Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Интерфейсы и устройства вычислительных машин

ОТЧЕТ

по лабораторной работе № 6

на тему

РАБОТА С ИНТЕРФЕЙСОМ BLUETOOTH

Студент: Н.Д. Георгиев

Преподаватель: Т. С. Жук

МИНСК 2024

# 1 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

## Цель работы

Изучить интерфейсы взаимодействия с Bluetooth и способы получения информации о подключаемых устройствах, реализовать мониторинг устройств, подключаемых через интерфейс Bluetooth.

## 1.2 Исходные данные к работе

– передать файл с ноутбука на умную колонку и реализовать механизм автоматического воспроизведения после передачи;

– передать файл с ноутбука на ноутбук и воспроизвести;

– передать файл с ноутбука на ноутбук и реализовать механизм автоматического воспроизведения после передачи;

– передать файл с ноутбука на телефон и реализовать механизм автоматического воспроизведения после передачи.

# 2 ОБЗОР ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

**2.1 Winsock**

Для изучения возможностей Windows Sockets 2 (Winsock)[1], применяемого для установления Bluetooth-соединений между устройствами, был использован официальный сайт Microsoft. В документации сайта подробно описаны функции и методы работы с Winsock[1], включая настройку соединений, передачу данных и обработку сетевых ошибок. Эти материалы предоставили необходимые сведения для реализации Bluetooth-соединений, объясняя, как эффективно использовать Winsock[1] для обмена данными и управления соединением между устройствами в Windows.

# 3 ВЫПОЛНЕНИЕ РАБОТЫ

## 3.1 Обоснование выбора архитектуры

Выбор многокомпонентной архитектуры с разделением на backend и frontend можно обосновать следующими причинам:

1. Разделение обязанностей. Разделение на серверную (backend) и клиентскую (frontend) части позволяет более эффективно управлять логикой приложения. Backend отвечает за сбор, обработку и сохранение данных о Bluetooth-устройствах. Это позволяет выделить ключевые процессы в один компонент и сделать его независимым от пользовательского интерфейса. Frontend, в свою очередь, сосредотачивается исключительно на визуализации и взаимодействии с пользователем.

2. Упрощение поддержки и расширяемости. Многокомпонентная архитектура делает проект более гибким. Если потребуется изменить способ сбора данных, это можно сделать, не затрагивая клиентскую часть. Аналогично, обновление интерфейса не требует модификации внутренней логики. Это значительно упрощает внесение изменений и поддержку проекта в долгосрочной перспективе.

3. Производительность. Сбор и запись данных в промежуточный объект данных выполняется на стороне backend, что позволяет разделить нагрузку между компонентами. Это может улучшить отзывчивость клиентской части, так как она занимается только отображением данных, а не их обработкой.

4. Взаимодействие через именованные каналы и файлы: Именованные каналы обеспечивают быстрый и надежный обмен данными между frontend и backend, позволяя им работать асинхронно. Файлы, так же, как и каналы, используются для взаимодействия backend и frontend, а также для сохранения данных на диск, что обеспечивает их сохранность и возможность восстановления в случае сбоев. Взаимодействие через файлы также упрощает процесс отладки, так как промежуточные данные можно легко проверить.

## 3.2 Обоснование выбора формата пользовательского интерфейса

Формат пользовательского интерфейса на основе графического интерфейса – CLI, был выбран, поскольку он является наиболее простым и понятным для опытных пользователей. В отличие от графических интерфейсов, CLI обеспечивает интуитивно понятное взаимодействие. GUI и CLI могут быть более удобны для рядового пользователя если не перегружены, что делает CLI предпочтительным выбором для удобного и доступного отображения данных.

## 3.3 Схема взаимодействия компонентов архитектуры

**3.3.1 Компоненты архитектуры**

Бэкенд: привилегированное исполняемое приложение сбора информации и обработки данных.

Фронтенд: пользовательское приложение для отображения информации об устройствах.

Приемник: пользовательское приложение для отображения принятых данных и отправленных сообщений.

