимо реализовать логику хранения логинов и паролей, ведения журнала, а также передачи самих данных. Со стороны клиентов необходимо реализовать взаимодействие с сервером, передавать, собирать файлы по кускам, а также оповещать сервер для заполнения журнала.

3. Технологическая часть

3.1. Используемые средства разработки

3.1.1. Язык программирования

Для написания данного курсового проекта был выбран язык программирования **Си**, который отличается минимализмом и легкостью компилирования с помощью однопроходного компилятора, который позволяет легко компилировать на разных машинах без правок под конкретные устройства. Язык является универсальным в рамках этого проекта, так, на нём будет писаться и сервер и клиент. Также учитывая, что проект пишется по Linux который тесно связан (написан) с **Си**.

При выборе языка основными критериями были:

- Многопоточный
- Динамический
- Низкоуровневый

Под динамическим в данном случае понималось, что программа может выполнять обширное количество задач во время обработки информации, которая может быть использована для проверки и разрешения доступа к объектам на время выполнения.

3.1.2. Среда разработки

При выборе среды разработки рассматриваются и другие возможные среды, которые помогут автоматизировать процесс разработки. Для разработки данной программы необходимо обеспечение средой следующих возможностей:

- удобные инструменты для отладки и поиска ошибок, в случае их возникновения;
- разработка более гибкой и надёжной программы путём обработки различных исключительных ситуаций, возникающих в результате некорректной работы программы;
- использование всплывающих подсказок во время написания кода программы, что обеспечивает значительную экономию времени и повышения уровня продуктивности.

Среда разработки **Visual Studio Code** поддерживает все эти возможности, и полностью поддерживается на Linux.

3.2. Основные результаты тестирования

Во время разработки программный продукт тщательно тестировался. Тестирование передачи файлов проводилось на разных типах текстовых файлов, у которых были разные размеры. После коммита файлов и повторной загрузки файлы открывались без проблем и ошибок. Список тестируемых ситуаций:

- вход под несуществующей учётной записью;
- вход под существующей учётной записью;
- целостность файла после успешной передачи;
- попытка передачи файла не выбранному пользователю.

3.3. Команда `bash: diff`

Для эффективного хранения данных на сервере, учитывая тот факт, что в приложении будет делаться большое количество коммитов, потребуется метод, с помощью которого можно будет эффективно хранить данные в промежутках между фиксациями. Поэтому на сервере будут храниться только журналы различий между предыдущими версиями и неполный файл.

Из-за отсутствия применимого API будет использована команда `diff` из `bash`, получающая в качестве параметра 2 текстовых файла и будет отображать различия между ними.

3.4. Команда `bash: patch`

Чтобы эффективно пользоваться файлами, созданными командой `diff`, будет использоваться ещё одна команда из `bash`: `patch`. Эта команда позволяет применять различие между 2 файлами, для внесения изменения в главный (исходный) файл. Необходимо перенаправить ввод этих команд в редактируемый файл, но затем оно может применяться к любому файлу, применяя отличия, указанные на входе.

3.5. SQLite3

SQLite3 – это библиотека, которая реализует способ управления реляционными базами данных. Данная библиотека была выбрана по причине того, что была уже знакома с предыдущего проекта, а также здесь операторы SQL

могут выполняться очень просто и без особых настроек и проблем, ведь хранимые данные зашифрованы. Эта библиотека была выбрана для использования в качестве пользовательского хранилища.

Таким образом, в каждой строке будет три поля: имя пользователя, пароль, права доступа, которые будут проверяться при входе в систему и далее в процессе использования приложения.

4. Экспериментальная часть

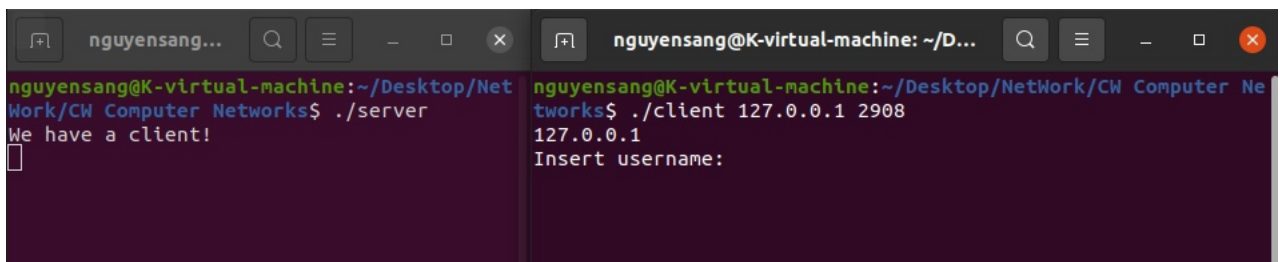
4.1. Условия эксперимента

Исследование результатов выполнения программы производилось при следующем аппаратном обеспечении:

- Процессор Intel(R) Core(TM)i5-10210U CPU @ 1.60GHz 2.11 GHz
- Оперативная память – 8 Гб
- Объем памяти (SSD) – 120 Гб
- Linux OS (Ubuntu)

4.2. Интерфейс программы

Программа при запуске требует сразу авторизацию, регистрация новых пользователей доступна только для администраторов.

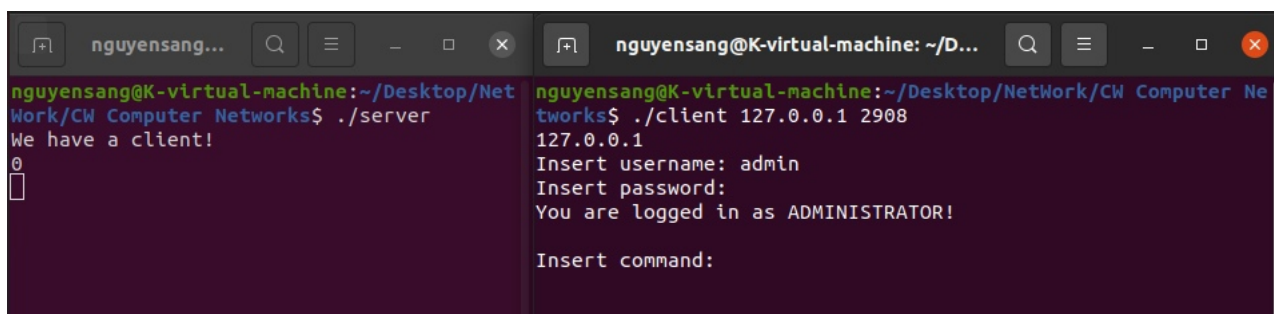


The image shows two terminal windows side-by-side. The left window has a title bar 'nguyensang...' and shows the command `nguyensang@K-virtual-machine:~/Desktop/Network/CW Computer Networks$./server` being executed, with the output 'We have a client!' and a cursor on the next line. The right window has a title bar 'nguyensang@K-virtual-machine: ~/D...' and shows the command `nguyensang@K-virtual-machine:~/Desktop/Network/CW Computer Networks$./client 127.0.0.1 2908` being executed, with the output '127.0.0.1' and 'Insert username:' followed by a cursor.

Рис. 7. Интерфейс программы (server&client)

Программа при запуске требует сразу авторизацию, регистрация новых пользователей доступна только для администраторов.

4.3. Результат работы программы



```
nguyensang@K-virtual-machine:~/Desktop/NetWork/CW Computer Networks$ ./server
We have a client!
0
[ ]

nguyensang@K-virtual-machine:~/Desktop/NetWork/CW Computer Networks$ ./client 127.0.0.1 2908
127.0.0.1
Insert username: admin
Insert password:
You are logged in as ADMINISTRATOR!

Insert command:
```

Рис. 8. Вход в систему от имени администратора.



```
Insert command:
status
[From Status]
Committed: ins.txt
Create: HDSD.txt
Committed: test.txt

Insert command:
[ ]
```

Рис. 9. Status.

```
Insert command:
upload ./work/ins.txt
Insert
Test

Hello

File was successfully sent!

Insert command:
status all
[From Status]
Create: ins.txt
Create: HDSD.txt
Create: test.txt

Insert command:
```

Рис. 10. Upload&status

```
[From Commit] The work files were sent!
[From Commit] The work files were received!

Insert command:
```

Рис. 11. Commit

```
Insert command:
revert 1
The revert succeeded!

Insert command:
```

Рис. 12. Revert

```
Insert command:
revert 1
The revert succeeded!

Insert command:
```

Рис. 13. Download

```
Insert command:  
showHistory  
[From History]  
Commit 1: "1"  
  
Commit 2: "2"  
  
Insert command:
```

Рис. 14. История коммитов

4.4. Выводы

Для увеличения скорости работы, и возможности работы с большими файлами можно убрать сжатие файлов и объединить её с дополнительным шифрованием современными средствами. Так же добавить хотя бы консольный интерфейс, для удобства работы с файлами, а также навигации не только в папке клиента, но и в свободном перемещении по смежным (на данном этапе реализации требовала *sudo — user* права, поэтому пока не добавлена в релизную версию). Также необходима реализация таблицы истории файлов (только для администраторов), с пронумерованными коммитами, если работать в больших командах. Команду `showDiff` нужно переписать для работы с большими файлами.

