Сетевой клиент для обработки векторов

Создано системой Doxygen 1.9.4

Глава 1

Алфавитный указатель классов

1.1 Классы

Классы с их кратким описанием.

Глава 2

Список файлов

2.1 Файлы

Полный список документированных файлов.

Глава 3

Классы

3.1 Класс FileHandler

Класс для операций чтения/записи файлов

#include <file_handler.h>

Открытые статические члены

- static VectorData readVectors (const std::string &filename)
  Читает векторы из бинарного файла
- static void writeResults (const std::string &filename, const std::vector< int16_t > &results)
  Записывает результаты в бинарный файл
- static std::pair< std::string, std::string > readCredentials (const std::string &filename)
  Читает учетные данные из файла конфигурации

3.1.1 Подробное описание

Класс для операций чтения/записи файлов

См. определение в файле file_handler.h строка 27

3.1.2 Методы

3.1.2.1 readCredentials()

std::pair< std::string, std::string > FileHandler::readCredentials (
const std::string & filename) [static]

Читает учетные данные из файла конфигурации

Аргументы

filename	Имя файла конфигурации

Возвращает

Пара (логин, пароль)

Исключения

std::runtime_error	при ошибках открытия или неверном формате файла

См. определение в файле file_handler.cpp строка 50

3.1.2.2 readVectors()

```
VectorData FileHandler::readVectors (
            const std::string & filename )   [static]
```

Читает векторы из бинарного файла

Аргументы

filename	Имя входного файла

Возвращает

Структура VectorData с прочитанными векторами

Исключения

std::runtime_error	при ошибках открытия или чтения файла

См. определение в файле file_handler.cpp строка 6

3.1.2.3 writeResults()

```
void FileHandler::writeResults (
            const std::string & filename,
            const std::vector< int16_t > & results )   [static]
```

Записывает результаты в бинарный файл

Аргументы

filename	Имя выходного файла
results	Вектор результатов для записи

Исключения

std::runtime_error	при ошибках создания или записи файла

См. определение в файле file_handler.cpp строка 37

Объявления и описания членов классов находятся в файлах:

- file_handler.h
- file_handler.cpp

3.2 Класс Interface

Класс для разбора командной строки и отображения справки

#include <interface.h>

Открытые статические члены

- static bool parseCommandLine (int argc, char *argv[], Params &params)
  Разбирает аргументы командной строки
- static void showHelp ()
  Отображает справку по использованию программы

3.2.1 Подробное описание

Класс для разбора командной строки и отображения справки

См. определение в файле interface.h строка 29

3.2.2 Методы

3.2.2.1 parseCommandLine()

```
bool Interface::parseCommandLine (
            int argc,
            char * argv[],
            Params & params )    [static]
```

Разбирает аргументы командной строки

Аргументы

argc	Количество аргументов
argv	Массив аргументов
params	Структура для сохранения разобранных параметров

Возвращает

true - успешный разбор, false - требуется выход (помощь или ошибка)

См. определение в файле interface.cpp строка 7

3.2.2.2 showHelp()

void Interface::showHelp ( ) [static]

Отображает справку по использованию программы

См. определение в файле interface.cpp строка 54

Объявления и описания членов классов находятся в файлах:

- interface.h
- interface.cpp

3.3 Класс LinuxAuthenticator

Класс для вычисления хеш-сумм с использованием Linux Crypto API через сокеты AF_ALG.

#include <linux_authenticator.h>

Открытые статические члены

- static std::string computeSHA224 (const std::string &data)
  Вычисляет SHA-224 хеш для строки данных
- static std::string computeSHA224 (const std::string &salt, const std::string &password)
  Вычисляет SHA-224 хеш для комбинации соли и пароля

Закрытые статические члены

- static int createCryptoSocket (const std::string &algorithm)
  Создает криптографический сокет для указанного алгоритма
- static std::vector< uint8_t > computeHash (int crypto_sock, const std::string &data)
  Вычисляет хеш для данных с использованием созданного криптосокета

3.3.1 Подробное описание

Класс для вычисления хеш-сумм с использованием Linux Crypto API через сокеты AF_ALG.

Предоставляет статические методы для вычисления SHA-224 хешей для данных и комбинации соли с паролем

См. определение в файле linux_authenticator.h строка 19

3.3.2 Методы

3.3.2.1 computeHash()

```
std::vector< uint8_t > LinuxAuthenticator::computeHash (
            int crypto_sock,
            const std::string & data )   [static], [private]
```

Вычисляет хеш для данных с использованием созданного криптосокета

Аргументы

crypto_sock	Дескриптор криптографического сокета
data	Данные для хеширования

Возвращает

Вектор байтов с вычисленным хешем

Исключения

std::runtime_error	при ошибках передачи данных или получения хеша

См. определение в файле linux_authenticator.cpp строка 39

3.3.2.2 computeSHA224() [1/2]

```
std::string LinuxAuthenticator::computeSHA224 (
            const std::string & data )   [static]
```

Вычисляет SHA-224 хеш для строки данных

Аргументы

data	Входные данные для хеширования

Возвращает

SHA-224 хеш в виде шестнадцатеричной строки

Исключения

std::runtime_error	при ошибках создания сокета или вычисления хеша

См. определение в файле linux_authenticator.cpp строка 71

3.3.2.3 computeSHA224() [2/2]

```
std::string LinuxAuthenticator::computeSHA224 (
            const std::string & salt,
            const std::string & password )    [static]
```

Вычисляет SHA-224 хеш для комбинации соли и пароля

Аргументы

salt	Соль для хеширования
password	Пароль для хеширования

Возвращает

SHA-224 хеш в виде шестнадцатеричной строки (соль + пароль)

Исключения

std::runtime_error	при ошибках создания сокета или вычисления хеша

См. определение в файле linux_authenticator.cpp строка 97

3.3.2.4 createCryptoSocket()

```
int LinuxAuthenticator::createCryptoSocket (
            const std::string & algorithm )    [static], [private]
```

Создает криптографический сокет для указанного алгоритма

Аргументы

algorithm	Название алгоритма хеширования (например, "sha224")

Возвращает

Дескриптор криптографического сокета

Исключения

std::runtime_error	при ошибках создания или привязки сокета

См. определение в файле linux_authenticator.cpp строка 13

Объявления и описания членов классов находятся в файлах:

- linux_authenticator.h
- linux_authenticator.cpp

3.4 Класс MockNetworkManager

Открытые члены

- MockNetworkManager (const std::string &addr, int p)
- bool connect ()
- bool authenticate (const std::string &login, const std::string &password)
- std::vector< int16_t > sendVectors (const std::vector< std::vector< int16_t > > &vectors)

3.4.1 Подробное описание

См. определение в файле test_network_manager.cpp строка 10

3.4.2 Конструктор(ы)

3.4.2.1 MockNetworkManager()

```
MockNetworkManager::MockNetworkManager (
            const std::string & addr,
            int p )   [inline]
```