**3.3.2 Взаимодействие компонентов**

Взаимодействие бэкенда с фронтендом: бэкенд записывает обработанные данные в файл, фронтенд читает этот файл для отображения информации пользователю. Фронтенд обрабатывает действия пользователя, получает название устройства, к которому пользователь хочет подключиться, передает их через именованный канал в бэкенд, который далее обрабатывает полученные от фронтенда данные и подключается к устройству если это возможно.

Взаимодействие бэкенда с приемником: бэкенд записывает данные, которые необходимо передать на другое устройство в сокет, а затем приемник читает записанные бэкендом данные из сокета и отображает.

**3.3.3 Файловое взаимодействие**

Бэкенд создает и обновляет файл, содержащий информацию об устройствах, фронтенд считывает эти файлы для обновления отображаемой информации.

**3.3.4 Канальное взаимодействие**

Фронтенд инициирует подключение к именованному каналу бэкенда, при запросе пользователя на подключение к другому устройству и передает данные об устройстве в канал, бэкенд считывает переданные данные и подключается к выбранному устройству если это возможно.

**3.3.5 Взаимодействие через сокеты**

Бэкенд записывает в сокет данные, которые пользователь хочет передать подключенному устройству. Далее приемник читает из сокета данные и выводит их на экран.

**3.3.4 Запуск и выполнение**

Бэкенд: запускается при запуске системы.

Фронтенд: запускается пользователем вручную.

Приемник: запускается пользователем вручную.

Эта архитектура обеспечивает разделение обязанностей: бэкенд обрабатывает данные, фронтенд фокусируется на взаимодействии с пользователем, а приемник ответственен за отображение данных на устройстве к которому подключился пользователь. Файловое и канальное взаимодействие между бэкендом и фронтендом предоставляет простой и надежный метод обмена данными в этой многокомпонентной системе, а механизм сокетов позволяет связать процесс приемника и процесс бэкенда на разных физических системах.

Схематично работа частей приложения представлена на рисунке 3.1.

