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-ВЕБИНАР»**

|  |  |  |
| --- | --- | --- |
| Студент |  | И.С. Андросов |
| Руководитель |  | Д.В. Деменковец |

Минск 2020

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) 13

[2.4 Захват экрана](#_Toc513665888) 16

[2.5 Передача данных по сети](#_Toc513665889) 17

[2.6 Работа с реестром](#_Toc513665889) 20

[3 Тестирование программного средства](#_Toc513665890) 21

[3.1 Тестирование функционала приложения](#_Toc513665889) 21

[4 Руководство пользователя](#_Toc513665891) 23

[4.1 Порядок использования](#_Toc513665893) 23

[Заключение](#_Toc513665894) 25

[Список использованных источников](#_Toc513665895) 26

[Приложение А. Исходный код программы](#_Toc513665896) 27

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 образования.

Онлайн-семинар или веб-конференция, вебинар – разновидность веб-конференции, проведение онлайн-встреч или презентаций через интернет. Во время веб-конференции каждый из участников находится у своего компьютера, а связь между ними поддерживается через Интернет посредством загружаемого приложения, установленного на компьютере каждого участника, или через веб-приложение. В последнем случае, чтобы присоединиться к конференции, нужно просто ввести URL в окне браузера.

В первые годы после появления Интернета термином «веб-конференция» часто называли ветку форума или доски объявлений. Позже термин получил значение общения именно в режиме реального времени.

История вебинаров берет свое начало в конце 1980-х годов. Именно тогда, в самом начале истории интернета, появились предшественники вебинаров – первые системы текстового общения в режиме реального времени. В середине 1990-х возникли веб-чаты и программы для мгновенного обмена сообщениями.

В 1995-м году компания PictureTel объявила о выпуске программы LiveShare Plus. Приложение позволяло демонстрировать экран, предоставлять другому пользователю удаленный доступ к компьютеру, передавать файлы и обмениваться текстовыми сообщениями.

Первые публичные веб-конференции стали доступны в мае 1996-го года благодаря компании Microsoft, которая анонсировала запуск NetMeeting. Программа представляла собой встроенный компонент для браузера Internet Explorer 3.0. NetMeeting позволил пользователям общаться и обмениваться данными в режиме реального времени.

Первое программное обеспечение для вебинаров, PlaceWare, вышло в том же году. Оно было выпущено компанией Xerox PARC. Программа позволяла одному или нескольким пользователям выступать с презентацией, на которой одновременно могли присутствовать сотни или даже тысячи посетителей со всего мира. Кроме того, PlaceWare предоставляла ряд других возможностей: проводить опросы аудитории, общаться в приватном чате, подключать к эфиру участников вебинара.

В 1998-м году Эрик Р. Корб зарегистрировал торговый знак «webinar», но позже это решение было оспорено в суде. Сейчас торговый знак «webinar» принадлежит компании InterCall.

В 1999-м была разработана программа WebEx Meeting Center, вскоре переименованная в WebEx. Она давала возможность проводить вебинары одновременно для ольшого числа слушателей.

С 2000-го года на рынке начало появляться все больше и больше сервисов для проведения вебинаров: GoToMeeting, GoToWebinar, ClickWebinar, MyOwnConference. С каждым годом в программах для вебинаров совершенствовались функции, появлялось больше возможностей. В настоящее время вебинар используется в рамках системы дистанционного обучения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Обзор аналогов

На данный момент существует множество вариаций приложений для вебинаров: с огромным числом пользователей, cвозможностью создания приватных конференций, с возможностью изменения заднего плана при передаче изображения с веб-камеры.

Одним из самых популярных является приложение Zoom. Свою популярность данная программа получила в целях удалённой работы, дистанционного обучения и социального общения с использованием интернета. Внешний вид данного программного средства представлен на рисунке 1.1.

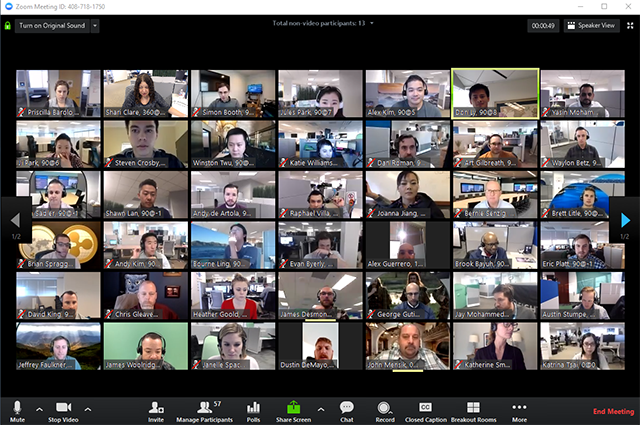


Рисунок 1.1 – Внешний вид программного средства Zoom

Данное приложение таже имеет много вопросов с безопасностью. В ноябре 2018-го года была обнаружена уязвимость безопасности, которая позволила удалённому неавторизованному злоумышленнику подделать UDP сообщения от участника собрания или сервера Zoom, чтобы задействовать функциональность целевого клиента. Это позволит злоумышленнику удалять посетителей с собраний, подделывать сообщения от пользователей или перехватывать общие экраны.

В июле 2019-го года исследователь безопасности Джонатан Лейтшух раскрыл уязвимость нулевого дня, которая позволяет любому веб-сайту принудительно присоединять пользователя macOS к вызову Zoom с активированной видеокамерой без разрешения пользователя.

Уязвимость нулевого дня – термин, обозначающий неустранённые уязвимости, а также вредоносные программы, против которых ещё не разработаны защитные механизмы. Сам термин означает, что у разработчиков было ноль дней на исправление дефекта: уязвимость или атака становится публично известна до момента выпуска производителем ПО исправлений ошибки, то есть потенциально уязвимость может эксплуатироваться на работающих копиях приложения без возможности защититься от неё.

После попытки удаления клиента Zoom в macOS программное обеспечение будет автоматически переустанавливаться в фоновом режиме, используя скрытый веб-сервер, который был установлен на компьютере во время первой установки и который остаётся активным после попытки удаления клиента. Zoom оперативно закрыл эту уязвимость.

Другим аналогом является линейка продуктов GoToMeeting. GoToMeeting – это веб-сервис, созданный компанией Citrix Online, являющейся подразделением корпорации Citrix Systems. Программное обеспечение предназначено для проведения веб-конференций и для удаленного просмотра рабочего стола, и позволяет пользователям общаться с другими людьми через интернет в режиме реального времени. GoToMeeting включает в себя следующие решения: GoToMeeting, GoToWebinar и GoToMeeting Corporate.

GoToMeeting – это базовое решение, позволяющее создавать конференции, в которых принимают участие до 15-ти человек. Компания предоставляет возможность бесплатно использовать этот продукт в течение испытательного периода. В отличие от GoToMeeting, GoToWebinar позволяет создавать вебинары для 1000-чи человек. Данное решение также может бесплатно использоваться в течение испытательного периода. GoToMeeting – это наиболее функциональное решение, позволяющее создавать как конференции до 25-ти человек, так и вебинары до 1000-чи человек. Кроме этого, в рамках решения предоставлены дополнительные средства для администрирования и управления пользователями. Данная линейка решений обладает удобным пользовательским интерфейсом.

Ниже представлены иллюстрации линейки продуктов GoToMeeting:

– на рисунке 1.2 окно конференции GoToMeeting;

– на рисунке 1.3 панель настроек в GoToMeeting.

