Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное средство «ОнлаЙн-тренажёр Р-159»**

|  |  |  |
| --- | --- | --- |
| Студент |  | И.С. Андросов |
| Руководитель |  | Д.С. Шулицкий |

Минск 2019

СОДЕРЖАНИЕ

[Введение](#_Toc513665878) 5

[1 Анализ предметной области](#_Toc513665879) 7

[1.1 Обзор аналогов](#_Toc513665880) 7

[1.2 Постановка задачи](#_Toc513665881) 11

[2 Разработка программного средства](#_Toc513665882) 12

[2.1 Структура программы](#_Toc513665883) 12

[2.2 Интерфейс программного средства](#_Toc513665884) 12

[2.3 Работа со звуковыми ресурсами](#_Toc513665885) 14

[2.4 Сетевая логика](#_Toc513665888) 14

[2.4.1 Серверная часть](#_Toc513665886) 16

[2.4.2 Клиентская часть](#_Toc513665887) 17

[3 Тестирование программного средства](#_Toc513665890) 23

[4 Руководство пользователя](#_Toc513665891) 26

[4.1 Интерфейс программы](#_Toc513665892) 26

[4.2 Порядок настройки](#_Toc513665893) 28

[Заключение](#_Toc513665894) 29

[Список использованных источников](#_Toc513665895) 30

[Приложение А. Исходный код программы](#_Toc513665896) 31

ВВЕДЕНИЕ

21-й век – век информационных и телекоммуникационных технологий. Всё больше отраслей специализируются в компьютерной сфере, так как это наиболее актуальное и современное направление. Новейшие научные достижения безудержно преображают мир вокруг нас. Большим изменениям подверглась и сфера коммуникаций. Современные средства связи обладают огромными преимуществами перед радиосвязью, но, будучи очень уязвимыми, могут быть ликвидированы при первой возможности. Радиосвязь же может быть использована локально. Таким образом, знание работы с радиостанциями является необходимым навыком.

Первый патент на беспроводную связь получил в [1872-ом году](https://ru.wikipedia.org/wiki/1872_%D0%B3%D0%BE%D0%B4) американский радиолюбитель, стоматолог по профессии [Малон Лумис](https://ru.wikipedia.org/wiki/%D0%9B%D1%83%D0%BC%D0%B8%D1%81,_%D0%9C%D0%B0%D0%BB%D0%BE%D0%BD), заявивший в [1866-ом году](https://ru.wikipedia.org/wiki/1866_%D0%B3%D0%BE%D0%B4) о том, что он открыл способ беспроволочной связи; в США изобретателем радио считают [Дэвида Хьюза](https://ru.wikipedia.org/wiki/%D0%A5%D1%8C%D1%8E%D0%B7,_%D0%94%D0%B5%D0%B2%D0%B8%D0%B4_%D0%AD%D0%B4%D0%B2%D0%B0%D1%80%D0%B4), а также [Томаса Эдисона](https://ru.wikipedia.org/wiki/%D0%AD%D0%B4%D0%B8%D1%81%D0%BE%D0%BD,_%D0%A2%D0%BE%D0%BC%D0%B0%D1%81_%D0%90%D0%BB%D0%B2%D0%B0) и [Николу Теслу](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%BA%D0%BE%D0%BB%D0%B0_%D0%A2%D0%B5%D1%81%D0%BB%D0%B0); в Германии – [Генриха Герца](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D1%80%D1%86,_%D0%93%D0%B5%D0%BD%D1%80%D0%B8%D1%85_%D0%A0%D1%83%D0%B4%D0%BE%D0%BB%D1%8C%D1%84); во Франции – [Эдуарда Бранли](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D0%BD%D0%BB%D0%B8,_%D0%AD%D0%B4%D1%83%D0%B0%D1%80%D0%B4); в ряде балканских стран – [Николу Теслу](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D0%BB%D0%B0,_%D0%9D%D0%B8%D0%BA%D0%BE%D0%BB%D0%B0); в Бразилии – Ланделя де Муру; в Англии – [Оливера Джозефа Лоджа](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B4%D0%B6,_%D0%9E%D0%BB%D0%B8%D0%B2%D0%B5%D1%80_%D0%94%D0%B6%D0%BE%D0%B7%D0%B5%D1%84); в Индии – [Джагадиша Чандру Боше](https://ru.wikipedia.org/wiki/%D0%91%D0%BE%D1%81,_%D0%94%D0%B6%D0%B0%D0%B3%D0%B4%D0%B8%D1%88_%D0%A7%D0%B0%D0%BD%D0%B4%D1%80%D0%B0); в России – [А. С. Попова](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BF%D0%BE%D0%B2,_%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D0%BD%D0%B4%D1%80_%D0%A1%D1%82%D0%B5%D0%BF%D0%B0%D0%BD%D0%BE%D0%B2%D0%B8%D1%87) и [Якова Наркевича-Иодко](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%80%D0%BA%D0%B5%D0%B2%D0%B8%D1%87-%D0%98%D0%BE%D0%B4%D0%BA%D0%BE,_%D0%AF%D0%BA%D0%BE%D0%B2_%D0%9E%D1%82%D1%82%D0%BE%D0%BD%D0%BE%D0%B2%D0%B8%D1%87).

Создателем первой успешной системы обмена информацией с помощью радиоволн на западе считается итальянский инженер [Гульельмо Маркони](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%BD%D0%B8,_%D0%93%D1%83%D0%BB%D1%8C%D0%B5%D0%BB%D1%8C%D0%BC%D0%BE).

В [СССР](https://ru.wikipedia.org/wiki/%D0%A1%D0%A1%D0%A1%D0%A0) и в бывших союзных республиках изобретателем радиотелеграфии считается [А. С. Попов](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BF%D0%BE%D0%B2,_%D0%90%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D0%BD%D0%B4%D1%80_%D0%A1%D1%82%D0%B5%D0%BF%D0%B0%D0%BD%D0%BE%D0%B2%D0%B8%D1%87). В опытах по радиосвязи, проведённых в физическом кабинете, а затем в саду Минного офицерского класса, прибор А. С. Попова обнаруживал излучение радиосигналов, посылаемых передатчиком, на расстоянии до 60-ти метров. На заседании Русского физико-химического общества в Петербурге 25-го апреля (седьмого мая) [1895-го года](https://ru.wikipedia.org/wiki/1895_%D0%B3%D0%BE%D0%B4) А. С. Попов продемонстрировал, как указано в протоколе заседания, «прибор, предназначенный для показывания быстрых колебаний в атмосферном электричестве». В СССР, с 1945-го года, седьмого мая стали отмечать как [День радио](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BD%D1%8C_%D1%80%D0%B0%D0%B4%D0%B8%D0%BE).

Изобретение радиосвязи дало начало таким научно-техническим направлениям, как [радиоастрономия](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%8F), [радиометрология](https://ru.wikipedia.org/w/index.php?title=%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F&action=edit&redlink=1), [радионавигация](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D0%BD%D0%B0%D0%B2%D0%B8%D0%B3%D0%B0%D1%86%D0%B8%D1%8F), [радиолокация](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D0%BB%D0%BE%D0%BA%D0%B0%D1%86%D0%B8%D1%8F), [радиоразведка](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D0%B2%D0%B5%D0%B4%D0%BA%D0%B0), [радиопротиводействие](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B4%D0%B8%D0%BE%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%BE%D1%80%D1%8C%D0%B1%D0%B0).

Радио послужило мощнейшим стимулом в исследовании и развитии электричества, и стало основой электроники. А электроника, в свою очередь, породила вычислительную технику.

Ультракороткие волны (УКВ) – традиционное в СССР название диапазона радиоволн, объединяющего метровые, дециметровые, сантиметровые и миллиметровые волны (или диапазоны очень высоких частот – ОВЧ, ультравысоких частот – УВЧ, сверхвысоких частот – СВЧ и крайне высоких частот – КВЧ). То есть это все радиоволны, длина которых менее 10-ти метров.

