

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Постановка задачи	4
1.2 Анализ существующих решений	4
1.2.1 tty-share	4
1.2.2 tmate	5
1.3 Модель клиент-сервер	5
1.4 Сокеты	6
1.4.1 Принципы сокетов	7
1.4.2 Основные функции сокетов	7
1.4.3 Типы сокетов	8
1.5 Обработка запросов клиентов	8
1.5.1 Обработка всех запросов в одном потоке	9
1.5.2 Обработка каждого запроса в отдельном потоке	9
1.5.3 Организация пула потоков	10
1.6 Протоколы	10
1.6.1 Протоколы транспортного уровня	10
1.6.2 Протоколы прикладного уровня	11
2 Конструкторская часть	13
2.1 Состав программного обеспечения	13
2.2 Функциональная модель	13
2.3 Сценарий использования	13
2.4 Проектирование протокола прикладного уровня	13
3 Технологическая часть	15
3.1 Используемые технологии	15
3.1.1 Язык программирования	15
3.1.2 ncurses	15
3.1.3 boost	16
3.2 Детали реализации	16
3.2.1 Реализация сервера	16

3.2.2	Реализация клиента	16
3.2.3	Передача пакетов	16
3.3	Примеры работы разработанного ПО	16
Заключение		17
Литература		18

Введение

Удаленное подключение к терминалу позволяет пользователю управлять работой удаленного сервера или рабочей станции по сети. Такой подход может быть полезен, например, системным администраторам для администрирования выделенных серверов и технической поддержки пользователей. На основе удаленного подключения реализуется возможность делиться сессией терминала, то есть использовать терминал несколькими пользователями из разных сетей одновременно. С помощью этой возможности можно совместно редактировать файлы, отлаживать какой-либо проект командой разработчиков и так далее.

Целью данной работой является разработка программного обеспечения, позволяющего удаленно подключения терминалу нескольких пользователей и его редактирования. Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ существующих решений;
- изучить существующие протоколы прикладного уровня;
- разработать свой протокол прикладного уровня для поставленной задачи;
- реализовать программное обеспечение (сервер и клиент) с использованием разработанного протокола для удаленного подключения к терминалу.

1 Аналитическая часть

В данном разделе проводится сравнительный анализ методов решения поставленной задачи, делается обоснованный выбор метода.

1.1 Постановка задачи

В соответствии с заданием необходимо разработать программное обеспечение для удаленного подключения к терминалу: серверное и клиентское приложение. Сервер должен поддерживать связь с множеством клиентов одновременно, обрабатывая запросы от каждого и рассылая изменения всем остальным. По необходимости сервер должен запустить команду терминала, высылая результат всем подключенным клиентам. Для решения этой задачи необходимо изучить предметную область и проанализировать существующие решения.

1.2 Анализ существующих решений

1.2.1 `tty-share`

`tty-share` [1] - инструмент, используемый для совместного редактирования терминала. Сеансом можно поделиться как в локальной сети, так и в удаленной. При совместном использовании терминала через сеть Internet, `tty-share` подключается к прокси-серверу, который является посредником при обмене данными между участниками. Экземпляр этого сервера может быть размещен как на вашей машине, так и на `tty-share.com`. Сквозное шифрование при таком подключении отсутствует.

К достоинствам `tty-share` можно отнести:

- возможность подключения к сессии из браузера;
- отсутствие зависимостей. `tty-share` представляет из себя статический кроссплатформенный двоичный файл, который не содержит зависимостей;

1.2.2 tmate

`tmate` [2] – ответвление терминального мультиплексора `tmux` [3], который обеспечивает безопасное, мгновенное и простое в использовании решение для совместного использования терминалов с помощью `ssh`-соединения [4]. На рисунке 1.1 представлена схема архитектуры `tmate` с различными компонентами.

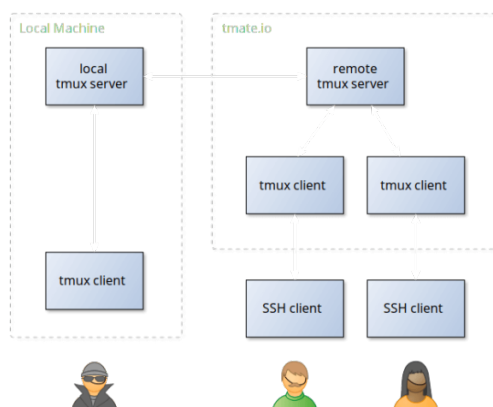


Рис. 1.1: Схема архитектуры `tmate`. Слева представлена архитектура при запуске в локальной сети, справа – в удаленной сети

Локальный, как и удаленный сервер `tmux` взаимодействует с протоколом поверх стерилизатора данных `msgpack` [5], который сжимается через `ssh`-соединение для повышения эффективности пропускной способности сети [2].