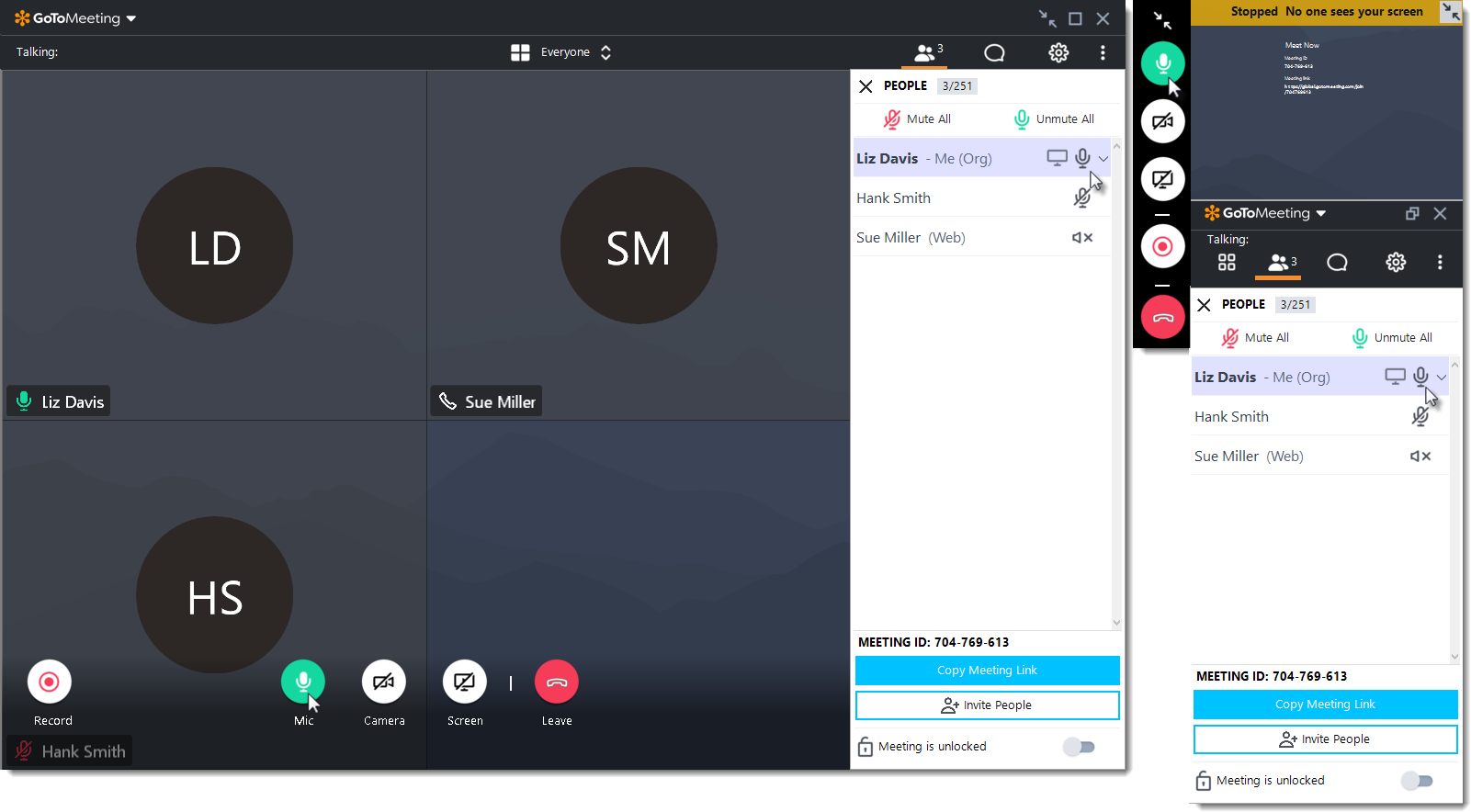


Рисунок 1.2 – Окно конференции GoToMeeting

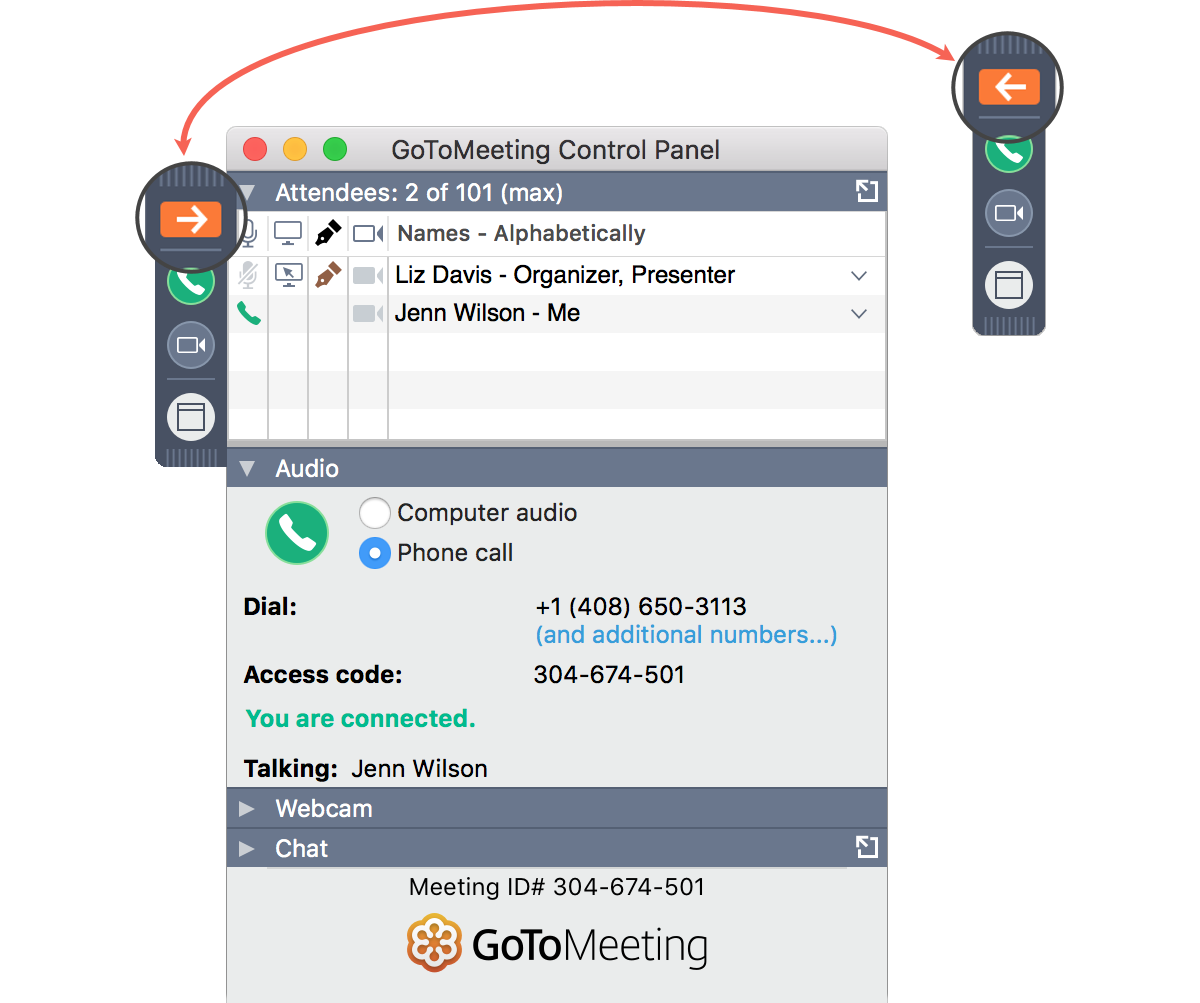


Рисунок 1.3 – Панель настроек в GoToMeeting

У вебинаров, как и у обычных семинаров, есть свои плюсы и минусы. Плюсы вебинара:

* высокая степень интерактивности – слушатели вовлечены в процесс обучения, задают вопросы для пояснения непонятных моментов, на которые сразу же получают ответы;
* в отличии от обычного семинара, слушатели вебинара по его окончании получают запись, которую позже можно использовать при закреплении материала. Запись выдается даже тем, кто не успел к нему подключиться;
* значительная экономия времени. Участникам и лектору не надо тратить время на дорогу к месту проведения семинара. Ко всему прочему они могут находиться в разных городах или даже странах. Присутствовать на вебинаре пользователи могут, где удобно;
* экономия средств. Стоимость обучения на вебинаре намного ниже открытого семинара. Помимо этого, нет необходимости тратиться на дорогу, питание и проживание. А затраты на организацию онлайн-семинара – минимальны и связаны с организацией рабочего места;
* Возможность соблюдения анонимности. Участники одного и того же вебинара могут не знать о личностях друг друга.

