Программа Клиента

Создано системой Doxygen 1.9.1

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

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 Client

Класс в котором находятся переменные и функция для подключения клиента к серверу

#include <Client.h>

Открытые члены

- int connection (string ip_adr, string portcon, string username, string password, string name_↩ original_file, string name_result_file)

  описания функции подключения клиента к серверу

Открытые атрибуты

- string name_original_file
- string name_result_file
- string name_auto_file
- string username
- string password
- string ip_adr
- int portcon

4.1.1 Подробное описание

Класс в котором находятся переменные и функция для подключения клиента к серверу

4.1.2 Методы

4.1.2.1 connection()

```
int Client::connection (
                string ip_adr,
                string portcon,
                string username,
                string password,
                string name_original_file,
                string name_result_file )
```

описания функции подключения клиента к серверу

подключение клиента к серверу и передача данных

Аргументы

ip_adr	айпи по которому устанавливается соединение
portcon	порт по которому устанавливается соединение
username	логин клиента
password	пароль клиента
name_original_file	файл с векторами для передачи серверу на обработку
name_result_file	файл для записи результатов обработки и отправки сервера клиенту

утсановка порта по умолчанию
```
*if(portcon == "") {
    portcon = "33333";
 }
```

Инициализация айпи
```
*char* ip_chars = new char[ip_adr.length()];
*ip_adr.copy(ip_chars, ip_adr.length());
*ip_chars[ip_adr.length()] = '\0';
```

Инициализируем переменную для порта
```
*unsigned int portik;
*portik = stoi(portcon);
```

Создаём структуру с введённым адресом и портом
```
*sockaddr_in * selfAddr = new (sockaddr_in);
*selfAddr->sin_family = AF_INET; // интернет протокол IPv4
*selfAddr->sin_port = 0;
*selfAddr->sin_addr.s_addr = 0;


*sockaddr_in * remoteAddr = new (sockaddr_in);
*remoteAddr->sin_family = AF_INET;      // интернет протокол IPv4
*remoteAddr->sin_port = htons(portik);
*remoteAddr->sin_addr.s_addr = inet_addr(ip_chars); //  адрес
```

буфер для передачи и приема данных
```
*char *buffer = new char[4096];
*strcpy(buffer,username.c_str());
```

вычисляем длину строки
```
*int msgLen = strlen(buffer);
```

создаём сокет
```
*int mySocket = socket(AF_INET, SOCK_STREAM, 0); //tcp протокол
```

связываем сокет с адрессом
```
*int rc = bind(mySocket,(const sockaddr *) selfAddr, sizeof(sockaddr_in));
```

установливаем соединение

```
*rc = connect(mySocket, (const sockaddr*) remoteAddr, sizeof(sockaddr_in));
```

Проверка соединения клиента с сервером

```
*if (rc == -1) {
*cout « "Error: failed connect to server.\n";
*close(mySocket);
*return 1;
 }
*else cout«"Connection is succesful"«endl;
```

Отправка логина серверу

```
*rc = send(mySocket, buffer, msgLen, 0);
*if(rc==-1)
 {
     throw error_client(string("Ошибка отправки логина"));
 }
*cout « "Мы отправляем логин: " « buffer « endl;
```

принимаем ответ в буффер

```
*rc = recv(mySocket, buffer, 4096, 0);
*if(rc==-1)
 {
     throw error_client(string("Ошибка получения ответа"));
 }
*string s1 = string(buffer);
*buffer[rc] = '\0'; // конец принятой строки

*cout « "Мы получаем соль: " « buffer « endl; // вывод полученного сообщения от сервера
```

Далее генерируется хэш и отправляется серверу

```
*string s2;
*string msg = s1 + password;
*s2 = MD5_hash(msg);

*strcpy(buffer,s2.c_str());
*rc = send(mySocket, buffer, s2.length(), 0);
*cout « "Мы отправляем хэш: " « buffer « endl;
```

Получене ответа об аутентификации

```
*rc = recv(mySocket, buffer, sizeof(buffer), 0);
*buffer[rc] = '\0'; // конец принятой строки
*if(rc==-1)
 {
     throw error_client(string("Ошибка получения ответа"));
 }
*cout « "Мы получаем ответ: " « buffer « endl; // вывод полученного сообщения от сервера
```

Открытие файла для записи результата

```
*ofstream fout(name_result_file, ios::binary | ios::out | ios::app);
*if(!fout)
 {
    throw error_client(string("Ошибка открытия файла"));
 }
*endcode
  Далее открывается файл с векторами для отправки
  @code
*uint32_t n;
*FILE *f; //описываем файловую переменную
*const char * c = name_original_file.c_str(); //конвертируем строку с названием файла с векторами в const char *
*f=fopen(c, "rb");
*if(f==NULL)
    {
        throw error_client(string("Ошибка открытия файла"));
    }
  endcode
*считываем из файла одуо целое число в переменную n которое является количеством векторов
  @code
*fread(&n, sizeof(uint32_t), 1, f);
```