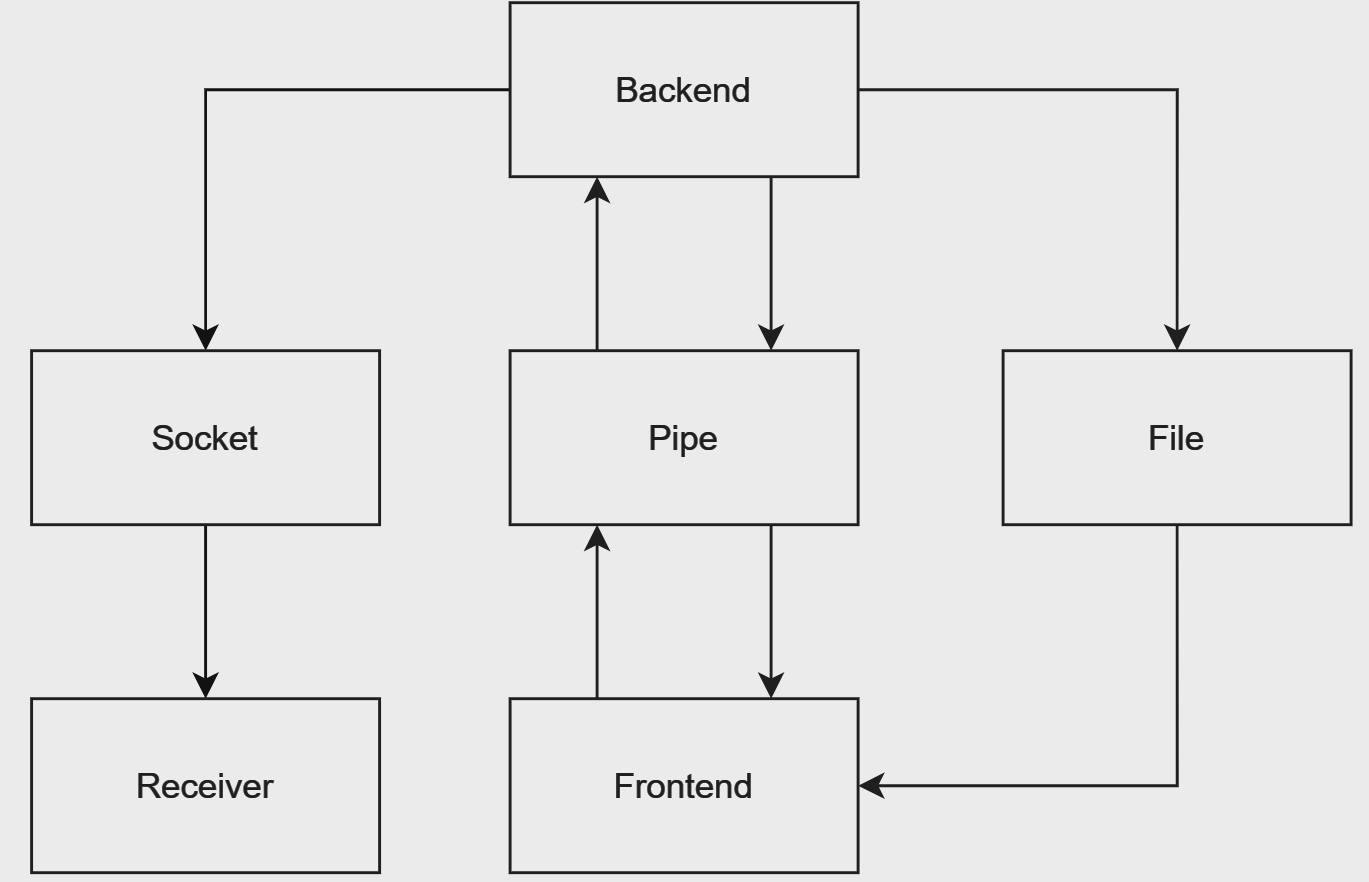
****

Рисунок 3.1 – Взаимодействие частей приложения.

## 3.4 Разработка программного обеспечения

## 3.4.1 Backend - часть

Код представляет собой программу на языке C++ для работы с Bluetooth-устройствами, включая их сканирование, выбор, подключение и передачу файлов. Программа использует библиотеки Winsock2, BluetoothAPIs и другие для взаимодействия с Bluetooth-устройствами на Windows. Описание каждой функции кода и структуры, представленных в коде:

vector<BLUETOOTH\_DEVICE\_INFO> devices: Вектор для хранения информации о найденных Bluetooth-устройствах. Каждый элемент содержит данные об одном устройстве, такие как имя, адрес, класс устройства и состояние подключения.

vector<pair<string, string>> file\_names: Вектор для хранения списка файлов, доступных для отправки. Каждый элемент представляет собой пару строк, где первая строка — это имя файла, а вторая — полный путь к файлу.

bool flag: Флаг для управления состоянием программы. Используется для завершения работы программы при необходимости.

SOCKET clientSocket: Сокет для связи с Bluetooth-устройством через протокол RFCOMM. Используется для подключения, передачи данных и управления соединением.

void menu():  
Отображает текстовое меню программы, содержащее команды для взаимодействия с пользователем, такие как выбор Bluetooth-устройств, подключение для отправки файла и выход из программы.

void printMusicFromDir(const std::wstring& folder\_path):  
Собирает список файлов из указанной директории на локальном диске (в данном случае папка с музыкой) и сохраняет их в вектор file\_names. Каждый файл представлен в виде пары (имя файла и полный путь).

void findBluetoothDevices():  
Выполняет поиск доступных Bluetooth-устройств в зоне действия. Использует API Windows для поиска устройств и сохраняет найденные устройства в вектор devices. Устройства классифицируются по их состоянию (подключены, аутентифицированы, известны и т.д.).

void printDeviceInfo(const BLUETOOTH\_DEVICE\_INFO& deviceInfo):  
Отвечает за вывод на экран детальной информации о конкретном Bluetooth-устройстве. Печатает адрес устройства, имя, класс устройства, а также состояние (подключено, аутентифицировано, запомнено и т.д.).

