Федеральное агентство связи (Россвязь)

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

По дисциплине «Сетевое программирование»

Тема: «**Разработка сетевого приложения.** Клиент на базе протокола FTP в пассивном режиме»

Выполнил:

Студент группы ИС-242

Пухов Максим Станиславович

Проверил:

ассистент кафедры ВС

Невейко Артём Игоревич

Новосибирск, 2025

Оглавление

[Постановка задачи 2](#_Toc194097575)

[Описание протокола 2](#_Toc194097576)

[Протокол FTP в пассивном режиме 2](#_Toc194097577)

[Описание реализации 3](#_Toc194097578)

[Работа программы 6](#_Toc194097579)

[Листинг 6](#_Toc194097580)

[Список источников 14](#_Toc194097581)

# Постановка задачи

Необходимо написать клиент на базе протокола FTP в пассивном режиме. Реализация на языке C++ для платформы Linux.

# Описание протокола

### **Протокол FTP в пассивном режиме**

FTP (File Transfer Protocol) — это сетевой протокол, предназначенный для передачи файлов между клиентом и сервером. В **пассивном режиме** клиент и сервер используют два соединения:

1. **Управляющее соединение** (Control Connection)
   * Клиент инициирует подключение к серверу по порту **21** (по умолчанию).
   * Используется для отправки команд и получения ответов от сервера.
2. **Данные соединения** (Data Connection)

* Сервер открывает временный порт и сообщает его клиенту.
* Клиент подключается к этому порту для передачи файлов.

#### **Алгоритм работы FTP в пассивном режиме**

1. Клиент устанавливает **управляющее соединение** с сервером, открывая TCP-соединение на порт **21**.
2. Клиент отправляет команду **PASV**, запрашивая пассивный режим.
3. Сервер отвечает кодом **227**, указывая на **IP** и **порт** для подключения.
4. Клиент подключается к указанному порту для передачи **данных**.
5. Сервер передает **данные** (файлы, список директорий) через это соединение

**Основные команды FTP**

#### **1. Управление соединением**

* USER <username> — отправляет имя пользователя для аутентификации.
* PASS <password> — отправляет пароль для аутентификации.
* QUIT — завершает сеанс FTP и разрывает соединение.

#### **2. Управление режимами передачи**

* PASV — включает **пассивный режим**, при котором сервер сообщает клиенту порт для передачи данных.
* PORT <h1,h2,h3,h4,p1,p2> — включает **активный режим**, где клиент сообщает серверу IP-адрес и порт для передачи данных.
* TYPE <A | I> — устанавливает тип передачи:
  + A (ASCII) — текстовый режим.
  + I (Image/Binary) — двоичный режим.

#### **3. Работа с файлами**

* RETR <filename> — загружает (скачивает) файл с сервера.
* STOR <filename> — загружает (отправляет) файл на сервер.
* APPE <filename> — добавляет данные к существующему файлу на сервере.
* DELE <filename> — удаляет файл с сервера.
* RNFR <oldname> — указывает файл для переименования.
* RNTO <newname> — переименовывает файл.

#### **4. Работа с каталогами**

* PWD — показывает текущий каталог на сервере.
* CWD <directory> — меняет текущий каталог.
* CDUP — поднимается на уровень выше (аналог cd .. в терминале).
* MKD <directory> — создает новый каталог.
* RMD <directory> — удаляет каталог.

#### **5. Просмотр содержимого**

* LIST <directory> — выводит список файлов и папок в каталоге.
* NLST <directory> — показывает только имена файлов без детальной информации.

#### **6. Управление соединением и сессией**

* NOOP — проверяет активность соединения, отправляя пустую команду.
* ABOR — отменяет текущую передачу данных.
* STAT — получает информацию о состоянии соединения.
* SYST — запрашивает информацию о серверной ОС.

# Описание реализации

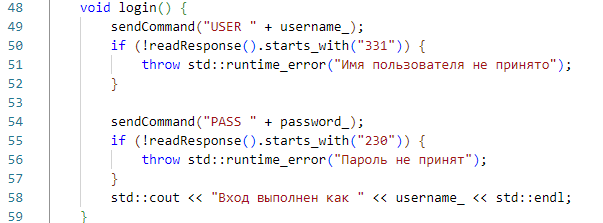
Программа представляет собой FTP-клиент, работающий в пассивном режиме. Она позволяет подключаться к FTP-серверу, авторизоваться, передавать файлы, изменять каталог и просматривать список файлов.