См. определение в файле test_network_manager.cpp строка 12

3.4.3 Методы

3.4.3.1 authenticate()

```
bool MockNetworkManager::authenticate (
            const std::string & login,
            const std::string & password )   [inline]
```

См. определение в файле test_network_manager.cpp строка 18

3.4.3.2 connect()

```
bool MockNetworkManager::connect ( )   [inline]
```

См. определение в файле test_network_manager.cpp строка 14

3.4.3.3 sendVectors()

```
std::vector< int16_t > MockNetworkManager::sendVectors (
            const std::vector< std::vector< int16_t > > & vectors )   [inline]
```

См. определение в файле test_network_manager.cpp строка 25

Объявления и описания членов класса находятся в файле:

- test_network_manager.cpp

3.5 Класс NetworkManager

Класс для управления сетевым соединением с сервером

#include <network_manager.h>

Открытые члены

- NetworkManager (const std::string &addr, int p)
  Конструктор NetworkManager.
- ~NetworkManager ()
  Деструктор NetworkManager.
- bool connect ()
  Устанавливает соединение с сервером
- void disconnect ()
  Закрывает соединение с сервером
- bool authenticate (const std::string &login, const std::string &password)
  Выполняет аутентификацию на сервере
- std::vector< int16_t > sendVectors (const VectorData &vectors)
  Отправляет векторы на сервер и получает результаты вычислений

Закрытые члены

- void sendData (const void *data, size_t size)

  Отправляет бинарные данные через сокет
- void receiveData (void *data, size_t size)

  Принимает бинарные данные через сокет
- std::string receiveString ()

  Принимает строку через сокет
- void sendString (const std::string &str)

  Отправляет строку через сокет

Закрытые данные

- int sockfd

  Дескриптор сетевого сокета
- std::string address

  Адрес сервера
- int port

  Порт сервера

3.5.1 Подробное описание

Класс для управления сетевым соединением с сервером

Обеспечивает подключение, аутентификацию и обмен данными с сервером

См. определение в файле network_manager.h строка 21

3.5.2 Конструктор(ы)

3.5.2.1 NetworkManager()

```
NetworkManager::NetworkManager (
            const std::string & addr,
            int p )
```

Конструктор NetworkManager.

Аргументы

addr	Адрес сервера
p	Порт сервера

См. определение в файле network_manager.cpp строка 13

3.5.2.2 ~NetworkManager()

```
NetworkManager::~NetworkManager ( )
```

Деструктор NetworkManager.

Автоматически закрывает соединение при уничтожении объекта

См. определение в файле network_manager.cpp строка 16

3.5.3 Методы

3.5.3.1 authenticate()

```
bool NetworkManager::authenticate (
            const std::string & login,
            const std::string & password )
```

Выполняет аутентификацию на сервере

Аргументы

login	Логин пользователя
password	Пароль пользователя

Возвращает

true - аутентификация успешна, false - аутентификация не удалась

Исключения

std::runtime_error	при ошибках сетевого обмена

См. определение в файле network_manager.cpp строка 79

3.5.3.2 connect()

```
bool NetworkManager::connect ( )
```

Устанавливает соединение с сервером

Возвращает

true - соединение установлено

Исключения

std::runtime_error	при ошибках создания сокета или подключения

См. определение в файле network_manager.cpp строка 20

3.5.3.3 disconnect()

```
void NetworkManager::disconnect ( )
```

Закрывает соединение с сервером

См. определение в файле network_manager.cpp строка 43

3.5.3.4 receiveData()

```
void NetworkManager::receiveData (
            void * data,
            size_t size )   [private]
```

Принимает бинарные данные через сокет

Аргументы

data	Указатель на буфер для данных
size	Ожидаемый размер данных в байтах

Исключения

std::runtime_error	при ошибке приема

См. определение в файле network_manager.cpp строка 58

3.5.3.5 receiveString()

```
std::string NetworkManager::receiveString ( )   [private]
```

Принимает строку через сокет

Возвращает

Принятая строка

Исключения

std::runtime_error	при ошибке приема

См. определение в файле network_manager.cpp строка 65

3.5.3.6 sendData()

```
void NetworkManager::sendData (
            const void * data,
            size_t size )   [private]
```

Отправляет бинарные данные через сокет

Аргументы

data	Указатель на данные
size	Размер данных в байтах

Исключения

std::runtime_error	при ошибке отправки

См. определение в файле network_manager.cpp строка 51

3.5.3.7 sendString()

```
void NetworkManager::sendString (
            const std::string & str )   [private]
```

Отправляет строку через сокет

Аргументы

str	Строка для отправки

Исключения

std::runtime_error	при ошибке отправки

См. определение в файле network_manager.cpp строка 75

3.5.3.8 sendVectors()

```
std::vector< int16_t > NetworkManager::sendVectors (
                const VectorData & vectors )
```

Отправляет векторы на сервер и получает результаты вычислений

Аргументы

vectors	Данные векторов для отправки

Возвращает

Вектор результатов вычислений

Исключения

std::runtime_error	при ошибках сетевого обмена

См. определение в файле network_manager.cpp строка 115

3.5.4 Данные класса

3.5.4.1 address

```
std::string NetworkManager::address  [private]
```

Адрес сервера

См. определение в файле network_manager.h строка 24

3.5.4.2 port

```
int NetworkManager::port  [private]
```

Порт сервера

См. определение в файле network_manager.h строка 25

3.5.4.3 sockfd

int NetworkManager::sockfd [private]

Дескриптор сетевого сокета

См. определение в файле network_manager.h строка 23

Объявления и описания членов классов находятся в файлах:

- network_manager.h
- network_manager.cpp

3.6 Структура Params

Структура для хранения параметров командной строки программы

#include <interface.h>

Открытые атрибуты

- std::string server_address
  Адрес сервера для подключения
- int server_port
  Порт сервера для подключения
- std::string input_file
  Путь к входному файлу с векторами
- std::string output_file
  Путь к выходному файлу для результатов
- std::string config_file
  Путь к файлу конфигурации с учетными данными

3.6.1 Подробное описание

Структура для хранения параметров командной строки программы

См. определение в файле interface.h строка 17

3.6.2 Данные класса

3.6.2.1 config_file

std::string Params::config_file

Путь к файлу конфигурации с учетными данными

См. определение в файле interface.h строка 22

3.6.2.2 input_file

std::string Params::input_file

Путь к входному файлу с векторами

См. определение в файле interface.h строка 20

3.6.2.3 output_file

std::string Params::output_file

Путь к выходному файлу для результатов

См. определение в файле interface.h строка 21

3.6.2.4 server_address

std::string Params::server_address

Адрес сервера для подключения

См. определение в файле interface.h строка 18

3.6.2.5 server_port

int Params::server_port

Порт сервера для подключения

См. определение в файле interface.h строка 19

Объявления и описания членов структуры находятся в файле:

- interface.h

3.7 Структура VectorData

Структура для хранения векторов, считанных из файла

#include <file_handler.h>

Открытые атрибуты

- std::vector< std::vector< int16_t > > vectors
  Коллекция векторов (каждый вектор - массив int16_t)

3.7.1 Подробное описание

Структура для хранения векторов, считанных из файла

См. определение в файле file_handler.h строка 19

3.7.2 Данные класса

3.7.2.1 vectors

std::vector<std::vector<int16_t> > VectorData::vectors

Коллекция векторов (каждый вектор - массив int16_t)

См. определение в файле file_handler.h строка 20

Объявления и описания членов структуры находятся в файле:

- file_handler.h

Глава 4

Файлы

4.1 file_handler.cpp

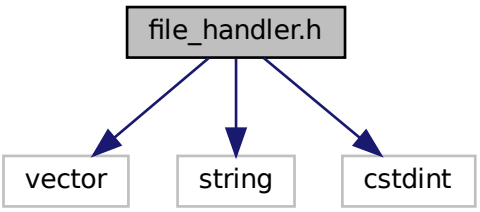
```
00001 #include "file_handler.h"
00002 #include <fstream>
00003 #include <stdexcept>
00004 #include <iostream>
00005
00006 VectorData FileHandler::readVectors(const std::string& filename) {
00007     std::ifstream file(filename, std::ios::binary);
00008     if (!file) {
00009         throw std::runtime_error("Не удалось открыть файл: " + filename);
00010     }
00011
00012     VectorData data;
00013     uint32_t num_vectors;
00014
00015     file.read(reinterpret_cast<char*>(&num_vectors), sizeof(num_vectors));
00016     if (!file) throw std::runtime_error("Ошибка чтения количества векторов");
00017
00018     std::cout « "Чтение " « num_vectors « " векторов из файла..." « std::endl;
00019
00020     for (uint32_t i = 0; i < num_vectors; ++i) {
00021         uint32_t vector_size;
00022         file.read(reinterpret_cast<char*>(&vector_size), sizeof(vector_size));
00023         if (!file) throw std::runtime_error("Ошибка чтения размера вектора");
00024
00025         std::vector<int16_t> vector(vector_size);
00026         file.read(reinterpret_cast<char*>(vector.data()), vector_size * sizeof(int16_t));
00027         if (!file) throw std::runtime_error("Ошибка чтения данных вектора");
00028
00029         data.vectors.push_back(vector);
00030
00031         std::cout « "  Вектор " « i + 1 « ": размер " « vector_size « std::endl;
00032     }
00033
00034     return data;
00035 }
00036
00037 void FileHandler::writeResults(const std::string& filename, const std::vector<int16_t>& results) {
00038     std::ofstream file(filename, std::ios::binary);
00039     if (!file) {
00040         throw std::runtime_error("Не удалось создать файл: " + filename);
00041     }
00042
00043     uint32_t num_results = results.size();
00044     file.write(reinterpret_cast<const char*>(&num_results), sizeof(num_results));
00045     file.write(reinterpret_cast<const char*>(results.data()), results.size() * sizeof(int16_t));
00046
00047     std::cout « "Записано " « num_results « " результатов в файл: " « filename « std::endl;
00048 }
00049
00050 std::pair<std::string, std::string> FileHandler::readCredentials(const std::string& filename) {
00051     std::ifstream file(filename);
00052     if (!file) {
00053         throw std::runtime_error("Не удалось открыть файл конфигурации: " + filename);
00054     }
00055
00056     std::string line;
00057     if (!std::getline(file, line)) {
00058         throw std::runtime_error("Пустой файл конфигурации: " + filename);
```

```
00059     }
00060
00061     size_t pos = line.find(':');
00062     if (pos == std::string::npos) {
00063         throw std::runtime_error("Неверный формат файла конфигурации: " + filename);
00064     }
00065
00066     std::string login = line.substr(0, pos);
00067     std::string password = line.substr(pos + 1);
00068
00069     std::cout « "Прочитаны учетные данные: логин=" « login « std::endl;
00070
00071     return {login, password};
00072 }
```

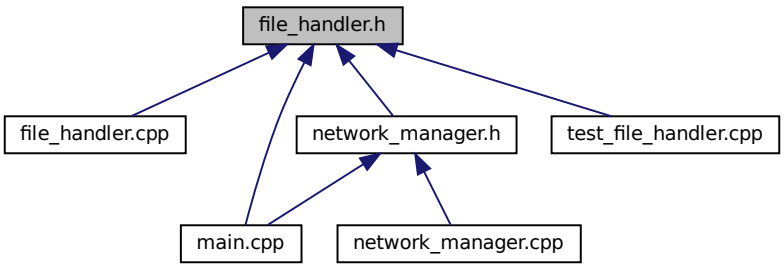
4.2 Файл file_handler.h

Заголовочный файл класса FileHandler для работы с файлами

#include <vector>
#include <string>
#include <cstdint>
Граф включаемых заголовочных файлов для file_handler.h:



Граф файлов, в которые включается этот файл:



Классы

- struct VectorData

  Структура для хранения векторов, считанных из файла
- class FileHandler

  Класс для операций чтения/записи файлов

4.2.1 Подробное описание

Заголовочный файл класса FileHandler для работы с файлами

Автор

Филимонов А.А.

Версия

1.0

Дата

15.12.2025

Авторство

ИБСТ ПГУ

См. определение в файле file_handler.h

4.3 file_handler.h

См. документацию.

```
00001
00010 #pragma once
00011 #include <vector>
00012 #include <string>
00013 #include <cstdint>
00014
00019 struct VectorData {
00020     std::vector<std::vector<int16_t» vectors;
00021 };
00022
00027 class FileHandler {
00028 public:
00035     static VectorData readVectors(const std::string& filename);
00036
00043     static void writeResults(const std::string& filename, const std::vector<int16_t>& results);
00044
00051     static std::pair<std::string, std::string> readCredentials(const std::string& filename);
00052 };
```