Минусы вебинара:

* как бы не старался лектор, но во время вебинара не достигается такая же эмоциональная связь, как во время живого общения. А это довольно важный аспект, от которого зависит эффективность обучения;
* так как вебинар – это новый инструмент для проведения тренингов и конференций, многие люди, привыкшие к работе с живой аудиторией, просто теряются перед монитором. В результате этого пропадает ритм выступления;
* иногда случается, что участники и лектор находятся в разных часовых поясах. Если для лектора вебинар начинается в 14:00, то для некоторых участников может начинаться в 23:00, что не совсем удобно.
  1. Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Онлайн-вебинар».

Исходя из рассмотренных аналогов можно определить основные требования к проекту.

В программном средстве планируется реализовать следующие функции:

* создание комнаты;
* подключение к комнате;
* возможность выхода и повторного подключения.

Логическая составляющая приложения будет включать в себя:

* алгоритмы захвата изображения;
* алгоритмы записи звука;
* отправку текстовых сообщений;
* передачу данных по сети;
* синхронизацию передачи данных.

Для разработки программного средства будет использоваться язык программирования С++ с использованием возможностей Windows API и среда разработки Visual Studio.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура программы

В данном приложении необходимо использовать следующие структурные блоки:

– Reg.h – файл заголовка, содержащий описания функций для работы с реестром с целью сохранения имени пользователя;

– StdAfx.h – файл заголовка, подключающий необходимые системные файлы;

– Trace.h – файл заголовка, содержащий макрос описания ошибок;

– Macro.h – файл заголовка, содержащий объявления макросов для обработки результатов функций WSAPI, используемых для взаимодействия со звуковыми устройствами;

– Role.h – файл заголовка, содержащий информацию о имени пользователя и его роли;

– Client.h – файл заголовка, содержащий структуру клиента: сокеты для предачи сообщений, звука и видео;

– Convert.h – файл заголовка, содержащий класс для преобразования string в wstring;

– WaveWriter.h – файл заголовка, содержащий класс описания файла, в котором собирается информация о структуре звука и звуковых данных.

– WindowControl.h – файл заголовка, описывающий класс, который реализуют все окна и элементы управления программы;

– Button.h – файл заголовка, содержащий класс кнопки;

– Label.h – файл заголовка, содержащий класс метки;

– TextBox.h – файл заголовка, содержащий класс поля ввода;

– Choice.h – файл заголовка, содержащий класс окна выбора роли пользователя;

– CreateRoomWindow.h – файл заголовка, содержащий класс окна создания комнаты;

– JoinRoomWindow.h – файл заголовка, содержащий класс окна подключения к комнате;

– RoomWindow.h – файл заголовка, содержащий класс главного окна вебинара, где и происходит взаимодействие между лектором и другими участника конференции;

* 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.

Для удобства разработки пользовательского интерфейса был создан класс WindowControl с полями имени, идентификатора и дескриптора окна. Также был добавлен вектор событий и методы для его обработки.

В качестве навигации был выбран очень простой и понятный большинству пользователей компонент Button. Компонент Label отвечает за отображение текста, а TextBox за возможность ввода информации, например, имя пользователя или сообщения в чате.

На рисунке 2.1 представлено отображение стартового окна программы.

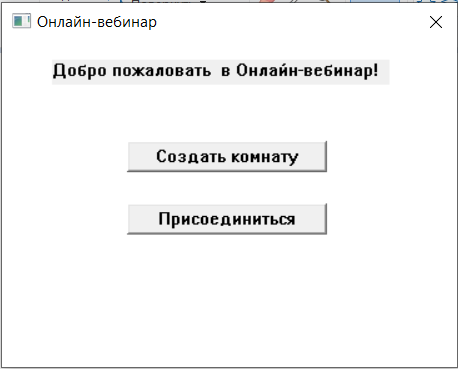


Рисунок 2.1 – Стартовое окно программы

* 1. Работа со звуковыми ресурсами

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

Управление звуком реализовано при помощи Windows Audio Session API или WASAPI. Файлы заголовков Audioclient.h и Audiopolicy.h определяют данный программный интерфейс.

WASAPI состоит из нескольких интерфейсов. Первый из них – это интерфейс IAudioClient. Чтобы получить доступ к интерфейсам WASAPI, нужно сначала получить ссылку на интерфейс IAudioClient конечного аудиоустройства, вызывая метод Activate с параметром iid, установленным на REFIID IID\_IAudioClient. Для инициализации потока на конечном устройстве происходит вызов метода Initialize. После инициализации потока можно получить ссылки на другие интерфейсы WASAPI, вызвав метод GetService.

Многие методы в WASAPI возвращают код ошибки, если конечное аудиоустройство, используемое клиентским приложением, становится недействительным. Именно поэтому были разработаны модули Macro.h и Trace.h. Например, макрос IF\_FAILED\_THROW выбрасывает ошибку, если вызов метода WASAPI возвращает код ошибки. Макрос SAFE\_RELEASE освобождает память, выделенную под объекты WASAPI.

При старте вебинара запускается поток ThreadSoundRecorder, который в свою очередь вызывает в цикле функцию RecordAudioStream, где и происходит запись звука. На очередном шаге цикла звуковые данные отправляются клиентам.

Чтобы получить размер выделенного буфера, который может отличаться от запрошенного размера, клиент вызывается метод GetBufferSize.

Чтобы переместить поток захваченных данных через буфер конечного устройства, поочередно вызывается метод GetBuffer и метод ReleaseBuffer. Доступ к данным в буфере конечного устройства производится в виде серии пакетов данных. Вызов GetBuffer извлекает из буфера следующий пакет захваченных данных. После чтения данных из пакета происходит вызов ReleaseBuffer, чтобы освободить пакет и сделать его доступным для других захваченных данных.

Размер пакета может варьироваться от одного вызова GetBuffer к другому. Перед вызовом GetBuffer происходит вызов метода GetNextPacketSize, чтобы заранее получить размер следующего пакета. Кроме того, клиент может вызвать метод GetCurrentPadding, чтобы получить общий объем захваченных данных, доступных в буфере. В любой момент размер пакета всегда меньше или равен общему объему захваченных данных в буфере.

Функция RecordAudioStream выделяет общий буфер длительностью в половину секунды. Выделенный буфер может иметь немного большую продолжительность. В основном цикле вызов функции Sleep заставляет программу ждать некоторое время. В начале каждого вызова Sleep общий буфер пуст или почти пуст. К моменту возврата из ожидания общий буфер примерно наполовину заполнен звуковыми данными.

После вызова метода Initialize поток остается открытым до тех пор, пока не будут освобождены все ссылки на интерфейс IAudioClient и все ссылки на интерфейсы служб, полученные с помощью метода GetService. Последний вызов Release закрывает поток.



Рисунок 2.2 – Блок-схема функции RecordAudioStream

* 1. Захват экрана

Работу данной части программы обеспечивает поток ThreadScreenShot.

Данные для отрисовки хранятся в буфере bmpBuffer. После начала конференции программа периодически получает данные о изображении на экране. Количество кадров в секунду определяется макросом FPS. Поток поочерёдно вызывает функцию TakeScreenShot, записывающую данные в буфер и обновляющую данные в дескрипторе hbmp, и передаёт данные остальным участникам конференции.

Для получения информации о текущем изображении на экране происходит вызов функции GetDC с параметром NULL. Функция GetDC извлекает дескриптор контекста устройства для клиентской области указанного окна или для всего экрана. Затем создаётся битмап объект с заданными шириной и высотой, для помещения большого изображения в меньший битмап, вызывается функция SetStretchBltMode с параметром HALFTONE. Режим HALFTONE отображает пиксели исходного прямоугольника в блоки пикселей целевого прямоугольника. Средний цвет целевого блока пикселей приблизительно соответствует цвету исходных пикселей.