Существует огромное множество УКВ радиостанций, наиболее известные из них:

* P-173;
* P-123МТ;
* P-111;
* P-159;
* P-180;
* P-181-5HУ

Р-159 – [советская](https://ru.wikipedia.org/wiki/%D0%A1%D0%A1%D0%A1%D0%A0) переносная широкодиапазонная войсковая ультракоротковолновая радиостанция для связи в звене управления Вооруженных Сил [рота](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D1%82%D0%B0)-[батальон](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D1%82%D0%B0%D0%BB%D1%8C%D0%BE%D0%BD). На вооружении [Советской Армии](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B2%D0%B5%D1%82%D1%81%D0%BA%D0%BE%D0%B9_%D0%90%D1%80%D0%BC%D0%B8%D0%B8) принята с начала [80-ых годов](https://ru.wikipedia.org/wiki/80-%D0%B5_%D0%B3%D0%BE%D0%B4%D1%8B).

Радиостанция приёмо-передающая, [симплексная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%81) с узкополосным телеграфированием и с тональным вызовом, а также с возможностью дистанционного управления в телефонном режиме. Конструкция ранцевая – наплечные ремни для переноски крепятся к корпусу. Также в комплекте имеется сумка радиста для переноски антенн и инструмента.

Радиостанция собрана по [трансиверной](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%B8%D0%B2%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%80%D0%B0%D0%B4%D0%B8%D0%BE%D1%81%D1%82%D0%B0%D0%BD%D1%86%D0%B8%D1%8F) схеме. Установка частоты производится с помощью малогабаритных поворотных галетных переключателей, информация с которых преобразуется в коэффициент деления для синтезатора использующего делитель с переменным коэффициентом деления.

Целью данного курсового проекта является разработка приложения «Онлайн-тренажёр Р-159».

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Обзор аналогов

Существует ряд тренажёров радиостанций, каждый из них имеет свои преимущества и недостатки. На рисунке 1.1 представлен один из наиболее удобных для освоения тренажёров радиостанции Р-130, работающей в коротковолновом диапазоне.



Рисунок 1.1 – Тренажёр радиостанции Р-130

В программном средстве есть пошаговые рекомендации по включению и настройки радиостанции, а также устройство блока выносного согласующего антенного устройства, которое изображено на рисунке 1.2. Также есть возможность самопроверки в самостоятельного режиме настройки устройства без подсказок приложения, таким образом, оно может быть использовано преподавательским составом.

Единственным недостатком данного приложения является неочевидная эксплуатация программного средства вместе с другими приложениями. Не подготовленный пользователь может совершить ряд ошибок, что испортит впечатление от получения навыков.



Рисунок 1.2 – Устройство блока выносного согласующего антенного устройства

Другим аналогом послужило приложения для создания тестов по различным радиостанциям, их тактико-технических характеристиках и использованию – MilitaryTest. Стандартная база данных программы включает такие устройства как: Р-111, Р-123, Р-130 и другие.

Главное меню приложения представлено на рисунке 1.3.

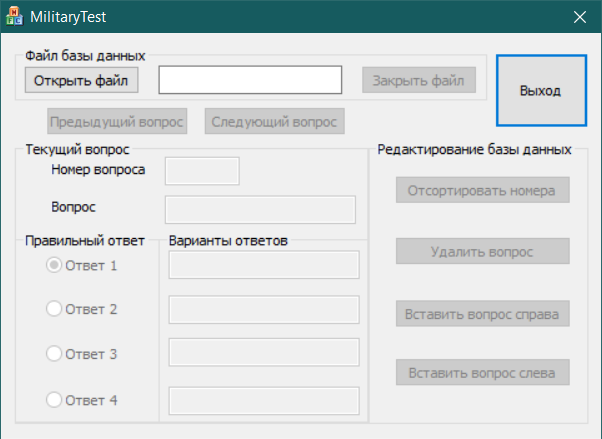


Рисунок 1.3 – Главное меню приложения MilitaryTest

К сожалению, на данный момент не существует полноценных аналогов тренажёра для Р-159. Различают две вариаций радиостанции Р-159: оригинальная советская войсковая и её модификация – Р-159М.

Самой успешной и популярной является оригинальная версия радиостанции. Свою популярность она получила за возможность переноса за спиной, широкого диапазона рабочих частот и прочности. Внешний вид данной радиостанции представлен на рисунке 1.4.



Рисунок 1.4 – Р-159 оригинальная советская войсковая

Радиостанция состоит из приёмопередатчика с аккумуляторным отсеком и аккумуляторными батареями, антенн, микротелефонной гарнитуры, противовеса и телеграфного ключа.

Тактико-технические характеристики:

* диапазон частот – 30,0 – 75,999 МГц;
* шаг перестройки – один кГц;
* вид работы – ТлФ (телефон и телеграф с частотной модуляцией), ТлГ (узкополосное телеграфирование);
* мощность передатчика – пять Ватт;
* элементная база – полупроводниковые приборы;
* вес рабочего комплекта – 14,5 килограмм;
* габариты 305×180×410 мм.

В [1988-ом году](https://ru.wikipedia.org/wiki/1988_%D0%B3%D0%BE%D0%B4) с учётом опыта [Афганской войны](https://ru.wikipedia.org/wiki/%D0%90%D1%84%D0%B3%D0%B0%D0%BD%D1%81%D0%BA%D0%B0%D1%8F_%D0%B2%D0%BE%D0%B9%D0%BD%D0%B0_(1979%E2%80%941989)) была выпущена модификация Р-159М.

Р-159М была предназначена для работы в комплекте с блоком засекречивающей аппаратурой связи Т-240С «Историк», состоящей из блока сопряжения и блока шифрования.

Ниже представлен аналог радиостанции, а также общий вид передней панели радиостанции.

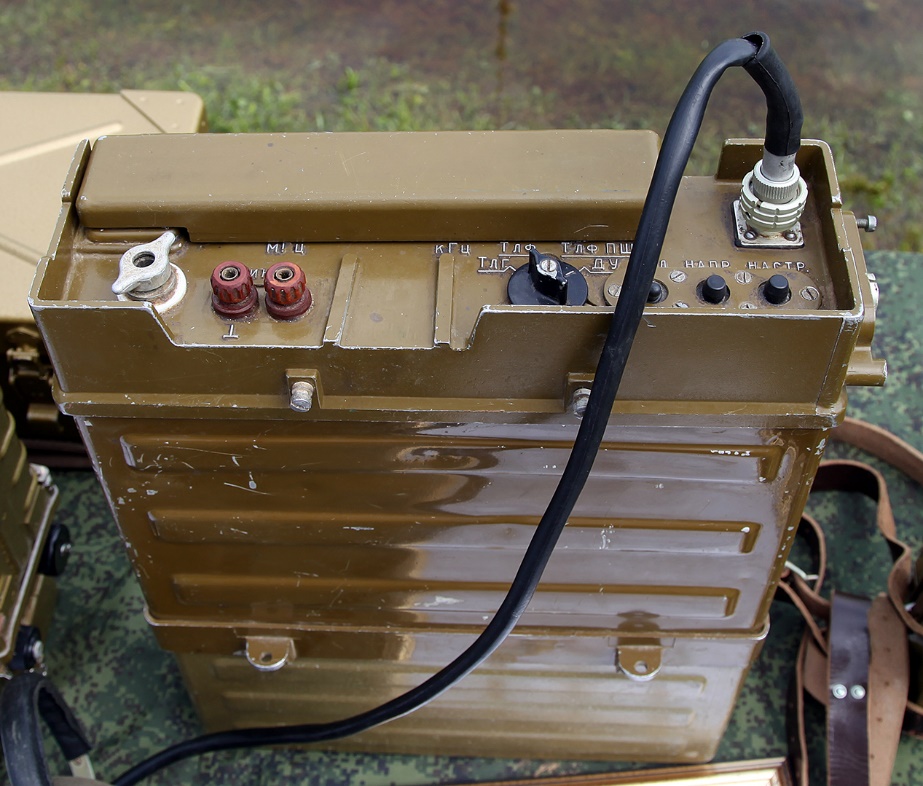


Рисунок 1.5 – Аналог Р-159М

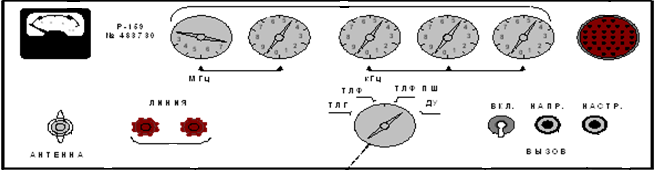


Рисунок 1.6 – Общий вид передней панели радиостанции

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Онлайн-тренажёр Р-159».

Будут разработаны алгоритмы инициализации сетевых ресурсов, передачи данных по сети и их валидации.