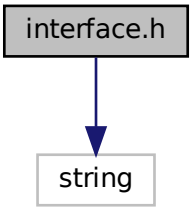
4.4 interface.cpp

```
00001 #include "interface.h"
00002 #include <iostream>
00003 #include <unistd.h>
00004 #include <cstdlib>
00005 #include <cstring>
00006
00007 bool Interface::parseCommandLine(int argc, char* argv[], Params& params) {
00008     // Установка значений по умолчанию
00009     params.server_port = 33333;
00010     params.config_file = "/.config/vclient.conf";
00011
00012     // Важно: сбросить optind для повторного использования getopt
00013     optind = 1;
00014     opterr = 1;  // Включаем вывод ошибок getopt
00015
00016     int opt;
00017     while ((opt = getopt(argc, argv, "a:p:i:o:c:h")) != -1) {
00018         switch (opt) {
00019             case 'a': // Адрес сервера
00020                 params.server_address = optarg;
00021                 break;
00022             case 'p': // Порт сервера
00023                 params.server_port = std::stoi(optarg);
00024                 break;
00025             case 'i': // Входной файл
00026                 params.input_file = optarg;
00027                 break;
00028             case 'o': // Выходной файл
00029                 params.output_file = optarg;
00030                 break;
00031             case 'c': // Конфигурационный файл
00032                 params.config_file = optarg;
00033                 break;
00034             case 'h': // Справка
00035                 showHelp();
00036                 return false;
00037             case '?': // Неизвестный параметр
00038                 // getopt уже вывел сообщение об ошибке
00039                 showHelp();
00040                 return false;
00041         }
00042     }
00043
00044     // Проверка обязательных параметров
00045     if (params.server_address.empty() || params.input_file.empty() || params.output_file.empty()) {
00046         std::cerr « "Ошибка: обязательные параметры не указаны\n";
00047         showHelp();
00048         return false;
00049     }
00050
00051     return true;
00052 }
00053
00054 void Interface::showHelp() {
00055     std::cout « "Использование: client [OPTIONS]\n";
00056     std::cout « "Обязательные параметры:\n";
00057     std::cout « "  -a ADDRESS    Адрес сервера\n";
00058     std::cout « "  -i FILE       Входной файл с векторами\n";
00059     std::cout « "  -o FILE       Выходной файл для результатов\n";
00060     std::cout « "Опциональные параметры:\n";
00061     std::cout « "  -p PORT       Порт сервера (по умолчанию: 33333)\n";
00062     std::cout « "  -c FILE       Файл конфигурации (по умолчанию: /.config/vclient.conf)\n";
00063     std::cout « "  -h            Показать эту справку\n";
00064     std::cout « "\nПример:\n";
00065     std::cout « "  ./client -a 127.0.0.1 -i vectors.bin -o results.bin -p 33333\n";
00066 }
```

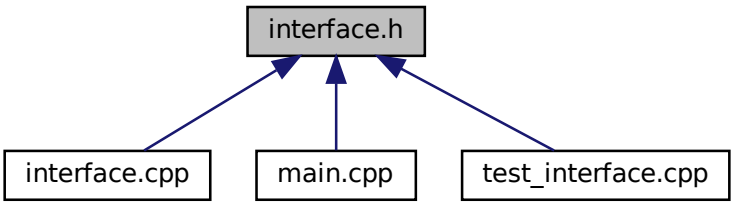
4.5 Файл interface.h

Заголовочный файл для обработки параметров командной строки

#include <string>
Граф включаемых заголовочных файлов для interface.h:



Граф файлов, в которые включается этот файл:



Классы

- struct Params

  Структура для хранения параметров командной строки программы
- class Interface

  Класс для разбора командной строки и отображения справки

4.5.1 Подробное описание

Заголовочный файл для обработки параметров командной строки

Автор

Филимонов А.А.

Версия

1.0

Дата

15.12.2025

Авторство

ИБСТ ПГУ

См. определение в файле interface.h

4.6 interface.h

См. документацию.

```
00001
00010 #pragma once
00011 #include <string>
00012
00017 struct Params {
00018     std::string server_address;
00019     int server_port;
00020     std::string input_file;
00021     std::string output_file;
00022     std::string config_file;
00023 };
00024
00029 class Interface {
00030 public:
00038     static bool parseCommandLine(int argc, char* argv[], Params& params);
00039
00043     static void showHelp();
00044 };
```

4.7 linux_authenticator.cpp

```
00001 #include "linux_authenticator.h"
00002 #include <linux/if_alg.h>
00003 #include <sys/socket.h>
00004 #include <unistd.h>
00005 #include <stdexcept>
00006 #include <cstring>
00007 #include <iomanip>
00008 #include <sstream>
00009 #include <iostream>
00010 #include <algorithm>
00011 #include <cctype>
00012
00013 int LinuxAuthenticator::createCryptoSocket(const std::string& algorithm) {
00014     // Создание криптосокета
00015     int sock = socket(AF_ALG, SOCK_SEQPACKET, 0);
00016     if (sock == -1) {
00017         throw std::runtime_error("Ошибка создания криптосокета");
00018     }
00019
00020     // Заполнение структуры для хеш-функции
00021     struct sockaddr_alg sa = {
00022         .salg_family = AF_ALG,
00023         .salg_type = "hash",
00024     };
00025
00026     // копирование имени алгоритма
00027     std::strncpy(reinterpret_cast<char*>(sa.salg_name), algorithm.c_str(), sizeof(sa.salg_name) - 1);
00028     sa.salg_name[sizeof(sa.salg_name) - 1] = '\0';
00029
00030     // Привязка сокета к алгоритму
00031     if (bind(sock, (struct sockaddr*)&sa, sizeof(sa)) == -1) {
00032         close(sock);
00033         throw std::runtime_error("Ошибка привязки криптосокета к алгоритму: " + algorithm);
00034     }
00035
00036     return sock;
00037 }
00038
```

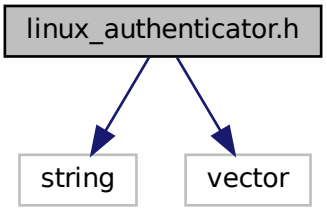
```
00039 std::vector<uint8_t> LinuxAuthenticator::computeHash(int crypto_sock, const std::string& data) {
00040     // Принятие соединения для криптоопераций
00041     int op_sock = accept(crypto_sock, nullptr, nullptr);
00042     if (op_sock == -1) {
00043         throw std::runtime_error("Ошибка принятия криптосоединения");
00044     }
00045 
00046     try {
00047         // Передача данных для вычисления хеша
00048         ssize_t sent = send(op_sock, data.c_str(), data.length(), 0);
00049         if (sent != static_cast<ssize_t>(data.length())) {
00050             close(op_sock);
00051             throw std::runtime_error("Ошибка передачи данных для хеширования");
00052         }
00053 
00054         // Получение результата
00055         std::vector<uint8_t> hash(28); // SHA-224 produces 28 bytes
00056         ssize_t received = recv(op_sock, hash.data(), hash.size(), 0);
00057         if (received != static_cast<ssize_t>(hash.size())) {
00058             close(op_sock);
00059             throw std::runtime_error("Ошибка получения хеша");
00060         }
00061 
00062         close(op_sock);
00063         return hash;
00064 
00065     } catch (...) {
00066         close(op_sock);
00067         throw;
00068     }
00069 }
00070 
00071 std::string LinuxAuthenticator::computeSHA224(const std::string& data) {
00072     int crypto_sock = -1;
00073     try {
00074         // Подготовка: создание криптосокета
00075         crypto_sock = createCryptoSocket("sha224");
00076 
00077         // Вычисление: передача данных и получение хеша
00078         std::vector<uint8_t> hash = computeHash(crypto_sock, data);
00079 
00080         // Преобразование в hex-строку
00081         std::stringstream ss;
00082         for (uint8_t byte : hash) {
00083             ss « std::hex « std::setw(2) « std::setfill('0') « static_cast<int>(byte);
00084         }
00085 
00086         close(crypto_sock);
00087         return ss.str();
00088 
00089     } catch (...) {
00090         if (crypto_sock != -1) {
00091             close(crypto_sock);
00092         }
00093         throw;
00094     }
00095 }
00096 
00097 std::string LinuxAuthenticator::computeSHA224(const std::string& salt, const std::string& password) {
00098     // Конкатенация соли и пароля
00099     std::string data = salt + password;
00100     std::cout « "Вычисление SHA-224 для соли + пароля..." « std::endl;
00101     std::string hash = computeSHA224(data);
00102 
00103     return hash;
00104 }
```