#### Реализация FTP-клиента

1. **Подключение к серверу**
   * Создаётся управляющее соединение (socket(AF\_INET, SOCK\_STREAM, 0)).
   * Осуществляется подключение (connect).
   * Проверяется ответ сервера (код 220).



1. **Аутентификация**
   * Отправляется команда USER с именем пользователя.
   * В случае успешного ответа (331) отправляется пароль (PASS).
   * Успешная аутентификация подтверждается кодом 230.



1. **Передача данных в пассивном режиме**
   * Клиент создает data-соединение к указанному сервером порту.
   * Клиент отправляет команду (например, RETR или LIST) через управляющее соединение.
   * Сервер начинает передачу данных через data-соединение.
   * Сервер закрывает data-соединение после завершения передачи.



1. **Скачивание файла**
   * Устанавливается бинарный режим (TYPE I).
   * Создаётся соединение данных через пассивный режим.
   * Отправляется команда RETR для загрузки файла.
   * После подтверждения (150) принимаются данные (recv) и записываются в файл.
   * Ожидается финальное подтверждение (226).



1. **Загрузка файла**
   * Устанавливается бинарный режим (TYPE I).
   * Открывается локальный файл (ifstream).
   * Создаётся соединение данных через пассивный режим.
   * Отправляется команда STOR для загрузки файла.
   * Файл передаётся серверу (send).
   * Ожидается подтверждение успешной передачи (226).



1. **Просмотр списка файлов**
   * Включается пассивный режим.
   * Отправляется команда LIST.
   * Полученные данные выводятся в консоль.



1. **Смена каталога и просмотр текущего каталога**
   * CWD изменяет текущий каталог.
   * PWD возвращает текущий путь.



