Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные системы и среды»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.Д.Владымцев |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Кроссплатформенная система обмена сообщениями»**

БГУИР КП 1-40 04 01 002 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 053505  Вашкевич Николай Владимирович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc702603608)

[1 Архитектура вычислительной системы 3](#_Toc791370142)

[2 Платформа программного обеспечения 5](#_Toc862657394)

[3 Теоретическое обоснование разработки программного продукта 14](#_Toc776730184)

[4 Проектирование функциональных возможностей программы 20](#_Toc1935942257)

[5 Архитектура разрабатываемой программы 24](#_Toc1512791070)

[ЗАКЛЮЧЕНИЕ 32](#_Toc1655140668)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 33](#_Toc1117560812)

[ПРИЛОЖЕНИЕ А(обязательное)Исходный код программы 35](#_Toc1034860831)

[ПРИЛОЖЕНИЕ Б(обязательное)Функциональная схема алгоритма 46](#_Toc249836718)

[ПРИЛОЖЕНИЕ В(обязательное)Блок схема программы 47](#_Toc786255223)

[ПРИЛОЖЕНИЕ Г(обязательное)Графический интерфейс 48](#_Toc1794817091)

[ПРИЛОЖЕНИЕ Д(обязательное)Ведомость 49](#_Toc882122453)

****ВВЕДЕНИЕ****

В современном информационном мире все большее значение приобретает обмен сообщениями между различными устройствами и платформами. В связи с этим, разработка кроссплатформенных систем обмена сообщениями является актуальной задачей, которая позволяет упростить процесс обмена информацией между различными устройствами и платформами. Целью данной курсовой работы является разработка и реализация кроссплатформенной системы обмена сообщениями, которая позволит упростить процесс обмена информацией и повысить его надежность и безопасность.

Обмен сообщениями — это процесс передачи информации между двумя или более устройствами или программами. Сообщения могут быть различной природы — это могут быть текстовые сообщения, звуковые сигналы, изображения, файлы и другое. Обмен сообщениями широко используется в различных областях, включая интернет-коммуникации, бизнес-процессы, телекоммуникации, автоматизацию производственных процессов. При обмене сообщениями важными параметрами являются скорость передачи, надежность и безопасность, так как информация может быть критически важной, ее потеря или неправильная передача может привести к серьезным последствиям.

Цели курсового проекта:

1 Приобретение теоретических и практических навыков организации обмена текстовыми сообщениями с использованием протокола шифрования по открытому и закрытому ключу.

2 Реализация программного продукта по обмену сообщениями между пользователями.

Задачи курсового проекта:

1 Изучение существующих технологий и протоколов обмена сообщениями, их достоинств и недостатков.

2 Анализ требований к системе, установка основных критериев ее функционирования и надежности.

3 Разработка архитектуры системы, определение ее основных компонентов и интерфейсов.

4 Разработка и реализация кроссплатформенного приложения, осуществляющего обмен сообщениями.

# Архитектура вычислительной системы

Программный продукт разрабатывается с учётом возможности использования как на современных вычислительных системах, так и для систем старшего поколения. Данная цель будет достигнута использованием многопоточности при реализации как серверной, так и клиентской части.

Основной вычислительной системой для разработки ПО использовался ноутбук Lenovo IdeaPad 3, изображенный на рисунке 1:



Рисунок 1 - Внешний вид ноутбука Lenovo IdeaPad 3

Данная вычислительная система имеет следующие аппаратные характеристики:

1 Центральный процессор (CPU)[1] - устройство, которое отвечает за выполнение основных вычислительных задач и управление работой системы. В ноутбуке Lenovo IdeaPad 3 установлен процессор Intel Core i5 10th Generation с тактовой частотой 2.8 ГГц и четырьмя ядрами.

2 Оперативная память (RAM)[2] - устройство, которое служит для временного хранения данных, необходимых для выполнения вычислительных операций. Используемая вычислительная система оснащена оснащена 8 ГБ оперативной памяти типа DDR3.

3 Жесткий диск (HDD) - устройство для хранения данных на постоянной основе. В ноутбуке Lenovo IdeaPad 3 установлен жесткий диск емкостью 256 ГБ и скоростью вращения шпинделя 5400 об/мин.

4 Видеокарта (GPU) - устройство, которое отвечает за обработку графических данных. Ноутбук Lenovo IdeaPad 3 оснащен графической картой Nvidia MX330 с 2Gb видеопамяти.

5 Дисплей - устройство, которое отображает информацию на экране. Ноутбук Lenovo IdeaPad 3 имеет дисплей диагональю 15.6 дюйма с разрешением 1366x768 пикселей.

6 Беспроводные модули[3] - устройства, которые обеспечивают беспроводную связь. Ноутбук Lenovo IdeaPad 3 оснащен модулями Wi-Fi и Bluetooth.

7 Порты[4] - разъемы для подключения внешних устройств. Ноутбук Lenovo IdeaPad 3 имеет порты USB 3.1 Gen 1, USB 2.0, HDMI, аудиовыход и кардридер.

Выбор вычислительной системы с данным центральным процессором выбран по причине того, что процессоры Intel Core i5 10-го поколения отличаются улучшенной производительностью, за счёт увеличения тактовой частоты процессора и улучшения архитектуры. Также нельзя не отметить уменьшенное количество энергопотребления, что позволяет увеличить время работы от заряженного аккумулятора.

# Платформа программного обеспечения

Перед выполнением части курсового проекта, связанного непосредственно с разработкой программного обеспечения, необходимо определить требования к платформам, для которых будет разработано соответствующее ПО.

В первую очередь необходимо определить список операционных систем. Так как основными языками программирования были выбраны C++ и Python, то большее количество библиотек, которые непосредственно влияют на скорость и качество разработки систем обмена сообщений, будут обладать десктопные операционные системы, представленные на рисунке 2.1:

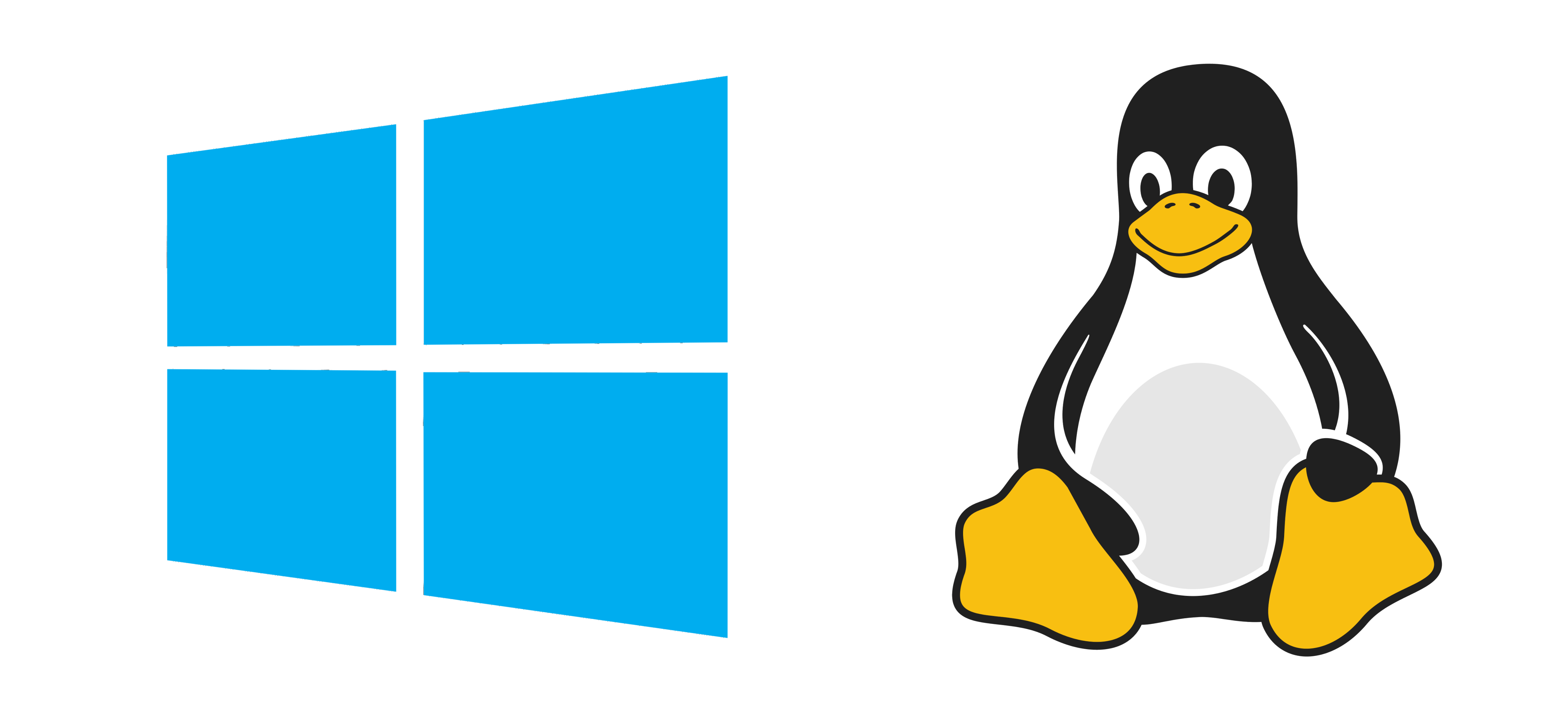


Рисунок 2.1 - Операционные системы Windows и Linux

В ходе выполнения курсовой работы стал вопрос выпора операционной системы (см. рисунок 2.1), под которую будет разрабатываться программное обеспечение.