4.8 Файл linux_authenticator.h

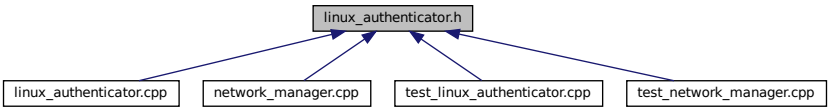
Заголовочный файл класса LinuxAuthenticator для работы с криптографией через AF_ALG.

```
#include <string>
#include <vector>
```

Граф включаемых заголовочных файлов для linux_authenticator.h:



Граф файлов, в которые включается этот файл:



Классы

- class LinuxAuthenticator

  Класс для вычисления хеш-сумм с использованием Linux Crypto API через сокеты AF_ALG.

4.8.1 Подробное описание

Заголовочный файл класса LinuxAuthenticator для работы с криптографией через AF_ALG.

Автор

Филимонов А.А.

Версия

1.0

Дата

15.12.2025

Авторство

ИБСТ ПГУ

См. определение в файле linux_authenticator.h

4.9 linux_authenticator.h

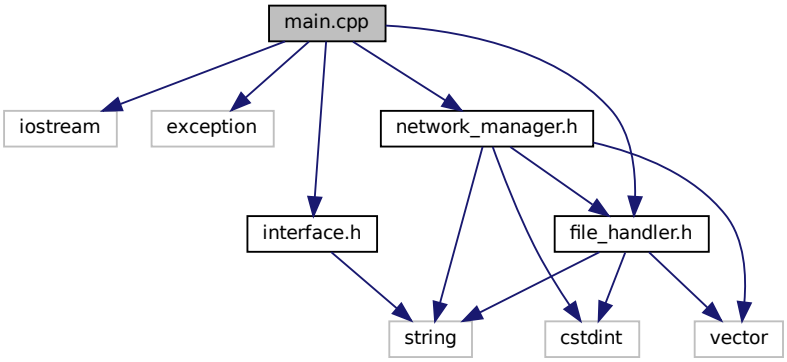
См. документацию.

```
00001
00010 #pragma once
00011 #include <string>
00012 #include <vector>
00013
00019 class LinuxAuthenticator {
00020 public:
00027     static std::string computeSHA224(const std::string& data);
00028
00036     static std::string computeSHA224(const std::string& salt, const std::string& password);
00037
00038 private:
00045     static int createCryptoSocket(const std::string& algorithm);
00046
00054     static std::vector<uint8_t> computeHash(int crypto_sock, const std::string& data);
00055 };
```

4.10 Файл main.cpp

Главный модуль сетевого клиента

#include <iostream>
#include <exception>
#include "interface.h"
#include "file_handler.h"
#include "network_manager.h"
Граф включаемых заголовочных файлов для main.cpp:



Функции

- int main (int argc, char *argv[])

  Точка входа в программу

4.10.1 Подробное описание

Главный модуль сетевого клиента

Автор

Филимонов А.А.

Версия

1.0

Дата

15.12.2025

Авторство

ИБСТ ПГУ

Программа подключается к серверу, выполняет аутентификацию, отправляет векторы данных и сохраняет результаты вычислений

См. определение в файле main.cpp

4.10.2 Функции

4.10.2.1 main()

```
int main (
                int argc,
                char * argv[] )
```

Точка входа в программу

Аргументы

argc	Количество аргументов командной строки
argv	Массив аргументов командной строки

Возвращает

0 - успешное завершение, 1 - ошибка выполнения

Основной алгоритм работы программы:

1. Разбор параметров командной строки
2. Чтение учетных данных и векторов из файлов
3. Подключение к серверу и аутентификация
4. Отправка векторов и получение результатов
5. Сохранение результатов в файл

См. определение в файле main.cpp строка 30

4.11 main.cpp

См. документацию.

```
00001
00012 #include <iostream>
00013 #include <exception>
00014 #include "interface.h"
00015 #include "file_handler.h"
00016 #include "network_manager.h"
00017
00030 int main(int argc, char* argv[]) {
00031     try {
00032         Params params;
00033
00034         // Разбор командной строки
00035         if (!Interface::parseCommandLine(argc, argv, params)) {
00036             return 1;
00037         }
00038
00039         std::cout « "=== Сетевой клиент ===" « std::endl;
00040         std::cout « "Сервер: " « params.server_address « ":" « params.server_port « std::endl;
00041         std::cout « "Входной файл: " « params.input_file « std::endl;
00042         std::cout « "Выходной файл: " « params.output_file « std::endl;
00043
00044         // Чтение учетных данных
00045         auto [login, password] = FileHandler::readCredentials(params.config_file);
00046
00047         // Чтение векторов
00048         VectorData vectors = FileHandler::readVectors(params.input_file);
00049
00050         // Сетевое взаимодействие
00051         NetworkManager netManager(params.server_address, params.server_port);
00052         netManager.connect();
00053
00054         if (!netManager.authenticate(login, password)) {
00055             std::cerr « "Ошибка аутентификации" « std::endl;
00056             return 1;
00057         }
00058
00059         std::vector<int16_t> results = netManager.sendVectors(vectors);
00060         netManager.disconnect();
00061
00062         // Сохранение результатов
00063         FileHandler::writeResults(params.output_file, results);
00064
00065         std::cout « "============================================" « std::endl;
00066         std::cout « "Программа успешно завершена!" « std::endl;
00067
00068     } catch (const std::exception& e) {
00069         std::cerr « "Ошибка: " « e.what() « std::endl;
00070         return 1;
00071     }
00072
00073     return 0;
00074 }
```