# Работа программы

# 

# Листинг

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456 | #include <iostream>  #include <string>  #include <vector>  #include <sstream>  #include <fstream>  #include <stdexcept>  #include <cstring>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <unistd.h>  #include <string\_view>  class FTPClient {  public:      FTPClient(const std::string& host, int port, const std::string& username, const std::string& password)  // Инициализация параметров          : host\_(host), port\_(port), username\_(username), password\_(password), control\_sock\_(-1) {}      ~FTPClient() {          if (control\_sock\_ != -1) close(control\_sock\_); // Закрывает сокет при уничтожении объекта      }      void connectToServer() {          control\_sock\_ = socket(AF\_INET, SOCK\_STREAM, 0); // Создание TCP-сокета          if (control\_sock\_ == -1) {              throw std::runtime\_error("Не удалось создать сокет: " + std::string(strerror(errno)));          }          sockaddr\_in server\_addr{}; // Структура адреса сервера          server\_addr.sin\_family = AF\_INET; // Семейство адресов (IPv4)          server\_addr.sin\_port = htons(port\_); // Порт в сетевом порядке байт          if (inet\_pton(AF\_INET, host\_.c\_str(), &server\_addr.sin\_addr) <= 0) {              throw std::runtime\_error("Неверный адрес: " + host\_ + " (используйте '127.0.0.1' для локального хоста)"); // Преобразование IP          }          if (connect(control\_sock\_, (sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1) {              throw std::runtime\_error("Не удалось подключиться к FTP-серверу по адресу " + host\_ + ":" +                                      std::to\_string(port\_) + ": " + std::string(strerror(errno))); // Подключение          }          std::string response = readResponse(); // Чтение приветственного сообщения сервера          if (!response.starts\_with("220")) {              throw std::runtime\_error("Сервер не принял соединение: " + response);          }          std::cout << "Подключено к FTP-серверу: " << host\_ << ":" << port\_ << std::endl;      }      void login() {          sendCommand("USER " + username\_); // Отправка имени пользователя          if (!readResponse().starts\_with("331")) {              throw std::runtime\_error("Имя пользователя не принято");          }          sendCommand("PASS " + password\_); // Отправка пароля          if (!readResponse().starts\_with("230")) {              throw std::runtime\_error("Пароль не принят");          }          std::cout << "Вход выполнен как " << username\_ << std::endl;      }      bool enterPassiveMode(std::string& ip, int& port) {          sendCommand("PASV"); // Запрос пассивного режима          std::string response = readResponse();          if (!response.starts\_with("227")) {              std::cerr << "Пассивный режим не поддерживается: " << response << std::endl;              return false;          }          size\_t start = response.find('(');          size\_t end = response.find(')');          if (start == std::string::npos || end == std::string::npos) {              std::cerr << "Неверный формат ответа PASV" << std::endl;              return false;          }          std::vector<int> values;          std::stringstream ss(response.substr(start + 1, end - start - 1));          std::string token;          while (std::getline(ss, token, ',')) {              values.push\_back(std::stoi(token));          }          if (values.size() != 6) {              std::cerr << "Неверные данные PASV" << std::endl;              return false;          }          ip = std::to\_string(values[0]) + "." + std::to\_string(values[1]) + "." +              std::to\_string(values[2]) + "." + std::to\_string(values[3]);          port = values[4] \* 256 + values[5];          return true;      }      void downloadFile(const std::string& remotePath, const std::string& localPath) {          sendCommand("TYPE I"); // Установка бинарного режима          if (!readResponse().starts\_with("200")) {              throw std::runtime\_error("Не удалось установить бинарный режим");          }          std::string data\_ip;          int data\_port;          if (!enterPassiveMode(data\_ip, data\_port)) {              throw std::runtime\_error("Не удалось войти в пассивный режим");          }          int data\_sock = createDataSocket(data\_ip, data\_port);          sendCommand("RETR " + remotePath); // Запрос на скачивание          if (!readResponse().starts\_with("150")) {              close(data\_sock);              throw std::runtime\_error("Сервер отклонил команду RETR");          }          std::ofstream file(localPath, std::ios::binary);          if (!file) {              close(data\_sock);              throw std::runtime\_error("Не удалось открыть локальный файл: " + localPath);          }          char buffer[4096];          ssize\_t bytes\_received;          while ((bytes\_received = recv(data\_sock, buffer, sizeof(buffer), 0)) > 0) {              file.write(buffer, bytes\_received);          }          if (bytes\_received < 0) {              throw std::runtime\_error("Ошибка получения данных: " + std::string(strerror(errno)));          }          file.close();          close(data\_sock);          if (!readResponse().starts\_with("226")) {              throw std::runtime\_error("Передача файла не удалась");          }          std::cout << "Файл загружен: " << localPath << std::endl;      }      void uploadFile(const std::string& localPath, const std::string& remotePath) {          sendCommand("TYPE I");          if (!readResponse().starts\_with("200")) {              throw std::runtime\_error("Не удалось установить бинарный режим");          }          std::ifstream file(localPath, std::ios::binary);          if (!file) {              throw std::runtime\_error("Не удалось открыть локальный файл: " + localPath);          }          