Server

Создано системой Doxygen 1.9.4

Глава 1

Документация сервера обработки векторов

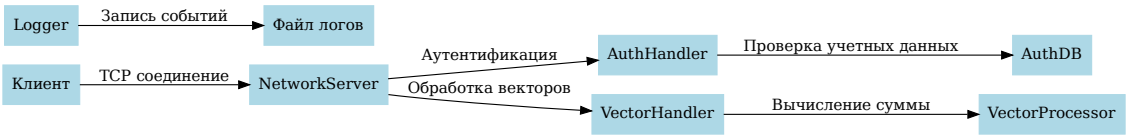
1.1 Введение

Сервер обработки векторов — это многокомпонентное приложение для аутентификации клиентов и обработки числовых векторных данных. Сервер поддерживает протокол аутентификации на основе SHA224 и предоставляет функционал для суммирования векторов с контролем переполнения.

1.2 Основные возможности

- Аутентификация клиентов с использованием хэширования SHA224 с солью
- Обработка векторных данных с контролем переполнения
- Многопоточное логирование в файл
- Конфигурация через командную строку
- Загрузка базы данных клиентов из файла

1.3 Архитектура системы



1.3.1 Основные компоненты

1.3.1.1 NetworkServer

Основной класс сервера, управляющий сетевыми соединениями и координирующий работу других компонентов.

1.3.1.2 AuthHandler

Обработчик аутентификации, реализующий проверку учетных данных клиентов с использованием SHA224.

1.3.1.3 VectorHandler

Обработчик векторных данных, читающий векторы из сети и вычисляющий их суммы с контролем переполнения.

1.3.1.4 NetworkUtils

Набор утилит для работы с сетью, преобразования данных и работы с сетевым порядком байт.

1.3.1.5 Logger

Потокобезопасная система логирования с записью в файл.

1.4 Протокол взаимодействия

1.4.1 Протокол аутентификации

1. Клиент отправляет данные в формате: <логин><72_hex_символа>
   - 72 шестнадцатеричных символа = 16 символов соли + 56 символов хэша SHA224
   - Хэш вычисляется как: SHA224(salt_hex + password)
2. Сервер проверяет логин и пароль в базе данных
3. Сервер вычисляет хэш и сравнивает с клиентским
4. Сервер отправляет ответ: "OK" (успех) или "ERR" (ошибка)

1.4.2 Протокол обработки векторов

1. Клиент отправляет количество векторов (uint32_t, сетевой порядок байт)
2. Для каждого вектора:
   - Размер вектора (uint32_t, сетевой порядок байт)
   - Данные вектора (размер × uint32_t, сетевой порядок байт)
3. Сервер для каждого вектора:
   - Вычисляет сумму с ограничением (см. VectorProcessor::sumClamp)
   - Отправляет результат (int32_t, сетевой порядок байт)

1.5 Ограничения

- Максимальное количество векторов на сессию: 100,000
- Максимальный размер одного вектора: 10,000,000 элементов
- Размер данных аутентификации: до 255 байт
- Сервер работает в однопоточном режиме

1.6 Зависимости

Boost.Program_options - для обработки командной строки

Crypto++ - для вычисления SHA224 хэшей

Стандартная библиотека C++17 - для базового функционала

1.7 Ссылки и дополнительная информация

NetworkServer - Основной класс сервера

AuthHandler - Обработчик аутентификации

VectorHandler - Обработчик векторов

Logger - Система логирования

NetworkUtils - Сетевые утилиты

Заметки

Для работы сервера требуется файл базы данных клиентов в формате "логин:пароль"

Предупреждения

Сервер не предназначен для обработки крайне больших векторов (>10^7 элементов)

См. также

VectorProcessor::sumClamp() для подробностей об алгоритме суммирования

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список документированных файлов.

Глава 4

Классы

4.1 Класс AuthDB

Класс для работы с базой данных аутентификации

#include <authdb.h>

Открытые члены

- AuthDB ()=default
  Конструктор по умолчанию
- void loadFromFile (const std::string &filename)
  Загрузка базы данных из файла
- bool findPassword (const std::string &login, std::string &outPassword) const
  Поиск пароля по логину

Закрытые данные

- std::unordered_map< std::string, std::string > db
  Хэш-таблица для хранения пар логин-пароль

4.1.1 Подробное описание

Класс для работы с базой данных аутентификации

4.1.2 Методы

4.1.2.1 findPassword()

```
bool AuthDB::findPassword (
            const std::string & login,
            std::string & outPassword ) const
```

Поиск пароля по логину

Ищет пароль по логину в базе данных

Аргументы

login	Логин пользователя
outPassword	Ссылка на строку для записи найденного пароля

Возвращает

true если логин найден, false в противном случае

Выполняет поиск в хэш-таблице по ключу (логину). Если логин найден, пароль записывается в outPassword.