Заключение

В рамках курсового проекта была реализована система управления версиями. С помощью разработанной программы можно существенно сократить время разработки ПО разрабатываемого несколькими людьми (командой), а также облегчить нахождение багов которые вызваны после определённого коммита. Также при тестировании программы были выявлены основные недочёты, связанные с вводом неправильных данных, проведён анализ основных проблем, которые могут возникнуть в процессе разработки и поддержки программного обеспечения. При дальнейшей разработке можно добавить улучшения:

- шифрование файлов современными средствами;
- использование облачных технологий;
- сжатие файлов;

Список литературы

- [1] POSIX thread (pthread) libraries: [ЭЛ. РЕСУРС]
Режим доступа: <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads>.
(Дата обращения: 04.11.2021)
- [2] Unix Domain Socket nedir ve ne işe yarar. [ЭЛ. РЕСУРС]
Режим доступа: <https://medium.com/@gokhansengun/unix-domain-socket-nedir-ve-ne-i%C5%9Fe-yarar-c72fe8decb30>»
(Дата обращения: 12.12.2021)
- [3] Yazılım Çorbası - TCP/UDP/Netlink ve Linux [ЭЛ. РЕСУРС]
Режим доступа: <https://yazilimcorbasi.blogspot.com/2012/03/tcp-ve-linux.html>
(Дата обращения: 10.12.2021)
- [4] GNU Bash [ЭЛ. РЕСУРС]
Режим доступа: <https://www.gnu.org/software/bash/>
(Дата обращения: 12.12.2021)
- [5] SQLite Documentation [ЭЛ. РЕСУРС]
Режим доступа: <https://sqlite.org/docs.html>
(Дата обращения: 22.11.2021)
- [6] Git Documentation [ЭЛ. РЕСУРС]
Режим доступа: <https://git-scm.com/doc>
(Дата обращения: 11.12.2021)
- [7] Versiyon Kontrol Sistemi (VCS) Nedir? [ЭЛ. РЕСУРС]
Режим доступа: <https://ceaksan.com/tr/versiyon-kontrol-sistemi-vcs-nedir>
(Дата обращения: 20.11.2021)
- [8] Modified Quantum Three Pass Protocol Based on Hybrid Cryptosystem.[ЭЛ. РЕСУРС]
Режим доступа: <http://i-rep.emu.edu.tr:8080/jspui/bitstream/11129/2327/1/abdullah.pdf>
(Дата обращения: 10.12.2021)
- [9] Three-Pass Protocol Implementation in Caesar Cipher Classic Cryptography. [ЭЛ. РЕСУРС]
Режим доступа: <https://osf.io/7u5jh/download/?format=pdf>
(Дата обращения: 10.12.2021)

- [10] FILE TRANSFER PROTOCOL (FTP). [ЭЛ. РЕСУРС]
Режим доступа: <https://tools.ietf.org/html/rfc959>
(Дата обращения: 10.12.2021)

Приложение

В приложении приводятся основные части исходных текстов курсовой работы.

login.c

```
#include "login.h"
structuser getUser()
{
    structuser currentUser;
    currentUser.state =NOTLOGGEDIN;
    printf("Insert username: ");
    /* username */
    if (fgets(currentUser.username,NMAX,stdin) == 0)
    {
        printf("Error while reading username");
    }
    currentUser.username[strlen(currentUser.username) - 1] = 0;
    printf("Insert password: ");
    intn;
    /* password */
    getPassword(currentUser.password,stdin);
    currentUser.password[strlen(currentUser.password) - 1] = 0;
    returncurrentUser;
};

voidgetPassword(char *lineptr, FILE *stream)
{
    structtermios old,new1;
    intnread;
    /* Turn echoing off and fail if we can't. */
    if (tcgetattr(fileno(stream), &old) != 0)
        return;
    new1=old;
    new1.c_lflag &= ~ECHO;
    if (tcsetattr(fileno(stream),TCSAFLUSH, &new1) != 0)
        return;
    /* Read the password. */
    fgets(lineptr,NMAX,stream);
    /* Restore terminal. */
    (void)tcsetattr(fileno(stream),TCSAFLUSH, &old);
};
```