В программном средстве планируется реализовать следующие функции:

* одновременная работа большого числа пользователей;
* работа в телеграфном режиме;
* работа в телефонном режиме;
* выбор и передача рабочей частоты.

Логическая составляющая приложения будет включать в себя:

* состояние выбранного режима;
* состояние частоты передачи;
* алгоритмы валидации и передачи данных;
* взаимодействие структурных блоков приложения.

Для разработки программного средства будет использоваться язык программирования C++, операционная система Windows 10 и среды разработки Embarcadero RAD studio 10.2 и Visual Studio 2019.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура программы

Данное приложение включает в себя две составляющие: серверную часть, обеспечивающая обмен данными между пользователями, и клиентскую часть, в которой требуется использовать два структурных блока:

– MainUnit – модуль, отображающий главное окно программы и обеспечивающий взаимодействие со вторым модулем;

– ClientLogic – модуль, отвечающий за генерацию данных и передачи в

дальнейшем их по сети;

В ходе разработки программного продукта были использованы две среды программирования, на это есть ряд причин. Visual Studio 2019 предоставляет огромные возможности для редактирования исходного кода, лексического, синтаксического и даже семантического анализа написанного кода в реальном времени, и удобную подсветку синтаксиса. Эти факторы позволяют ускорить процесс разработки и избежать непредвиденных ошибок, что сэкономило значительное количество времени в процессе тестирования серверной части. Недостатком же является отсутствие простого создания интерфейса. Именно поэтому для разработки клиентской части была выбрана среда Embarcadero RAD studio 10.2.

Данный подход не только позволил эффективно разрабатывать приложения, но и показать совместимость структурных блоков, полученных в результате работы в различных средах.

* 1. Интерфейс программного средства

Внешний вид и удобность в использовании являются одними из главных критериев качества программного средства. Поэтому взаимодействие приложения с пользователем необходимо организовать максимально интуитивно и просто.

Клиентская часть разработана в среде Embarcadero RAD studio 10.2 для быстрого доступа к компонентам, реализующих графический интерфейс.

В качестве навигации были выбраны очень простые и понятные большинству пользователей компоненты TButton, TChekBox, TRadioButton и TComboBox. Также пользователю предлагается ввести данные IP и порта в соответствующие компоненты TEdit серверного приложения для работы в онлайн режиме.

На рисунке 2.1 представлено главное окно программы.

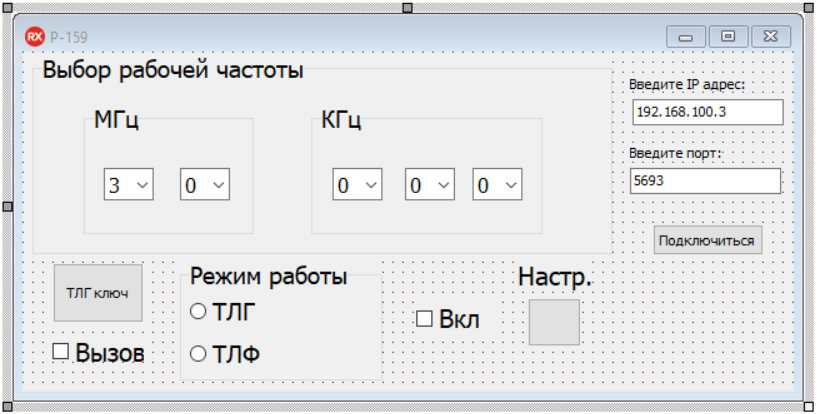


Рисунок 2.1 – Главное окно программы

Серверное приложение работает в консольном режиме, что показывает его принадлежность к исполнению логической составляющей тренажёра. Для начала работы, как и в клиентской части, нужно ввести данные IP и порта.

Отображение серверного приложение представлено на рисунке 2.2.

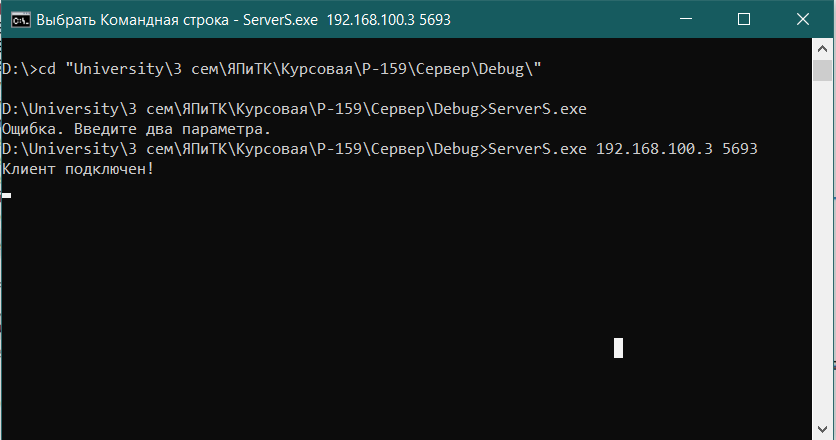


Рисунок 2.2 – Отображения серверного приложения

Интерфейс программного средства получился компактным и позволяет работать сразу с несколькими приложениями одновременно с одного устройства.

* 1. Работа со звуковыми ресурсами

Программное средство взаимодействует с определенным набором ресурсов, обеспечивающих некоторые звуковые эффекты.

Управление звуком реализовано при помощи стандартной системной функции PlaySound, а также библиотеки SDL. Различные случаи использования функции PlaySound приведены ниже.

if (currentMod == 0 && isEnabled && incomFreq == frequency)

{

PlaySound(TEXT("TLG.wav"), NULL, SND\_ASYNC);

break;

}

В качестве параметров передается файл в виде строки, который будет воспроизводиться, параметр использования ресурса, а также специальные флаги SND\_ASYNC и SND\_LOOP, определяющие асинхронный и циклический характеры воспроизведения. В данном участке кода воспроизводится звук телеграфа.

PlaySound(0, 0, SND\_PURGE);

Данный пример использования функции PlaySound позволяет прервать воспроизведение звука при наличии флага SND\_PURGE.

Библиотека SDL была использована для записи и воспроизведения голоса пользователя при передаче в телефонном режиме. Для начала работы с со звуком нужно определить и проинициализировать записывающее и воспроизводящее устройства. Это происходит в самом начале работы программы – на этапе создания главного окна.

На рисунке 2.3 изображена блок-схема инициализации звуковых устройств.



Рисунок 2.3 – Блок-схема инициализации звуковых устройств

Задачу инициализации выполняют следующие функции, которые вызываются в соответствующем порядке:

1. SDL\_Init( SDL\_INIT\_VIDEO | SDL\_INIT\_AUDIO ) – начало работы с библиотекой;
2. loadMedia() – загрузка устройств воспроизведения и записи;
3. loadRecordingDevice() – инициализация записывающего устройства;
4. loadPlaybackDevice() – инициализация воспроизводящего устройства;
5. initBuffer() – подсчёт размера буфера для хранения очередного голосового сообщения и выделение памяти под него;

Каждый этап зависит от предыдущего, что объясняет такую очерёдность вызовов, а также производит проверку корректного выполнения функций для приемлемой работы программного средства.

После успешного выделения памяти для аудиоданных происходит выполнение функции в другом потоке посредством выполнения следующей строки кода.

CreateThread(NULL, NULL,(LPTHREAD\_START\_ROUTINE) initRecording, NULL, NULL, NULL);

InitReocording – это функция запуска записи голоса, выполняющая постоянный цикл при условии, что флаг quit равен false. Функция имеет несколько состояний, которые хранятся в переменной currentState, тип которой представляет собой перечисление.

enum RecordingState

{

START\_RECORDING,

RECORDING

};

При воспроизведении звука используется аналогичная функция playAudio(), за исключением того, что у неё нет переменной состояния. Так как в ней заранее известны данные, подлежащие выводу.

* 1. Сетевая логика

Интернет – основа работы приложения. Для работы с сетью используется стандартное API Windows – Windows Sockets 2. Поэтому вначале обоих приложений подключается заголовочный файл winsock2.h.

#include <winsock2.h>

Winsock позволяет создавать расширенные приложения для Интернета, локальной сети и другие сетевые приложения для передачи данных приложений независимо от используемого сетевого протокола. Благодаря Winsock программистам предоставляется доступ к расширенным сетевым возможностям Windows, таким как многоадресная передача.