Проведём сравнительных анализ этих двух операционных систем, определим их главные преимущества и недостатки.

Windows — это операционная система, разработанная компанией Microsoft, которая является одной из самых популярных операционных систем в мире. Windows предназначен для управления компьютером и выполнения задач пользователем. Ядро Windows имеет многослойную архитектуру, которая включает в себя уровень аппаратной абстракции, уровень управления процессами, уровень управления памятью и уровень управления устройствами. Эти уровни работают вместе, чтобы обеспечить надежность и производительность ОС. Windows поддерживает многопользовательскую работу, что означает, что несколько пользователей могут работать на одном компьютере в то же время. Для этого каждому пользователю выделяется отдельная учетная запись, которая содержит его настройки и персональные данные. Windows также имеет систему доступа к ресурсам, которая позволяет контролировать доступ к файлам, папкам и другим ресурсам на компьютере. Windows имеет множество встроенных компонентов и служб, которые обеспечивают дополнительную функциональность, такую как служба безопасности, служба интернет-соединений, служба печати и другие. Эти компоненты и службы могут работать как самостоятельные процессы или как службы фоновой задачи.

Windows также поддерживает множество интерфейсов программирования приложений (API), которые позволяют разработчикам создавать программы для Windows и представлены на рисунке 2.2:

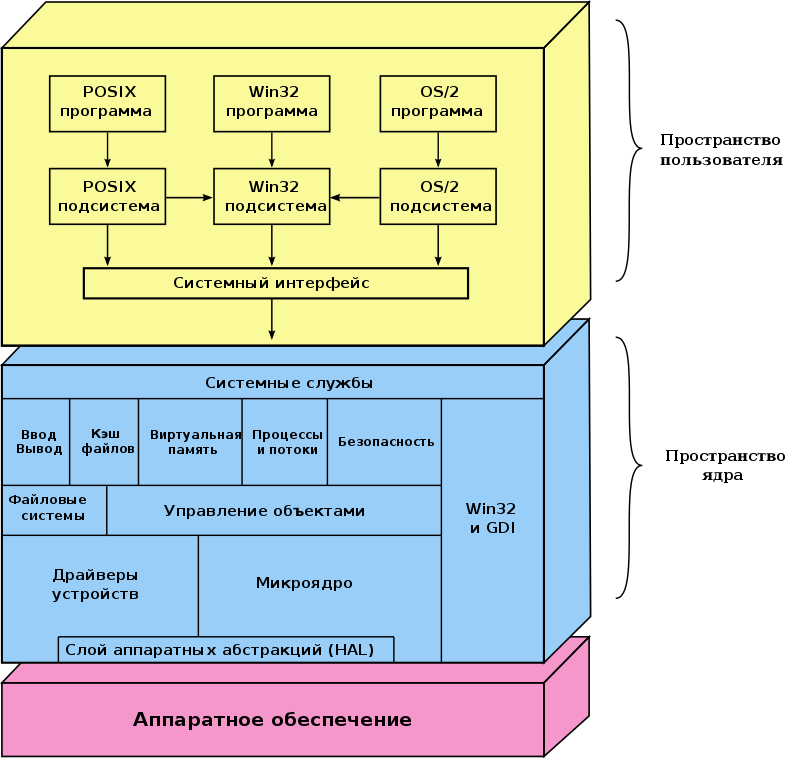


Рисунок 2.2 - Cхема всех API реализованных в OC Windows

Некоторые из наиболее популярных API включают в себя Win32 API, .NET Framework, Windows Runtime API и другие. Кроме того, Windows имеет интегрированную среду разработки Visual Studio, которая облегчает создание приложений для ОС Windows. В целом, Windows является высокоуровневой операционной системой, которая обладает широким функционалом и поддерживает множество устройств и программных платформ. В целом, Windows является одной из самых популярных операционных систем в мире, обладающей широким функционалом и поддерживающей множество устройств и программных платформ. Как системный программист, я отвечаю за создание и поддержку программ, которые работают на этой платформе, а также за обеспечение безопасности и защиты ОС Windows от вредоносных программ и хакерских атак.

Linux — это операционная система с открытым исходным кодом, которая была разработана Линусом Торвальдсом, который представлен на рисунке 2.3:



Рисунок 2.3 - Основатель ОС Linux Линус Торвальдс

В 1991 году и с тех пор стала популярной среди пользователей по всему миру. Одной из главных особенностей Linux является то, что она предоставляет пользователю полный контроль над системой. В отличие от Windows, где многие настройки скрыты от пользователя, в Linux можно настроить практически все параметры системы. Это может быть полезно для пользователей, которые хотят получить максимальный контроль над своей системой и настроить ее под свои потребности.

Linux также является многозадачной и многопользовательской операционной системой. Это означает, что несколько пользователей могут работать на одном компьютере в то же время, и каждый пользователь имеет свою учетную запись. Кроме того, Linux позволяет запускать множество приложений и процессов одновременно, что делает ее очень гибкой и мощной.

Linux имеет широкий спектр программного обеспечения, доступного для установки и использования. Большинство программ, которые работают на Windows, также могут работать на Linux, хотя иногда требуются небольшие изменения или адаптация. Кроме того, Linux имеет свой собственный набор программного обеспечения, который включает в себя браузеры, текстовые редакторы, офисные пакеты, графические редакторы и многое другое. Одним из главных преимуществ Linux является то, что она является операционной системой с открытым исходным кодом. Это означает, что любой желающий может просмотреть и изменить исходный код системы или любой ее компоненты. Это делает Linux более гибкой и безопасной, так как сообщество разработчиков постоянно работает над улучшением и обновлением системы.

Кроме того, Linux имеет ряд инструментов и программ, которые помогают с обслуживанием и настройкой системы, таких как командная строка, представленная на рисунке 2.4:



Рисунок 2.4 - Командная строка в ОС Linux

Также присутствуют утилиты мониторинга и диагностики, система пакетного управления и многое другое. Эти инструменты позволяют пользователям более гибко настраивать и управлять системой в соответствии с их потребностями.

Однако, существуют и недостатки у Linux. Один из них — это то, что она не имеет такой широкой поддержки программного обеспечения, как Windows. Это связано с тем, что Linux имеет меньшую долю рынка, чем Windows, и разработчики программного обеспечения могут не заинтересованы в создании версий своих продуктов для Linux. Кроме того, для работы с Linux необходимы некоторые навыки работы с командной строкой и понимание основных принципов работы операционной системы.

В целом, Linux является мощной операционной системой, которая может быть использована в различных сферах, включая серверные приложения, встроенные системы и десктопные компьютеры. Она обладает широким спектром инструментов и программного обеспечения, а также является более безопасной и надежной операционной системой, чем Windows. Однако, для работы с ней необходимо иметь некоторые навыки и понимание основных принципов ее работы.

Теперь конкретизируем и обоснуем выбор таких языков программирования как C++ и Python, в качестве основного из предложенных двух был выбран Python. C++ и Python — это два популярных языка программирования, которые используются в различных областях и имеют свои преимущества и недостатки.

C++ — это компилируемый язык программирования. Компилируемый язык программирования — это язык программирования, в котором программа переводится (компилируется) в машинный код, который может быть непосредственно исполнен процессором компьютера.