4.12 network_manager.cpp

```
00001 #include "network_manager.h"
00002 #include "linux_authenticator.h"
00003 #include <sys/socket.h>
00004 #include <netinet/in.h>
```

```
00005 #include <arpa/inet.h>
00006 #include <unistd.h>
00007 #include <stdexcept>
00008 #include <cstring>
00009 #include <iostream>
00010 #include <algorithm>
00011 #include <cctype>
00012
00013 NetworkManager::NetworkManager(const std::string& addr, int p)
00014     : sockfd(-1), address(addr), port(p) {}
00015
00016 NetworkManager::~NetworkManager() {
00017     disconnect();
00018 }
00019
00020 bool NetworkManager::connect() {
00021     sockfd = socket(AF_INET, SOCK_STREAM, 0);
00022     if (sockfd == -1) {
00023         throw std::runtime_error("Ошибка создания сокета");
00024     }
00025
00026     sockaddr_in serv_addr{};
00027     serv_addr.sin_family = AF_INET;
00028     serv_addr.sin_port = htons(port);
00029
00030     if (inet_pton(AF_INET, address.c_str(), &serv_addr.sin_addr) <= 0) {
00031         throw std::runtime_error("Неверный адрес: " + address);
00032     }
00033
00034     std::cout « "Подключение к " « address « ":" « port « "..." « std::endl;
00035     if (::connect(sockfd, (sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
00036         throw std::runtime_error("Ошибка подключения к " + address + ":" + std::to_string(port));
00037     }
00038
00039     std::cout « "Подключение установлено успешно" « std::endl;
00040     return true;
00041 }
00042
00043 void NetworkManager::disconnect() {
00044     if (sockfd != -1) {
00045         close(sockfd);
00046         sockfd = -1;
00047         std::cout « "Соединение закрыто" « std::endl;
00048     }
00049 }
00050
00051 void NetworkManager::sendData(const void* data, size_t size) {
00052     ssize_t bytes_sent = send(sockfd, data, size, 0);
00053     if (bytes_sent != static_cast<ssize_t>(size)) {
00054         throw std::runtime_error("Ошибка отправки данных");
00055     }
00056 }
00057
00058 void NetworkManager::receiveData(void* data, size_t size) {
00059     ssize_t bytes_received = recv(sockfd, data, size, 0);
00060     if (bytes_received != static_cast<ssize_t>(size)) {
00061         throw std::runtime_error("Ошибка приема данных");
00062     }
00063 }
00064
00065 std::string NetworkManager::receiveString() {
00066     char buffer[1024];
00067     ssize_t bytes_received = recv(sockfd, buffer, sizeof(buffer) - 1, 0);
00068     if (bytes_received <= 0) {
00069         throw std::runtime_error("Ошибка приема строки");
00070     }
00071     buffer[bytes_received] = '\0';
00072     return std::string(buffer);
00073 }
00074
00075 void NetworkManager::sendString(const std::string& str) {
00076     sendData(str.c_str(), str.length());
00077 }
00078
00079 bool NetworkManager::authenticate(const std::string& login, const std::string& password) {
00080     try {
00081         //Отправка логина
00082         sendString(login);
00083         std::cout « "Отправлен логин: " « login « std::endl;
00084
00085         //Получение соли от сервера или ERR
00086         std::string salt_response = receiveString();
00087         std::cout « "Получена соль от сервера " «std::endl;
00088
00089         if (salt_response == "ERR") {
00090             std::cerr « "Сервер вернул ошибку идентификации" « std::endl;
00091             return false;
```

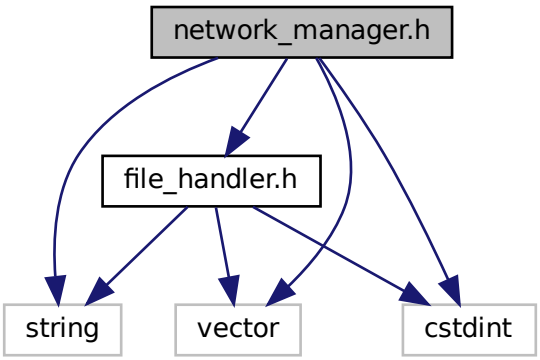
```
00092         }
00093
00094         //Вычисление хеша SHA-224(SALT16 || PASSWORD)
00095         std::string hash = LinuxAuthenticator::computeSHA224(salt_response, password);
00096
00097         // ПРЕОБРАЗОВАНИЕ В ВЕРХНИЙ РЕГИСТР
00098         std::transform(hash.begin(), hash.end(), hash.begin(),
00099                        [](unsigned char c) { return std::toupper(c); });
00100
00101         sendString(hash);
00102         std::cout « "Отправлен хеш SHA-224 " « std::endl;
00103
00104         // Получение результата аутентификации
00105         std::string auth_response = receiveString();
00106         std::cout « "Результат аутентификации: " « auth_response « std::endl;
00107
00108         return auth_response == "OK";
00109
00110     } catch (const std::exception& e) {
00111         throw std::runtime_error("Ошибка аутентификации: " + std::string(e.what()));
00112     }
00113 }
00114
00115 std::vector<int16_t> NetworkManager::sendVectors(const VectorData& vectors) {
00116     try {
00117         std::vector<int16_t> results;
00118
00119         // Отправка количества векторов (uint32_t)
00120         uint32_t num_vectors = vectors.vectors.size();
00121         sendData(&num_vectors, sizeof(num_vectors));
00122         std::cout « "Отправлено векторов: " « num_vectors « std::endl;
00123
00124         // Отправка каждого вектора и получение результатов
00125         for (size_t i = 0; i < vectors.vectors.size(); ++i) {
00126             const auto& vector = vectors.vectors[i];
00127
00128             // Отправка размера вектора (uint32_t)
00129             uint32_t vector_size = vector.size();
00130             sendData(&vector_size, sizeof(vector_size));
00131
00132             // Отправка значений вектора (int16_t)
00133             sendData(vector.data(), vector_size * sizeof(int16_t));
00134
00135             // Получение результата вычислений (int16_t)
00136             int16_t result;
00137             receiveData(&result, sizeof(result));
00138             results.push_back(result);
00139
00140             std::cout « "  Вектор " « i + 1 « " (размер " « vector_size « ") -> результат: " « result « std::endl;
00141         }
00142
00143         std::cout « "Все векторы успешно обработаны" « std::endl;
00144         return results;
00145
00146     } catch (const std::exception& e) {
00147         throw std::runtime_error("Ошибка отправки векторов: " + std::string(e.what()));
00148     }
00149 }
```

4.13 Файл network_manager.h

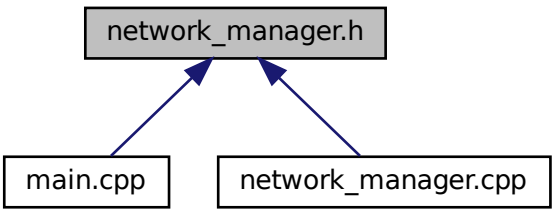
Заголовочный файл класса NetworkManager для сетевого взаимодействия

```
#include <string>
#include <vector>
#include <cstdint>
#include "file_handler.h"
```

Граф включаемых заголовочных файлов для network_manager.h:



Граф файлов, в которые включается этот файл:



Классы

- class NetworkManager

  Класс для управления сетевым соединением с сервером

4.13.1 Подробное описание

Заголовочный файл класса NetworkManager для сетевого взаимодействия

Автор

Филимонов А.А.