Данный режим медленнее и требует большей обработки исходного изображения, чем при других режимах, но производит изображения более высокого качества.

Для получения данных картинки вызывается функция GetDIBits, одним из параметром которой является указать на буфер, в котором будут храниться данные.



Рисунок 2.3 – Блок-схема функции TakeScreenShot



Рисунок 2.4 – Блок-схема функции ThreadScreenShot

* 1. Передача данных по сети

Интернет – основа передачи данных в приложении. Для работы с сетью используется стандартное API Windows – Windows Sockets. Поэтому вначале приложения подключается заголовочный файл winsock.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 с его экспортированными функциями и стеками протоколов. Он использует парадигму сокетов, которая впервые была популяризирована UNIX от Berkeley Software Distribution или BSD. Позднее он был адаптирован для Windows в Windows Sockets 1.1, с которой приложения Windows Sockets 2 обратно совместимы. В [4] содержится документация по работе с сокетами на Windows.

Для передачи данных был выбран протокол TCP с целью предотвращения потери пакетов, а также стабильного подключения пользователей к конференции.

В модуле Client.h содержится объявление структуры клиента. В поле name хранится имя пользователя, подключённого к вебинару. chatSocket хранит дескриптор сокета чата конференции. Сокеты videoSocket и voiceSocket используются для передачи графических и звуковых данных соответственно.

После запуска конференции пользователем происходит инициализация Windows Sockets API вызовом функции WSAStartup. Как и большинство системных функций, результатом работы является код выполнения, который нужно проверять в целях предотвращения и логирования ошибок.

На рисунке 2.5 представлено отображение окна с ошибкой инициализации чата.

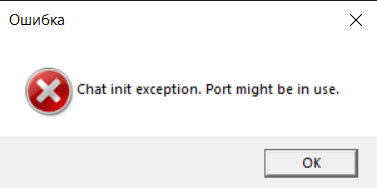


Рисунок 2.5 – Окно с ошибкой инициализации чата

В переменной servChatSocket хранится дескриптор сокет прослушивания подключения к чату. Запуск прослушивания происходит при вызове функции listen.

Для корректной работы программы требовалось применение критической секции, так как подключения пользователей должны быть синхронизированы. В программном средстве несколько потоков могут обращаться к сокетам, разделёнными между собой. В целях предотвращения ситуации, когда один поток пытается передать данные клиенту, который подключился, например, к чату, но не успел инициализировать сокет передачи звука или трансляции изображения.

Все процессы, которые вызывают функции Winsock, должны инициализировать использование библиотеки Windows Sockets перед выполнением других вызовов функций Winsock. Это также обеспечивает поддержку Winsock в системе.

После инициализации сокетов, пользователь, создавший конференцию, подключается к чату, затем другие пользователи могут подключиться к вебинару. После установки соединения с сервером пользователь отправляет имя пользователя, которое будет отображаться в чате.

Отдельного внимания заслуживает функция ThreadFuncVideoListener, в которой происходит получение данных изображения и создание битмап объекта. В данной функции потребовалось следить за числом полученных байт от сервера, чтобы сформировать корректное изображение.



Рисунок 2.6 – Блок-схема функции ThreadFuncVideoListener

* 1. Работа с реестром

Функции для работы с реестром, а также необходимые макросы описаны в заголовочном файле Reg.h. Реестр был использован для сохранения имени пользователя при повторном использовании приложения. Были реализованы следующие функции:

– CreateRegistryKey – функция, создающая ключ, в котором хранятся данные, необходимые программе;

– WriteStringInRegistry – функция, предназначенная для записи строки по ключу;

– ReadStringFromRegistry – функция, позволяющая получить строковое значение по ключу.

При запуске программы в исходном файле source.cpp происходит создание ключа «Webinar», далее по ходу выполнения программы при необходимости получения значения имени пользователя из реестра вызывается функция ReadStringFromRegistry по ключу «userName», для обновления данного значения вызывается функция WriteStringInRegistry.

При записи строкового значения в реестр необходимо было учитывать тот факт, что приложение использовало строки типа LPWSTR, а для записи нужно было использовать строки типа LPCSTR. Для решения данной проблемы был разработан класс Convert, определённый в файле заголовка Convert.h. Данный класс имеет два метода:

– WstrToStr – преобразование типа wstring в string;

– StrToWStr – преобразование типа string в wstring;

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Тестирование функционала приложения

В ходе тестирования приложения были выявлены некоторые недостатки программного средства.

Была составлена таблица 3.1, показывающая ожидаемые и реальные результаты, полученные при заданных условиях, она представлена ниже.

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Передача аудиоданных клиентам | Успешная передача данных за конечное время | Успешная передача данных за конечное время |
| 2. | Подключение к серверу клиента | Успешное подключение | Успешное подключение |
| 3. | Смена записывающего устройства | Остановка и возобновление передачи | Остановка и возобновление передачи |
| 4. | Отправка сообщения в чат | Отображение сообщения | Отображение сообщения |
| 5. | Непрерывная запись голоса | Безотказная работа приложения | Безотказная работа приложения |
| 6. | Память, выделяемая для голосовых сообщений | более 1 мегабайт | менее 64 килобайт |
| 7. | Подключение пользователей под одним именем | Продолжение работы клиента | Продолжение работы клиента |
| 8. | Выключение вебинара во время передачи звука и изображения | Продолжение работы клиента | Продолжение работы клиента |
| 9. | Передача изображения | Корректное отображение | Смещение изображения |
| 10. | Отключение пользователя | Вывод сообщения о выходе | Вывод сообщения о выходе |

При запуске вебинара и подключению к нему с одного устройства, ошибок при передаче изображения и звука не наблюдалось, так как данные мгновенно передавались другому экземпляру приложения минуя отправку по сети. Но при подключении к конференции с другого устройства было обнаружено смещение передаваемого изображения.

Смещение изображения представлено на следующем рисунке.

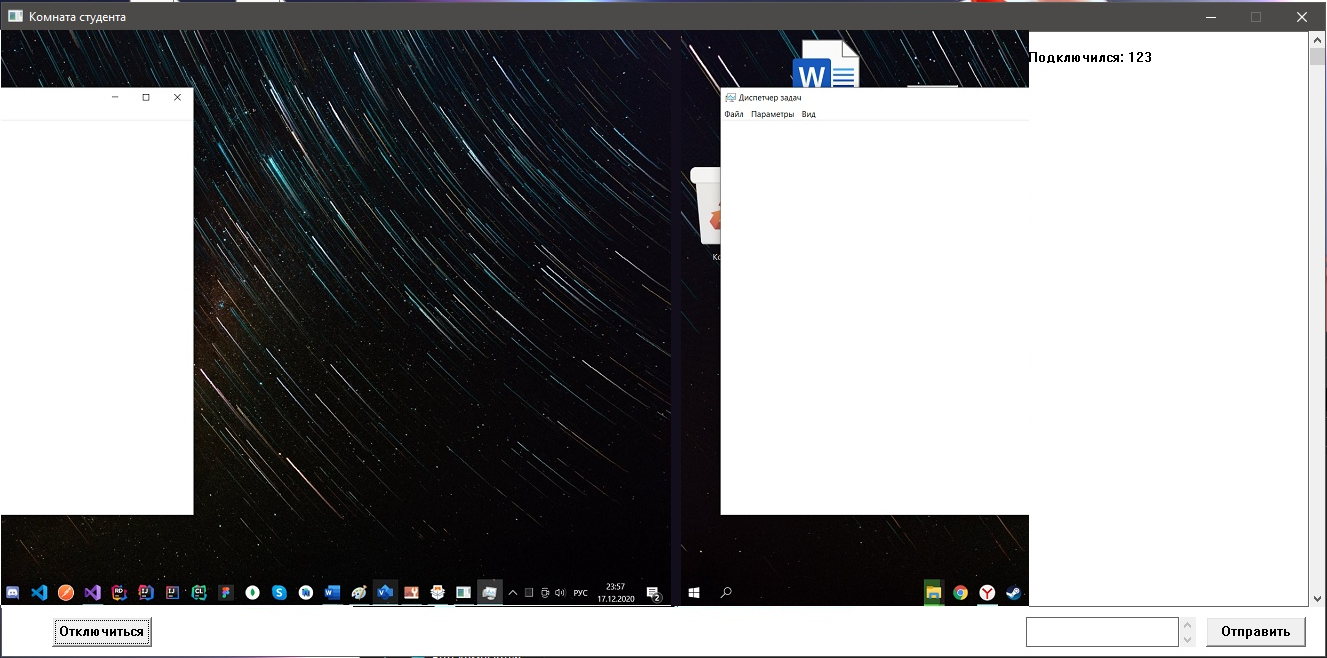


Рисунок 3.1 – Смещение изображения

Функции send и recv возвращают целочисленное значение количества отправленных или полученных байтов, соответственно, или ошибку. Каждая функция также принимает одни и те же параметры: активный сокет, буфер символов, количество байтов для отправки или получения и любые используемые флаги.