Аргументы

login	Логин пользователя для поиска
outPassword	Ссылка на строку для записи найденного пароля

Возвращает

true если логин найден в базе данных, false если логин отсутствует

Заметки

Время поиска: в среднем O(1), в худшем случае O(n)

Постусловие

Если возвращено true, outPassword содержит пароль пользователя Если возвращено false, out↩ Password остается неизменным

4.1.2.2 loadFromFile()

```
void AuthDB::loadFromFile (
            const std::string & filename )
```

Загрузка базы данных из файла

Загружает базу данных клиентов из текстового файла

Аргументы

filename	Имя файла в формате "login:password" по строкам

Исключения

std::runtime_error	если файл не может быть открыт

Формат файла: каждая строка содержит пару "логин:пароль" Пустые строки игнорируются. Если формат строки некорректен (нет символа ':'), она пропускается.

Аргументы

filename	Путь к файлу базы данных

Исключения

std::runtime_error	если файл не может быть открыт

Заметки

Предыдущее содержимое базы данных очищается перед загрузкой

Предупреждения

Файл должен быть в формате UTF-8 или ASCII

Постусловие

База данных содержит все корректные пары логин-пароль из файла

Объявления и описания членов классов находятся в файлах:

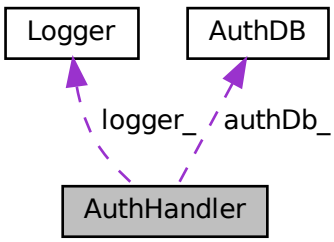
- authdb.h
- authdb.cpp

4.2 Класс AuthHandler

Класс для обработки аутентификации клиентов

#include <auth_handler.h>

Граф связей класса AuthHandler:



Открытые члены

- AuthHandler (Logger &logger, AuthDB &authDb)

  Конструктор обработчика аутентификации
- bool authenticate (int client_fd, std::string &out_login)

  Основной метод аутентификации
- bool parseAuthData (const std::string &data, std::string &login, std::string &salt_hex, std::string &hash_hex)

  Парсинг данных аутентификации
- bool verifyHash (const std::string &login, const std::string &password, const std::string &salt_hex, const std::string &client_hash_hex)

  Проверка хэша пароля

Закрытые члены

- bool sendResponse (int client_fd, bool success)

  Отправка ответа клиенту
- std::string computeSHA224 (const std::string &data)

  Вычисление SHA224 хэша

Закрытые данные

- Logger & logger_

  Ссылка на объект логгера
- AuthDB & authDb_

  Ссылка на базу данных аутентификации

4.2.1 Подробное описание

Класс для обработки аутентификации клиентов

4.2.2 Конструктор(ы)

4.2.2.1 AuthHandler()

```
AuthHandler::AuthHandler (
            Logger & logger,
            AuthDB & authDb )
```

Конструктор обработчика аутентификации

Создает обработчик аутентификации

Аргументы

logger	Логгер для записи событий
~~authDb~~	~~База данных аутентификации~~
logger	Ссылка на логгер для записи событий аутентификации
authDb	Ссылка на базу данных аутентификации

4.2.3 Методы

4.2.3.1 authenticate()

```
bool AuthHandler::authenticate (
            int client_fd,
            std::string & out_login )
```

Основной метод аутентификации

Выполняет процесс аутентификации клиента

Аргументы

client_fd	Файловый дескриптор клиентского сокета
out_login	Ссылка на строку для записи аутентифицированного логина

Возвращает

true если аутентификация успешна, false в противном случае

Процесс аутентификации состоит из следующих шагов:

1. Чтение данных аутентификации из сокета (до 255 байт)
2. Парсинг данных: извлечение логина, соли и хэша
3. Поиск пароля в базе данных по логину
4. Вычисление хэша на стороне сервера и сравнение с клиентским
5. Отправка результата клиенту ("OK" или "ERR")

Аргументы

client_fd	Файловый дескриптор клиентского сокета
out_login	Ссылка на строку для записи аутентифицированного логина

Возвращает

true если аутентификация успешна, false в противном случае

Заметки

Максимальный размер данных аутентификации: 255 байт
Формат данных: <логин><72 шестнадцатеричных символа> где 72 символа = 16 символов соли + 56 символов хэша SHA224

Постусловие

Если аутентификация успешна, out_login содержит логин клиента

4.2.3.2 computeSHA224()

```
std::string AuthHandler::computeSHA224 (
            const std::string & data )   [private]
```

Вычисление SHA224 хэша

Вычисляет SHA224 хэш от переданных данных

Аргументы

data	Данные для хэширования

Возвращает

Хэш в виде hex строки

Использует библиотеку CryptoPP для вычисления хэша. Результат возвращается в виде шестнадцатеричной строки.