void printDevices():  
Выводит на экран список всех найденных Bluetooth-устройств. Эта функция вызывает findBluetoothDevices() для обновления списка перед выводом.

void printMusic():  
Выводит список доступных для отправки файлов. Использует функцию printMusicFromDir() для сбора файлов в папке D:\\LB6\\lab6Client\\files, а затем выводит их на экран с нумерацией для выбора.

int sendFile():  
Реализует отправку файла на подключенное Bluetooth-устройство. Пользователь выбирает файл из списка, после чего файл отправляется через сокет. Размер файла отправляется первым сообщением, затем файл передается порциями. Процесс повторяется до тех пор, пока пользователь не завершит отправку.

void initSocketToSend():  
Инициализирует сокетное соединение с выбранным Bluetooth-устройством. Пользователь выбирает устройство из списка, после чего устанавливается соединение через протокол RFCOMM. После успешного подключения вызывается функция sendFile() для передачи файла.

int main():  
Основная функция программы. Организует цикл меню, где пользователь может выбирать команды для выполнения. В зависимости от выбора пользователя вызываются соответствующие функции для работы с Bluetooth-устройствами или отправки файлов.

## 3.4.2 Frontend – часть

Этот код представляет собой серверное приложение на языке C++ для работы с Bluetooth-соединениями. Сервер ожидает подключения от клиента по протоколу RFCOMM и принимает файлы через Bluetooth-сокет. После получения файла сервер автоматически воспроизводит его. Описание каждой функции кода и структуры, представленных в коде:

Основные функции и структуры:

int getFile(SOCKET clientSocket):  
Функция для приема файла от клиента через Bluetooth-соединение. Принимает размер файла, после чего получает файл по частям и сохраняет его на диск. После успешного завершения приема файла программа автоматически воспроизводит его с помощью системной команды system().

SOCKET initServerSocket():  
Функция для инициализации серверного Bluetooth-сокета. Настраивает сокет для работы с протоколом RFCOMM и устанавливает его на прослушивание входящих подключений.

int main():  
Основная функция, которая инициализирует Bluetooth-сервер, ожидает подключения клиентов и принимает файлы.

# 4 ВЫВОДЫ

Цель работы достигнута: изучен интерфейс Bluetooth и способы получения информации о подключаемых устройствах. Разработана программа для мониторинга Bluetooth-устройств, которая позволяет сканировать, подключать и отсоединять устройства, а также извлекать и отображать данные о подключенных устройствах, такие как их адреса, имена, типы, классы устройств и другие параметры. Программа использует сокеты и технологии Windows для взаимодействия с Bluetooth-устройствами, обеспечивая детальный контроль за состоянием подключения. Реализован удобный графический интерфейс, который позволяет пользователю управлять устройствами, передавать файлы и отправлять сообщения на устройства. Особенность реализации — использование низкоуровневых методов для работы с Bluetooth-соединениями, что обеспечивает высокую степень контроля над процессом подключения и передачи данных.

# ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

[1] Microsoft [Электронный ресурс]. – Windows Sockets 2. – Режим доступа: [Windows Sockets 2 - Win32 apps | Microsoft Learn](https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2) – Дата доступа: 16.11.2024.

**ПРИЛОЖЕНИЕ А**

(обязательное)

Исходный текст программы

Модуль «Backend»:

Файл «Client\_main.cpp»:

001. #include <iostream>

002. #include <fstream>

003. #include <vector>

004. #include <Winsock2.h>

005. #include <Ws2bth.h>

006. #include <BluetoothAPIs.h>

007. #include <conio.h>

008. #include <map>

009. #include <sstream>

010. #include <iomanip>

011. #include <filesystem>

012. #include <stdlib.h>

013.

014. #pragma comment(lib, "Ws2\_32.lib")

015. #pragma comment(lib, "Bthprops.lib")

016. using namespace std;

017. namespace fs = std::filesystem;

018.

019. vector<BLUETOOTH\_DEVICE\_INFO> devices;

020. vector<pair<string, string>> file\_names;

021.