Winsock следует модели Windows Open System Architecture – WOSA. Он определяет стандартный интерфейс поставщика услуг между интерфейсом прикладного программирования (API) с его экспортированными функциями и стеками протоколов. Он использует парадигму сокетов, которая впервые была популяризирована UNIX от Berkeley Software Distribution (BSD). Позднее он был адаптирован для Windows в Windows Sockets 1.1, с которой приложения Windows Sockets 2 обратно совместимы.

В качестве структуры для хранения информации о передаваемых данных по сети была выбрана переменная типа Packet, где Packet – перечисление, содержащее значения P\_KEY и P\_AUDIO.

enum Packet

{

P\_KEY,

P\_AUDIO

};

Серверная часть начинает работу с инициализации WSA, код которой приведён ниже.

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if(WSAStartup(DLLVersion, &wsaData) != 0) {

cout << "Ошибка инициализации WSA!" << endl;

exit(1);

}

Клиентская часть выполняет аналогичный код, но при ошибке инициализации выполнение программы приостанавливается.

* + 1. Серверная часть

Основная задача данного структурного блока – ретрансляция полученных данных всем пользователям. Сервер «ожидает» подключения очередного участника сети, а после добавляет его в массив сокетов пользователей под своим индексом. Не исключено, что при подсоединении может произойти непредвиденная ошибка, в этом случае будет выведено сообщение об ошибке установления связи.

SOCKET Connections[100];

int Counter = 0;

Как только пользователь подключился, ему присваивается своё место, а также новый поток принятия сетевых пакетов, реализованных в функции ClientHandler. Ниже представлен код, реализующий данный функционал.

SOCKET sListen = socket(AF\_INET, SOCK\_STREAM, NULL);

bind(sListen, (SOCKADDR\*)&addr, sizeof(addr));

listen(sListen, SOMAXCONN);

SOCKET newConnection;

for(int i = 0; i < 100; i++) {

newConnection = accept(sListen, (SOCKADDR\*)&addr, &sizeofaddr);

if(newConnection == 0) {

cout << "Ошибка подключения клиента.";

} else {

cout << "Клиент подключен!";

Connections[i] = newConnection;

Counter++;

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)

ClientHandler, (LPVOID)(i), NULL, NULL);

}

}

Функция socket создает сокет, связанный с конкретным поставщиком транспортных услуг. Сокет – название программного интерфейса для обеспечения обмена данными между процессами.

Принятием и обработкой пакетов, как было сказано выше, занимается функция ClientHandler. При обнаружении несоответствия принятого пакета данных с допустимыми значениями в функции ProcessPacket, пользователь уведомляется о данном происшествии и номере принятых данных.

Если же передача была корректной, то происходит вызов ретранслирующей подпрограммы:

– sendTelegraph – функция, вызываемая при значении переменной packettype равной P\_KEY;

– sendAudio – подпрограмма передачи голосового сообщения;

Ниже на рисунка 2.4 и 2.5 приведены блок-схемы функций ClientHandler и ProcessPacket соответсвенно.



Рисунок 2.4 – Блок-схема функции ClientHandler



Рисунок 2.5 – Блок-схема функции ProcessPacket

* + 1. Клиентская часть

Интерфейс клиентской части обеспечивает управление тренажёра. Для отслеживания состояния приложения объявлены следующие переменные.

int frequency = 30000;

int currentMod = 2;

int isCalling = 0;

int isEnabled = 0;

Их названия говорят сами за себя, а именно: первая переменная характеризует частоту передачи в диапазоне от 30000 до 79999 герц, вторая – за текущий способ отправки информации, isCalling и isEnabled за включенное или выключенное положение записи звука и подачи питания соответственно. Также были выбраны константные значения типа const int для переключения режима передачи сообщений.

const int TLG = 0;

const int TLF = 1;

Подключение к серверу и работа с другими пользователями начинается после ввода IP адреса и порта с последующим нажатием кнопки «Подключиться». Происходит присвоение введённых значений в соответствующие поля переменной addr типа SOCKADDR\_IN, а также попытка установления связи с сервером. При положительном ответе последнего, происходит выполнение функции ClientHandler в новом потоке.

int connectToServer(char \*IP, int Port)

{

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if(WSAStartup(DLLVersion, &wsaData) != 0)

{

exit(1);

}

SOCKADDR\_IN addr;

int sizeofaddr = sizeof(addr);

addr.sin\_addr.s\_addr = inet\_addr(IP);

addr.sin\_port = htons(Port);

addr.sin\_family = AF\_INET;

Connection = socket(AF\_INET, SOCK\_STREAM, NULL);

if (connect(Connection, (SOCKADDR\*)&addr, sizeof(addr)) != 0)

{

exit(1);

}

CreateThread(NULL,NULL,( LPTHREAD\_START\_ROUTINE ) ClientHandler, NULL, NULL, NULL);

}

Структура подпрограммы ClientHandler аналогична серверному варианту, но в данном приложении после обработки пакета происходит воспроизведение звука телеграфии или переданного буфера в функции playAudio().

В функции processPacket происходит идентификация переменной packettype и выполнение воспроизведения звука телеграфа или же звука пользователя, отправившего сообщение. Но перед непосредственным воспроизведением клиент получает данные о частоте, на которой была получена транзакция, сравнение её с частотой приёмника и принятием решения об игнорировании или обработки данных.

Для передачи и приёма всех данных по сети используется функции:

– send(SOCKET s, const char \*buf, int len, int flags);

– recv(SOCKET s, char \*buf, int len, int flags);

Несложно заметить, что они принимают одинаковые параметры, но работают по-разному. Первым параметром передаётся переменная типа SOCET, отвечающая за то, куда отправить или откуда будут получены требуемые значения. Второй параметр характеризует указатель на буфер, который следует передать или же получить. Третий – длина в байтах данных в буфере, на которые указывает параметр buf. И последний – набор флагов, которые определяют способ выполнения вызова. Этот параметр создается с помощью побитового оператора ИЛИ с любым из следующих значений:

– MSG\_DONTROUTE указывает, что данные не должны подлежать маршрутизации;

* MSG\_OOB отправляет данные OOB;

Если ошибки не возникает, send возвращает общее количество отправленных байтов, которое может быть меньше, чем число, запрошенное для отправки в параметре len. В противном случае возвращается значение SOCKET\_ERROR, и конкретный код ошибки можно получить, вызвав функцию WSAGetLastError().

Следующие участки кода показывают примеры использования приёма и передачи данных по сети в программе.

int msg\_size;

int frequency;

recv(Connections[index], (char\*)& frequency, sizeof(int), NULL);

recv(Connections[index], (char\*)& msg\_size, sizeof(unsigned long

int), NULL);

vector<char> msg(msg\_size + 1, 0);

recv(Connections[index], msg.data(), msg\_size, NULL);

Этот участок кода в свою очередь принимает данные на сервере с целью дальней обработки и ретрансляции.

void sendAudio()

{

Packet packettype = P\_AUDIO;

int msg\_size = gBufferByteSize;

send(Connection, (char\*)&packettype, sizeof(Packet), NULL);

send(Connection, (char\*)&frequency, sizeof(int), NULL);

send(Connection, (char\*)&msg\_size, sizeof(unsigned long int),

NULL);

send(Connection, gRecordingBuffer, msg\_size, NULL);

Sleep(10);

}

А данная функция посылает голосовой буфер на сервер с клиентского приложения.

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Разработка онлайн тренажёра Р-159 велась с использованием системы контроля версий GitHub, позволившая сохранять состояние программы на каждом отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений. Также можно добавлять к каждому этапу новые файлы, или удалять устаревшие варианты. После накопления определённого количества групп изменений, их следует отправить на удалённый репозиторий, где видна вся история приложения и разница между каждым новым «коммитом».

На рисунке 3.1 изображён пример изменений, отображаемых на GitHub.