Аргументы

data	Строка данных для хэширования

Возвращает

Хэш SHA224 в виде hex строки (56 символов)

Заметки

Размер хэша SHA224: 28 байт (224 бита) = 56 hex символов

См. также

CryptoPP::SHA224

4.2.3.3 parseAuthData()

```
bool AuthHandler::parseAuthData (
            const std::string & data,
            std::string & login,
            std::string & salt_hex,
            std::string & hash_hex )
```

Парсинг данных аутентификации

Парсит данные аутентификации, полученные от клиента

Аргументы

data	Сырые данные от клиента
login	Ссылка на строку для записи логина
salt_hex	Ссылка на строку для записи соли в hex
hash_hex	Ссылка на строку для записи хэша в hex

Возвращает

true если парсинг успешен, false в противном случае

Формат данных: все символы кроме последних 72 - логин, последние 72 символа - шестнадцатеричные данные (16 символов соли + 56 символов хэша)

Аргументы

data	Сырые данные от клиента (логин + hex)
login	Ссылка на строку для записи извлеченного логина
salt_hex	Ссылка на строку для записи соли в hex (16 символов)
hash_hex	Ссылка на строку для записи хэша в hex (56 символов)

Возвращает

true если парсинг успешен, false в противном случае

Предусловие

Длина data должна быть не менее 72 символов

Последние 72 символа должны быть корректной hex строкой

Постусловие

Если возвращено true, параметры содержат извлеченные данные

Заметки

Логин может быть пустым, если data состоит ровно из 72 hex символов

4.2.3.4 sendResponse()

```
bool AuthHandler::sendResponse (
                int client_fd,
                bool success )   [private]
```

Отправка ответа клиенту

Отправляет клиенту результат аутентификации

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
success	Результат аутентификации

Возвращает

true если отправка успешна, false в противном случае

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
success	Результат аутентификации

Возвращает

true если отправка успешна, false в противном случае

Заметки

Формат ответа: "OK" при успехе, "ERR" при неудаче

Длина ответа: 2 байта для "OK", 3 байта для "ERR"

4.2.3.5 verifyHash()

```
bool AuthHandler::verifyHash (
                const std::string & login,
                const std::string & password,
                const std::string & salt_hex,
                const std::string & client_hash_hex )
```

Проверка хэша пароля

Проверяет корректность хэша пароля

Аргументы

login	Логин пользователя
password	Пароль из базы данных
salt_hex	Соль в hex формате
client_hash_hex	Хэш от клиента в hex формате

Возвращает

true если хэши совпадают, false в противном случае

Процесс проверки:

1. Конкатенирует соль (hex) и пароль (plaintext)
2. Вычисляет SHA224 от результата конкатенации
3. Сравнивает полученный хэш с хэшем от клиента

Аргументы

login	Логин пользователя (для логирования)
password	Пароль из базы данных в plaintext
salt_hex	Соль в hex формате (16 символов, 8 байт)
client_hash_hex	Хэш от клиента в hex формате (56 символов)

Возвращает

true если хэши совпадают, false в противном случае

Заметки

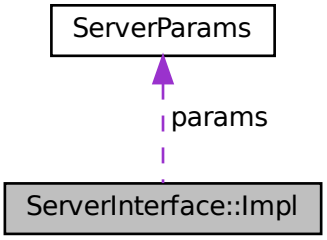
Используется схема: hash = SHA224(salt || password)
Сравнение выполняется с защитой от timing-атак через VerifyBufsEqual

Объявления и описания членов классов находятся в файлах:

- auth_handler.h
- auth_handler.cpp

4.3 Структура ServerInterface::Impl

Граф связей класса ServerInterface::Impl:



Открытые атрибуты

- po::options_description desc {"Allowed options"}
- ServerParams params
- po::variables_map vm

Объявления и описания членов структуры находятся в файле:

- serverInterface.cpp

4.4 Класс Logger

Класс для потокобезопасного логирования в файл

#include <logger.h>

Открытые члены

- Logger (const std::string &filename)
  Конструктор логгера
- ~Logger ()
  Деструктор логгера
- void info (const std::string &msg)
  Запись информационного сообщения
- void error (const std::string &msg)
  Запись сообщения об ошибке
- void warning (const std::string &msg)
  Запись предупреждающего сообщения

Закрытые члены

- void write (const std::string &level, const std::string &msg)
  Основной метод записи в лог

Закрытые данные

- std::mutex mtx
  Мьютекс для синхронизации доступа к файлу
- std::ofstream ofs
  Поток для записи в файл

4.4.1 Подробное описание

Класс для потокобезопасного логирования в файл

4.4.2 Конструктор(ы)

4.4.2.1 Logger()

```
Logger::Logger (
            const std::string & filename )   [explicit]
```

Конструктор логгера

Создает логгер с привязкой к файлу

Аргументы

filename	Имя файла для записи логов

Исключения

std::runtime_error	если файл не может быть открыт

Открывает файл для записи в режиме добавления (append). Если файл не существует, он будет создан. Если файл существует, новые записи будут добавляться в конец.