Компиляция происходит в несколько этапов, представленных на рисунке 2.5:

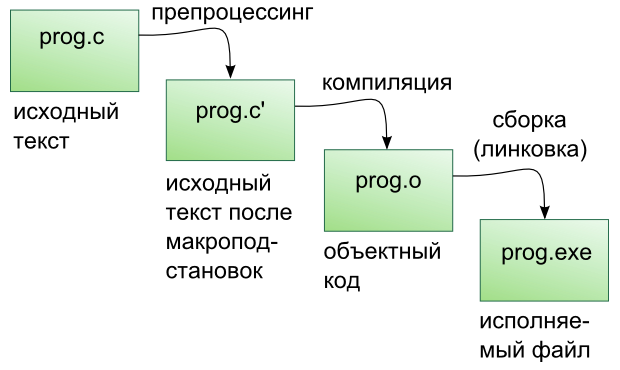


Рисунок 2.5 - Этапы компиляции программного обеспечения

Сначала исходный код на языке программирования анализируется компилятором и преобразуется в промежуточный код, который затем проходит оптимизацию. Затем промежуточный код переводится в машинный код, который может быть непосредственно исполнен на процессоре.

Компилируемые языки программирования имеют ряд преимуществ. Они обычно работают быстрее, чем интерпретируемые языки, потому что машинный код может быть напрямую исполнен процессором. Кроме того, компиляция обычно происходит один раз для каждой версии программы, что означает, что скомпилированная программа может быть запущена множество раз без необходимости повторной компиляции. Это делает компилируемые языки привлекательными для разработки крупных и сложных программ, которые должны быть эффективными по скорости работы.

Однако, компилируемые языки программирования также имеют некоторые недостатки. Они обычно более сложны в использовании и требуют от программиста более высокого уровня знаний и навыков. Кроме того, компиляция может занимать значительное время для больших программ, и если программа изменяется, то может потребоваться повторная компиляция. С++ широко используется для разработки системного и прикладного программного обеспечения, игр, мобильных приложений и других программ. C++ имеет высокую производительность и управляет ресурсами напрямую, что делает его эффективным для работы с памятью, устройствами ввода-вывода и другими аппаратными компонентами.

C++ также является объектно-ориентированным языком программирования. Объектно-ориентированный язык программирования (ООП) — это парадигма программирования, которая ориентирована на представление программы в виде набора объектов, которые взаимодействуют друг с другом для выполнения задач.

В ООП программный код организуется в классы, которые содержат данные и методы для работы с этими данными. Объекты представляют экземпляры классов, которые могут быть созданы во время выполнения программы.

Основные принципы ООП представлены на рисунке 2.6:



Рисунок 2.6 - Основные принципы объектно-ориентированного программирования

Наследование позволяет создавать новые классы на основе существующих классов, включая их данные и методы. Полиморфизм позволяет использовать один и тот же метод с различными объектами, что упрощает написание более гибких и многократно используемых программ. Инкапсуляция обеспечивает доступ к данным только через методы класса, что позволяет скрыть детали реализации и защитить данные от несанкционированного доступа.

Объектно-ориентированные языки программирования, такие как Java, Python и C++, широко используются для создания сложных программных систем, таких как веб-приложения, игры и операционные системы. ООП позволяет создавать модульный и переносимый код, упрощает поддержку и расширение программного обеспечения, а также улучшает его качество. Однако, C++ может быть сложен в использовании и требует более высокого уровня экспертизы по сравнению с другими языками программирования.

Python — это интерпретируемый язык программирования.

Интерпретируемый язык программирования — это язык, который выполняется непосредственно на компьютере, без предварительной компиляции в машинный код. Вместо этого, код интерпретируется на ходу, построчно, при помощи специальной программы, называемой интерпретатором.

Python - один из самых популярных интерпретируемых языков программирования. При написании программ на Python код сохраняется в текстовых файлах с расширением .py. Эти файлы затем передаются интерпретатору Python, который читает и анализирует код и выполняет его инструкции. Кроме того, Python использует динамическую типизацию, что означает, что тип переменной может изменяться в процессе выполнения программы.

Python имеет простой и лаконичный синтаксис, который облегчает чтение и написание кода. Он также предлагает множество встроенных функций и библиотек, которые упрощают написание программ и решение задач в различных областях. Python широко используется для разработки веб-приложений, научных вычислений, машинного обучения, автоматизации задач и многих других областей.

Одним из главных преимуществ интерпретируемых языков, таких как Python, является их быстрота и удобство отладки. Поскольку код выполняется непосредственно на компьютере, программист может быстро увидеть результаты своей работы и легко исправлять ошибки. Кроме того, Python позволяет интерактивный режим работы, что дает возможность быстро проверять и исправлять код на лету.

Python также поддерживает множество библиотек и фреймворков, которые упрощают разработку приложений и решение задач в различных областях. Например, библиотека NumPy и фреймворк TensorFlow позволяют работать с научными вычислениями и машинным обучением, а фреймворк Django облегчает разработку веб-приложений. Однако, Python может иметь более низкую производительность по сравнению с C++, особенно при работе с большими объемами данных. Python также может иметь проблемы с управлением памятью, что может привести к утечкам памяти.

Также может быть полезно знать как C++, так и Python, и использовать их в зависимости от задачи. Если требуется высокая производительность и управление памятью, то C++ может быть более подходящим выбором. Если же нужна быстрая разработка и простой синтаксис, то Python может быть лучшим выбором. Стоит учитывать, что Python является одним из самых популярных языков программирования, что означает, что он имеет большое сообщество разработчиков и множество готовых решений. Это может быть особенно полезно при работе с задачами в области научных вычислений, машинного обучения и веб-разработки.

С другой стороны, C++ имеет более низкий уровень абстракции и ближе к аппаратному уровню, что позволяет программистам более тесно контролировать процессор и память компьютера. Это может быть полезно при работе с высокопроизводительными приложениями и системами реального времени, где каждая микросекунда имеет значение.

Подведём итог анализа двух высокоуровневых языков и оформим полученный вывод на рисунке 2.7:



Рисунок 2.7 - Основные отличительные характеристики компилируемых и интерпретируемых ЯП

# Теоретическое обоснование разработки программного продукта

В настоящее время в мире существует большое количество операционных систем, устройств и приложений, что приводит к необходимости обмена данными между ними. С каждым годом растёт количество людей, вовлечённых в использование мессенджеров, рабочих чатов и программ для обмена сообщениями. Также можно обмениваться не только текстовой информацией, тогда уже из этого вытекает огромный спектр возможностей для организации передачи данных. Это могут быть различные устройства IoT, мобильные приложения, веб-приложения и т.д. Каждое из этих устройств может работать на различной платформе и использовать разные языки программирования, что усложняет задачу обмена сообщениями между ними. Нельзя не отметить, что различные приложения и устройства могут использовать разные протоколы обмена сообщениями, что приводит к необходимости реализации поддержки нескольких протоколов в одной системе. Например, устройства IoT могут использовать протокол MQTT, веб-приложения - HTTP, а мобильные приложения - WebSocket. Безусловно, важным фактором является безопасность обмена сообщениями[5]. Существует ряд уязвимостей, связанных с обменом сообщениями, таких как отказ в обслуживании, подделка сообщений, перехват. Необходимо предусмотреть защиту от таких угроз.

Тогда из вышеперечисленных факторов программный продукт должен обладать одним из протоколов защиты информации при передаче сообщений, так как разработка ведётся под десктопные операционные системы, открывается возможность передачи информации через сокеты, реализованные в каждой из выбранных операционных систем.

Рассмотрим уже готовые протоколы шифрования передачи сообщений.

Симметричное шифрование[6] — это метод шифрования, при котором один и тот же ключ используется для зашифровки и расшифровки сообщения. Это означает, что и отправитель, и получатель должны знать один и тот же ключ.

Алгоритм симметричного шифрования состоит из двух основных этапов[7]: зашифровки и расшифровки. На этапе зашифровки данные преобразуются с помощью ключа таким образом, чтобы стать непонятными и нечитаемыми без знания ключа. На этапе расшифровки зашифрованные данные преобразуются обратно в исходное сообщение с помощью того же ключа, который использовался для зашифровки.

Симметричное шифрование широко используется в различных областях, в том числе в криптографии, защите данных и безопасности сети. Некоторые из самых известных алгоритмов симметричного шифрования — это AES (Advanced Encryption Standard), DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard) и IDEA (International Data Encryption Algorithm).

Преимущества симметричного шифрования включают быстродействие и простоту реализации, а также высокий уровень защиты данных, если ключ сохраняется в секрете. Однако, существует риск компрометации ключа, что может привести к раскрытию зашифрованных данных. Кроме того, если ключ используется для шифрования и расшифровки большого количества сообщений, то утечка ключа может привести к компрометации всех этих сообщений. Графическое представление симметричного шифрования изображено на рисунке 3.1:



Рисунок 3.1 - Графическое представление симметричного шифрования

Асимметричное шифрование[8], также известное как криптография с открытым ключом, — это метод шифрования, в котором используется пара ключей: открытый и закрытый. Каждый участник процесса шифрования может иметь свою пару ключей, состоящую из открытого и закрытого ключа. Открытый ключ используется для шифрования сообщений, а закрытый ключ используется для расшифровки. Открытый ключ может быть распространен любым участником сети, так как он не является секретным. Закрытый ключ остается в тайне у владельца пары ключей.