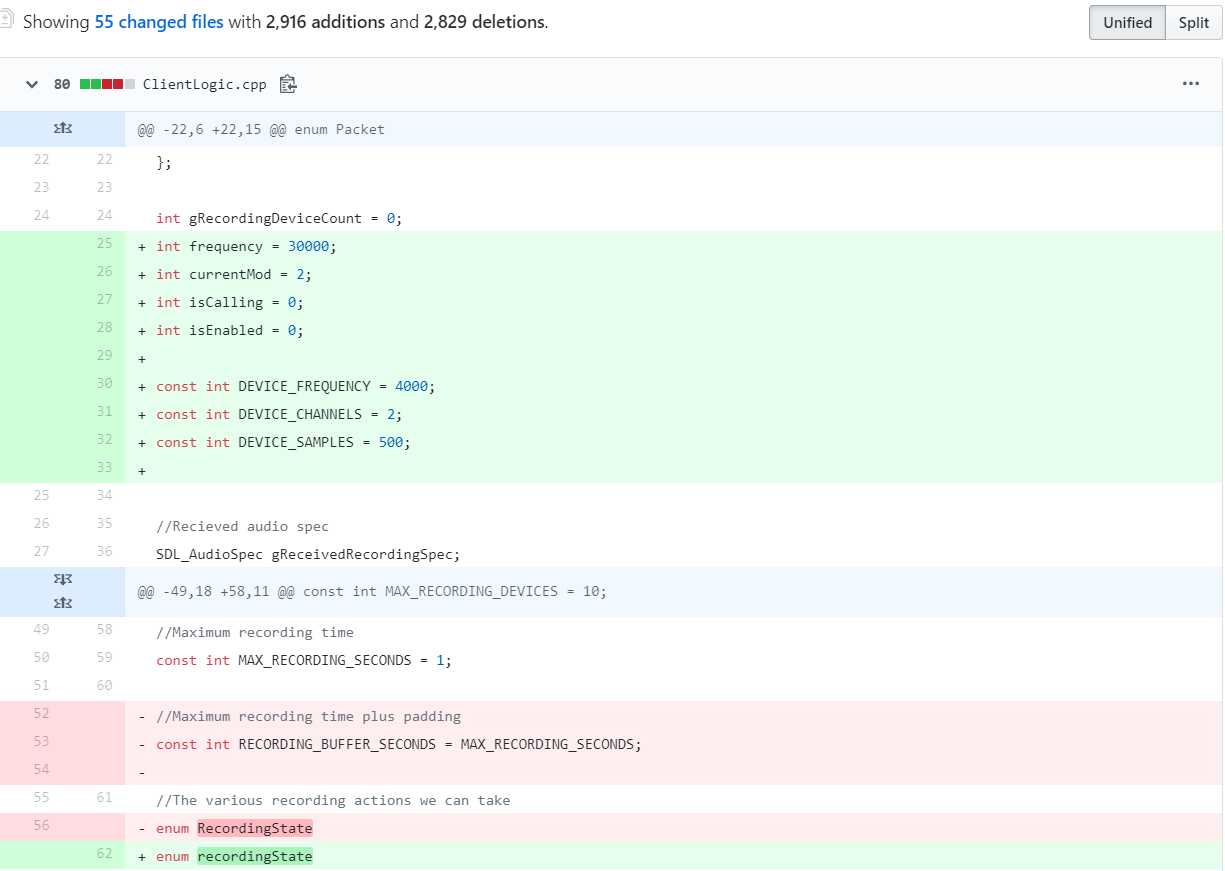


Рисунок 3.1 – Пример изменений, отображаемых на GitHub

В ходе тестирования приложения были выявлены некоторые недостатки интерфейса и сетевой логики программного средства. Была составлена таблица, показывающая ожидаемые и реальные результаты, полученные при заданных условия.

Таблица 1 - Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Передача аудиоданных на сервер и на клиент | Успешная передача данных за конечное время | Нераспознанный пакет с номером -858993460 |
| 2. | Подключение к серверу клиента | Успешное подключение | Успешное подключение |
| 3. | Смена рабочей частоты во время передачи данных | Изменение конечного получателя | Изменение конечного получателя |
| 4. | Смена режима передачи | Смена посылаемых данных | Смена посылаемых данных |
| 5. | Непрерывная запись голоса | Безотказная работа приложения | Ожидание окончания записи сообщения |
| 6. | Память, выделяемая для голосовых сообщений | < 1 мегабайт | 12 мегабайт |

Была обнаружена проблема при передаче аудиоданных на сервер и с него на клиент. Размер одного буфера составляет 32 килобайт, что приводит к небольшой задержке перед полной передачей, но изначально приложение не учитывало этого и попытка отправления данных с сервера приводила к ряду ошибок и некорректному воспроизведению. Эта проблема была решена добавлением задержки после отправки очередного сообщения. Исправленный участок кода принял следующий вид.

Packet msgtype = P\_AUDIO;

send(Connections[i], (char\*)& msgtype, sizeof(Packet), NULL);

send(Connections[i], (char\*)& frequency, sizeof(int), NULL);

send(Connections[i], (char\*)& msg\_size, sizeof(unsigned long

int), NULL);

send(Connections[i], msg.data(), msg\_size, NULL);

Sleep(900);

Второй сложностью стало обеспечение одновременной работы клиентского приложения и записи звуковых данных. Это приводило к ожиданию окончания записи, во время которого никак нельзя было взаимодействовать с интерфейсом. Было решено выполнять функцию записи в другом потоке. Добавленная строка кода представлена ниже.

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)initRecording, NULL, NULL, NULL);

Ещё одним недостатком стало огромное количество памяти, выделяемое для аудиоданных. Изначально планировалось записывать по 5 секунд непрерывно в хорошем качестве, а уже потом отправлять всё количество байт на сервер. Буфер занимал 12 мегабайт и очень сильно замедлял работу приложения как клиентского, так и серверного, а при большом количестве пользователей распознавание членораздельной речи стало невозможным. Решением стало использование менее ресурсоёмкого качества записи, что явилось более целесообразным для качества реально воспроизводимых звуков радиостанцией, а также запись в буфер по одной секунде. Исправленный код данного участка приведён ниже.

desiredRecordingSpec.freq = 4000;

desiredRecordingSpec.format = AUDIO\_F32;

desiredRecordingSpec.channels = 2;

desiredRecordingSpec.samples = 500;

desiredRecordingSpec.callback = audioRecordingCallback;

Остальные проблемы были связаны с недочётами на стадии проектирования и были устранены на стадии разработки.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Интерфейс программы

Интерфейс программного средства может показаться неудобным или неочевидным, но разработан на основе настоящей панели радиостанции. Такое решение позволяет получить максимум практических навыков в работе с радиостанцией и приведёт к минимуму ошибок при эксплуатации реальных аналогов.

Сервер следует запускать до работы в основном приложении. Для начала работы следует вызвать программу из командной строки с переданными значениями IP и порта. Пример ввода данных в консоли изображён на рисунке 4.1.

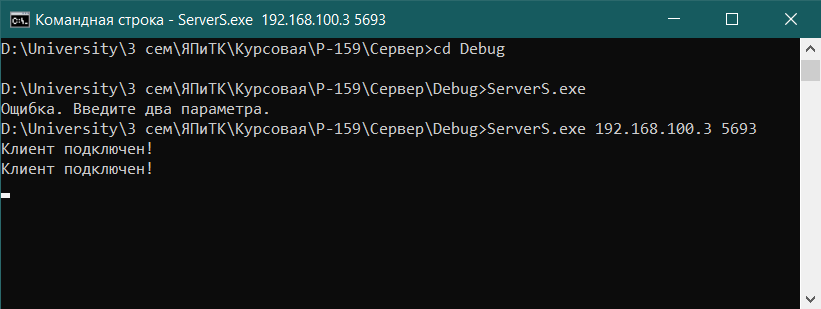


Рисунок 4.1 – Пример ввода данных в консоли

Всегда можно получить информацию по вводимым значения при мощи запуска программы с флагом «/?». При некорректном предоставлении данных будет выведено сообщение с просьбой верифицировать адрес и порт.

Сервер уведомит о получении неправильных данных или безуспешном подключении пользователя, такой случай представлен на рисунке 4.2.