отправка количества векторов на сервер

```
*rc = send(mySocket, &n, sizeof(n), 0);
```

cout << "Мы отправляем кол-во векторов: " << n << endl; // вывод полученного сообщения от сервера fout<<n<<endl; далее идёт отправка количество чисел, содержащиеся в векторе

```
*for(int i=0; i<n; i++)
 {
*uint32_t size;
*double d;
    fread(&size, sizeof(uint32_t), 1, f);
```

```
    uint32_t size1;
    size1=(4+size*sizeof(size));
    cout«"Размер  "«i+1«"-го  вектора  "«size1«" байт"«endl;
    send(mySocket, &size, sizeof(size), 0);//отправка размера вектора
```

далее отправляются сами векторы массивом на сервер

```
double array[size]={0};
cout«"Векторы в строке  ";
for(int j=0;j<size;j++)
{
   fread(&d, sizeof(double), 1, f);
   array[j]=d;                //заполнение вектора в массив
   cout«d«"   ";     //вывод вектора в строке
}
cout«endl;
send(mySocket, &array, sizeof(array), 0);
cout « "Мы отправляем сам вектор: " « array « endl; // вывод полученного сообщения от сервера
double sum = 0;
recv(mySocket, &sum, sizeof(sum), 0);
cout « "Мы получаем ответ: " « sum « endl; // вывод полученного сообщения от сервера
fout «sum« endl;
```

} После обмена файлами закрываются сокет и файлы

```
*close(mySocket);
  fout.close();
```

return 0;

Объявления и описания членов классов находятся в файлах:

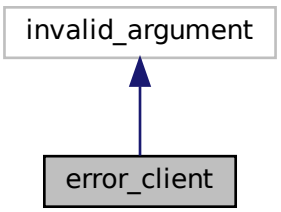
- Client.h
- Client.cpp

4.2 Класс error_client

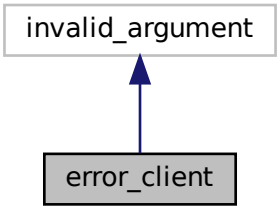
Класс для обработки ошибок возникающих при работе клиента

#include <Client.h>

Граф наследования:error_client:



Граф связей класса error_client:



Открытые члены

- error_client (const string &what_arg)
- error_client (const char *what_arg)

4.2.1 Подробное описание

Класс для обработки ошибок возникающих при работе клиента

Объявления и описания членов класса находятся в файле:

- Client.h

Глава 5

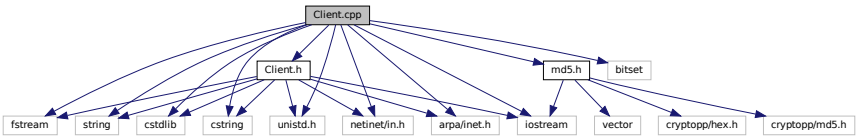
Файлы

5.1 Файл Client.cpp

Файл взаимодействие с сервером

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "md5.h"
#include "Client.h"
#include <bitset>
```

Граф включаемых заголовочных файлов для Client.cpp:



Функции

- void Exception (const string &why, const int exitCode)

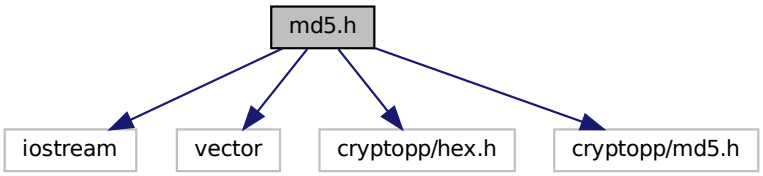
5.1.1 Подробное описание

Файл взаимодействие с сервером

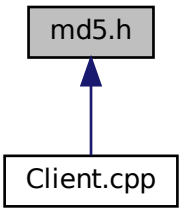
5.2 Файл md5.h

Заголовочный файл для модуля md5.

#include <iostream>
#include <vector>
#include <cryptopp/hex.h>
#include <cryptopp/md5.h>
Граф включаемых заголовочных файлов для md5.h:



Граф файлов, в которые включается этот файл:



Макросы

- #define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1

Функции

- string MD5_hash (string msg)

5.2.1 Подробное описание

Заголовочный файл для модуля md5.

Автор

Рогашевский А.В.

Версия

1.0

Дата

06.02.2023

Авторство

ИБСТ ПГУ

Предметный указатель