К достоинствам `tmate` можно отнести:

- возможность подключения как в локальной сети, так и в удаленной;
- использование `ssh`-сессий;
- возможность разместить `tmate`-сервер как на своей машине, так и на серверах `tmate.io`.

1.3 Модель клиент-сервер

В модели клиент-сервер роли определены: сервер предоставляет ресурсы и службы одному или нескольким клиентам, которые обращаются

к серверу за обслуживанием. В качестве примеров серверов можно привести веб-серверы, почтовые серверы и файловые серверы. Каждый из этих серверов предоставляет ресурсы для клиентских устройств, таких как настольные компьютеры, ноутбуки, планшеты и смартфоны. Большинство серверов могут устанавливать отношение «один ко многим» с клиентами, что означает, что один сервер может предоставлять ресурсы нескольким клиентам одновременно. Когда клиент запрашивает соединение с сервером, сервер может либо принять, либо отклонить это соединение. Если соединение принято, сервер устанавливает и поддерживает соединение с клиентом по определенному протоколу. Например, почтовый клиент может запросить SMTP-соединение с почтовым сервером для отправки сообщения. Затем приложение SMTP на почтовом сервере запросит проверку подлинности у клиента, например адрес электронной почты и пароль. Если эти учетные данные совпадают с учетной записью на почтовом сервере, сервер отправит электронное письмо целевому получателю. Часто клиенты и серверы взаимодействуют через компьютерную сеть на разных аппаратных средствах, но и клиент и сервер могут находиться в одной и той же системе. Хост сервера запускает одну или несколько серверных программ, которые совместно используют свои ресурсы с клиентами. Клиент не предоставляет общий доступ ни к одному из своих ресурсов, но запрашивает данные или службу у сервера. Поэтому клиенты инициируют сеансы связи с серверами, которые ожидают входящих запросов. Клиенту не известно о том, как работает сервер при выполнении запроса и доставке ответа. Клиент должен только понимать ответ, основанный на хорошо известном прикладном протоколе, т.е. содержание и форматирование данных для запрашиваемой службы. Клиенты и серверы обмениваются сообщениями в шаблоне обмена сообщениями запрос-ответ. Клиент отправляет запрос, а сервер возвращает ответ.

1.4 Сокеты

Сокет (англ. socket [6]) — конечный пункт передачи данных, абстракция, обозначающая одно из окончаний сетевого соединения. Каждая из программ, устанавливающих соединение, должна иметь собственный сокет.

Сокеты – это интерфейс прикладного программирования для сетевых приложений TCP/IP. Интерфейс сокетов был создан в восьмидесятых годах для операционной системы UNIX. Позднее интерфейс сокетов был перенесен в Microsoft Windows. Сокеты до сих пор используются в приложениях для сетей TCP/IP. Сетевые приложения используют сокет, как виртуальные разъемы для обмена данными между собой. Сокеты бывают трех видов: клиентские, слушающие и серверные.

1.4.1 Принципы сокетов

Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы (в UNIX непривилегированные процессы не могут использовать порты меньше 1024). Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т.д. Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут общаться любые локальные процессы путём чтения/записи из него. Сокеты типа INET доступны из сети и требуют выделения номера порта [7]. Обычно клиент явно подсоединяется к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером.

1.4.2 Основные функции сокетов

В таблицах 1.1 - 1.3 представлены общие функции, функции сервера и клиента соответственно.

Функция	Описание
<code>socket</code>	Создать новый сокет и вернуть файловый дескриптор
<code>send</code>	Отправить данные по сети
<code>receive</code>	Получить данные по сети
<code>close</code>	Закрыть соединение

Таблица 1.1: Общие функции

Функция	Описание
<code>bind</code>	Связать сокет с IP-адресом и портом
<code>listen</code>	Слушает порт и ждет когда будет установлено соединение
<code>accept</code>	Принять запрос на установку соединения

Таблица 1.2: Функции сервера

1.4.3 Типы сокетов

Различают два типа сокетов:

- сокет с предварительным установлением соединения, когда до начала передачи данных устанавливаются адреса сокетов отправителя и получателя данных – такие сокет соединяются друг с другом и остаются соединенными до окончания обмена данными;
- сокет без установления соединения, когда соединение до начала передачи данных не устанавливается, а адреса сокетов отправителя и получателя передаются с каждым сообщением.

Если тип сокета – виртуальный канал, то сокет должен устанавливать соединение, если же тип сокета – датаграмма, то, как правило, это сокет без установления соединения, хотя последнее не является требованием.