022. bool flag = false;

023. SOCKET clientSocket;

024.

025. // Menu display

026. void menu() {

027. cout << "\n\t\t\t================== Menu ==================\n";

028. cout << "\n\t\t\t[p] - List available devices";

029. cout << "\n\t\t\t[s] - Connect and send a file";

030. cout << "\n\t\t\t[e] - Exit program\n";

031. cout << "\n\t\t\t==========================================\n\n";

032. }

033.

034. // Collect files from a folder

035. void printMusicFromDir(const std::wstring& folder\_path) {

036. for (const auto& entry : fs::directory\_iterator(folder\_path)) {

037. pair<string, string> files("", "");

038. files.first = entry.path().filename().string();

039. files.second = entry.path().string();

040. file\_names.push\_back(files);

041. }

042. }

043.

044. // Find Bluetooth devices

045. void findBluetoothDevices() {

046. devices.clear();

047. BLUETOOTH\_DEVICE\_SEARCH\_PARAMS searchParams;

048. searchParams.dwSize = sizeof(BLUETOOTH\_DEVICE\_SEARCH\_PARAMS);

049. searchParams.fReturnAuthenticated = TRUE;

050. searchParams.fReturnConnected = TRUE;

051. searchParams.fReturnRemembered = TRUE;

052. searchParams.fReturnUnknown = TRUE;

053. searchParams.cTimeoutMultiplier = 1;

054.

055. BLUETOOTH\_DEVICE\_INFO deviceInfo = { sizeof(BLUETOOTH\_DEVICE\_INFO), 0 };

056. HBLUETOOTH\_DEVICE\_FIND hFind;

057.

058. hFind = BluetoothFindFirstDevice(&searchParams, &deviceInfo);

059.

060. if (hFind == NULL) {

061. cerr << "No devices found. Ensure Bluetooth is enabled on your device." << endl;

062. return;

063. }

064.

065. do {

066. devices.push\_back(deviceInfo);

067. } while (BluetoothFindNextDevice(hFind, &deviceInfo));

068.

069. BluetoothFindDeviceClose(hFind);

070. }

071.

072.

073. void printDeviceInfo(const BLUETOOTH\_DEVICE\_INFO& deviceInfo) {

074. wcout << L"Address: " << hex << uppercase

075. << setw(12) << setfill(L'0') << deviceInfo.Address.ullLong << endl;

076.

077. wcout << L"Name: " << deviceInfo.szName << endl;

078.

079. // Печать класса устройства (major/minor class)

080. wcout << L"Class: " << hex << uppercase << deviceInfo.ulClassofDevice << endl;

081.

082. // Определяем major class

083. uint8\_t majorClass = (deviceInfo.ulClassofDevice & 0x1F00) >> 8;

084. wcout << L"Major Class: " << majorClass << endl;

085.

086. // Appearance (интерпретируем как значение класса устройства)

087. wcout << L"Appearance: " << deviceInfo.ulClassofDevice << endl;

088.

089. wcout << L"Paired: " << (deviceInfo.fAuthenticated ? L"Yes" : L"No") << endl;

090. wcout << L"Trusted (Remembered): " << (deviceInfo.fRemembered ? L"Yes" : L"No") << endl;

091. wcout << L"Connected: " << (deviceInfo.fConnected ? L"Yes" : L"No") << endl;

092.

093. // Legacy Pairing

094. wcout << L"Legacy Pairing: " << L"No" << endl;

095.

096. wcout << L"--------------------------------------------" << endl;

097. }

098.

099. void printDevices() {

100. findBluetoothDevices();

101. if (devices.empty()) {

102. cerr << "No Bluetooth devices found. Ensure Bluetooth is enabled on your system." << endl;

103. return;

104. }

105.

106. for (size\_t i = 0; i < devices.size(); ++i) {

107. wcout << L"Device " << i << L":" << endl;

108. printDeviceInfo(devices[i]);

109. }

110. }

111.

112. // Print available files

113. void printMusic() {

114. file\_names.clear();

115.