Когда отправитель хочет отправить сообщение получателю, он использует открытый ключ получателя для шифрования сообщения. Затем получатель использует свой закрытый ключ для расшифровки сообщения. Таким образом, только получатель, у которого есть закрытый ключ, может прочитать сообщение. Асимметричное шифрование является более безопасным, чем симметричное шифрование, так как закрытый ключ никогда не раскрывается и остается только у владельца пары ключей. Это предотвращает возможность расшифровки сообщения третьими лицами.

Однако, асимметричное шифрование требует более высокой вычислительной мощности[9], чем симметричное шифрование, так как процесс шифрования и расшифровки является более сложным. Кроме того, в отличие от симметричного шифрования, асимметричное шифрование не гарантирует аутентификацию, что может быть решено с помощью электронных подписей. Графическое представление асинхронного шифрования сообщения изображено на рисунке 3.2:



Рисунок 3.2 - Графическое представление симметричного шифрования

Касательно цифровой подписи в обеспечении информационной безопасности, определим понятие цифровой подписи.

Цифровая подпись[10] — это математический алгоритм, который позволяет создать уникальный идентификатор, подтверждающий авторство и целостность данных. Она используется для аутентификации отправителя и обеспечения целостности сообщений в мессенджерах и других системах обмена сообщениями.

Для создания цифровой подписи используется ассиметричное шифрование. Отправитель создает уникальный хэш сообщения и шифрует его своим секретным ключом. Полученный результат добавляется к сообщению и отправляется получателю. Получатель расшифровывает хэш с помощью открытого ключа отправителя и сравнивает его с хэшем сообщения. Если хэши совпадают, то сообщение не было изменено в процессе передачи и отправитель подтверждает свое авторство.

Использование цифровой подписи позволяет предотвратить подделку сообщений и убедиться в том, что сообщение было отправлено именно от того, кто указан в качестве отправителя. Это особенно важно для систем, где отправитель и получатель могут находиться в разных местах и не имеют возможности лично проверить подлинность сообщения. Графическое представление цифровой подписи сообщения изображено на рисунке 3.3:

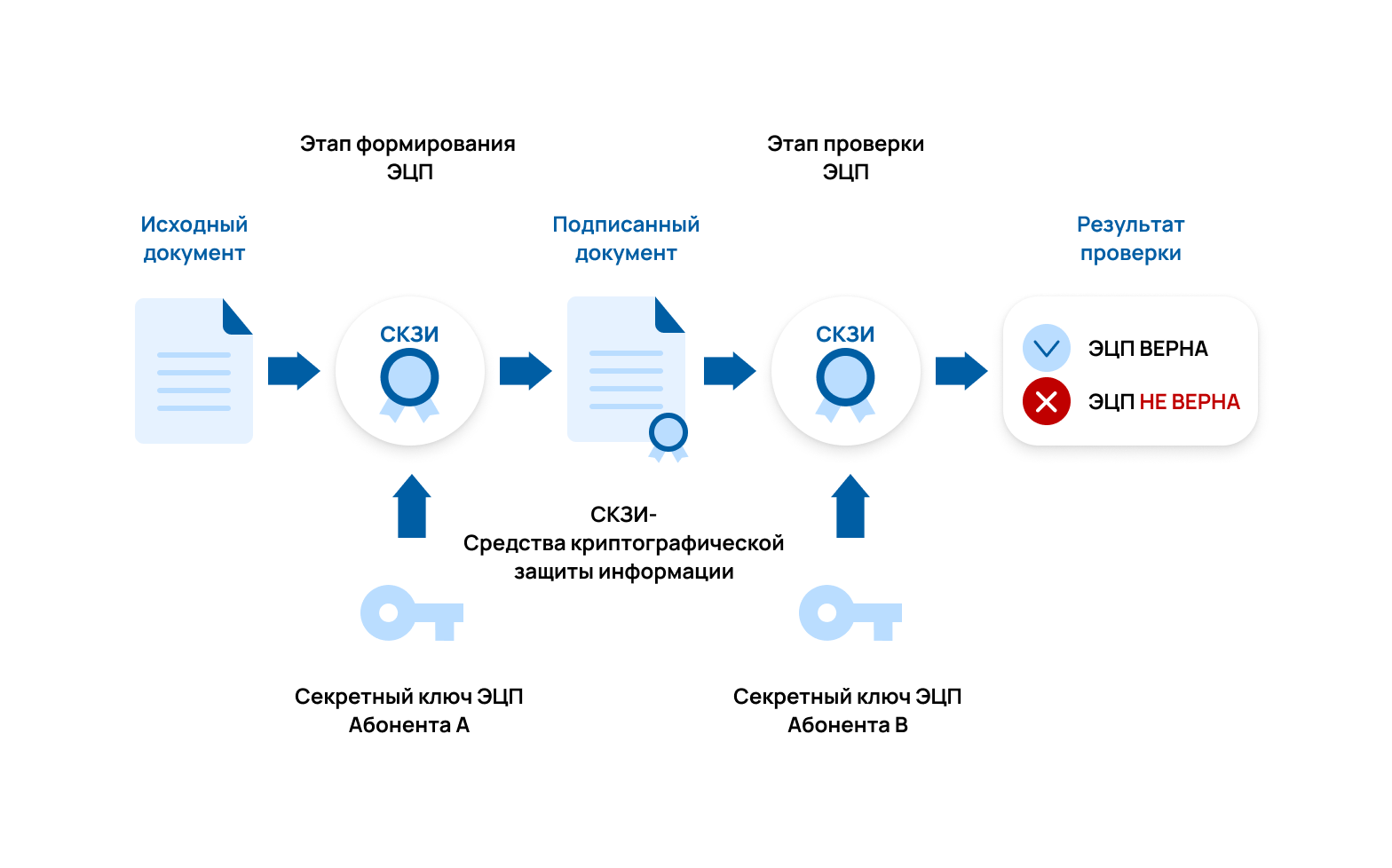


Рисунок 3.3 - Графическое представление цифровой подписи

Что касается механизмов передачи данных между компьютерами, как было упомянуто ранее, в разрабатываемом программном продукте для передачи данных будет использоваться интерфейс сокетов. Определим понятие сокетов.

Сокеты[11] — это программный интерфейс, который обеспечивает взаимодействие между приложениями через сетевые протоколы. Они используются для передачи данных между процессами или узлами в сети. Сокеты взаимодействуют с сетевыми протоколами на уровне транспортного и прикладного уровней модели OSI.

Модель OSI[12] (Open Systems Interconnection) — это стандарт, который определяет семь уровней для коммуникации между устройствами в компьютерных сетях. Каждый уровень выполняет определенную функцию и используется для передачи данных между устройствами. Основные используемые уровни модели OSI представлены на рисунке 3.4:

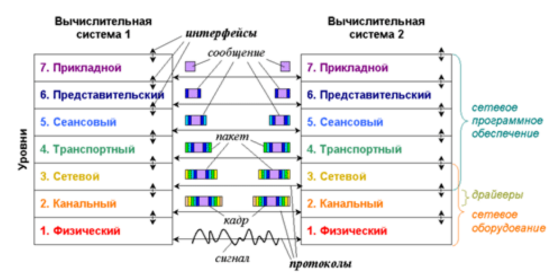


Рисунок 3.4 - Уровни модели OSI

1 Физический уровень (Physical layer) — отвечает за передачу битов данных через физическую среду передачи, такую как провода, оптоволокно, радиоволны. Он определяет характеристики электрических, механических и функциональных интерфейсов, которые позволяют физически связывать устройства.

2 Канальный уровень (Data Link layer) — обеспечивает надежную передачу данных между соседними устройствами, связанными вместе в локальной сети. Он управляет физической адресацией, контролирует поток данных, обнаруживает и исправляет ошибки передачи данных.

3 Сетевой уровень (Network layer) — отвечает за маршрутизацию и доставку данных от отправителя к получателю, включая промежуточные узлы. Он определяет, какие маршруты должны быть использованы для доставки данных, а также обеспечивает переход от логических адресов (например, IP-адресов) к физическим адресам (например, MAC-адресам).

4 Транспортный уровень (Transport layer) — обеспечивает надежную передачу данных между конечными устройствами и контролирует поток данных, управляет соединениями и обнаруживает, исправляет ошибки. Он также может обеспечивать механизмы управления качеством обслуживания (QoS) и контролировать нагрузку на сеть.