1.5 Обработка запросов клиентов

При приходе клиентского запроса у сервера имеется несколько вариантов действий:

Функция	Описание
<code>connect</code>	Установить соединение

Таблица 1.3: Функции клиента

- обрабатывать все запросы в одном потоке;
- обрабатывать каждый запрос в отдельном потоке;
- организовать пул потоков.

1.5.1 Обработка всех запросов в одном потоке

Это решение подходит только для ограниченного числа случаев, в которых количество клиентов невелико, и обращаются они к серверу не часто. Эта самая простая схема работы: минимум потоков, минимум ресурсов, нет синхронизации. Необходимо построить очередь входящих запросов, чтобы они не терялись при последовательной обработке. Но при большом количестве клиентов, клиенты будут долго ждать ответа от сервера.

1.5.2 Обработка каждого запроса в отдельном потоке

Для каждого клиентского запроса создается отдельный поток. Такая схема работает следующим образом: первичный поток приложения прослушивает клиентские запросы и при поступлении каждого создает новый поток, передавая ему клиентский пакет (данные или команду). Созданный поток выполняет соответствующую обработку, передает результаты обратно клиенту, или же помещает их в БД, и завершает свое существование.

Основные недостатки такой модели:

- частое создание и завершение потоков;
- малое время работы потока;
- нерегулируемое количество потоков;

- в большинстве случаев отсутствие очереди клиентских запросов;
- большое количество переключений контекстов рабочих потоков.

Для решения этих проблем и предназначен пул потоков.

1.5.3 Организация пула потоков

Имеется главный поток приложения, прослушивающий клиентские запросы. Пул потоков создается заранее или при поступлении первого запроса. При поступлении запроса главный поток выбирает поток из пула и передает ему запрос. Если все потоки заняты обработкой, то есть активны, пакет ставится в очередь и ждет освобождения одного из потоков. Алгоритмы добавления потоков в пул и определения оптимального размера пула сильно зависят от решаемой задачи [8].

1.6 Протоколы

Протокол – набор правил, определяющих взаимодействие двух одноименных уровней модели взаимодействия открытых систем в различных абонентских ЭВМ. Стек протоколов — набор взаимодействующих сетевых протоколов.

Наиболее популярные стеки протоколов: TCP/IP, IPX/SPX, NetBIOS/SMB, DECnet, SNA и OSI. Большинство протоколов (все из перечисленных, кроме SNA) одинаковы на физическом и канальном уровне, но на других уровнях как правило используют разные протоколы.

1.6.1 Протоколы транспортного уровня

Протоколы транспортного уровня предназначены для обеспечения непосредственного информационного обмена между двумя пользовательскими процессами. Существует два типа протоколов транспортного уровня

– сегментирующие протоколы и не сегментирующие протоколы доставки дейтаграмм [9].

Протоколы доставки дейтаграмм просты для реализации, однако, не обеспечивают гарантированной и достоверной доставки сообщений.

В качестве протоколов транспортного уровня в сети Internet могут быть использованы два протокола:

- UDP – User Datagram Protocol;
- TCP – Transmission Control Protocol.

Описание принципов построения протокола UDP приведено в RFC 768. Для передачи сообщений UDP используются пакеты IP. Сообщения UDP в данном случае размещаются в поле данных переносящего их пакета.

Протокол TCP используется для обеспечения надежного информационного обмена на транспортном уровне в сетях Internet.

Существует достаточно много причин, которые могут помешать пакету, который передается в сети, успешно достичь станции назначения. Таким образом, если не будут использованы специальные методы для обеспечения гарантированной доставки, принятое сообщение может существенным образом отличаться от того сообщения, которое было передано.

Надежный информационный обмен предполагает следующие возможности:

- потоковый обмен;
- использование виртуальных соединений;
- буферизированная передача данных;
- неструктурированный поток;
- обмен в режиме полного дуплекса.

1.6.2 Протоколы прикладного уровня

Протоколы прикладного уровня описывают взаимодействие между клиентской и серверной частями программы. Прикладные протоколы работа-

ют на верхнем уровне модели OSI. Они обеспечивают взаимодействие приложений и обмен данными между ними.

Далее будет подробнее рассмотрен протокол HTTP.

«Классическая» схема HTTP-сеанса выглядит так:

- установление TCP-соединения;
- запрос клиента;
- ответ сервера;
- разрыв TCP-соединения.

Запрос клиента содержит в себе заголовок и тело запроса. Заголовок содержит в себе строку состояния, которая имеет следующий формат: **метод запроса, URL ресурса, версия протокола HTTP**. Метод, указанный в строке состояния, определяет способ взаимодействия на ресурс, URL которого задан в той же строке. Методу может принимать значения GET, POST, HEAD, PUT, DELETE и т.д.