Версия

1.0

Дата

15.12.2025

Авторство

ИБСТ ПГУ

См. определение в файле network_manager.h

4.14 network_manager.h

См. документацию.

```
00001
00010 #pragma once
00011 #include <string>
00012 #include <vector>
00013 #include <cstdint>
00014 #include "file_handler.h"
00015
00021 class NetworkManager {
00022 private:
00023     int sockfd;
00024     std::string address;
00025     int port;
00033     void sendData(const void* data, size_t size);
00034
00041     void receiveData(void* data, size_t size);
00042
00048     std::string receiveString();
00049
00055     void sendString(const std::string& str);
00056
00057 public:
00063     NetworkManager(const std::string& addr, int p);
00064
00069     ~NetworkManager();
00070
00076     bool connect();
00077
00081     void disconnect();
00082
00090     bool authenticate(const std::string& login, const std::string& password);
00091
00098     std::vector<int16_t> sendVectors(const VectorData& vectors);
00099 };
```

4.15 test_file_handler.cpp

```
00001 #include <UnitTest++/UnitTest++.h>
00002 #include <fstream>
00003 #include <cstdio>
00004 #include "file_handler.h"
00005
00006 int main()
00007 {
00008     return UnitTest::RunAllTests();
00009 }
00010
00011 namespace {
00012     bool removeFile(const std::string& filename) {
00013         return std::remove(filename.c_str()) == 0;
00014     }
00015 }
00016
```

```
00017 SUITE(FileHandlerTests) {
00018     TEST(POSITIVE_2_1_ReadVectors) {
00019         std::string filename = "test_input.bin";
00020         std::ofstream file(filename, std::ios::binary);
00021         uint32_t num_vectors = 2;
00022         file.write(reinterpret_cast<const char*>(&num_vectors), sizeof(num_vectors));
00023         uint32_t size1 = 3; std::vector<int16_t> vec1 = {1, 2, 3};
00024         file.write(reinterpret_cast<const char*>(&size1), sizeof(size1));
00025         file.write(reinterpret_cast<const char*>(vec1.data()), size1 * sizeof(int16_t));
00026         uint32_t size2 = 2; std::vector<int16_t> vec2 = {10, 20};
00027         file.write(reinterpret_cast<const char*>(&size2), sizeof(size2));
00028         file.write(reinterpret_cast<const char*>(vec2.data()), size2 * sizeof(int16_t));
00029         file.close();
00030 
00031         VectorData data = FileHandler::readVectors(filename);
00032         CHECK_EQUAL(2u, data.vectors.size());
00033         CHECK_EQUAL(3u, data.vectors[0].size());
00034         CHECK_EQUAL(1, data.vectors[0][0]);
00035         CHECK_EQUAL(2, data.vectors[0][1]);
00036         CHECK_EQUAL(3, data.vectors[0][2]);
00037         CHECK_EQUAL(10, data.vectors[1][0]);
00038         CHECK_EQUAL(20, data.vectors[1][1]);
00039 
00040         removeFile(filename);
00041     }
00042 
00043     TEST(POSITIVE_2_2_WriteResults) {
00044         std::string filename = "test_output.bin";
00045         std::vector<int16_t> results = {42, 123, -456};
00046         FileHandler::writeResults(filename, results);
00047 
00048         std::ifstream file(filename, std::ios::binary);
00049         uint32_t num;
00050         file.read(reinterpret_cast<char*>(&num), sizeof(num));
00051         CHECK_EQUAL(3u, num);
00052         std::vector<int16_t> read_results(num);
00053         file.read(reinterpret_cast<char*>(read_results.data()), num * sizeof(int16_t));
00054         CHECK_EQUAL(42, read_results[0]);
00055         CHECK_EQUAL(123, read_results[1]);
00056         CHECK_EQUAL(-456, read_results[2]);
00057 
00058         removeFile(filename);
00059     }
00060 
00061     TEST(POSITIVE_2_3_WriteEmptyResults) {
00062         std::string filename = "test_empty.bin";
00063         std::vector<int16_t> results;
00064         FileHandler::writeResults(filename, results);
00065 
00066         std::ifstream file(filename, std::ios::binary);
00067         uint32_t num;
00068         file.read(reinterpret_cast<char*>(&num), sizeof(num));
00069         CHECK_EQUAL(0u, num);
00070 
00071         removeFile(filename);
00072     }
00073 
00074     TEST(POSITIVE_2_4_ReadCredentials) {
00075         std::string filename = "test_config.conf";
00076         std::ofstream file(filename);
00077         file << "testuser:testpass";
00078         file.close();
00079 
00080         auto [login, password] = FileHandler::readCredentials(filename);
00081         CHECK_EQUAL("testuser", login);
00082         CHECK_EQUAL("testpass", password);
00083 
00084         removeFile(filename);
00085     }
00086 
00087     TEST(NEGATIVE_2_6_NonExistentFile) {
00088         CHECK_THROW(FileHandler::readVectors("nonexistent.bin"), std::runtime_error);
00089     }
00090 
00091     TEST(NEGATIVE_2_7_NoColon) {
00092         std::string filename = "bad_config.conf";
00093         std::ofstream file(filename);
00094         file << "testuser testpass";
00095         file.close();
00096         CHECK_THROW(FileHandler::readCredentials(filename), std::runtime_error);
00097         removeFile(filename);
00098     }
00099 
00100     TEST(NEGATIVE_2_8_EmptyFile) {
00101         std::string filename = "empty.conf";
00102         std::ofstream file(filename);
00103         file.close();
```

```
00104         CHECK_THROW(FileHandler::readCredentials(filename), std::runtime_error);
00105         removeFile(filename);
00106     }
00107
00108     TEST(NEGATIVE_2_9_OnlyLogin) {
00109         std::string filename = "only_login.conf";
00110         std::ofstream file(filename);
00111         file « "login:";
00112         file.close();
00113         auto [login, password] = FileHandler::readCredentials(filename);
00114         CHECK_EQUAL("login", login);
00115         CHECK_EQUAL("", password);
00116         removeFile(filename);
00117     }
00118
00119     TEST(NEGATIVE_2_10_OnlyPassword) {
00120         std::string filename = "only_pass.conf";
00121         std::ofstream file(filename);
00122         file « ":password";
00123         file.close();
00124         auto [login, password] = FileHandler::readCredentials(filename);
00125         CHECK_EQUAL("", login);
00126         CHECK_EQUAL("password", password);
00127         removeFile(filename);
00128     }
00129 }
```