create_usersDB.c

```
#include <sqlite3.h>
#include <stdio.h>
/*
 * Script that creates a database by inserting a admin and guest (
    normal)
    user.
 */
```

```

intcallback(void *, int, char **, char **);
sqlite3 *openUsersDB();
voidcreateUsersTable(sqlite3 *db);
intmain(void) {
    sqlite3 *db =openUsersDB();
    createUsersTable(db);
    sqlite3_close(db);
    return 0;
}
sqlite3 *openUsersDB() {
    sqlite3 *db;
    char *err_msg = 0;
    int rc =sqlite3_open("users.db", &db);
    if (rc !=SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n",
            sqlite3_errmsg(db));
        sqlite3_close(db);
        return 0;
    }
    returndb;
}

voidcreateUsersTable(sqlite3 *db) {
    int rc;
    char *err_msg = 0;
    char *sql = "DROP TABLE IF EXISTS Users;"
        "CREATE TABLE Users(Name TEXT PRIMARY KEY, Password TEXT,
        Right INTEGER);"
        "INSERT INTO Users VALUES('admin', 'admin', 0);"
        "INSERT INTO Users VALUES('user', 'user', 1)";
    rc =sqlite3_exec(db, sql, 0, 0, &err_msg);
    if (rc !=SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", err_msg);
        sqlite3_free(err_msg);
        sqlite3_close(db);
        return;
    }
}

```

transferFiles.c

```

#include "transferFiles.h"

int sendFile(int sd, char *fileName, int key) {
    FILE *in = fopen(fileName, "r");
    /* Check if the file exists. */
    if (doesFileExist(fileName) == 0) {
        return 0;
    }

    char *sendBuff = (char *)malloc(MAXBUFFLENGTH * sizeof(char));

```

```

bzero(sendBuff, MAXBUFFLENGTH);
int buffSize;
/* reading from the file */
if((buffSize = fread(sendBuff, sizeof(char), MAXBUFFLENGTH, in))
    >0) {
    printf("%s\n", sendBuff);
    fflush(stdout);

    /* send the read content to the server. */
    if(writelnFdWithTPP(sd, sendBuff, key) == 0) {
        printf("ERROR: Failed to send file %s.\n", fileName);
        return 0;
    }
    bzero(sendBuff, MAXBUFFLENGTH);
}

/* free mem. */
free(sendBuff);
return 1;
}

int receiveFile(int sd, char *fileName, int key) {
    FILE *out = fopen(fileName, "w");
    /* checking to see if I could open the file. */
    if (out == 0) {
        printf("Error while opening the file.\n");
        return 0;
    }

    int buffSize;
    int writeSize;
    char *recvbuf;
    /* reading the contents of the file. */
    if((recvbuf = readFromFdWithTPP(sd, key)) != 0) {
        if (strlen(recvbuf) != 0) {
            printf("%s\n", recvbuf);
            fflush(stdout);
        }
        /* write the contents in the file. */
        for (int i = 0; i < strlen(recvbuf); i++) {
            fprintf(out, "%c", recvbuf[i]);
        }
    }
    fclose(out);
    free(recvbuf);

    return 1;
}

int doesFileExist(const char *filename) {

```

```

    struct stat st;
    /* check if the file exists. */
    int result = stat(filename, &st);

    return result == 0;
}

int removeFilesFromDirectory(char *directoryPath) {
    DIR *directory;
    struct dirent *inFile;
    /* open the directory. */
    if (NULL == (directory = opendir(directoryPath))) {
        printf("Failed to open the directory!\n");

        return 0;
    }
    char *filePath = (char *)malloc(MAXFILENAMELENGTH * sizeof(char)
        );

    /* browse through the files in the directory. */
    while ((inFile = readdir(directory))) {
        if (!strcmp (inFile->d_name, "."))
            continue;
        if (!strcmp (inFile->d_name, ".."))
            continue;
        sprintf(filePath, "%s%s", directoryPath, inFile->d_name);
        /* deleting the file. */
        if (remove(filePath) == -1) {
            return 0;
        }
    }

    return 1;
}