5 Сеансовый уровень (Session layer) — устанавливает, управляет и завершает соединения между устройствами в сети. Он обеспечивает механизмы управления сеансами, управления протоколами и управления исключительными ситуациями.

6 Уровень представления (Presentation layer) — обеспечивает преобразование данных в удобный для передачи формат. Он может выполнять сжатие, шифрование и декодирование данных.

7 Прикладной уровень (Application layer) — это уровень, который обрабатывает данные, создаваемые приложениями.

На уровне транспортного уровня OSI-модели сокеты используют протоколы TCP[13] (Transmission Control Protocol) и UDP (User Datagram Protocol) для обмена данными между узлами в сети. Протокол TCP обеспечивает надежную и упорядоченную передачу данных между узлами в сети. Сокеты, работающие с TCP, обычно используются для передачи крупных объемов данных, таких как файлы, мультимедийные данные и т.д. При использовании TCP, сокеты устанавливают соединение между двумя узлами в сети, после чего данные передаются в виде потока, и получатель может быть уверен, что данные будут доставлены в правильном порядке и без потерь. Протокол UDP, в свою очередь, обеспечивает не надежную, но более быструю передачу данных между узлами в сети. Сокеты, работающие с UDP, обычно используются для передачи маленьких объемов данных, таких как команды или небольшие сообщения. При использовании UDP, сокеты просто отправляют данные без установления соединения, и получатель может получить данные в любом порядке и с потерями. UDP и TCP протоколы графически изображены на рисунке 3.5:

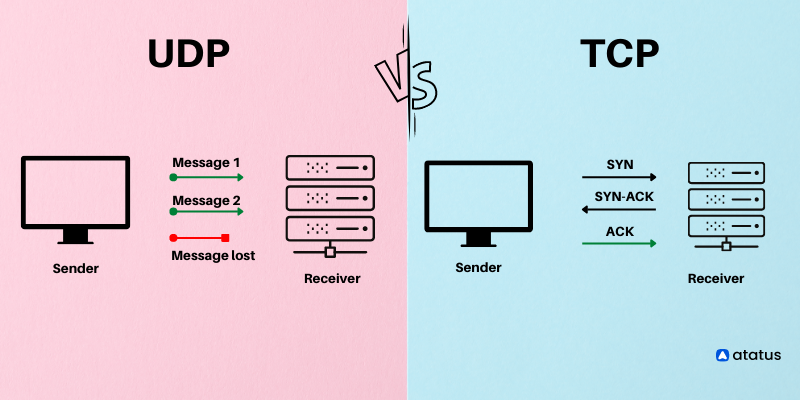


Рисунок 3.5 - Графическое представление UDP

# Проектирование функциональных возможностей программы

Для корректной и удобной работы программы для ПК по передаче сообщений ей необходимо обладать следующими функциями:

1 Установление соединения между сервером и клиентом.

2 Установление соединения с базой данных.

3 Асинхронное шифрование передаваемых сообщений от клиента к серверу.

4 Корректный вывод графического интерфейса.

Первая функция является неотъемлемой частью, так как без нее в принципе программа не сможет выполнить свою задачу. Данная функция также крайне важна по причине того, что она дает следующие возможности:

1 Установка соединения: программа позволяет клиенту и серверу установить соединение между собой. Для этого обе стороны должны иметь доступ к сети и знать адреса друг друга.

2 Аутентификация: программа может проверить подлинность клиента или сервера, используя различные методы аутентификации, такие как логин и пароль, сертификаты или цифровые подписи.

3 Обмен данными: после установки соединения клиент и сервер могут обмениваться данными, используя различные протоколы, такие как HTTP, FTP, SMTP и другие. Программа может обеспечить надежность и целостность передачи данных, используя различные механизмы, такие как проверка контрольной суммы, шифрование и подпись.

4 Управление соединением[14]: программа может управлять соединением между клиентом и сервером, включая открытие и закрытие соединения, управление таймаутами, переподключение при обрыве связи и т.д.

5 Обработка ошибок: программа должна уметь обрабатывать ошибки, которые могут возникнуть в процессе установления соединения или обмена данными, и сообщать об этих ошибках клиенту или серверу.

Графически это может выглядеть как рисунок 4.1:

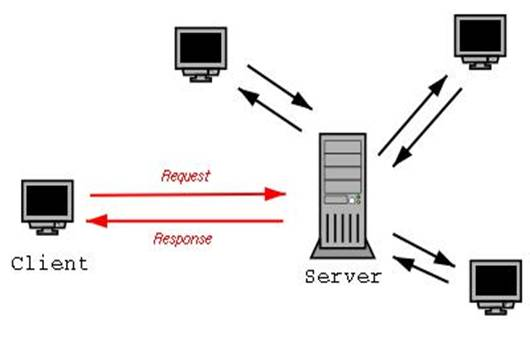


Рисунок 4.1 - Установление соединения сервера с клиентами

Вторая функция установления соединения с базой данных может обеспечить следующие возможности:

1 Выполнение запросов к базе данных: после установления соединения с базой данных, программа может отправлять запросы для извлечения, добавления, изменения или удаления данных. Для этого может использоваться язык SQL (Structured Query Language), который позволяет работать с данными в базе.

2 Обработка ошибок[15]: при установлении соединения с базой данных могут возникнуть ошибки, такие как неверные учетные данные, проблемы с доступностью базы данных и другие. Программа должна иметь механизмы обработки таких ошибок и уведомления пользователя о проблемах.

3 Управление транзакциями: транзакции используются для гарантирования целостности данных в базе данных. При установлении соединения с базой данных программа может начать транзакцию, выполнить несколько запросов и завершить транзакцию. Если при выполнении запросов произойдет ошибка, программа может отменить выполняемую транзакцию и привести базу данных в первоначальное состояние.

4 Управление соединениями: при работе с базой данных может возникнуть необходимость в управлении соединениями. Например, программа может создавать пул соединений для повышения производительности и снижения нагрузки на базу данных. Также программа может устанавливать тайм-ауты для соединений, чтобы избежать блокировки при невозможности установления соединения в течение определенного времени.

5 Шифрование и защита данных: при установлении соединения с базой данных, программа может использовать шифрование данных для защиты конфиденциальной информации, такой как учетные данные и другие данные, передаваемые между клиентом и сервером. Также может использоваться аутентификация и авторизация для обеспечения безопасности доступа к базе данных.

Третья функция асинхронного шифрования передаваемых сообщений от клиента к серверу обеспечивает:

1 Конфиденциальность: сообщения между клиентом и сервером шифруются с использованием симметричного или асимметричного алгоритма шифрования, что позволяет скрыть содержание сообщений от посторонних лиц.

2 Целостность: используется цифровая подпись для обеспечения целостности передаваемых сообщений. При получении сообщения сервер проверяет его цифровую подпись, чтобы убедиться, что сообщение не было изменено в процессе передачи.

3 Аутентификация: при использовании асимметричного шифрования клиент может отправить серверу свой публичный ключ, чтобы сервер мог убедиться в том, что сообщения действительно отправляет именно клиент, а не кто-то другой, пытающийся выдать себя за клиента.

4 Защита от повторной передачи: при использовании асимметричного шифрования можно использовать метки времени для обеспечения защиты от повторной передачи сообщений. Каждое сообщение получает уникальную метку времени, которая позволяет серверу отвергнуть сообщение, если оно уже было обработано ранее.

5 Уменьшение нагрузки на сеть[16]: при использовании асинхронного шифрования передаваемых сообщений можно уменьшить объем передаваемых данных. Это связано с тем, что при асимметричном шифровании не нужно передавать ключ шифрования, так как он уже известен обеим сторонам.

Четвёртая функция корректного вывода графического интерфейса предполагает следующее:

1 Использование библиотек графического интерфейса: существует множество библиотек, которые позволяют разработчикам создавать графический интерфейс. Они предоставляют готовые элементы интерфейса, такие как кнопки, поля ввода, списки и т.д., которые можно использовать для создания пользовательского интерфейса. Некоторые из наиболее популярных библиотек включают Tkinter[17], PyQt и wxWidgets.

2 Правильное использование шрифтов: шрифты являются важной частью графического интерфейса. Они могут быть использованы для обозначения заголовков, кнопок и других элементов. При разработке программы необходимо убедиться, что выбранные шрифты являются читаемыми и хорошо видны на экране.

3 Размещение элементов интерфейса: размещение элементов интерфейса важно для обеспечения понятности и удобства использования программы. Элементы должны быть размещены таким образом, чтобы пользователи могли легко находить нужную информацию и выполнять необходимые действия.