116. wstring path = L"D:\\LB6\\lab6Client\\files";

117. printMusicFromDir(path);

118. int i = 0;

119. for (const auto& file\_name : file\_names) {

120. cout << i++ << ". " << file\_name.first << endl;

121. }

122. }

123.

124. // Send a file to the Bluetooth device

125. int sendFile() {

126. int temp = 0;

127.

128. while (true) {

129. cout << endl;

130. printMusic();

131. int pickFile;

132.

133. cout << "Select a file to send: ";

134. cin >> pickFile;

135. if (pickFile < 0 || static\_cast<size\_t>(pickFile) >= file\_names.size()) {

136. cerr << "Invalid file selection. Please try again." << endl;

137. continue;

138. }

139.

140. string pathToFile = file\_names[pickFile].second;

141. ifstream fileToSend(pathToFile, ios::binary);

142.

143. if (!fileToSend) {

144. cerr << "Failed to open file: " << pathToFile << endl;

145. return -1;

146. }

147. cout << pathToFile;

148.

149. // Определяем размер файла

150. fileToSend.seekg(0, ios::end);

151. int fsize = fileToSend.tellg();

152. fileToSend.seekg(0, ios::beg);

153.

154. string file\_size = to\_string(fsize);

155. send(clientSocket, file\_size.c\_str(), 100, 0);

156.

157. char buffer[1024];

158. while (!fileToSend.eof()) {

159. fileToSend.read(buffer, sizeof(buffer));

160. int bytesRead = fileToSend.gcount();

161. int bytesSent = send(clientSocket, buffer, bytesRead, 0);

162. if (bytesSent == SOCKET\_ERROR) {

163. cerr << "Error writing to socket!" << endl;

164. return -1;

165. }

166. }

167. fileToSend.close();

168.

169. cout << "Continue? 1 - Yes, 0 - No: ";

170. cin >> temp;

171. if (temp == 0) break;

172. }

173. return 0;

174. }

175.

176. // Initialize socket connection

177. void initSocketToSend() {

178. cout << "Enter the device number: " << endl;

179. int choice;

180. cin >> choice;

181.

182. BLUETOOTH\_DEVICE\_INFO deviceInfo = devices.at(choice);

183. BTH\_ADDR addr = deviceInfo.Address.ullLong;

184.

185. WSADATA wsaData;

186. if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

187. cerr << "Failed to initialize Winsock!" << endl;

188. exit(1);

189. }

190.

191. SOCKADDR\_BTH sockaddrBthServer = { 0 };

192. sockaddrBthServer.addressFamily = AF\_BTH;

193. sockaddrBthServer.serviceClassId = RFCOMM\_PROTOCOL\_UUID;

194. sockaddrBthServer.port = 4;

195. sockaddrBthServer.btAddr = addr;

196.

197. clientSocket = socket(AF\_BTH, SOCK\_STREAM, BTHPROTO\_RFCOMM);

198. if (clientSocket == INVALID\_SOCKET) {

199. cerr << "Error creating client socket!" << endl;

200. WSACleanup();

201. exit(1);

202. }

203.

204. if (connect(clientSocket, reinterpret\_cast<sockaddr\*>(&sockaddrBthServer), sizeof(sockaddrBthServer)) == SOCKET\_ERROR) {

205. cerr << "Error connecting to the server!" << endl;

206. closesocket(clientSocket);

207. WSACleanup();

208. exit(1);

209. }

210. wcout << L"Connected to the server." << endl;

211.

212. flag = false;

213. cout << endl;

214. sendFile();

215.

216. closesocket(clientSocket);

217. WSACleanup();

218. }

219.

220. int main() {

221. setlocale(LC\_ALL, "English");

222. char i = 0;

223.

224. menu();

225.

226. while (true) {

227. cin >> i;

228. getchar();

229.

230. switch (i) {

231. case 'p':

232. system("cls");

233. menu();

234. cout << endl;

235. printDevices();

236. cout << endl;

237. break;

238.

239. case 's':

240. system("cls");

241. menu();

242. cout << endl;

243. printDevices();

244. initSocketToSend();

245. cout << endl;

246. break;

247.