```

server.c

```

#include "connection.h"

int initServer()
{
    /* Data structures required for connection. */
    struct sockaddr_in server;
    struct sockaddr_in from;

    int sd;
    /* I create the socket through which acceptance will be made. */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        printf("[server] Error creating socket!\n");
    }
}

```

```

    exit(-1);
}

/* using the SO_REUSEADDR option */
int on = 1;
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

/* I initialize the sockaddr structures. */
bzero(&server, sizeof(server));
bzero(&from, sizeof(from));

/* Set family */
server.sin_family = AF_INET;
/* I accept connection to any available address. */
server.sin_addr.s_addr = htonl(INADDR_ANY);
/* I accept connections to the PORT port. */
server.sin_port = htons(PORT);

/* Bind the socket to the sockaddr structure */
if (bind(sd, (struct sockaddr *)&server, sizeof(struct sockaddr))
    == -1)
{
    printf("[server] Binding error!\n");
    exit(-1);
}
/* We start listening for connections. */
if (listen(sd, MAXCLIENTS) == -1)
{
    printf("[server] Listening error!\n");
    exit(-1);
}

return sd;
}

int connectToServer(char *ip, char *port)
{
    int sd;
    struct sockaddr_in server;
    int serverPort = atoi(port);

    /* Create client socket. */
    if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        printf("[client] Socket() error.\n");
        exit(-1);
    }

    /* We set the server data. */
    server.sin_family = AF_INET;

```

```

server.sin_addr.s_addr = inet_addr(ip);
server.sin_port = htons(serverPort);

/* We make the connection. */
if (connect(sd, (struct sockaddr *)&server, sizeof(struct
    sockaddr)) == -1)
{
    printf("[client] Connection error!\n");
    exit(-1);
}

return sd;
}

struct command receiveCommand(int sd, int key)
{
    struct command commandReceived;
    /* I read the command name from the socket. */
    if ((commandReceived.commandName = readFromFdWithTPP(sd, key))
        == 0)
    {
        printf("[server] Error reading command name!\n");
        commandReceived.commandName = (char *) malloc(MAXCOMMANDELENGTH
            * sizeof(char));
        strcpy(commandReceived.commandName, "Error");
        return commandReceived;
    }
    int x;
    char c;
    /* I read the junk character that remains in the buffer. */
    if (read(sd, &c, sizeof(char)) < 1)
    {
        printf("[server] Error reading junk character!\n");
        strcpy(commandReceived.commandName, "Error");
        return commandReceived;
    }
    /* I read the number of parameters. */
    if (read(sd, &x, sizeof(int)) < sizeof(int))
    {
        printf("[server] Error reading nrParameters!\n");
        strcpy(commandReceived.commandName, "Error");
        return commandReceived;
    }
    /* Convert it to the host format. */
    x = ntohl(x);
    commandReceived.nrParameters = x;
    commandReceived.parameters = (char **) malloc(MAXNRPARAMETERS *
        sizeof(char *));
    /* I read all the parameters. */
    for (int i = 0; i < commandReceived.nrParameters; i++)

```

```

{
    commandReceived.parameters[i] = readFromFdWithTPP(sd, key);
    /* I read the junk character that remains in the buffer. */
    if (read(sd, &c, sizeof(char)) < 1)
    {
        printf("[server]Error reading junk character!\n");
        strcpy(commandReceived.commandName, "Error");
        return commandReceived;
    }
}

return commandReceived;
}

int sendCommand(int sd, struct command commandToBeSent, int key)
{
    /* Send the command name. */
    if (writeInFdWithTPP(sd, commandToBeSent.commandName, key) == 0)
    {
        return -1;
    }

    /*Send the number of parameters converted to the network format.
    */
    commandToBeSent.nrParameters = htonl(commandToBeSent.
        nrParameters);
    if (write(sd, &commandToBeSent.nrParameters, sizeof(int)) == 0)
    {
        return -1;
    }
    commandToBeSent.nrParameters = ntohl(commandToBeSent.
        nrParameters);
    /* Send parameter by parameter. */
    for (int i = 0; i < commandToBeSent.nrParameters; i++)
    {
        if (writeInFdWithTPP(sd, commandToBeSent.parameters[i], key)
            == 0)
        {
            return -1;
        }
    }
    return 0;
}

int writeInFd(int fd, char *buff)
{
    // NOT USED
    int n = strlen(buff);
    if (write(fd, &n, sizeof(int)) < sizeof(int))