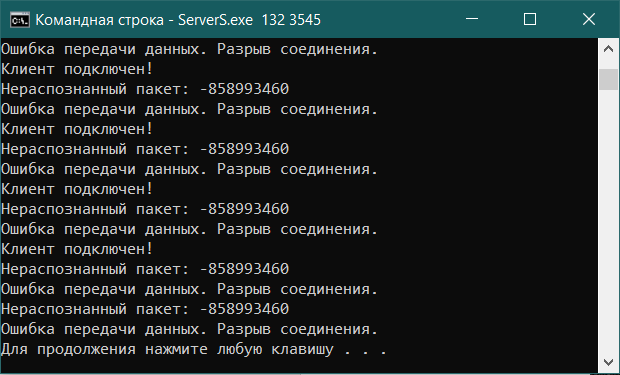


Рисунок 4.2 – Случай уведомления сервера об ошибке

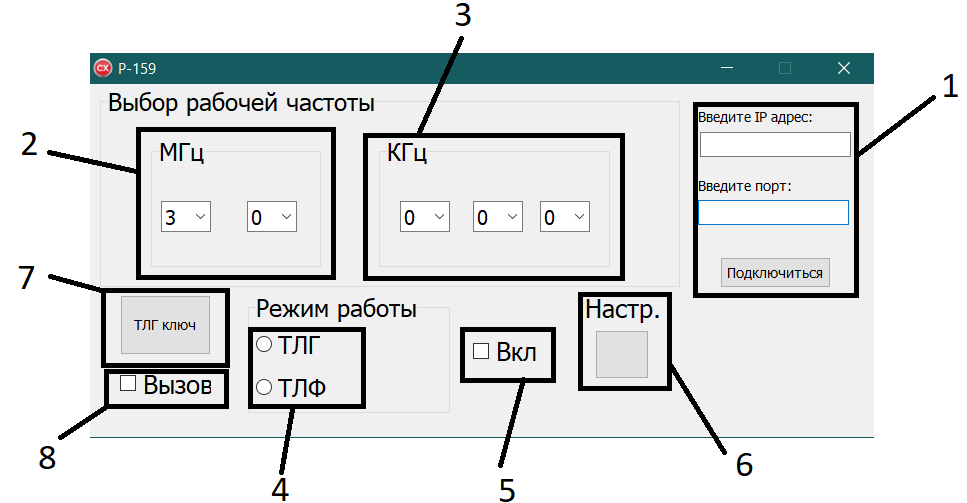


Рисунок 4.3 – Главное окно программы

Главное окно программы, изображенное на рисунке 4.3, содержит следующие элементы:

1. Поля ввода IP адреса и порта, кнопка «Подключится» организуют подключение к серверу по заданным значениям;
2. Поля выбора текущей рабочей частоты в десятках и единицах мегагерц соответственно;
3. Поля настройки частоты в сотнях, десятках и единицах килогерц;
4. Выбор режима работы: телеграфный или телефонный;
5. Включение и выключение тренажёра;
6. Настройка рабочей частоты;
7. Нажатие телеграфного ключа;
8. Переключение режимов вызов и приём в телефонном режиме;

При запуске тренажёра происходит инициализация ресурсов, которая может сопровождаться исключениями, если система не поддерживает какой-либо компонент или библиотеку. Когда происходит ошибка, вызовы останавливаются, а пользователь уведомляется о соответствующей ошибке, пример которой изображён на рисунке 4.4.

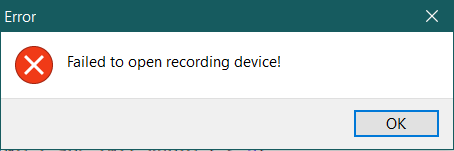


Рисунок 4.4 – Пример ошибки при загрузке записывающего устройства

* 1. Порядок настройки

Существуют принятые правила использования радиостанций, их подготовки, выставления начального состояния и настройки, которыми следует руководствоваться при эксплуатации реальных прототипов. Это связано с внутренним устройством настоящего устройства, которое подвержено износу. Соблюдение правил позволить продлить работу радиостанции и избежать неожиданностей.

Начальное состояние:

* checkBox «Вкл» в выключенном состоянии;
* checkBox «Вызов» в выключенном состоянии.

Порядок настройки:

1. В ввода IP адреса и порта ввести корректны данные о сервере;
2. Нажать на кнопку «Подключиться»;
3. Выбрать предпочитаемую частоту;
4. Выбрать режим передачи;
5. Нажать на «Вкл»;
6. Нажать на кнопку «Настр.»

После этого приложение готово к передаче данных, если требуется предавать сообщения голосом, то переключить «Вызов» в активное состояние в соответствующем режиме, при этом приём как телефонных сообщений, так и телеграфных будет невозможен. Для приёма аудиоданных и их воспроизведения нужно отключить «Вызов», а для передачи и приёма телеграфных сообщений, выбрать режим «ТЛГ».

ЗАКЛЮЧЕНИЕ

Развитие телекоммуникаций не прекращает расти, человечество с каждым годом достигает новых скоростей и качества передачи данных, но новые знания вырабатываются на основе предыдущих. Поэтому так важно не только знать теоретические основы радиосвязи, но и применять их на практике.

В рамках данного курсового проекта было разработано программное средство «Онлайн-тренажёр Р-159», которое обеспечит полезное и увлекательное времяпрепровождение. Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

* одновременная работа большого числа пользователей;
* работа в телеграфном режиме;
* работа в телефонном режиме;
* выбор и передача рабочей частоты.

Для успешного выполнения всех поставленных целей потребовалось изучить библиотеку SDL, основы компьютерных систем и сетей, объектно-ориентированные возможности языка C++, изучить основные принципы данной парадигмы, а также освоить взаимодействия существующих компонентов.

Существует много возможностей для дальнейшего улучшения приложения. Одним из самых простых направлений является добавление подробного описания настройки и включения приложения, адаптация приложения для других операционных систем. Клиентская часть может быть дополнена другими радиостанциями, а серверная часть – интерфейсом с таблицей подключённых пользователей, их позывных и качества сети.

Использование данного приложения позволит не только провести время с пользой, но и стимулирует работу памяти, а также даст полезные навыки в работе с радиостанцией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке C++ / Гирберд Шилдт. Учебный курс: / Гирберд Шилдт. – СПб: изд. С. В. Малгачёва, 2001. – 231 с;
2. RAD Studio Product Documentation – Embarcadero Technologies [Электронный ресурс]. – Электронные данные. – Режим доступа: http://docs.embarcadero.com/products/rad\_studio;
3. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003;
4. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
5. Windows Sockets 2 – Win32 apps | Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/\_winsock/;
6. Культин, Н. А. Основы программирования в C++ 2-е издание – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 182с;
7. Культин, Н. А. Основы программирования в C++ – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 167с;
8. Хлебостроев, В. С. Программирование графики в Rad Studio – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 243с;
9. Графические возможности Rad Studio (электронный ресурс). – Электронные данные. – Режим доступа: http://embarcadero.gym5cheb.ru.

ПРИЛОЖЕНИЕ А

Исходный код программы

**MainUnit.h**

#ifndef MainUnitH

#define MainUnitH

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

#include <Vcl.ExtCtrls.hpp>

#include <Vcl.CheckLst.hpp>

#include <Vcl.Dialogs.hpp>

#include "SDL/include/SDL.h"

#include <Vcl.MPlayer.hpp>

#include "ClientLogic.h"

class TMainForm : public TForm

{

\_\_published: // IDE-managed Components

TGroupBox \*GroupBox1;

TRadioGroup \*rgMod;

TButton \*Button;

TGroupBox \*gbMH;

TGroupBox \*gbKH;

TComboBox \*cbMTenth;

TComboBox \*cbMOnes;

TComboBox \*cbKHundred;

TComboBox \*cbKTenth;

TComboBox \*cbKOnes;

TButton \*btnSetFrequency;

TCheckBox \*cbEnable;

TLabel \*Label1;

TCheckBox \*CheckBox2;

TEdit \*IPEdt;

TEdit \*PortEdt;

TLabel \*Label2;

TLabel \*Label3;

TButton \*ConnectBtn;

void \_\_fastcall FormCreate(TObject \*Sender);

void \_\_fastcall ButtonClick(TObject \*Sender);

void \_\_fastcall rgModClick(TObject \*Sender);

void \_\_fastcall cbEnableClick(TObject \*Sender);

void \_\_fastcall CheckBox2Click(TObject \*Sender);

void \_\_fastcall btnSetFrequencyClick(TObject \*Sender);

void \_\_fastcall ConnectBtnClick(TObject \*Sender);

private: // User declarations

public: // User declarations

\_\_fastcall TMainForm(TComponent\* Owner);

};

extern PACKAGE TMainForm \*MainForm;

#endif

**MainUnit.cpp**

#include "stdafx.h"

#pragma comment(lib, "ws2\_32.lib")

#include <winsock2.h>

#include <iostream>

#include <stdio.h>

#include <string>

#include <sstream>

#pragma warning(disable: 4996)

#include <vcl.h>

#pragma hdrstop

#include "MainUnit.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TMainForm \*MainForm;

\_\_fastcall TMainForm::TMainForm(TComponent\* Owner)