std::string data\_ip;          int data\_port;          if (!enterPassiveMode(data\_ip, data\_port)) {              throw std::runtime\_error("Не удалось войти в пассивный режим");          }          int data\_sock = createDataSocket(data\_ip, data\_port);          sendCommand("STOR " + remotePath);          if (!readResponse().starts\_with("150")) {              close(data\_sock);              throw std::runtime\_error("Сервер отклонил команду STOR");          }          char buffer[4096];          while (file.read(buffer, sizeof(buffer)) || file.gcount() > 0) {              if (send(data\_sock, buffer, file.gcount(), 0) < 0) {                  throw std::runtime\_error("Ошибка отправки данных: " + std::string(strerror(errno)));              }          }          file.close();          close(data\_sock);          if (!readResponse().starts\_with("226")) {              throw std::runtime\_error("Передача файла не удалась");          }          std::cout << "Файл загружен на сервер: " << remotePath << std::endl;      }      void listFiles() {          sendCommand("PASV");          std::string response = readResponse();          if (!response.starts\_with("227")) {              throw std::runtime\_error("Не удалось войти в пассивный режим");          }          std::string data\_ip;          int data\_port;          if (!enterPassiveMode(data\_ip, data\_port)) {              throw std::runtime\_error("Не удалось разобрать ответ PASV");          }          int data\_sock = createDataSocket(data\_ip, data\_port);          sendCommand("LIST");          if (!readResponse().starts\_with("150")) {              close(data\_sock);              throw std::runtime\_error("Сервер отклонил команду LIST");          }          char buffer[4096];          ssize\_t bytes\_received;          std::string file\_list;          while ((bytes\_received = recv(data\_sock, buffer, sizeof(buffer), 0)) > 0) {              file\_list.append(buffer, bytes\_received);          }          close(data\_sock);          if (!readResponse().starts\_with("226")) {              throw std::runtime\_error("Получение списка файлов не удалось");          }          std::cout << "Список файлов:\n" << file\_list << std::endl;      }      void createDirectory(const std::string& directory) {          sendCommand("MKD " + directory);          std::string response = readResponse();          if (!response.starts\_with("257")) {              throw std::runtime\_error("Не удалось создать папку: " + response);          }          std::cout << "Папка создана: " << directory << std::endl;      }      void changeDirectory(const std::string& directory) {          sendCommand("CWD " + directory);          std::string response = readResponse();          if (!response.starts\_with("250")) {              throw std::runtime\_error("Не удалось сменить каталог: " + response);          }          std::cout << "Текущий каталог изменен на: " << directory << std::endl;      }      std::string getCurrentDirectory() {          sendCommand("PWD");          std::string response = readResponse();          if (!response.starts\_with("257")) {              throw std::runtime\_error("Не удалось получить текущий каталог: " + response);          }          size\_t start = response.find('"');          size\_t end = response.rfind('"');          if (start != std::string::npos && end != std::string::npos && start != end) {              return response.substr(start + 1, end - start - 1);          }          return response;      }        void removeDirectory(const std::string& directory) {          sendCommand("RMD " + directory);          std::string response = readResponse();          if (!response.starts\_with("250")) {              throw std::runtime\_error("Не удалось удалить папку: " + response);          }          std::cout << "Папка удалена: " << directory << std::endl;      }      void deleteFile(const std::string& filePath) {          sendCommand("DELE " + filePath);          std::string response = readResponse();          if (!response.starts\_with("250")) {              throw std::runtime\_error("Не удалось удалить файл: " + response);          }          std::cout << "Файл удален: " << filePath << std::endl;      }      void renameFileOrDirectory(const std::string& oldName, const std::string& newName) {          sendCommand("RNFR " + oldName);          std::string response = readResponse();          if (!response.starts\_with("350")) {              throw std::runtime\_error("Не удалось выбрать файл/папку для переименования: " + response);          }          sendCommand("RNTO " + newName);          response = readResponse();          if (!response.starts\_with("250")) {              throw std::runtime\_error("Не удалось переименовать: " + response);          }          std::cout << "Успешно переименовано: " << oldName << " -> " << newName << std::endl;      }  private:      int control\_sock\_; // Сокет для управляющего соединения      std::string host\_; // Адрес сервера      int port\_; // Порт (обычно 21)      std::string username\_; // Имя пользователя      std::string password\_; // Пароль      void sendCommand(std::string\_view cmd) {          std::string command = std::string(cmd) + "\r\n"; // Добавление FTP-окончания          if (send(control\_sock\_, command.c\_str(), command.size(), 0) < 0) { // Отправка через сокет              throw std::runtime\_error("Не удалось отправить команду: " + std::string(strerror(errno)));          }      }      std::string readResponse() {   // Чтение данных из сокета, пока не встретится "\r\n"          std::string response;          char buffer[1024];          ssize\_t bytes\_received;          while ((bytes\_received = recv(control\_sock\_, buffer, sizeof(buffer) - 1, 0)) > 0) {              buffer[bytes\_received] = '\0';              response += buffer;              if (response.find("\r\n") != std::string::npos) break;          }          if (bytes\_received < 0) {              throw std::runtime\_error("Ошибка чтения ответа: " + std::string(strerror(errno)));          }          return response;      }      int createDataSocket(const std::string& ip, int port) {          int data\_sock = socket(AF\_INET, SOCK\_STREAM, 0);          if (data\_sock == -1) {              throw std::runtime\_error("Не удалось создать сокет данных: " + std::string(strerror(errno)));          }          sockaddr\_in data\_addr{};          data\_addr.sin\_family = AF\_INET;          data\_addr.sin\_port = htons(port);          inet\_pton(AF\_INET, ip.c\_str(), &data\_addr.sin\_addr);          if (connect(data\_sock, (sockaddr\*)&data\_addr, sizeof(data\_addr)) == -1) {              close(data\_sock);              throw std::runtime\_error("Не удалось подключить сокет данных: " + std::string(strerror(errno)));          }          return data\_sock;      }  };  int main() {      try {          std::string server;          int port = 21;          std::string username;          std::string password;            std::cout << "Введите адрес сервера: ";          std::cin >> server;          std::cout << "Введите имя пользователя: ";          std::cin >> username;          std::cout << "Введите пароль: ";          std::cin >> password;           // Ввод параметров подключения          FTPClient ftp(server, port, username, password);          ftp.connectToServer();          ftp.login();            while (true) {              std::cout << "\nМеню:" << std::endl;              std::cout << "1. Скачать файл" << std::endl;              std::cout << "2. Загрузить файл" << std::endl;              std::cout << "3. Показать список файлов" << std::endl;              std::cout << "4. Создать каталог" << std::endl;              std::cout << "5. Сменить каталог" << std::endl;              std::cout << "6. Показать текущий каталог" << std::endl;              std::cout << "7. Удалить каталог" << std::endl;              std::cout << "8. Удалить файл" << std::endl;              std::cout << "9. Переименовать файл или каталог" << std::endl;              std::cout << "q. Выйти" << std::endl;              std::cout << "Выберите действие: ";                std::string choice;              std::cin >> choice;                if (choice == "1") {                  std::string remoteFile, localFile;                  std::cout << "Введите имя удаленного файла: ";                  std::cin >> remoteFile;                  std::cout << "Введите имя локального файла: ";                  std::cin >> localFile;                  try {                      ftp.downloadFile(remoteFile, localFile);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "2") {                  std::string localFile, remoteFile;                  std::cout << "Введите имя локального файла: ";                  std::cin >> localFile;                  std::cout << "Введите имя удаленного файла: ";                  std::cin >> remoteFile;                  try{                      ftp.uploadFile(localFile, remoteFile);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "3") {                  try {                      ftp.listFiles();                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "4") {                  std::string directory;                  std::cout << "Введите имя каталога: ";                  std::cin >> directory;                  try {                      ftp.createDirectory(directory);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "5") {                  std::string directory;                  std::cout << "Введите путь к каталогу: ";                  std::cin >> directory;                  try {                      ftp.changeDirectory(directory);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "6") {                  try {                      std::cout << "Текущий каталог: " << ftp.getCurrentDirectory() << std::endl;                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "7") {                  std::string directory;                  std::cout << "Введите имя каталога для удаления: ";                  std::cin >> directory;                  try {                      ftp.removeDirectory(directory);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "8") {                  std::string filePath;                  std::cout << "Введите имя файла для удаления: ";                  std::cin >> filePath;                  try {                      ftp.deleteFile(filePath);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "9") {                  std::string oldName, newName;                  std::cout << "Введите текущее имя файла/папки: ";                  std::cin >> oldName;                  std::cout << "Введите новое имя: ";                  std::cin >> newName;                  try {                      ftp.renameFileOrDirectory(oldName, newName);                  } catch (const std::exception& e) {                      std::cout << e.what() << std::endl;                  }              } else if (choice == "q") {                  std::cout << "Выход..." << std::endl;                  break;              } else {                  std::cout << "Неверный выбор. Попробуйте снова." << std::endl;              }          }      } catch (const std::exception& e) {          std::cerr << "Ошибка: " << e.what() << std::endl;          return 1;      }      return 0;  } |

# 

# Список источников

1. Павский К. В Введение в разработку сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP): Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2020. – 91 с.
2. Павский К. В., Ефимов А. В. Разработка сетевых приложений (протоколы TCP/IP, клиент-сервер, PCAP, Boost.ASIO) : Учебное пособие / Сибирский государственный университет телекоммуникаций и информатики. – Новосибирск, 2018. – 80 с.
3. FTP: Электронный ресурс [https://ru.wikipedia.org/wiki/FTP]