Аргументы

filename	Путь к файлу лога

Исключения

std::runtime_error	если файл не может быть открыт для записи

Заметки

Формат открытия файла: std::ios::app (добавление в конец)

4.4.2.2 ~Logger()

Logger::~Logger ( )

Деструктор логгера

Закрывает файловый поток, если он был открыт. Гарантирует корректное освобождение ресурсов.

4.4.3 Методы

4.4.3.1 error()

void Logger::error (
const std::string & msg)

Запись сообщения об ошибке

Записывает сообщение об ошибке в лог

Аргументы

msg	Текст сообщения
msg	Текст сообщения об ошибке

Заметки

Уровень: ERROR

4.4.3.2 info()

void Logger::info (
const std::string & msg)

Запись информационного сообщения

Записывает информационное сообщение в лог

Аргументы

msg	Текст сообщения
msg	Текст информационного сообщения

Заметки

Уровень: INFO

4.4.3.3 warning()

void Logger::warning (
const std::string & msg)

Запись предупреждающего сообщения

Записывает предупреждающее сообщение в лог

Аргументы

msg	Текст сообщения
msg	Текст предупреждающего сообщения

Заметки

Уровень: WARNING

4.4.3.4 write()

void Logger::write (
const std::string & level,
const std::string & msg) [private]

Основной метод записи в лог

Основной метод записи сообщения в лог

Аргументы

level	Уровень логирования
msg	Текст сообщения

Формат записи: [YYYY-MM-DD HH:MM:SS] LEVEL: сообщение Время берется с точностью до секунды. Метод потокобезопасен благодаря использованию мьютекса.

Аргументы

level	Уровень логирования (INFO, ERROR, WARNING)
msg	Текст сообщения для записи

Заметки

Использует локальную блокировку мьютекса для предотвращения пересечения записей от разных потоков

Объявления и описания членов классов находятся в файлах:

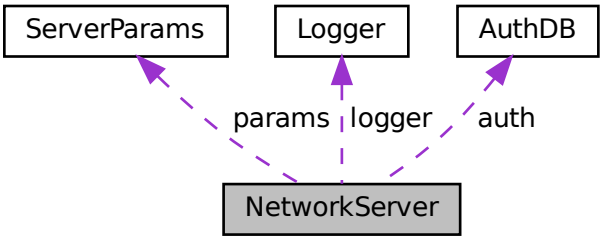
- logger.h
- logger.cpp

4.5 Класс NetworkServer

Основной класс сетевого сервера

#include <network_server.h>

Граф связей класса NetworkServer:



Открытые члены

- NetworkServer (const ServerParams &p, Logger &lg, AuthDB &a)
  Конструктор сервера
- ~NetworkServer ()
  Деструктор сервера
- void run ()
  Запуск основного цикла сервера
- void requestStop ()
  Запрос остановки сервера
- bool isRunning () const
  Проверка состояния работы сервера

Закрытые члены

- void createSocket ()
  Создание слушающего сокета
- void serveClient (int client_fd)
  Обслуживание подключенного клиента

Закрытые данные

- int listen_fd = -1
  Файловый дескриптор слушающего сокета
- ServerParams params
  Параметры конфигурации сервера
- Logger & logger
  Ссылка на объект логгера
- AuthDB & auth
  Ссылка на базу данных аутентификации
- std::atomic< bool > running {true}
  Флаг работы сервера

4.5.1 Подробное описание

Основной класс сетевого сервера

Осуществляет прием подключений, аутентификацию клиентов и обработку их запросов

4.5.2 Конструктор(ы)

4.5.2.1 NetworkServer()

```
NetworkServer::NetworkServer (
            const ServerParams & p,
            Logger & lg,
            AuthDB & a )
```

Конструктор сервера

Создает сетевой сервер с заданными параметрами

Аргументы

p	Параметры конфигурации сервера
lg	Логгер для записи событий
a	База данных аутентификации
p	Параметры конфигурации сервера
lg	Логгер для записи событий
a	База данных аутентификации

Заметки

Дескриптор сокета инициализируется значением -1 (невалидный)

4.5.2.2 ~NetworkServer()

```
NetworkServer::~NetworkServer ( )
```

Деструктор сервера

Закрывает слушающий сокет, если он был открыт. Гарантирует освобождение системных ресурсов.