4.16 test_interface.cpp

```
00001 #include <UnitTest++/UnitTest++.h>
00002 #include <iostream>
00003 #include <cstring>
00004 #include "interface.h"
00005
00006 int main() {
00007     return UnitTest::RunAllTests();
00008 }
00009
00010 char* strdup(const char* s) {
00011     size_t len = strlen(s) + 1;
00012     char* copy = new char[len];
00013     std::strcpy(copy, s);
00014     return copy;
00015 }
00016
00017 SUITE(InterfaceTests) {
00018     TEST(POSITIVE_HelpFlag) {
00019         char* arg0 = strdup("./client");
00020         char* arg1 = strdup("-h");
00021         char* argv[] = {arg0, arg1, nullptr};
00022         int argc = 2;
00023
00024         Params params;
00025         bool result = Interface::parseCommandLine(argc, argv, params);
00026
00027         CHECK(!result);
00028
00029         delete[] arg0;
00030         delete[] arg1;
00031     }
00032
00033     TEST(NEGATIVE_MissingRequiredParams) {
00034         char* arg0 = strdup("./client");
00035         char* arg1 = strdup("-a");
00036         char* arg2 = strdup("127.0.0.1");
00037         char* argv[] = {arg0, arg1, arg2, nullptr};
00038         int argc = 3;
00039
00040         Params params;
00041         bool result = Interface::parseCommandLine(argc, argv, params);
00042
00043         CHECK(!result);
00044
00045         delete[] arg0;
00046         delete[] arg1;
00047         delete[] arg2;
00048     }
00049
00050     TEST(NEGATIVE_InvalidOption) {
00051         char* arg0 = strdup("./client");
00052         char* arg1 = strdup("-x");
00053         char* arg2 = strdup("value");
00054         char* argv[] = {arg0, arg1, arg2, nullptr};
```

```
00055         int argc = 3;
00056
00057         Params params;
00058         bool result = Interface::parseCommandLine(argc, argv, params);
00059
00060         CHECK(!result);
00061
00062         delete[] arg0;
00063         delete[] arg1;
00064         delete[] arg2;
00065     }
00066 }
```

4.17 test_linux_authenticator.cpp

```
00001 #include <UnitTest++/UnitTest++.h>
00002 #include "linux_authenticator.h"
00003
00004 int main()
00005 {
00006     return UnitTest::RunAllTests();
00007 }
00008
00009 SUITE(LinuxAuthenticatorTests) {
00010     TEST(POSITIVE_3_1_ComputeSHA224Data) {
00011         std::string data = "test";
00012         std::string hash = LinuxAuthenticator::computeSHA224(data);
00013         CHECK_EQUAL(56u, hash.length());
00014         for (char c : hash) {
00015             CHECK((c >= '0' && c <= '9') || (c >= 'a' && c <= 'f') || (c >= 'A' && c <= 'F'));
00016         }
00017     }
00018
00019     TEST(POSITIVE_3_4_SameInputSameHash) {
00020         std::string salt = "1234567890abcdef";
00021         std::string pass = "password";
00022         std::string hash1 = LinuxAuthenticator::computeSHA224(salt, pass);
00023         std::string hash2 = LinuxAuthenticator::computeSHA224(salt, pass);
00024         CHECK_EQUAL(hash1, hash2);
00025     }
00026
00027     TEST(POSITIVE_3_5_DifferentPasswords) {
00028         std::string salt = "1234567890abcdef";
00029         std::string hash1 = LinuxAuthenticator::computeSHA224(salt, "pass1");
00030         std::string hash2 = LinuxAuthenticator::computeSHA224(salt, "pass2");
00031         CHECK(hash1 != hash2);
00032     }
00033
00034     TEST(POSITIVE_3_6_DifferentSalts) {
00035         std::string pass = "password";
00036         std::string hash1 = LinuxAuthenticator::computeSHA224("salt1", pass);
00037         std::string hash2 = LinuxAuthenticator::computeSHA224("salt2", pass);
00038         CHECK(hash1 != hash2);
00039     }
00040
00041     TEST(POSITIVE_3_7_HashLength) {
00042         std::string salt = "1234567890abcdef";
00043         std::string pass = "password";
00044         std::string hash = LinuxAuthenticator::computeSHA224(salt, pass);
00045         CHECK_EQUAL(56u, hash.length());
00046     }
00047
00048     TEST(POSITIVE_3_8_NonEmptyHash) {
00049         std::string data = "testdata";
00050         std::string hash = LinuxAuthenticator::computeSHA224(data);
00051         CHECK(!hash.empty());
00052     }
00053
00054     TEST(POSITIVE_3_9_SpecialChars) {
00055         std::string salt = "salt@#";
00056         std::string pass = "p@ssw#rd!";
00057         std::string hash = LinuxAuthenticator::computeSHA224(salt, pass);
00058         CHECK(!hash.empty());
00059         CHECK_EQUAL(56u, hash.length());
00060     }
00061
00062     TEST(POSITIVE_3_2_HexOnly) {
00063         std::string salt = "0123456789abcdef";
00064         std::string pass = "pass";
00065         std::string hash = LinuxAuthenticator::computeSHA224(salt, pass);
00066         for (char c : hash) {
00067             CHECK((c >= '0' && c <= '9') || (c >= 'a' && c <= 'f'));
00068         }
```

```
00069     }
00070 }
```

4.18 test_network_manager.cpp

```
00001 #include <UnitTest++/UnitTest++.h>
00002 #include <algorithm>
00003 #include "linux_authenticator.h"
00004
00005 int main()
00006 {
00007     return UnitTest::RunAllTests();
00008 }
00009
00010 class MockNetworkManager {
00011 public:
00012     MockNetworkManager(const std::string& addr, int p) {}
00013
00014     bool connect() {
00015         return true;
00016     }
00017
00018     bool authenticate(const std::string& login, const std::string& password) {
00019         std::string salt = "1234567890abcdef";
00020         std::string hash = LinuxAuthenticator::computeSHA224(salt, password);
00021         std::transform(hash.begin(), hash.end(), hash.begin(), ::toupper);
00022         return true;
00023     }
00024
00025     std::vector<int16_t> sendVectors(const std::vector<std::vector<int16_t»& vectors) {
00026         std::vector<int16_t> results;
00027         for (const auto& vec : vectors) {
00028             results.push_back(static_cast<int16_t>(vec.size() * 10));
00029         }
00030         return results;
00031     }
00032 };
00033
00034 SUITE(NetworkManagerTests) {
00035     TEST(POSITIVE_4_1_Connect) {
00036         MockNetworkManager net("127.0.0.1", 33333);
00037         CHECK(net.connect());
00038     }
00039
00040     TEST(POSITIVE_4_2_Authenticate) {
00041         MockNetworkManager net("127.0.0.1", 33333);
00042         CHECK(net.authenticate("testuser", "testpass"));
00043     }
00044
00045     TEST(POSITIVE_4_3_SendVectors) {
00046         MockNetworkManager net("127.0.0.1", 33333);
00047         std::vector<std::vector<int16_t» vectors = {{1,2,3}, {10}};
00048         auto results = net.sendVectors(vectors);
00049         CHECK_EQUAL(2u, results.size());
00050         CHECK_EQUAL(30, results[0]);
00051         CHECK_EQUAL(10, results[1]);
00052     }
00053 }
```

Предметный указатель