```

```

{
    printf("Typing error!\n");
    return 0;
}
if (write(fd, buff, n + 1) < n + 1)
{
    printf("Typing error!\n");
    return 0;
}
return 1;
};

char *readFromFd(int fd)
{
    // NOT USED
    int n, m;
    if (read(fd, &n, sizeof(int)) < sizeof(int))
    {
        return 0;
    }
    if (n == 0)
    {
        return 0;
    }
    fflush(stdout);
    char *buff = (char *)malloc(n * sizeof(char));
    if (read(fd, buff, n) < n)
    {
        return 0;
    }

    buff[n] = 0;

    return buff;
};

int writeInFdWithTPP(int fd, char *buff, int key)
{
    /* STEP 1: Encrypt and send the message */
    int n = strlen(buff);

    encryptData(buff, strlen(buff), key);

    if (write(fd, &n, sizeof(int)) < sizeof(int))
    {
        printf("Typing error!\n");
        return 0;
    }
    if (write(fd, buff, n + 1) < n + 1)

```



```

{
    printf("Typing error!\n");
    return 0;
}
decryptData(buff, strlen(buff), key);

/* STEP 2: I read the encrypted message twice. */
int length, m;
if (read(fd, &length, sizeof(int)) < sizeof(int))
{
    return 0;
}
if (length == 0)
{
    return 0;
}
char *buffEncrypted = (char *)malloc(length * sizeof(char));
if (read(fd, buffEncrypted, length) < length)
{
    return 0;
}
buffEncrypted[length] = 0;

char c;
if (read(fd, &c, sizeof(char)) < 0)
{
    printf("Error while reading the junk character!\n");
}

/* STEP 3: I decrypt the message and forward it. */
n = strlen(buffEncrypted);
decryptData(buffEncrypted, n, key);
if (write(fd, &n, sizeof(int)) < sizeof(int))
{
    printf("Typing error!\n");
    return 0;
}
if (write(fd, buffEncrypted, n + 1) < n + 1)
{
    printf("Typing error!\n");
    return 0;
}

return 1;
};

char *readFromFdWithTPP(int fd, int key)
{
    /* I read the encrypted message */
    int n, m;

```

```

if (read(fd, &n, sizeof(int)) < sizeof(int))
{
    return 0;
}
if (n == 0)
{
    return 0;
}
char *buff = (char *)malloc(n * sizeof(char));
if (read(fd, buff, n) < n)
{
    return 0;
}

buff[n] = 0;

char c;
if (read(fd, &c, sizeof(char)) < 0)
{
    printf("Error while reading the junk character!\n");
}

/* I encrypt the message again and send it via fd. */
encryptData(buff, n, key);
if (write(fd, &n, sizeof(int)) < sizeof(int))
{
    printf("Typing error!\n");
    return 0;
}
if (write(fd, buff, n + 1) < n + 1)
{
    printf("Typing error!\n");
    return 0;
}

/* I read the message and decrypt it to get the correct message.
   */
if (read(fd, &n, sizeof(int)) < sizeof(int))
{
    return 0;
}
if (n == 0)
{
    return 0;
}
if (read(fd, buff, n) < n)
{
    return 0;
}

```

```

    buff[n] = 0;

    decryptData(buff, n, key);

    return buff;
};

void encryptData(char *data, int length, int key)
{
    /* I encrypt data according to Caesar's cipher. */
    for (int i = 0; i < length; i++)
    {
        data[i] += key;
    }
}

void decryptData(char *data, int length, int key)
{
    /* I decrypt the data according to Caesar's cipher. */
    for (int i = 0; i < length; i++)
    {
        data[i] -= key;
    }
}

void readJunkFromFD(int fd)
{
    /* Fac non-blocking read. */
    fcntl(fd, F_SETFL, fcntl(fd, F_GETFL) | O_NONBLOCK);
    char *buff = malloc(MAXJUNK * sizeof(char));
    read(fd, buff, MAXJUNK);
    /* Redo read blocker. */
    fcntl(fd, F_SETFL, fcntl(fd, F_GETFL) & ~O_NONBLOCK);

    /* Free memory. */
    free(buff);
}

```