4.5.3 Методы

4.5.3.1 createSocket()

void NetworkServer::createSocket ( ) [private]

Создание слушающего сокета

Создает и настраивает слушающий сокет

Исключения

std::system_error	при ошибках создания сокета

Выполняет последовательность действий:

1. Создание TCP сокета (AF_INET, SOCK_STREAM)
2. Установка опции SO_REUSEADDR для быстрого переиспользования порта
3. Привязка сокета к указанному адресу и порту
4. Перевод сокета в режим прослушивания

Исключения

std::system_error	при ошибках создания/настройки сокета

Постусловие

listen_fd содержит валидный дескриптор слушающего сокета

Заметки

Очередь ожидающих соединений установлена в 5 (стандартное значение)

4.5.3.2 isRunning()

bool NetworkServer::isRunning ( ) const

Проверка состояния работы сервера

Проверяет, работает ли сервер

Возвращает

true если сервер работает, false в противном случае

4.5.3.3 requestStop()

void NetworkServer::requestStop ( )

Запрос остановки сервера

Запрашивает остановку сервера

Устанавливает флаг running в false и закрывает слушающий сокет. Это приводит к выходу из основного цикла accept().

Заметки

Потокобезопасен благодаря использованию std::atomic<bool>

4.5.3.4 run()

void NetworkServer::run ( )

Запуск основного цикла сервера

Запускает основной цикл работы сервера

Алгоритм работы:

1. Создание слушающего сокета
2. Цикл while(running): a. Ожидание подключения клиента (accept) b. Принятие соединения c. Обработка клиента в serveClient() d. Закрытие клиентского сокета
3. Логирование завершения работы

   Заметки

   Сервер работает в однопоточном (последовательном) режиме
   Цикл прерывается при установке флага running в false

   См. также

   serveClient()

4.5.3.5 serveClient()

void NetworkServer::serveClient (
int client_fd) [private]

Обслуживание подключенного клиента

Обслуживает подключенного клиента

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета

Процесс обслуживания состоит из двух этапов:

1. Аутентификация:
   - Создание AuthHandler
   - Проверка учетных данных клиента
   - Если аутентификация не пройдена - закрытие соединения
2. Обработка векторов:
   - Создание VectorHandler
   - Чтение и обработка векторных данных
   - Отправка результатов клиенту

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета

Исключения

Может	генерировать исключения из AuthHandler и VectorHandler

Заметки

Оба этапа выполняются в одном потоке последовательно
При ошибке на любом этапе соединение закрывается

Объявления и описания членов классов находятся в файлах:

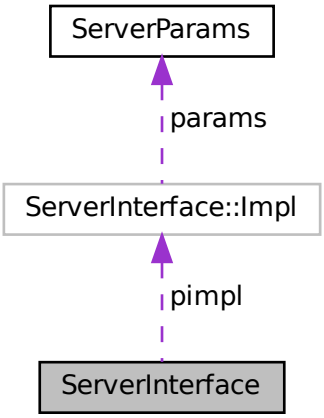
- network_server.h
- network_server.cpp

4.6 Класс ServerInterface

Класс для обработки параметров командной строки сервера

#include <serverInterface.h>

Граф связей класса ServerInterface:



Классы

- struct Impl

Открытые члены

- ServerInterface ()
  Конструктор интерфейса командной строки
- bool parse (int argc, char **argv)
  Парсинг аргументов командной строки
- ServerParams getParams () const
  Получение параметров сервера
- std::string getDescription () const
  Получение описания параметров командной строки

Закрытые данные

- Impl * pimpl
  Указатель на реализацию (PImpl)

4.6.1 Подробное описание

Класс для обработки параметров командной строки сервера

Использует паттерн PImpl для сокрытия реализации

4.6.2 Методы

4.6.2.1 getDescription()

std::string ServerInterface::getDescription ( ) const

Получение описания параметров командной строки

Возвращает

Строка с описанием всех доступных опций

4.6.2.2 getParams()

ServerParams ServerInterface::getParams ( ) const

Получение параметров сервера

Возвращает

Структура с параметрами конфигурации

4.6.2.3 parse()

```
bool ServerInterface::parse (
            int argc,
            char ** argv )
```

Парсинг аргументов командной строки

Аргументы

argc	Количество аргументов
argv	Массив аргументов

Возвращает

true если парсинг успешен, false если требуется показать справку

Объявления и описания членов классов находятся в файлах:

- serverInterface.h
- serverInterface.cpp

4.7 Структура ServerParams

Структура параметров конфигурации сервера

#include <server_params.h>

Открытые атрибуты

- int port = 33333
  Порт для прослушивания
- std::string address = "127.0.0.1"
  IP-адрес для привязки
- std::string logFile = "server.log"
  Путь к файлу логов
- std::string clientsDbFile = "clients.db"
  Путь к файлу базы данных клиентов
- bool help = false
  Флаг запроса справки

4.7.1 Подробное описание

Структура параметров конфигурации сервера

Объявления и описания членов структуры находятся в файле:

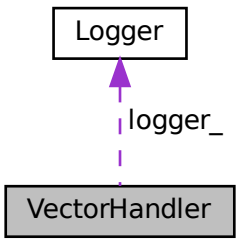
- server_params.h

4.8 Класс VectorHandler

Класс для обработки векторных запросов от клиентов

#include <vector_handler.h>

Граф связей класса VectorHandler:



Открытые члены

- VectorHandler (Logger &logger)
  Конструктор обработчика векторов
- void process (int client_fd, const std::string &login)
  Основной метод обработки векторов
- bool readVector (int client_fd, std::vector< uint32_t > &vector)
  Чтение вектора из сокета
- int32_t processVector (const std::vector< uint32_t > &vector)
  Обработка одного вектора
- bool sendResult (int client_fd, int32_t result)
  Отправка результата клиенту

Закрытые члены

- uint32_t readVectorCount (int client_fd)
  Чтение количества векторов
- bool validateVectorCount (uint32_t count)
  Валидация количества векторов
- bool validateVectorSize (uint32_t size)
  Валидация размера вектора

Закрытые данные

- Logger & logger_
  Ссылка на объект логгера

4.8.1 Подробное описание

Класс для обработки векторных запросов от клиентов

4.8.2 Конструктор(ы)

4.8.2.1 VectorHandler()

VectorHandler::VectorHandler (
Logger & logger) [explicit]

Конструктор обработчика векторов

Создает обработчик векторных запросов

Аргументы

logger	Логгер для записи событий
logger	Логгер для записи событий обработки векторов

4.8.3 Методы

4.8.3.1 process()

```
void VectorHandler::process (
            int client_fd,
            const std::string & login )
```

Основной метод обработки векторов

Основной метод обработки векторов для аутентифицированного клиента

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
login	Логин аутентифицированного пользователя

Исключения

std::runtime_error	при ошибках обработки

Процесс обработки:

1. Чтение количества векторов (uint32_t, сетевой порядок байт)
2. Для каждого вектора: a. Чтение размера вектора b. Чтение данных вектора c. Обработка вектора (суммирование с ограничением) d. Отправка результата клиенту
3. Логирование прогресса и статистики

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
login	Логин аутентифицированного пользователя (для логирования)

Исключения

std::runtime_error	при ошибках чтения/записи или неверных данных

Заметки

Максимальное количество векторов: 100,000
Максимальный размер одного вектора: 10,000,000 элементов
Каждые 10 векторов логируется прогресс обработки

4.8.3.2 processVector()

```
int32_t VectorHandler::processVector (
            const std::vector< uint32_t > & vector )
```

Обработка одного вектора

Обрабатывает один вектор (суммирование с ограничением)

Аргументы

vector	Вектор для обработки

Возвращает

Результат обработки (сумма с ограничением)

Аргументы

vector	Вектор для обработки

Возвращает

Результат обработки (сумма элементов с ограничением)

Заметки

Использует VectorProcessor::sumClamp() для вычисления

4.8.3.3 readVector()

```
bool VectorHandler::readVector (
            int client_fd,
            std::vector< uint32_t > & vector )
```

Чтение вектора из сокета

Читает один вектор из сокета

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
vector	Ссылка на вектор для записи данных

Возвращает

true если чтение успешно, false в противном случае

Читает размер вектора (uint32_t), затем читает данные вектора. Размер вектора проверяется на корректность.

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
vector	Ссылка на вектор для записи данных

Возвращает

true если чтение успешно, false в противном случае

Заметки

Использует recvAll() для гарантированного чтения всех данных

Постусловие

Если возвращено true, vector содержит прочитанные данные

4.8.3.4 readVectorCount()

uint32_t VectorHandler::readVectorCount (
int client_fd) [private]

Чтение количества векторов

Читает количество векторов для обработки

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета

Возвращает

Количество векторов для обработки

Исключения

std::runtime_error	при неверном количестве

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета

Возвращает

Количество векторов для обработки

Исключения

std::runtime_error	при ошибке чтения или неверном значении

Заметки

Значение читается в сетевом порядке байт и проверяется на корректность

4.8.3.5 sendResult()

```
bool VectorHandler::sendResult (
                int client_fd,
                int32_t result )
```

Отправка результата клиенту

Отправляет результат обработки вектора клиенту

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
result	Результат обработки

Возвращает

true если отправка успешна, false в противном случае

Аргументы

client↩ _fd	Файловый дескриптор клиентского сокета
result	Результат обработки вектора

Возвращает

true если отправка успешна, false в противном случае

Заметки

Результат отправляется в сетевом порядке байт

4.8.3.6 validateVectorCount()

```
bool VectorHandler::validateVectorCount (
            uint32_t count )  [private]
```

Валидация количества векторов

Проверяет корректность количества векторов

Аргументы

count	Проверяемое количество

Возвращает

true если количество допустимо, false в противном случае

Аргументы

count	Проверяемое количество векторов

Возвращает

true если количество допустимо, false в противном случае

Заметки

Допустимый диапазон: 1..100,000

4.8.3.7 validateVectorSize()

```
bool VectorHandler::validateVectorSize (
            uint32_t size )  [private]
```