248. case 'e':

249. system("cls");

250. devices.clear();

251. file\_names.clear();

252. flag = true;

253. exit(0);

254. break;

255.

256. default:

257. break;

258. }

259. }

260. return 0;

261. }

Модуль «Frontend»:

Файл «Server\_main.cpp»:

001. #include <iostream>

002. #include <fstream>

003. #include <Winsock2.h>

004. #include <Ws2bth.h>

005. #include <BluetoothAPIs.h>

006. #include <locale.h>

007.

008. #pragma comment(lib, "Ws2\_32.lib")

009. #pragma comment(lib, "Bthprops.lib")

010.

011. using namespace std;

012.

013. // Функция для получения файла

014. int getFile(SOCKET clientSocket) {

015. char fileSize[1000];

016. memset(fileSize, '\0', 1000);

017.

018. // Получение размера файла

019. if (!recv(clientSocket, fileSize, 1000, 0))

020. return -1;

021. int dataSize = atoi(fileSize);

022.

023. if (dataSize <= 0)

024. return 0;

025.

026. cout << "Receiving file" << endl;

027.

028. // Открытие файла для записи

029. ofstream outputFile("D:/receivedFile.mp3", ios::out | ios::binary | ios::trunc);

030. while (dataSize > 0) {

031. char buffer[1024];

032. int bytes = recv(clientSocket, buffer, min(1024, dataSize), 0);

033. outputFile.write(buffer, bytes);

034. dataSize -= bytes;

035. }

036. outputFile.close();

037.

038. // Автоматическое воспроизведение файла

039. system("D:/receivedFile.mp3");

040.

041. return 0;

042. }

043.

044. // Инициализация серверного сокета

045. SOCKET initServerSocket() {

046. SOCKADDR\_BTH sockaddrBthServer = { 0 };

047. sockaddrBthServer.addressFamily = AF\_BTH;

048. sockaddrBthServer.serviceClassId = RFCOMM\_PROTOCOL\_UUID;

049. sockaddrBthServer.port = BT\_PORT\_ANY;

050.

051. SOCKET serverSocket = socket(AF\_BTH, SOCK\_STREAM, BTHPROTO\_RFCOMM);

052. bind(serverSocket, reinterpret\_cast<sockaddr\*>(&sockaddrBthServer), sizeof(sockaddrBthServer));

053.

054. int addrLen = sizeof(sockaddrBthServer);

055. getsockname(serverSocket, reinterpret\_cast<sockaddr\*>(&sockaddrBthServer), &addrLen);

056.

057. return serverSocket;

058. }

059.

060. // Ожидание клиента

061. SOCKET initClientSocket(SOCKET serverSocket) {

062. SOCKADDR\_BTH sockaddrBthClient = { 0 };

063. int clientAddrSize = sizeof(sockaddrBthClient);

064. SOCKET clientSocket = accept(serverSocket, reinterpret\_cast<sockaddr\*>(&sockaddrBthClient), &clientAddrSize);

065.

066. return clientSocket;

067. }

068.

069. int main() {

070. setlocale(LC\_ALL, "Russian");

071.

072. // Инициализация WSA

073. WSADATA wsaData;

074. WSAStartup(MAKEWORD(2, 2), &wsaData);

075.

076. // Инициализация сервера

077. SOCKET serverSocket = initServerSocket();

078. listen(serverSocket, SOMAXCONN);

079. cout << "Server has started" << endl;

080.

081. while (true) {

082. cout << "Waiting for connection" << endl;

083. SOCKET clientSocket = initClientSocket(serverSocket);

084. cout << "Connection established" << endl;

085.

086. // Прием файлов

087. while (true) {

088. int result = getFile(clientSocket);

089. if (result == -1)

090. break;

091. }

092.

093. cout << "Client has disconnected" << endl << endl;

094. closesocket(clientSocket);

095. }

096.

097. cout << "Server is shutting down" << endl;

098. closesocket(serverSocket);

099. WSACleanup();

100.

101. return 0;

102. }