Ответ сервера клиенту начинается со строки состояния, которая имеет следующую форму: **версия протокола, код ответа, сообщение**. Версия протокола задается в том же формате, что и в запросе клиента. Код ответа – трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером. Пояснительное сообщение дублируют код ответа в символьном виде.

Вывод

В поставленной необходимо чтобы пакет, отправленный клиентом, гарантировано доходил до сервера и данные не терялись при передаче. Таким образом, в качестве протокола транспортного уровня был выбран протокол TCP. Были проанализированы аналоги и рассмотрены особенности существующих протоколов прикладного уровня для дальнейшего проектирования собственного протокола прикладного уровня. В качестве механизма обработки клиентов был выбрана обработка запросов в одном потоке.

2 Конструкторская часть

В данном разделе рассматривается процесс проектирования структуры программного обеспечения.

2.1 Состав программного обеспечения

Программное обеспечение состоит из клиент-серверного приложения.

Структура разрабатываемого проекта:

- сервер – программа, обрабатывающая запросы клиентов и являющаяся источником информации для них;
 - клиент – программа, которая является инициатором соединения и ...
- TODO

2.2 Функциональная модель

TODO

2.3 Сценарий использования

TODO

2.4 Проектирование протокола прикладного уровня

TODO

Вывод

В данном разделе был спроектирована структура разрабатываемого программного обеспечения.

3 Технологическая часть

В данном разделе рассматривается выбор технологий для реализации поставленной задачи, листинги реализации разработанного программного обеспечения и приведены результаты работы ПО.

3.1 Используемые технологии

3.1.1 Язык программирования

Для реализации данного проекта был выбран язык программирования C++ [10] по нескольким причинам:

- наличие для языка C++ необходимых для реализации задачи библиотек;
- скорость исполнения кода;
- хорошее знание языка C++;

3.1.2 ncurses

С помощью библиотеки **ncurses** [11], предназначенной для управления ввода-вывода на терминал и позволяющей задавать экранные координаты и цвет выводимых символов, реализуется окно терминала на стороне клиента. Библиотека **ncurses** была выбрана в связи с отсутствием хорошо задокументированных аналогов.

3.1.3 boost

3.2 Детали реализации

3.2.1 Реализация сервера

3.2.2 Реализация клиента

3.2.3 Передача пакетов

3.3 Примеры работы разработанного ПО

Вывод

В данном разделе был обоснован выбор технологий для решения поставленной задачи, рассмотрены листинги реализованного программного обеспечения и приведены результаты работы ПО.

Заключение

В ходе выполнения данной работы:

- был проведен анализ существующих решений;
- были изучены протокол прикладного уровня HTTP;
- был разработан свой протокол прикладного уровня для поставленной задачи;
- было реализовано программное обеспечение (сервер и клиент) с использованием разработанного протокола для удаленного подключения к терминалу.

Таким образом, цель курсовой работы была достигнута – было разработано ПО в составе клиента и сервера для удаленного подключения к терминалу нескольких пользователей одновременно.

Литература

- [1] tty-share terminal sharing [Электронный ресурс]. Режим доступа: <https://tty-share.com/> (дата обращения: 08.12.2021).
- [2] tmate – Instant terminal sharing [Электронный ресурс]. Режим доступа: <https://tmate.io/> (дата обращения: 08.12.2021).
- [3] tmux source code - GitHub [Электронный ресурс]. Режим доступа: <https://github.com/tmux/tmux> (дата обращения: 08.12.2021).
- [4] SSH Communications Security [Электронный ресурс]. Режим доступа: <https://www.ssh.com/> (дата обращения: 08.12.2021).
- [5] MessagePack: It's like JSON. But fast and small. [Электронный ресурс]. Режим доступа: <https://msgpack.org/index.html> (дата обращения: 08.12.2021).
- [6] What Is a Socket? (The Java™ Tutorials) [Электронный ресурс]. Режим доступа: <https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html> (дата обращения: 08.12.2021).
- [7] Сокеты - сетевое программирование [Электронный ресурс]. Режим доступа: <https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html> (дата обращения: 08.12.2021).
- [8] Многопоточность [Электронный ресурс]. Режим доступа: <https://se.ifmo.ru/documents/10180/1422934/prog-concurrency.pdf/da25f4c8-02fc-506d-79da-7ea9a2186dac> (дата обращения: 08.12.2021).
- [9] Протоколы транспортного уровня [Электронный ресурс]. Режим доступа: http://opds.spbsut.ru/data/_uploaded/mu/itm_psu/vlss_present/psu-09_tcp_udp.pdf (дата обращения: 08.12.2021).
- [10] Standard C++ [Электронный ресурс]. Режим доступа: <https://isocpp.org/> (дата обращения: 08.12.2021).

- [11] NCURSES Programming [Электронный ресурс]. Режим доступа: <https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/> (дата обращения: 08.12.2021).