Для решения данной проблемы были использованы значения, возвращаемые из данных функций. Каждое последующее получение буфера сопровождалось проверкой на окончание получения очередного изображения, и только после этого создавался битмап объект.

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

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Порядок использования

Стартовое окно программы, изображенное на рисунке 4.1, содержит следующие элементы:

– «Создать комнату»;

– «Присоединиться».

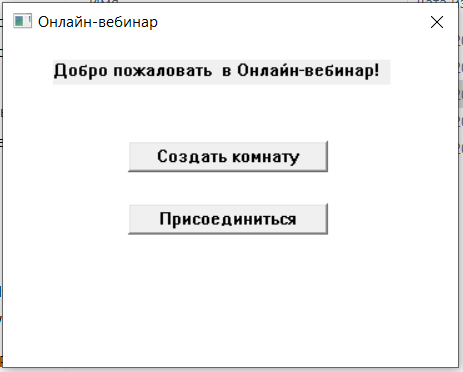


Рисунок 4.1 – Стартовое окно программы

После выбора любой из данных опций пользователю требуется ввести имя. Окно ввода имени представлено на следующем рисунке.

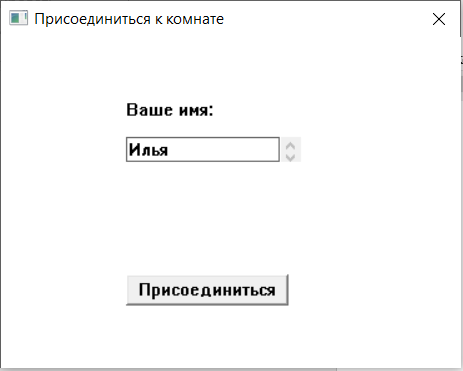


Рисунок 4.2 – Окно ввода имени

Далее открывается главное окно конференции. Создатель вебинара должен начать на кнопку «Начать» для старта трансляции. Для передачи звука будет использовано стандартное устройство записи звука, определённое Windows. В время конференции можно менять устройство записи, что исключает необходимость перезапуска комнаты при отключении микрофона.

Подключающиеся же пользователи в свою очередь должны нажать на кнопку «Подключиться», чтобы участвовать в конференции.

Далее представлено изображение главного окна программы.

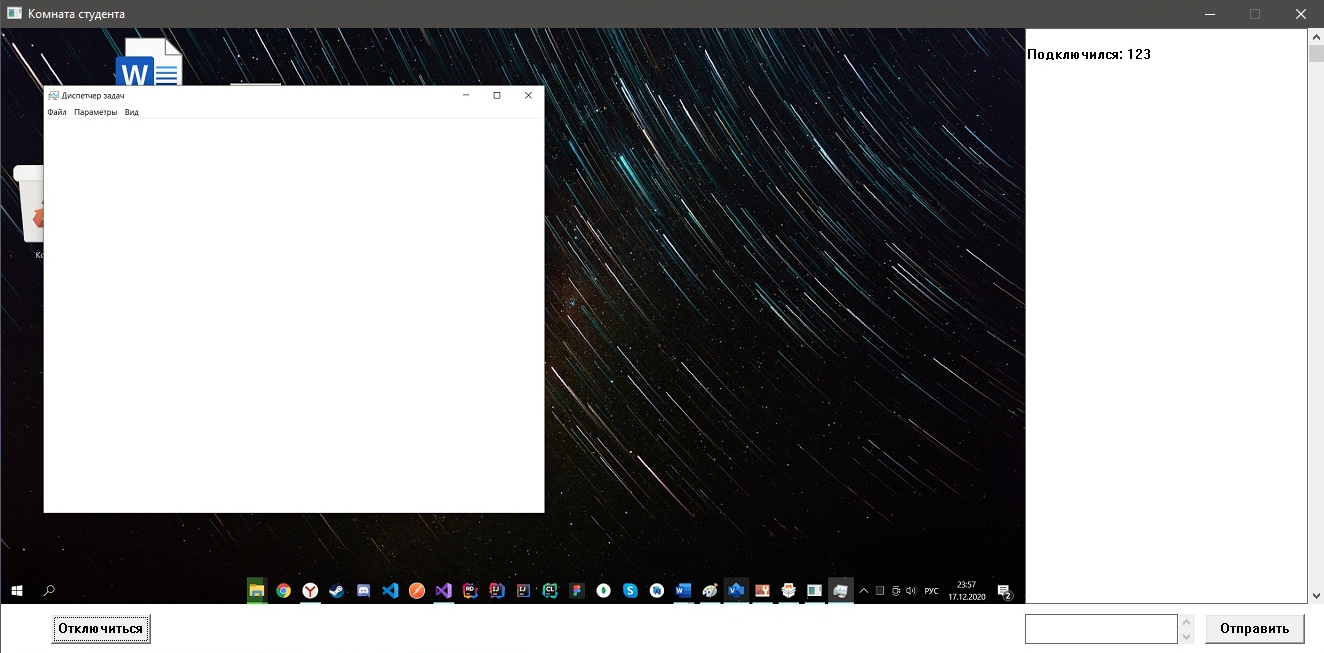


Рисунок 4.3 – Главное окно программы

В случае некорректной загрузки какого-либо из ресурсов, например, при неудачном подключении к комнате, будет выведено соответствующее сообщение, которое представлено на рисунке 4.4, и некоторые элементы приложения станут недоступны.

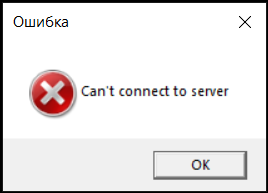


Рисунок 4.4 – Сообщение при некорректном подключении к комнате

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта было разработано программное средство «Онлайн-вебинар», которое обеспечит дистанционное проведение собраний и лекций. Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

* создание комнаты;
* подключение к комнате;
* возможность выхода и повторного подключения;
* алгоритмы захвата изображения;
* алгоритмы записи звука;
* отправка текстовых сообщений;
* передача данных по сети;
* синхронизация передачи данных.

Для успешного выполнения всех поставленных целей потребовалось изучить возможности Windows API, в частности Windows Sockets, WASAPI, работу с реестром. Также потребовалось применить знания объектно-ориентированных возможностей языка C++, основные принципы данной парадигмы, а также освоить взаимодействия существующих компонентов и классов.

Существует много возможностей для дальнейшего улучшения приложения. Одним из направлений является добавление функции использования веб-камеры и записи конференции. Ещё одним вариантом развития является адаптация проекта для запуска на устройствах с низкой разрешающей способностью экрана и переход к кроссплатформенной разработке.