: TForm(Owner)

{

}

void \_\_fastcall TMainForm::FormCreate(TObject \*Sender)

{

if(SDL\_Init( SDL\_INIT\_VIDEO | SDL\_INIT\_AUDIO ) < 0)

{

MessageDlg("Failed to initialize SDL!", mtError,

TMsgDlgButtons()<<mbOK, 0);

}

else

{

//Load media

if( !loadMedia() )

{

MessageDlg("Failed to load media!", mtError,

TMsgDlgButtons()<<mbOK, 0);

}

else

{

//Device failed to open

if( loadRecordingDevice() )

{

//Report error

MessageDlg("Failed to open recording device!",

mtError, TMsgDlgButtons()<<mbOK, 0);

}

//Device opened successfully

else

{

//Device failed to open

if( loadPlaybackDevice() )

{

//Report error

MessageDlg("Failed to open playback

device!", mtInformation,

TMsgDlgButtons() << mbOK, 0);

}

//Device opened successfully

else

{

initBuffer();

}

}

}

}

}

void \_\_fastcall TMainForm::ButtonClick(TObject \*Sender)

{

sendTLG();

}

void \_\_fastcall TMainForm::rgModClick(TObject \*Sender)

{

switch (rgMod->ItemIndex)

{

case TLG:

{

toggleMod(TLG);

break;

}

case TLF:

{

toggleMod(TLF);

break;

}

}

}

void \_\_fastcall TMainForm::cbEnableClick(TObject \*Sender)

{

toggleEnabled();

}

void \_\_fastcall TMainForm::CheckBox2Click(TObject \*Sender)

{

toggleCalling();

}

void \_\_fastcall TMainForm::btnSetFrequencyClick(TObject \*Sender)

{

frequency = cbMTenth->Text.ToInt() \* 10000 + cbMOnes-\ >Text.ToInt() \* 1000 + cbKHundred->Text.ToInt() \* 100 +

cbKTenth->Text.ToInt() \* 10 + cbKOnes->Text.ToInt() \* 1;

}

void \_\_fastcall TMainForm::ConnectBtnClick(TObject \*Sender)

{

connectToServer(AnsiString(IPEdt->Text).c\_str(), PortEdt-

>Text.ToInt());

}

**ClientLogic.h**

#include "MainUnit.h"

#ifndef ClientLogicH

#define ClientLogicH

SOCKET Connection;

SOCKET ConnectSocket = INVALID\_SOCKET;

const int TLG = 0;

const int TLF = 1;

int isEnabled = 0;

int isCalling = 0;

int currentMod = 2;

int frequency = 30000;

int connectToServer(char \*IP, int Port);

bool loadMedia();

bool loadRecordingDevice();

bool loadPlaybackDevice();

void sendTLG();

void initBuffer();

void ClientHandler();

void toggleEnabled();

void toggleCalling();

void toggleMod(int mod);

#endif

**ClientLogic.cpp**

#include "stdafx.h"

#pragma comment(lib, "ws2\_32.lib")

#include <winsock2.h>

#include <iostream>

#include <stdio.h>

#include <string>

#include <sstream>

#pragma warning(disable: 4996)

#pragma hdrstop

#include "ClientLogic.h"

#pragma package(smart\_init)

//Number of available devices

enum Packet

{

P\_KEY,

P\_AUDIO

};

int gRecordingDeviceCount = 0;

int frequency = 30000;

int currentMod = 2;

int isCalling = 0;

int isEnabled = 0;

const int DEVICE\_FREQUENCY = 4000;

const int DEVICE\_CHANNELS = 2;

const int DEVICE\_SAMPLES = 500;

//Recieved audio spec

SDL\_AudioSpec gReceivedRecordingSpec;

SDL\_AudioSpec gReceivedPlaybackSpec;

//Recording data buffer

Uint8\* gRecordingBuffer = NULL;

//Size of data buffer

Uint32 gBufferByteSize = 0;

//Position in data buffer

Uint32 gBufferBytePosition = 0;

//Maximum position in data buffer for recording

Uint32 gBufferByteMaxPosition = 0;

//Audio device IDs

SDL\_AudioDeviceID recordingDeviceId = 0;

SDL\_AudioDeviceID playbackDeviceId = 0;

//Maximum number of supported recording devices

const int MAX\_RECORDING\_DEVICES = 10;

//Maximum recording time

const int MAX\_RECORDING\_SECONDS = 1;

//The various recording actions we can take

enum recordingState

{

START\_RECORDING,

RECORDING

};

void audioRecordingCallback( void\* userdata, Uint8\* stream, int

len )

{

//Copy audio from stream

memcpy( &gRecordingBuffer[ gBufferBytePosition ], stream, len );

//Move along buffer

gBufferBytePosition += len;

}

void audioPlaybackCallback( void\* userdata, Uint8\* stream, int

len)

{

//Copy audio to stream

memcpy( stream, &gRecordingBuffer[ gBufferBytePosition ], len );

//Move along buffer

gBufferBytePosition += len;

}

SDL\_AudioSpec getAudioDevice(SDL\_AudioSpec device)

{

SDL\_zero(device);

device.freq = DEVICE\_FREQUENCY;

device.format = AUDIO\_F32;

device.channels = DEVICE\_CHANNELS;

device.samples = DEVICE\_SAMPLES;

device.callback = audioRecordingCallback;

return device;

}

bool loadRecordingDevice()

{

//Get selection index

const int index = 0;

SDL\_AudioSpec desiredRecordingSpec =

getAudioDevice(desiredRecordingSpec);

//Open recording device

recordingDeviceId = SDL\_OpenAudioDevice(

SDL\_GetAudioDeviceName( index, SDL\_TRUE ), SDL\_TRUE, &desiredRecordingSpec, &gReceivedRecordingSpec, SDL\_AUDIO\_ALLOW\_FORMAT\_CHANGE );

return recordingDeviceId == 0;

}

bool loadPlaybackDevice()

{

SDL\_AudioSpec desiredPlaybackSpec = getAudioDevice(desiredPlaybackSpec);

//Open playback device

playbackDeviceId = SDL\_OpenAudioDevice( NULL, SDL\_FALSE, &desiredPlaybackSpec, &gReceivedPlaybackSpec, SDL\_AUDIO\_ALLOW\_FORMAT\_CHANGE );

return playbackDeviceId == 0;

}

void sendAudio()

{

Packet packettype = P\_AUDIO;

int msg\_size = gBufferByteSize;

send(Connection, (char\*)&packettype, sizeof(Packet), NULL);

send(Connection, (char\*)&frequency, sizeof(int), NULL);

send(Connection, (char\*)&msg\_size, sizeof(unsigned long int),

NULL);

send(Connection, gRecordingBuffer, msg\_size, NULL);

Sleep(10);

}

void recording()

{

//Lock callback

SDL\_LockAudioDevice( recordingDeviceId );

//Finished recording

if( gBufferBytePosition > gBufferByteMaxPosition )

{

//Stop recording audio

SDL\_PauseAudioDevice( recordingDeviceId, SDL\_TRUE );

//Go on to next state

gBufferBytePosition = 0;

//Start playback

//SDL\_PauseAudioDevice( playbackDeviceId, SDL\_FALSE );

if (currentMod == 1 && isCalling && isEnabled)

{

sendAudio();

}

}

//Unlock callback

SDL\_UnlockAudioDevice( recordingDeviceId );

}

void startRecording()

{

//Go back to beginning of buffer

gBufferBytePosition = 0;

//Start recording

SDL\_PauseAudioDevice( recordingDeviceId, SDL\_FALSE );

}

int initRecording()

{

//Main loop flag

bool quit = false;

//Set the default recording state

recordingState currentState = START\_RECORDING;

//While application is running

while( !quit )

{

//Do current state event handling

switch( currentState )

{

//User getting ready to record

case START\_RECORDING:

{

startRecording();

//Go on to next state

currentState = RECORDING;

break;

}

case RECORDING:

{

recording();

//Go on to next state

currentState = START\_RECORDING;

break;

}

}

}

return 0;

}

void initBuffer()

{

//Calculate per sample bytes

int bytesPerSample = gReceivedRecordingSpec.channels \* (

SDL\_AUDIO\_BITSIZE( gReceivedRecordingSpec.format ) / 8 );

//Calculate bytes per second

int bytesPerSecond = gReceivedRecordingSpec.freq \*

bytesPerSample;