Валидация размера вектора

Проверяет корректность размера вектора

Аргументы

size	Проверяемый размер

Возвращает

true если размер допустим, false в противном случае

Аргументы

size	Проверяемый размер вектора

Возвращает

true если размер допустим, false в противном случае

Заметки

Допустимый диапазон: 1..10,000,000

Объявления и описания членов классов находятся в файлах:

- vector_handler.h
- vector_handler.cpp

4.9 Класс VectorProcessor

Класс для обработки векторов чисел с контролем переполнения

#include <vector_processor.h>

Открытые статические члены

- static int32_t sumClamp (const std::vector< uint32_t > &v)
  Суммирует элементы вектора с контролем переполнения

4.9.1 Подробное описание

Класс для обработки векторов чисел с контролем переполнения

4.9.2 Методы

4.9.2.1 sumClamp()

int32_t VectorProcessor::sumClamp (
const std::vector< uint32_t > & v) [static]

Суммирует элементы вектора с контролем переполнения

Использует 64-битный аккумулятор для избежания переполнения, возвращает результат в диапазоне [0, 2^31-1]

Аргументы

v	Вектор 32-битных беззнаковых целых чисел

Возвращает

Сумма элементов, приведенная к int32_t с учетом ограничений

Объявления и описания членов классов находятся в файлах:

- vector_processor.h
- vector_processor.cpp

Глава 5

Файлы

5.1 auth_handler.h

```
1 #ifndef AUTH_HANDLER_H
2 #define AUTH_HANDLER_H
3
4 #include <string>
5 #include "logger.h"
6 #include "authdb.h"
7
12 class AuthHandler {
13 public:
19     AuthHandler(Logger& logger, AuthDB& authDb);
20
27     bool authenticate(int client_fd, std::string& out_login);
28
37     bool parseAuthData(const std::string& data, std::string& login,
38                        std::string& salt_hex, std::string& hash_hex);
39
48     bool verifyHash(const std::string& login, const std::string& password,
49                     const std::string& salt_hex, const std::string& client_hash_hex);
50
51 private:
52     Logger& logger_;
53     AuthDB& authDb_;
54
61     bool sendResponse(int client_fd, bool success);
62
68     std::string computeSHA224(const std::string& data);
69 };
70
71 #endif
```

5.2 authdb.h

```
1 #pragma once
2 #include <string>
3 #include <unordered_map>
4
9 class AuthDB {
10 public:
14     AuthDB() = default;
15
21     void loadFromFile(const std::string& filename);
22
29     bool findPassword(const std::string& login, std::string& outPassword) const;
30
31 private:
32     std::unordered_map<std::string,std::string> db;
33 };
```

5.3 logger.h

```
1 #pragma once
2 #include <string>
3 #include <mutex>
4 #include <fstream>
5
10 class Logger {
11 public:
17     explicit Logger(const std::string& filename);
18
22     ~Logger();
23
28     void info(const std::string& msg);
29
34     void error(const std::string& msg);
35
40     void warning(const std::string& msg);
41
42 private:
43     std::mutex mtx;
44     std::ofstream ofs;
45
51     void write(const std::string& level, const std::string& msg);
52 };
```

5.4 network_server.h

```
1 #ifndef NETWORK_SERVER_H
2 #define NETWORK_SERVER_H
3
4 #include "server_params.h"
5 #include <atomic>
6 #include <cstdint>
7
8 class Logger;
9 class AuthDB;
10
16 class NetworkServer {
17 public:
24     NetworkServer(const ServerParams& p, Logger& lg, AuthDB& a);
25
29     ~NetworkServer();
30
34     void run();
35
39     void requestStop();
40
45     bool isRunning() const;
46
47 private:
52     void createSocket();
53
58     void serveClient(int client_fd);
59
60     int listen_fd = -1;
61     ServerParams params;
62     Logger& logger;
63     AuthDB& auth;
64     std::atomic<bool> running{true};
65 };
66
67 #endif
```

5.5 network_utils.h

```
1 #ifndef NETWORK_UTILS_H
2 #define NETWORK_UTILS_H
3
4 #include <string>
5 #include <cstddef>
6 #include <cstdint>
7 #include <netinet/in.h>
8
9 namespace NetworkUtils {
17     ssize_t recvAll(int fd, void* buf, size_t len);
18
25     std::string bytesToHex(const unsigned char* data, size_t length);
26
```

```
34     bool hexToBytes(const std::string& hex, unsigned char* output, size_t output_len);
35
41     std::string sockaddrToString(const sockaddr_in& addr);
42
48     bool isValidHex(const std::string& str);
49
56     uint32_t readNetworkUint32(int fd);
57
64     bool sendNetworkUint32(int fd, uint32_t value);
65 }
66
67 #endif
```

5.6 server_params.h

```
1 #ifndef SERVER_PARAMS_H
2 #define SERVER_PARAMS_H
3
4 #include <string>
5
10 struct ServerParams {
11     int port = 33333;
12     std::string address = "127.0.0.1";
13     std::string logFile = "server.log";
14     std::string clientsDbFile = "clients.db";
15     bool help = false;
16 };
17
18 #endif
```