4 Использование цветовой схемы: выбор правильной цветовой схемы помогает сделать интерфейс более привлекательным и приветливым. Некоторые цвета могут быть использованы для привлечения внимания к ключевым элементам интерфейса, тогда как другие могут использоваться для задания фона и общей темы.

5 Работа с событиями: графический интерфейс обычно включает в себя множество событий, которые могут быть обработаны программой. Это может включать клики мыши, нажатия клавиш, перемещения курсора и т.д. Правильная обработка событий может помочь убедиться, что программа работает правильно и быстро реагирует на пользовательские действия.

# Архитектура разрабатываемой программы

В основе программного продукта лежат модули графического интерфейса, клиентской и серверной части. В самом начале нужно подключить компьютер к любой проводной или беспроводной сети, чтобы программный продукт мог корректно определить IP-адрес этой сети и использовать его как основной адрес сети.

def get\_ip\_address(ifname):

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

return socket.inet\_ntoa(fcntl.ioctl(

s.fileno(),

0x8915, # SIOCGIFADDR

struct.pack('256s', ifname[:15].encode('utf-8'))

)[20:24])

После определения IP-адреса и порта для создания соединения, запускается основной класс Main с методом run(), который по заданному буферу генерирует приватный и публичный ключ[18], которые нам необходимы для дальнейшего создания механизма приёма/передачи сообщений. Далее принимается сокет подключенного клиента и для него создаётся объект Chat куда передаются созданные ранее ключи шифрования и запускается отдельный поток класса Main[19].

class Main:

"""

def run:

It will generate keys and wait for connections

"""

def run():

username\_exist = False

print(f"[[magenta]\*[/magenta]] Buffer: {buffer}")

print("[[cyan]+[/cyan]] RSA key generation...")

public\_key, private\_key = API.create\_keys(buffer)

print("[[cyan]+[/cyan]] RSA key generated")

while True:

client, addr = server.accept()

chat = Chat(client, private\_key, public\_key)

multi\_conn = Thread(target=chat.run)

multi\_conn.start()

Сервер отправляет закодированное сообщение о защищённом соединении и если всё в программном продукте отрабатывает как нужно, то от клиента приходит подтверждение и далее идёт процесс аутентификации, успешный результат которого отправляет клиенту ключи, по которым он будет шифровать и расшифровывать данные[20]. После этого данный поток приходит в режим ожидания и при получении сообщения расшифровывает его, обрабатывает и отправляет уже по месту назначения. Это реализовано в функциях send\_friend и send\_group.

def run(self):

try:

username\_exist = False

# password\_md5 = hashlib.md5(password\_md5.encode()).hexdigest()

self.client.send("protected".encode())

print(f'Protected sended')

nickname = self.client.recv(buffer).decode()

id = self.client.recv(buffer).decode()

for cl in clients:

if cl[1] == id or cl[2] == nickname:

username\_exist = True

if not username\_exist:

self.client.send("/id\_accepted".encode())

print("[[yellow]?[/yellow]] Client connected")

clients.append((self.client, id, nickname))

else:

# Send message: "exit" to client

self.client.send("/exit".encode())

self.client.close()

except:

pass

try:

API.send\_buffer(self.client, buffer)

except:

pass

time.sleep(0.5)

send\_keys = API.Send\_keys(

self.public\_key,

self.private\_key,

self.client)

self.rsa\_api = API.RSA(self.public\_key, self.private\_key)

self.chat\_api = API.Chat(self.private\_key, self.public\_key)

send\_keys.public()

time.sleep(0.5)

send\_keys.private()

time.sleep(0.5)

api\_chat = API.Chat(self.private\_key, self.public\_key)

while True:

self.parse\_msg(

api\_chat.recv(self.client, buffer)

)

def send\_friend(self, id: int, msg: str):

for cln in clients:

if int(cln[1]) == id:

self.chat\_api.send(cln[0], msg)

db.send\_message(

int(cln[0]),

int(cln[3]),

msg,

False

)

def send\_group(self, snd\_id: int, gr\_id: int, msg: str):

groups\_filter = list(filter(

lambda x: self.compare\_group(

db.get\_groups(x[1]), gr\_id

), clients

))

for cl in groups\_filter:

self.chat\_api.send(

cl[0], msg

)

db.send\_message(

snd\_id,

gr\_id,

msg,

True

)

def compare\_group(self, group\_list: list, id: int):

for gr in group\_list:

if gr[0] == id:

return True

return False

def parse\_msg(self, msg):

msg = msg.decode('utf-8')

msg = msg.split('^\*$$')

if msg[2] == 'fr':

self.send\_friend(int(msg[3]), msg[4])

elif msg[2] == 'gr':

self.send\_group(int(msg[0]), int(msg[3]), msg[4])

Со стороны клиента в этот момент происходило следующее: устанавливалось соединение с сервером, если оно успешно, то генерировался графический интерфейс, заполнялись соответствующие поля, чтобы провести регистрацию нового пользователя или авторизацию и аутентификацию уже существующего[21]. В коде это реализовано следующим образом:

def login(self):

id = db.login(

self.username\_entry.get(),

self.password\_entry.get()

)

if id != -1:

self.id = id

self.msg\_queue = client.launch\_client(self.username\_entry.get(), self.id)

self.show\_messaging()

else:

self.id = -1

self.invalid\_login\_label.config(text="Invalid login/password")

def signup(self):

if db.register(

self.username\_entry.get(),

self.password\_entry.get(),

self.number\_entry.get()

):

self.show\_login()

def launch\_client(username, id):

main = Main()

queue = main.connect(username, id)

return queue

def connect(self, username: str, id: str):

conn = Connection()

self.ip, self.port = conn.get\_server()

while True:

try:

s.connect((self.ip, self.port))

except ConnectionRefusedError:

cls()

print("[red]ERROR[/red]: Connection refused")

self.ip, self.port = conn.get\_server()

self.username, self.username\_styled, = conn.get\_username()

is\_protected = s.recv(1024).decode()

if is\_protected == "protected":

self.password = conn.get\_password()

self.send\_password(self.password)

else:

break

is\_protected = s.recv(1024).decode()

print(f'is\_protected decoded!')

# Send username to server and wait 0.5s

self.send\_username(username)

time.sleep(0.5)

self.send\_id(id)

# Get buffer from server

buffer = self.get\_buffer()

# Decompress keys with zlib

public\_key = zlib.decompress(s.recv(buffer))

private\_key = zlib.decompress(s.recv(buffer))

# Key load

client\_key = API.Load\_keys(public\_key, private\_key)

self.private\_key, self.public\_key = client\_key.load\_all()

# API

self.chat\_api = API.Chat(self.private\_key, self.public\_key)

result\_queue = queue.Queue()

chat = Chat(self.chat\_api, username, id)

chat\_thread = threading.Thread(target=chat.run, args=(result\_queue, ))

chat\_thread.start()

return result\_queue.get()

Далее если всё прошло успешно, то пользователь имеет возможность выбрать из списка своих друзей пользователя или отправить текстовое сообщение в группу. Для этого выбирается в выпадающем списке соответствующая группа или пользователь, вводится сообщение, нажимается кнопка Send, сообщение шифруется закрытым ключом пользователя и передаётся на сервер, где вновь обрабатывается[22], отправляется по месту назначения и записывается в базу данных.

def send\_message(self):

option = self.selected\_option.get()

dst, index = self.get\_selected\_item()

index = self.fr\_list[index][0]

if option == 'friends':

msg = f'fr'

elif option == 'groups':

msg = f'gr'

msg = f'{msg}^\*$${index}^\*$${self.message\_entry.get()}'

print(f'Send message: {msg}')

self.msg\_queue.put(msg)

def write(self, q):

try:

while True:

msg = q.get()

print(f'Send message: {msg}')

sys.stdout.write("\033[F")

msg = f'{self.username}^\*$${self.id}^\*$${msg}'

self.chat\_api.send(msg)

q.task\_done()

except KeyboardInterrupt:

s.send("/exit".encode())

sys.exit(1)

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была рассмотрена тема "Кроссплатформенная система обмена сообщениями". Были рассмотрены основные принципы и подходы к разработке такой системы, а также технологии, используемые для ее создания.

В ходе работы были изучены различные аспекты разработки мессенджеров, включая безопасность, шифрование, установление соединения с базой данных, работу с сокетами, а также графический интерфейс пользователя. В процессе работы была разработана кроссплатформенная система обмена сообщениями, обеспечивающая безопасность и защиту данных пользователей. Все возможности и функционал системы были протестированы и успешно проверены. В целом, разработка кроссплатформенной системы обмена сообщениями является актуальной темой, так как мессенджеры становятся все более популярными среди пользователей во всем мире. Разработка такой системы обмена сообщениями, обеспечивающей безопасность и защиту данных пользователей, является важным шагом в обеспечении безопасности в сфере коммуникаций.