Использование данного приложения позволит улучшить качество проведения собраний, также оно может быть использовано в образовательных целях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке C++ / Гирберд Шилдт. Учебный курс: / Гирберд Шилдт. – СПб: изд. С. В. Малгачёва, 2001. – 231 с;
2. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003;
3. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
4. Windows Sockets 2 – Win32 apps | Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/\_winsock/;
5. WASAPI – Win32 apps | Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/coreaudio/wasapi;
6. Programming reference for the Win32 API– Электронные данные. – режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/>
7. Registry reference – Win32 apps | Microsoft Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/sysinfo/registry-reference;
8. Культин, Н. А. Основы программирования в C++ 2-е издание – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 182с;
9. Культин, Н. А. Основы программирования в C++ – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 167с;
10. ОСиСП презентация/Кирилл Сурков, Дмитрий Сурков, Юрий Четырько – 172сл.

ПРИЛОЖЕНИЕ А

Исходный код программы

#include <memory>

#include <Windows.h>

#include <Winsock.h>

#include <string>

#include <tchar.h>

#include <iostream>

#include <vector>

#include <conio.h>

#include <exception>

#include <Mmsystem.h>

#include <audioclient.h>

#include <mmdeviceapi.h>

#include "RoomWindow.h"

#include "Button.h"

#include "TextBox.h"

#include "Role.h"

#include "Client.h"

#include "Mmsystem.h"

#include "mmreg.h"

#include "WaveWriter.h"

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

#pragma comment(lib, "Winmm.lib")

#define FPS 10

#define BUFFER\_SIZE 1024

#define REFTIMES\_PER\_SEC 500000

#define REFTIMES\_PER\_MILLISEC 500

#define MAX\_LOOP\_BEFORE\_STOP 20

const CLSID CLSID\_MMDeviceEnumerator = \_\_uuidof(MMDeviceEnumerator);

const IID IID\_IMMDeviceEnumerator = \_\_uuidof(IMMDeviceEnumerator);

const IID IID\_IAudioClient = \_\_uuidof(IAudioClient);

const IID IID\_IAudioCaptureClient = \_\_uuidof(IAudioCaptureClient);

HRESULT RecordAudioStream(const WCHAR\* fileName);

DWORD WINAPI ThreadSoundRecorder(LPVOID \_param)

{

HRESULT hr = CoInitializeEx(NULL, COINIT\_APARTMENTTHREADED | COINIT\_DISABLE\_OLE1DDE);

if (SUCCEEDED(hr))

{

while (true) {

RecordAudioStream(L"capture.wav");

PlaySound(L"capture.wav", NULL, SND\_ASYNC | SND\_NODEFAULT);

}

CoUninitialize();

}

return 0;

}

HRESULT RecordAudioStream(const WCHAR\* fileName)

{

HRESULT hr = S\_OK;

IMMDeviceEnumerator\* pEnumerator = NULL;

IMMDevice\* pDevice = NULL;

IAudioClient\* pAudioClient = NULL;

IAudioCaptureClient\* pCaptureClient = NULL;

WAVEFORMATEX\* pwfx = NULL;

REFERENCE\_TIME hnsRequestedDuration = REFTIMES\_PER\_SEC;

REFERENCE\_TIME hnsActualDuration;

UINT32 bufferFrameCount;

WaveWriter waveWriter;

UINT32 uiFileLength = 0;

BOOL bExtensibleFormat = FALSE;

try

{

IF\_FAILED\_THROW(CoCreateInstance(CLSID\_MMDeviceEnumerator, NULL, CLSCTX\_ALL, IID\_IMMDeviceEnumerator, (void\*\*)&pEnumerator));

IF\_FAILED\_THROW(pEnumerator->GetDefaultAudioEndpoint(eCapture, eConsole, &pDevice));

IF\_FAILED\_THROW(pDevice->Activate(IID\_IAudioClient, CLSCTX\_ALL, NULL, (void\*\*)&pAudioClient));

IF\_FAILED\_THROW(pAudioClient->GetMixFormat(&pwfx));

switch (pwfx->wFormatTag)

{

case WAVE\_FORMAT\_EXTENSIBLE:

TRACE((L"WAVE\_FORMAT\_EXTENSIBLE"));

bExtensibleFormat = TRUE;

WAVEFORMATEXTENSIBLE\* pWaveFormatExtensible = reinterpret\_cast<WAVEFORMATEXTENSIBLE\*>(pwfx);

break;

}

IF\_FAILED\_THROW(pAudioClient->Initialize(AUDCLNT\_SHAREMODE\_SHARED, 0, hnsRequestedDuration, 0, pwfx, NULL));

IF\_FAILED\_THROW(pAudioClient->GetBufferSize(&bufferFrameCount));

IF\_FAILED\_THROW(pAudioClient->GetService(IID\_IAudioCaptureClient, (void\*\*)&pCaptureClient));

IF\_FAILED\_THROW(waveWriter.Initialize(fileName, bExtensibleFormat) ? S\_OK : E\_FAIL);

hnsActualDuration = (double)REFTIMES\_PER\_SEC \* bufferFrameCount / pwfx->nSamplesPerSec;

IF\_FAILED\_THROW(pAudioClient->Start());

BOOL bDone = FALSE;

UINT32 packetLength = 0;

UINT32 numFramesAvailable;

BYTE\* pData;

DWORD flags;

int iLoop = 0;

while (bDone == FALSE)

{

Sleep(hnsActualDuration / REFTIMES\_PER\_MILLISEC / 2);

IF\_FAILED\_THROW(pCaptureClient->GetNextPacketSize(&packetLength));

while (packetLength != 0)

{

IF\_FAILED\_THROW(pCaptureClient->GetBuffer(&pData, &numFramesAvailable, &flags, NULL, NULL));

waveWriter.WriteWaveData(pData, numFramesAvailable \* pwfx->nBlockAlign) ? S\_OK : E\_FAIL;

uiFileLength += numFramesAvailable;

IF\_FAILED\_THROW(pCaptureClient->ReleaseBuffer(numFramesAvailable));

IF\_FAILED\_THROW(pCaptureClient->GetNextPacketSize(&packetLength));

}

if (iLoop++ == MAX\_LOOP\_BEFORE\_STOP)

bDone = TRUE;

}

}

catch (HRESULT) {}

if (hr == S\_OK && pwfx != NULL)

waveWriter.FinalizeHeader(pwfx, uiFileLength, bExtensibleFormat);

if (pAudioClient)

{

pAudioClient->Reset();

LOG\_HRESULT(pAudioClient->Stop());

SAFE\_RELEASE(pAudioClient);

}

CoTaskMemFree(pwfx);

SAFE\_RELEASE(pCaptureClient);

SAFE\_RELEASE(pEnumerator);

SAFE\_RELEASE(pDevice);

return hr;

}

namespace webinar

{

const int SCREEN\_WIDTH = 1920;

const int SCREEN\_HEIGHT = 1080;

const int CHAT\_PORT = 5567;

const int VOICE\_LISTEN\_PORT = 9663;

const int VIDEO\_LISTEN\_PORT = 9881;

const int SCREENSHOT\_MS = 1000 / FPS;

bool isEndOfWebinar = true;

const char\* HOST\_ADRESS = "192.168.100.4";

HBITMAP hbmp = NULL;

HINSTANCE Hrw = NULL;

std::vector<WindowControl\*> formControls;

CRITICAL\_SECTION critSec = { 0 };

WSADATA verSoc = { 0 };

SOCKET clientChatSocket = NULL;

SOCKET servChatSocket = NULL;

SOCKET videoSocket = NULL;

SOCKET voiceSocket = NULL;

const int CLIENT\_COUNT = 30;

int connectedClientsCount = 0;

std::vector<client> clientSockets;

void InitChat();

void InitVoiceSocket();

void InitVideoSocket();

void ConnectToChat();

void ConnectToVoice();

void ConnectToVideo();

int TakeScreenShot();

void DrawImage(HWND hwnd, HDC hdc);

const long videoRecvSize = 70000000;

char videoRecvBuff[videoRecvSize];

char voiceBuf[8294400];

char\* bmpBuffer = 0;

DWORD WINAPI ThreadChatListener(LPVOID);

DWORD WINAPI ThreadChatConnectionsListener(LPVOID);

DWORD WINAPI ThreadScreenShot(LPVOID \_param);