//Calculate buffer size

gBufferByteSize = MAX\_RECORDING\_SECONDS \* bytesPerSecond;

//Calculate max buffer use

gBufferByteMaxPosition = MAX\_RECORDING\_SECONDS \*

bytesPerSecond;

//Allocate and initialize byte buffer

gRecordingBuffer = new Uint8[ gBufferByteSize ];

memset( gRecordingBuffer, 0, gBufferByteSize );

CreateThread(NULL, NULL, ( LPTHREAD\_START\_ROUTINE )

initRecording, NULL, NULL, NULL);

}

bool loadMedia()

{

//Loading success flag

bool success = true;

//Get capture device count

gRecordingDeviceCount = SDL\_GetNumAudioDevices( SDL\_TRUE );

//No recording devices

if( gRecordingDeviceCount < 1 )

{

MessageDlg("Unable to get audio capture device!",

mtError, TMsgDlgButtons()<<mbOK, 0);

success = false;

}

else //At least one device connected

{

//Cap recording device count

if( gRecordingDeviceCount > MAX\_RECORDING\_DEVICES )

{

gRecordingDeviceCount = MAX\_RECORDING\_DEVICES;

}

}

return success;

}

void playAudio()

{

int msg\_size;

recv(Connection, (char\*)&msg\_size, sizeof(unsigned long int),

NULL);

recv(Connection, gRecordingBuffer, msg\_size, NULL);

//Main loop flag

bool quit = false;

//Go on to next state

gBufferBytePosition = 0;

//Start playback

SDL\_PauseAudioDevice( playbackDeviceId, SDL\_FALSE );

while (!quit)

{

//Lock callback

SDL\_LockAudioDevice( playbackDeviceId );

//Finished playback

if( gBufferBytePosition > gBufferByteMaxPosition )

{

//Stop playing audio

SDL\_PauseAudioDevice( playbackDeviceId,SDL\_TRUE );

//Go on to next message

quit = true;

}

//Unlock callback

SDL\_UnlockAudioDevice( playbackDeviceId );

}

}

bool processPacket(Packet packettype) {

int incomFreq;

switch(packettype)

{

case P\_KEY:

{

recv(Connection, (char\*)&incomFreq, sizeof(int),

NULL);

if (currentMod == 0 && isEnabled && incomFreq ==

frequency)

{

PlaySound(TEXT("TLG.wav"), NULL, SND\_ASYNC);

break;

}

}

case P\_AUDIO:

{

recv(Connection, (char\*)&incomFreq, sizeof(int),

NULL);

if (currentMod == 1 && !isCalling && isEnabled &&

incomFreq == frequency)

{

playAudio();

break;

}

}

}

return true;

}

void sendTLG()

{

if (currentMod == 0 && isEnabled)

{

Packet packettype = P\_KEY;

send(Connection, (char\*)&packettype, sizeof(Packet),

NULL);

send(Connection, (char\*)&frequency, sizeof(int), NULL);

Sleep(10);

}

}

void clientHandler()

{

Packet packettype;

while(true)

{

recv(Connection, (char\*)&packettype, sizeof(Packet),

NULL);

Sleep(100);

if(!processPacket(packettype))

{

break;

}

}

closesocket(Connection);

}

void toggleEnabled()

{

isEnabled = !isEnabled;

}

void toggleMod(int mod)

{

currentMod = mod;

}

void toggleCalling()

{

isCalling = !isCalling;;

}

void setFrequency(int MTenth, int MOnes, int KHundred, int KTenth, int KOnes)

{

frequency = MTenth \* 10000 + MOnes \* 1000 + KHundred \* 100 + KTenth \* 10 + KOnes \* 1;

}

int connectToServer(char \*IP, int Port)

{

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if(WSAStartup(DLLVersion, &wsaData) != 0)

{

exit(1);

}

SOCKADDR\_IN addr;

int sizeofaddr = sizeof(addr);

addr.sin\_addr.s\_addr = inet\_addr(IP);

addr.sin\_port = htons(Port);

addr.sin\_family = AF\_INET;

Connection = socket(AF\_INET, SOCK\_STREAM, NULL);

if (connect(Connection, (SOCKADDR\*)&addr, sizeof(addr)) != 0)

{

exit(1);

}

CreateThread(NULL, NULL, ( LPTHREAD\_START\_ROUTINE )

clientHandler, NULL, NULL, NULL);

}

**ServerS.cpp**

#include "stdafx.h"

#pragma comment(lib, "ws2\_32.lib")

#include <winsock2.h>

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#pragma warning(disable: 4996)

SOCKET connections[100];

int counter = 0;

using namespace std;

enum Packet

{

P\_KEY,

P\_AUDIO

};

bool sendTelegraph(int index)

{

int frequency;

try

{

recv(connections[index], (char\*)& frequency,

sizeof(int), NULL);

}

catch (...)

{

return false;

}

for (int i = 0; i < counter; i++)

{

if (i == index)

{

continue;

}

Packet msgtype = P\_KEY;

send(connections[i], (char\*)& msgtype, sizeof(Packet),

NULL);

send(connections[i], (char\*)& frequency, sizeof(int),

NULL);

}

return true;

}

bool sendAudio(int index)

{

int msg\_size;

int frequency;

try

{

recv(connections[index], (char\*)& frequency,

sizeof(int), NULL);

recv(connections[index], (char\*)& msg\_size,

sizeof(unsigned long int), NULL);

}

catch(...)

{

return false;

}

vector<char> msg(msg\_size + 1, 0);

try

{

recv(connections[index], msg.data(), msg\_size, NULL);

}

catch (...)

{

return false;

}

for (int i = 0; i < counter; i++)

{

if (i == index)

{

continue;

}

Packet msgtype = P\_AUDIO;

send(connections[i], (char\*)& msgtype, sizeof(Packet),

NULL);

send(connections[i], (char\*)& frequency, sizeof(int),

NULL);

send(connections[i], (char\*)& msg\_size, sizeof(unsigned

long int), NULL);

send(connections[i], msg.data(), msg\_size, NULL);

Sleep(900);

}

return true;

}

bool processPacket(int index, Packet packettype)

{

switch (packettype)

{

case P\_KEY:

{

return sendTelegraph(index);

}

case P\_AUDIO:

{

return sendAudio(index);

}

default:

{

cout << "Нераспознанный пакет: " << packettype << endl;

return false;

}

}

}

void сlientHandler(int index)

{

Packet packettype;

while (true)

{

recv(connections[index], (char\*)& packettype, sizeof(Packet), NULL);

if (!processPacket(index, packettype))

{

cout << "Ошибка передачи данных. Разрыв соединения." << endl;

break;

}

}

closesocket(connections[index]);

}

bool checkParams(int argc, char\* argv[])

{

for (int i = 0; i < argc; i++)

{

if (!strcmp(argv[i], "/?"))

{

cout << "Для старта сервера передайте первыми двумя параметрами данные IP и порта.";

return false;

}

}

if (argc < 3)

{

cout << "Ощибка. Введите два параметра.";

return false;

}

return true;

}

bool initWSA()

{

WSAData wsaData;

WORD DLLVersion = MAKEWORD(2, 1);

if (WSAStartup(DLLVersion, &wsaData) != 0)

{

cout << "Ошибка инициализации WSA!" << endl;

return false;

}

return true;

}

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

{

setlocale(LC\_ALL, "Russian");

if (!checkParams(argc, argv) || !initWSA())

{

return 0;

}

SOCKADDR\_IN addr;

int sizeofaddr = sizeof(addr);

try

{

addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

addr.sin\_port = htons(stoi(argv[2]));

}

catch (...)

{

cout << "Ошибка. Введите корректные данные IP и порта." << endl;

return 0;

}

addr.sin\_family = AF\_INET;

SOCKET sListen = socket(AF\_INET, SOCK\_STREAM, NULL);

bind(sListen, (SOCKADDR\*)&addr, sizeof(addr));

listen(sListen, SOMAXCONN);

SOCKET newConnection;

for(int i = 0; i < 100; i++)

{

newConnection = accept(sListen, (SOCKADDR\*)&addr, &sizeofaddr);

if(newConnection == 0)

{

cout << "Ошибка подключения клиента.";

}

else

{

cout << "Клиент подключен!\n";

connections[i] = newConnection;

counter++;

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)сlientHandler, (LPVOID)(i), NULL, NULL);

}

}

system("pause");

return 0;

}