Итогом работы можно считать выполненными поставленные ранее цели и задачи:

Цели курсового проекта:

1 Приобретение теоретических и практических навыков организации обмена текстовыми сообщениями с использованием протокола шифрования по открытому и закрытому ключу.

2 Реализация программного продукта по обмену сообщениями между пользователями.

Задачи курсового проекта:

1 Изучение существующих технологий и протоколов обмена сообщениями, их достоинств и недостатков.

2 Анализ требований к системе, установка основных критериев ее функционирования и надежности.

3 Разработка архитектуры системы, определение ее основных компонентов и интерфейсов.

4 Разработка и реализация кроссплатформенного приложения, осуществляющего обмен сообщениями.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Самые важные аппаратные характеристики компьютера [Электронный ресурс]:Unetway. Режим доступа: <https://studfile.net/preview/7209950/page:13/>. Дата доступа: 01.12.2022.

[2] Материнская плата. Схема и характеристики. [Электронный ресурс]: 2hpc. – Режим доступа: <https://2hpc.ru/>. Дата доступа: 01.12.2022.

[3] Микропроцессоры. Основные характеристики. [Электронный ресурс]: bstudy. – Режим доступа: <https://bstudy.net/814892/informatika/osnovnye_harakteristiki_mikroprotsessorov>. Дата доступа: 01.12.2022.

[4] Оперативная память. Установка. [Электронный ресурс]: okeygeek. –Режим доступа: <https://okeygeek.ru/kak-pravilno-ustanovit-operativnuyu-pamyat-v-kompyuter/>. Дата доступа: 01.12.2022

[5] Alan Moore Python GUI Programming with Tkinter. / Alan Moore. – Packt Publishing, 2018 г. – 425 c.

[6] Top Programming Languages 2022 [Электронный ресурс]. – Режим доступа: https://spectrum.ieee.org/top-programming-languages-2022. Дата доступа: 01.05.2023

[7] Niels Ferguson, Bruce Schneier, Tadayoshi Kohno Cryptography Engineering: Design Principles and Practical Applications. / Niels Ferguson и другие. – Wiley, 2010. – 384 с.

[8] Bruce Schneier Applied Cryptography: Protocols, Algorithms and Source Code in C. / Bruce Schneier – Wiley, 2015. – 784 с.

[9] Top Programming Languages 2022 [Электронный ресурс]. – Режим доступа: <https://spectrum.ieee.org/top-programming-languages-2022>.

[10] Жан-Филипп Омассон Serious Cryptography: A Practical Introduction to Modern Encryption. / Жан-Филипп Омассон. – Manning, 2017. – 312 с.

[11] Brandon Rhodes Foundations of Python Network Programming. / Brandon Rhoades – Apress, 2014 г. – 824 c.

[128Fred Piper, Sean Murphy Cryptography: A Very Short Introduction. / Fred Piper, Sean Murphy – Oxford University Press, 2002. – 160 с.

[13] Justin Seitz Black Hat Python: Python Programming for Hackers and Pentesters. / Justin Seitz – No Starch Press, 2014 г. – 352 с.

[14] Christof Paar , Jan Pelzl Understanding Cryptography: A Textbook for Students and Practitioners. / Christof Paar , Jan Pelzl – Springer, 2009. – 290 с.

[15] Marius Iulian Mihailescu Pro Cryptography and Cryptanalysis: Creating Advanced Algorithms with C# and .NET. / Marius Iulian Mihailescu – Wiley, 2020. – 588 с.

[16] Mihai Catalin Teodosiu Python 3 Network Programming - Build 5 Network Applications. / Mihai Catalin Teodosiu – Packt Publishing, 2018 г. – 125 c.

[17] Handbook of Applied Cryptography [Электронный ресурс]. – Режим доступа: <https://doc.lagout.org/network/3_Cryptography/CRC%20Press%20-%20Handbook%20of%20applied%20Cryptography.pdf>

[18] А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин, Александр Алферов Основы криптографии. / А. Ю. Зубов, А. С. Кузьмин и другие– Гелиос АРВ, 2005. – 457 с.

[19] А. А. Молдовян, Н. А. Молдовян, Н. Д. Гуц, Б. В. Изотов Криптография. Скоростные шифры. / А. А. Молдовян, Н. А. Молдовян и другие – БВХ-Петербург, 2002. – 1256 с.

[20] Структурное проектирование [Электронный ресурс]: Studme. Режим доступа: <https://studme.org/192620/informatika/strukturnoe_proektirovanie>. Дата доступа: 01.05.2023.

[21] Структурное проектирование [Электронный ресурс]: Studfile. Режим доступа: <https://studfile.net/preview/7209950/page:13/>. Дата доступа: 01.05.2023.

[22] Методология проектирования программных продуктов [Электронный ресурс]: StudIzba. – Режим доступа: <https://studizba.com/lectures/informatika-i-programmirovanie/lekcii-po-informatike-i-programmirovaniyu/4303-metodologiya-proektirovaniya-programmnyh-produktov.html>. Дата доступа: 01.05.2023.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

server.py

import socket

import rsa

import zlib

import time

import json

import hashlib

import sys

from threading import Thread

from rich import print

sys.path.append('client')

import ips

sys.path.append('db-utils/ORM-for-DB')

import db

# Vars

clients = []

nicknames = []

ip = ips.get\_ip\_address('wlp0s20f3')

port: int = 8869

buffer: int = 1024

# welcome\_message: str = config\_json["welcome\_message"]

# protected\_by\_password: bool = config\_json["protected\_by\_password"]

# password: str = config\_json["password"]

# Create socket

server = socket.socket()

# Start listing

server.bind((ip, port))

print(f"[[green]![/green]] Listing: {ip}:{port}")

server.listen(32)

class API:

def create\_keys(buffer: int):

public\_key, private\_key = rsa.newkeys(buffer)

return public\_key, private\_key

def send\_buffer(s, buffer: int):

s.send(str(buffer).encode())

class Chat:

def \_\_init\_\_(self, priv\_key, pub\_key) -> None:

self.priv\_key = priv\_key

self.pub\_key = pub\_key

def send(self, s, msg: str):

s.send(rsa.encrypt(msg.encode(), self.pub\_key))

def recv(self, s, buffer: int):

msg = s.recv(buffer)

msg = rsa.decrypt(msg, self.priv\_key)

print(f'Recieved decrypted message: {msg}')

return msg

class Send\_keys:

def \_\_init\_\_(self, pub\_key, priv\_key, client) -> None:

self.client = client

self.pub\_key = pub\_key

self.priv\_key = priv\_key

def private(self):

private\_key\_exported = rsa.PrivateKey.save\_pkcs1(self.priv\_key)

# compressing

private\_key\_exported = zlib.compress(private\_key\_exported, 4)

self.client.send(private\_key\_exported)

def public(self):

public\_key\_exported = rsa.PublicKey.save\_pkcs1(self.pub\_key)

# compressing

public\_key\_exported = zlib.compress(public\_key\_exported, 4)

self.client.send(public\_key\_exported)

class RSA:

def \_\_init\_\_(self, pub\_key, priv\_key) -> None:

self.pub\_key = pub\_key

self.priv\_key = priv\_key

def encrypt(self, msg: str):

return rsa.encrypt(msg.encode(), self.pub\_key)

def decrypt(self, msg: bytes):

return rsa.decrypt(msg, self.priv\_key)

class Chat:

Args: client (socket), private\_key, public\_key

def joined (nickname: str):

It will send a message when a client disconnect

def welcome\_message (bytes: bytes):

It will send the clients the encrypted welcome message

def send\_to\_clients (message: bytes):

It sends clients a message, but it won't be able to send it to itself

def remove\_client (client):

removes clients from the client list

def middle:

When a customer enters the chat, perform this function.

Send clients a message announcing that a client has logged in,

then wait for a message from the client and then send it to all

def run:

It's where this class will launch

def \_\_init\_\_(self, client, private\_key, public\_key) -> None:

self.client = client

self.private\_key = private\_key

self.public\_key = public\_key

def joined(self, nickname: str):

self.send\_to\_clients(self.rsa\_api.encrypt(

f"[green]{nickname}[/green] has joined."))

def send\_to\_clients(self, msg: bytes):

for client in clients:

if client != self.client:

try:

client.send(msg)

except BaseException:

self.remove\_client(client)

def remove\_client(self, client):

print("[[yellow]?[/yellow]] Client disconnected")

index\_list = list(map(lambda x: x[0], clients))

index = index\_list.index(client)

# Remove from list

clients.remove(client)

# Get username from socket

nickname = nicknames[index]

self.send\_to\_clients(self.rsa\_api.encrypt(

f"[green]{nickname}[/green] has left."))