DWORD WINAPI ThreadSoundRecorder(LPVOID \_param);

DWORD WINAPI ThreadFuncVideoListener(LPVOID \_param);

DWORD WINAPI ThreadFuncVoiceListener(LPVOID \_param);

DWORD WINAPI ThreadClientListen(LPVOID \_param);

std::string messagesData;

bool ClickSend(WPARAM, LPARAM);

bool ClickConnect(WPARAM, LPARAM);

bool ClickStart(WPARAM, LPARAM);

RoomWindow::RoomWindow(TCHAR \* strWindowClassName, HINSTANCE hInstance, int x, int y, int width, int heigth, int iShowWindow)

{

\_wndClass.cbSize = sizeof(\_wndClass);

\_wndClass.style = CS\_HREDRAW | CS\_VREDRAW;

\_wndClass.lpfnWndProc = WndProc;

\_wndClass.lpszMenuName = nullptr;

\_wndClass.lpszClassName = strWindowClassName;

\_wndClass.cbWndExtra = NULL;

\_wndClass.cbClsExtra = NULL;

\_wndClass.hIcon = LoadIcon(nullptr, IDI\_WINLOGO);

\_wndClass.hIconSm = LoadIcon(nullptr, IDI\_WINLOGO);

\_wndClass.hCursor = LoadCursor(nullptr, IDC\_ARROW);

\_wndClass.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(WHITE\_BRUSH));

\_wndClass.hInstance = hInstance;

if (!RegisterClassEx(&\_wndClass))

{

throw std::exception("Window Class Initial Error");

}

WindowControl::\_hwndWindow = CreateWindow(\_wndClass.lpszClassName,

strWindowClassName,

WS\_OVERLAPPEDWINDOW & ~WS\_THICKFRAME & ~WS\_MAXIMIZEBOX,

x, y, width, heigth,

HWND(NULL),

NULL,

HINSTANCE(hInstance),

NULL);

if (!WindowControl::\_hwndWindow)

{

throw std::exception("Window Cerate Error");

}

Hrw = (HINSTANCE)\_hwndWindow;

ShowWindow(WindowControl::\_hwndWindow, iShowWindow);

UpdateWindow(WindowControl::\_hwndWindow);

WindowControl::\_strWindowName = reinterpret\_cast<wchar\_t>(strWindowClassName);

}

RoomWindow::~RoomWindow()

{

WSACleanup();

}

LRESULT RoomWindow::WndProc(HWND hWnd,

UINT uMsg,

WPARAM wParam,

LPARAM lParam)

{

try

{

switch (uMsg)

{

case WM\_CREATE:

{

formControls.push\_back(new TextBox(hWnd, \_T(""), 1024, 0, 300, 576, 11));

formControls.push\_back(new TextBox(hWnd, \_T(""), 1024, 586, 170, 30, 12));

formControls.push\_back(new Button(hWnd, \_T("Отправить"), 1204, 586, 100, 30, 13));

formControls[2]->SetEvent(ClickSend, WM\_COMMAND);

if (userRole == TEACHER) {

formControls.push\_back(new Button(hWnd, \_T("Начать"), 50, 586, 100, 30, 15));

formControls[3]->SetEvent(ClickStart, WM\_COMMAND);

}

else {

formControls.push\_back(new Button(hWnd, \_T("Подключиться"), 50, 586, 100, 30, 10));

formControls[3]->SetEvent(ClickConnect, WM\_COMMAND);

}

return 0;

}case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc, memDC;

RECT rect;

HWND hwnd = (HWND)Hrw;

GetClientRect(hwnd, &rect);

hdc = BeginPaint(hwnd, &ps);

memDC = CreateCompatibleDC(hdc);

HBITMAP memBmp = CreateCompatibleBitmap(hdc, 1024, 576);

HBITMAP oldBitmap = (HBITMAP)SelectObject(memDC, memBmp);

DrawImage(hwnd, memDC);

BitBlt(hdc, 0, 0, 1024, 576, memDC, 0, 0, SRCCOPY);

DeleteDC(memDC);

DeleteObject(memBmp);

DeleteObject(oldBitmap);

EndPaint(hwnd, &ps);

return 0;

}

case WM\_COMMAND:

for (int i = 0; i < formControls.size(); i++)

{

if (formControls[i]->GatID() == LOWORD(wParam))

{

formControls[i]->EventStart(uMsg, wParam, lParam);

}

}

return 0;

case WM\_DESTROY:

PostQuitMessage(NULL);

return 0;

default:

return DefWindowProc(hWnd, uMsg, wParam, lParam);

}

}

catch (std::exception ex)

{

MessageBoxA(NULL, ex.what(), NULL, MB\_ICONERROR);

PostQuitMessage(-1);

return -1;

}

}

void DrawImage(HWND hwnd, HDC hdc) {

HDC hdcMem;

BITMAP bitmap;

HGDIOBJ oldBitmap;

hdcMem = CreateCompatibleDC(hdc);

oldBitmap = SelectObject(hdcMem, hbmp);

GetObject(hbmp, sizeof(bitmap), &bitmap);

SetStretchBltMode(hdc, HALFTONE);

StretchBlt(hdc, 0, 0, 1024, 576, hdcMem, 0, 0,SCREEN\_WIDTH, SCREEN\_HEIGHT, SRCCOPY);

SelectObject(hdcMem, oldBitmap);

DeleteDC(hdcMem);

DeleteObject(oldBitmap);

}

bool ClickStart(WPARAM wParam, LPARAM lParam) {

SetWindowTextA(formControls[3]->GetHandler(), "Завершить");

if (!isEndOfWebinar) {

isEndOfWebinar = true;

exit(0);

}

isEndOfWebinar = false;

InitChat();

InitVideoSocket();

InitVoiceSocket();

ConnectToChat();

return true;

}

bool ClickConnect(WPARAM wParam, LPARAM lParam) {

if (!isEndOfWebinar) {

exit(0);

}

isEndOfWebinar = false;

SetWindowTextA(formControls[3]->GetHandler(), "Отключиться");

ConnectToChat();

ConnectToVideo();

ConnectToVoice();

return true;

}

bool ClickSend(WPARAM wParam, LPARAM lParam)

{

if (clientChatSocket == NULL) {

MessageBoxA(NULL, "No socket", NULL, MB\_ICONERROR);

return false;

}

UINT lenOfMessageText = GetWindowTextLength(formControls[1]->GetHandler());

if (lenOfMessageText < 1)

return false;

std::string messageText;

messageText.resize(lenOfMessageText+1);

GetWindowTextA(formControls[1]->GetHandler(), &messageText[0], messageText.size());

send(clientChatSocket, &messageText[0], messageText.size(), NULL);

}

void Brodcast(std::string message)

{

for (int i = 0; i < clientSockets.size(); i++)

{

if (clientSockets[i].chatSocket && clientSockets[i].chatSocket != SOCKET\_ERROR)

{

int sizeSent = send(clientSockets[i].chatSocket,

&message[0],

strlen(&message[0]),

0);

if (sizeSent == SOCKET\_ERROR)

{

closesocket(clientSockets[i].chatSocket);

clientSockets[i].chatSocket = SOCKET\_ERROR;

}

}

}

}

void InitChat()

{

try

{

WSADATA verSoc = { 0 };

if (WSAStartup(0x0202, &verSoc))

{

throw("Can't initialize a socket servion");

}

{

servChatSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (servChatSocket == INVALID\_SOCKET)

{

WSACleanup();

throw("Can't create the server socket");

}

sockaddr\_in local\_addr = { 0 };

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port = htons(CHAT\_PORT);

local\_addr.sin\_addr.s\_addr = NULL;

if (bind(servChatSocket,

reinterpret\_cast<sockaddr\*>(&local\_addr),

sizeof(local\_addr)) == SOCKET\_ERROR)

{

closesocket(servChatSocket);

WSACleanup();

throw("Can't bind a socket with the sockaddr struct");

}

}

clientSockets = std::vector<client>(CLIENT\_COUNT);

if (listen(servChatSocket, clientSockets.size()) == SOCKET\_ERROR)

{

closesocket(servChatSocket);

WSACleanup();

throw("Can't set max amount of clients");

}

InitializeCriticalSection(&critSec);

CreateThread(NULL,NULL,ThreadChatConnectionsListener,NULL,NULL,NULL);

}

catch (std::exception& ex)

{

MessageBoxA(NULL, ex.what(), NULL, MB\_ICONERROR);

PostQuitMessage(-1);

}

catch (...)