5.7 serverInterface.h

```
1 #pragma once
2 #include <string>
3 #include "server_params.h"
4
10 class ServerInterface {
11 public:
15     ServerInterface();
16
23     bool parse(int argc, char** argv);
24
29     ServerParams getParams() const;
30
35     std::string getDescription() const;
36
37 private:
38     struct Impl;
39     Impl* pimpl;
40 };
```

5.8 vector_handler.h

```
1 #ifndef VECTOR_HANDLER_H
2 #define VECTOR_HANDLER_H
3
4 #include <string>
5 #include <cstdint>
6 #include "logger.h"
7 #include "vector_processor.h"
8
13 class VectorHandler {
14 public:
19     explicit VectorHandler(Logger& logger);
20
27     void process(int client_fd, const std::string& login);
28
35     bool readVector(int client_fd, std::vector<uint32_t>& vector);
36
42     int32_t processVector(const std::vector<uint32_t>& vector);
43
50     bool sendResult(int client_fd, int32_t result);
51
52 private:
53     Logger& logger_;
54
```

```
61     uint32_t readVectorCount(int client_fd);
62
68     bool validateVectorCount(uint32_t count);
69
75     bool validateVectorSize(uint32_t size);
76 };
77
78 #endif
```

5.9 vector_processor.h

```
1 #pragma once
2 #include <vector>
3 #include <cstdint>
4
9 class VectorProcessor {
10 public:
18     static int32_t sumClamp(const std::vector<uint32_t>& v);
19 };
```

Глава 6

Примеры

6.1 /home/ryaker4/local_git/Server/Server/network_utils.cpp

Преобразует массив байт в шестнадцатеричную строку в верхнем регистре

Преобразует массив байт в шестнадцатеричную строку в верхнем регистре Каждый байт преобразуется в два шестнадцатеричных символа. Используется для удобного отображения бинарных данных в логах и для передачи хэшей в текстовом формате.

Аргументы

data	Указатель на массив байт
length	Количество байт для преобразования

Возвращает

Строка, содержащая шестнадцатеричное представление данных

bytesToHex({0xDE, 0xAD, 0xBE, 0xEF}, 4) -> "DEADBEEF"

```
#include "network_utils.h"
#include <arpa/inet.h>
#include <cstring>
#include <iomanip>
#include <sstream>
#include <unistd.h>
#include <algorithm>
namespace NetworkUtils {
ssize_t recvAll(int fd, void* buf, size_t len) {
    char* p = static_cast<char*>(buf);
    size_t rem = len;
    while(rem > 0) {
        ssize_t r = recv(fd, p, rem, 0);
        if(r <= 0)
            return r;
        p += r;
        rem -= r;
    }
    return static_cast<ssize_t>(len);
}
std::string bytesToHex(const unsigned char* data, size_t length) {
    std::stringstream ss;
    ss « std::hex « std::setfill('0') « std::uppercase;
    for(size_t i = 0; i < length; ++i) {
        ss « std::setw(2) « static_cast<int>(data[i]);
    }
    return ss.str();
}
```

```
bool hexToBytes(const std::string& hex, unsigned char* output, size_t output_len) {
    if(hex.length() != output_len * 2) {
        return false;
    }

    auto hexCharToByte = [](char c) -> unsigned char {
        if(c >= '0' && c <= '9')
            return c - '0';
        if(c >= 'a' && c <= 'f')
            return 10 + (c - 'a');
        if(c >= 'A' && c <= 'F')
            return 10 + (c - 'A');
        return 0xFF;
    };

    for(size_t i = 0; i < output_len; i++) {
        unsigned char high = hexCharToByte(hex[i * 2]);
        unsigned char low = hexCharToByte(hex[i * 2 + 1]);
        if(high == 0xFF || low == 0xFF) {
            return false;
        }
        output[i] = (high « 4) | low;
    }

    return true;
}
std::string sockaddrToString(const sockaddr_in& addr) {
    char ipbuf[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &addr.sin_addr, ipbuf, sizeof(ipbuf));
    return std::string(ipbuf) + ":" + std::to_string(ntohs(addr.sin_port));
}
bool isValidHex(const std::string& str) {
    return std::all_of(str.begin(), str.end(), [](char c) {
        return (c >= '0' && c <= '9') ||
               (c >= 'a' && c <= 'f') ||
               (c >= 'A' && c <= 'F');
    });
}
uint32_t readNetworkUint32(int fd) {
    uint32_t value;
    if(recvAll(fd, &value, 4) != 4) {
        throw std::runtime_error("Failed to read uint32");
    }
    return value;
}
bool sendNetworkUint32(int fd, uint32_t value) {
    return send(fd, &value, 4, 0) == 4;
}
}
```

Предметный указатель