# Remove nickname

nicknames.remove(nickname)

while True:

try:

msg = self.client.recv(buffer).decode()

except BaseException as e:

print(f'Exception: {e}')

self.remove\_client(self.client)

break

def run(self):

try:

username\_exist = False

# password\_md5 = hashlib.md5(password\_md5.encode()).hexdigest()

self.client.send("protected".encode())

print(f'Protected sended')

nickname = self.client.recv(buffer).decode()

id = self.client.recv(buffer).decode()

for cl in clients:

if cl[1] == id or cl[2] == nickname:

username\_exist = True

if not username\_exist:

self.client.send("/id\_accepted".encode())

print("[[yellow]?[/yellow]] Client connected")

clients.append((self.client, id, nickname))

else:

# Send message: "exit" to client

self.client.send("/exit".encode())

self.client.close()

except:

pass

try:

API.send\_buffer(self.client, buffer)

except:

pass

time.sleep(0.5)

send\_keys = API.Send\_keys(

self.public\_key,

self.private\_key,

self.client)

self.rsa\_api = API.RSA(self.public\_key, self.private\_key)

self.chat\_api = API.Chat(self.private\_key, self.public\_key)

send\_keys.public()

time.sleep(0.5)

send\_keys.private()

time.sleep(0.5)

api\_chat = API.Chat(self.private\_key, self.public\_key)

while True:

self.parse\_msg(

api\_chat.recv(self.client,buffer))

def send\_friend(self, id: int, msg: str):

for cln in clients:

if int(cln[1]) == id:

self.chat\_api.send(cln[0], msg)

db.send\_message(

int(cln[0]),

int(cln[3]),

msg,

False

)

def send\_group(self, snd\_id: int, gr\_id: int, msg: str):

groups\_filter = list(filter(

lambda x: self.compare\_group(

db.get\_groups(x[1]), gr\_id

), clients

))

for cl in groups\_filter:

self.chat\_api.send(

cl[0], msg

)

db.send\_message(

snd\_id,

gr\_id,

msg,

True

)

def compare\_group(self, group\_list: list, id: int):

for gr in group\_list:

if gr[0] == id:

return True

return False

def parse\_msg(self, msg):

msg = msg.decode('utf-8')

msg = msg.split('^\*$$')

if msg[2] == 'fr':

self.send\_friend(int(msg[3]), msg[4])

elif msg[2] == 'gr':

self.send\_group(int(msg[0]), int(msg[3]), msg[4])

class Main:

"""

def run:

It will generate keys and wait for connections

"""

def run():

username\_exist = False

print(f"[[magenta]\*[/magenta]] Buffer: {buffer}")

print("[[cyan]+[/cyan]] RSA key generation...")

public\_key, private\_key = API.create\_keys(buffer)

print("[[cyan]+[/cyan]] RSA key generated")

while True:

client, addr = server.accept()

chat = Chat(client, private\_key, public\_key)

multi\_conn = Thread(target=chat.run)

multi\_conn.start()

if \_\_name\_\_ == "\_\_main\_\_":

Main.run()

client.py

import sys

import os

import socket

import rsa

import time

import zlib

import random

import threading

import hashlib

import queue

from ips import get\_ip\_address

from rich import print

from rich.console import Console

from rich.tree import Tree

from rich.align import Align

from anonfile import AnonFile

global buffer

# Vars

console = Console()

anon = AnonFile()

s = socket.socket()

buffer = 1024

class API:

class Chat:

def \_\_init\_\_(self, priv\_key, pub\_key) -> None:

self.priv\_key = priv\_key

self.pub\_key = pub\_key

def send(self, msg: str):

res = rsa.encrypt(msg.encode(), self.pub\_key)

print(f'RSA encrypted message: {res}')

s.send(res)

def recv(self, buffer: int):

msg = s.recv(buffer)

return rsa.decrypt(msg, self.priv\_key).decode()

class Load\_keys:

def \_\_init\_\_(self, pub\_key, priv\_key) -> None:

self.pub\_key = pub\_key

self.priv\_key = priv\_key

def private(self):

return rsa.PrivateKey.load\_pkcs1(self.priv\_key)

def public(self):

return rsa.PublicKey.load\_pkcs1(self.pub\_key)

def load\_all(self):

return rsa.PrivateKey.load\_pkcs1(

self.priv\_key), rsa.PublicKey.load\_pkcs1(self.pub\_key)

class Connection:  
 def \_\_init\_\_(self) -> None:

pass

def get\_server(self):

server\_ip = get\_ip\_address('wlp0s20f3')

server\_port = 8869

ips = ['local', 'localhost', '::1']

if server\_ip in ips:

server\_ip = "127.0.0.1"

return server\_ip, server\_port

def get\_password(self):

password = console.input("[b]Insert password[/b] :locked\_with\_key: : ")

return hashlib.md5(password.encode()).hexdigest()

def start(self):

ip, port = self.get\_server()

return ip, port

class Chat:

class chat\_api: RSA custom socket

var (str) username\_syled: (<Username>) {

It is the username but compliant for sending via socket

making all the username available

}

var (str) username: Current username

def \_\_init\_\_(self, chat\_api, id, username) -> None:

self.chat\_api = chat\_api

self.id = id

self.username = username

def receive(self):

while True:

try:

print(self.chat\_api.recv(buffer))

except rsa.pkcs1.DecryptionError:

pass

def write(self, q):

try:

while True:

msg = q.get()

print(f'Send message: {msg}')

sys.stdout.write("\033[F")

msg = f'{self.username}^\*$${self.id}^\*$${msg}'

self.chat\_api.send(msg)

q.task\_done()

except KeyboardInterrupt:

s.send("/exit".encode())

sys.exit(1)

def run(self, res\_queue):

q = queue.Queue()

receive\_process = threading.Thread(target=self.receive)

receive\_process.start()

write\_process = threading.Thread(target=self.write, args=(q, ))

write\_process.start()

res\_queue.put(q)

class Main:

def \_\_init\_\_(self) -> None:

pass

def connect(self, username: str, id: str):

conn = Connection()

self.ip, self.port = conn.get\_server()

while True:

try:

s.connect((self.ip, self.port))

except ConnectionRefusedError:

cls()

print("[red]ERROR[/red]: Connection refused")

self.ip, self.port = conn.get\_server()

self.username, self.username\_styled, = conn.get\_username()

is\_protected = s.recv(1024).decode()

if is\_protected == "protected":

self.password = conn.get\_password()

self.send\_password(self.password)

else:

break

is\_protected = s.recv(1024).decode()

print(f'is\_protected decoded!')

# Send username to server and wait 0.5s

self.send\_username(username)

time.sleep(0.5)

self.send\_id(id)

# Get buffer from server

buffer = self.get\_buffer()

# Decompress keys with zlib

public\_key = zlib.decompress(s.recv(buffer))

private\_key = zlib.decompress(s.recv(buffer))

# Key load

client\_key = API.Load\_keys(public\_key, private\_key)

self.private\_key, self.public\_key = client\_key.load\_all()

# API

self.chat\_api = API.Chat(self.private\_key, self.public\_key)

result\_queue = queue.Queue()

chat = Chat(self.chat\_api, username, id)

chat\_thread = threading.Thread(target=chat.run, args=(result\_queue, ))

chat\_thread.start()

return result\_queue.get()

def send\_username(self, username: str):

s.send(username.encode())

def send\_id(self, id: int):

s.send(str(id).encode())

confirm = s.recv(1024).decode()

if confirm != "/id\_accepted":

print("[red]ERROR[/red]: User exists")

s.close()

quit()

def send\_password(self, password: str):

s.send(password.encode())

confirm = s.recv(1024).decode()

# Confirm: /exit or /accepted

if confirm != "/accepted":

print("[red]ERROR[/red]: Incorrect password")

s.close()

quit()

def get\_buffer(self):

return int(s.recv(buffer).decode())

def get\_welcome\_message(self):

print(Align(self.chat\_api.recv(buffer), "center"))

def launch\_client(username, id):

main = Main()

queue = main.connect(username, id)

return queue

ПРИЛОЖЕНИЕ Б  
(обязательное)  
Функциональная схема алгоритма

ПРИЛОЖЕНИЕ В  
(обязательное)  
Блок схема программы

ПРИЛОЖЕНИЕ Г  
(обязательное)  
Графический интерфейс

ПРИЛОЖЕНИЕ Д  
(обязательное)  
Ведомость