{

MessageBoxA(NULL, "Chat init exception. Port might be in use.", NULL, MB\_ICONERROR);

PostQuitMessage(-1);

}

}

void InitVoiceSocket()

{

voiceSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (voiceSocket == INVALID\_SOCKET)

{

WSACleanup();

throw("Can't create the server socket");

}

sockaddr\_in local\_addr = { 0 };

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port = htons(VOICE\_LISTEN\_PORT);

local\_addr.sin\_addr.s\_addr = NULL;

if (bind(voiceSocket,

reinterpret\_cast<sockaddr\*>(&local\_addr),

sizeof(local\_addr)) == SOCKET\_ERROR)

{

closesocket(voiceSocket);

WSACleanup();

throw("Can't bind a socket with the sockaddr struct");

}

if (listen(voiceSocket, clientSockets.size()) == SOCKET\_ERROR)

{

closesocket(voiceSocket);

WSACleanup();

throw("Can't set max amount of clients");

}

CreateThread(NULL,NULL,ThreadSoundRecorder,NULL,NULL,NULL);

}

void InitVideoSocket()

{

videoSocket = socket(AF\_INET, SOCK\_STREAM, 0);

if (videoSocket == INVALID\_SOCKET)

{

WSACleanup();

throw("Can't create the server socket");

}

sockaddr\_in local\_addr = { 0 };

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_port = htons(VIDEO\_LISTEN\_PORT);

local\_addr.sin\_addr.s\_addr = NULL;

if (bind(videoSocket,

reinterpret\_cast<sockaddr\*>(&local\_addr),

sizeof(local\_addr)) == SOCKET\_ERROR)

{

closesocket(videoSocket);

WSACleanup();

throw("Can't bind a socket with the sockaddr struct");

}

if (listen(videoSocket, clientSockets.size()) == SOCKET\_ERROR)

{

closesocket(videoSocket);

WSACleanup();

throw("Can't set max amount of clients");

}

CreateThread(NULL, NULL, ThreadScreenShot, NULL, NULL, NULL);

}

void ConnectTo(int port, SOCKET\* Socket) {

if (\*Socket != NULL)

{

closesocket(\*Socket);

Socket = NULL;

WSACleanup();

}

if (WSAStartup(0x0202, &verSoc))

{

throw(std::exception("Can't initialize socket version: ", WSAGetLastError()));

}

\*Socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);

if (\*Socket == SOCKET\_ERROR)

{

WSACleanup();

throw(std::exception("Can't create socket: ", GetLastError()));

}

std::string ip = HOST\_ADRESS;

if (!strlen(&ip[0]))

{

throw(std::exception("Incorrect host ip"));

}

struct sockaddr\_in addrClient = { 0 };

addrClient.sin\_addr.s\_addr = inet\_addr(&ip[0]);

addrClient.sin\_family = AF\_INET;

addrClient.sin\_port = htons(port);

if (SOCKET\_ERROR == connect(\*Socket,

(struct sockaddr\*)&addrClient,

sizeof(addrClient)))

{

closesocket(\*Socket);

throw(std::exception("Can't connect to server"));

}

}

void ConnectToChat()

{

try

{

ConnectTo(CHAT\_PORT, &clientChatSocket);

CreateThread(NULL,

NULL,

ThreadChatListener,

reinterpret\_cast<LPVOID>(clientChatSocket),

NULL,

NULL);

send(clientChatSocket,

userName.c\_str(),

strlen(userName.c\_str()),

NULL);

}

catch (std::exception ex)

{

if (clientChatSocket != NULL)

{

closesocket(clientChatSocket);

clientChatSocket = NULL;

}

MessageBoxA(NULL, ex.what(), NULL, MB\_ICONERROR);

}

}

void ConnectToVoice()

{

ConnectTo(VOICE\_LISTEN\_PORT, &voiceSocket);

CreateThread(NULL,

NULL,

ThreadFuncVoiceListener,

NULL,

NULL,

NULL);

}

void ConnectToVideo()

{

ConnectTo(VIDEO\_LISTEN\_PORT, &videoSocket);

CreateThread(NULL,

NULL,

ThreadFuncVideoListener,

NULL,

NULL,

NULL);

}

int TakeScreenShot() {

HDC hScreen = GetDC(NULL);

int ScreenX = SCREEN\_WIDTH;

int ScreenY = SCREEN\_HEIGHT;

HDC hdcMem = CreateCompatibleDC(hScreen);

DeleteObject(hbmp);

hbmp = CreateCompatibleBitmap(hScreen, ScreenX, ScreenY);

HBITMAP hOld = (HBITMAP)SelectObject(hdcMem, hbmp);

SetStretchBltMode(hdcMem, HALFTONE);

StretchBlt(hdcMem,

0, 0,

SCREEN\_WIDTH, SCREEN\_HEIGHT,

hScreen,

0, 0,

ScreenX,

ScreenY,

SRCCOPY);

SelectObject(hdcMem, hOld);

BITMAPINFOHEADER bmi = { 0 };

bmi.biSize = sizeof(BITMAPINFOHEADER);

bmi.biPlanes = 1;

bmi.biBitCount = 32;

bmi.biWidth = SCREEN\_WIDTH;

bmi.biHeight = -SCREEN\_HEIGHT;

bmi.biCompression = BI\_RGB;

bmi.biSizeImage = 0;

if (!bmpBuffer) {

bmpBuffer = (char\*)malloc(4 \* SCREEN\_WIDTH \* SCREEN\_HEIGHT);

}

int s = GetDIBits(hdcMem, hbmp, 0, SCREEN\_HEIGHT, bmpBuffer, (BITMAPINFO\*)&bmi, DIB\_RGB\_COLORS);

ReleaseDC(GetDesktopWindow(), hScreen);

DeleteDC(hScreen);

DeleteDC(hdcMem);

DeleteObject(hOld);

return 4 \* SCREEN\_WIDTH \* SCREEN\_HEIGHT;

}

DWORD WINAPI ThreadChatListener(LPVOID \_param)

{

const int SIZE\_BUF = 1024;

SOCKET Socket = reinterpret\_cast<SOCKET>(\_param);

char buf\_recv[SIZE\_BUF] = { 0 };

int size\_recv;

do

{

memset(buf\_recv, 0, sizeof(buf\_recv));

size\_recv = recv(Socket,

buf\_recv,

sizeof(buf\_recv) - 1,

0);

int err = WSAGetLastError();

if (size\_recv && size\_recv != SOCKET\_ERROR)

{

messagesData += "\r\n";

messagesData += buf\_recv;

SetWindowTextA(formControls[0]->GetHandler(), messagesData.c\_str());

}

} while (size\_recv && size\_recv != SOCKET\_ERROR);

return 0;

}

DWORD WINAPI ThreadChatConnectionsListener(LPVOID \_param)

{

bool teacherConnect = true;

while (servChatSocket != SOCKET\_ERROR)

{

try

{

SOCKET connectingSocketClient = NULL;

sockaddr\_in clientAddrInfo = { 0 };

int sizeClientAddrInfo = sizeof(clientAddrInfo);

connectingSocketClient = accept(servChatSocket,

reinterpret\_cast<sockaddr\*>(&clientAddrInfo),

&sizeClientAddrInfo);

if (connectingSocketClient == INVALID\_SOCKET && servChatSocket != INVALID\_SOCKET)

throw("Couldn't connect a client");

if (connectedClientsCount > clientSockets.size())

{

char errorMesage[] = "Server is full";

send(connectingSocketClient, errorMesage, sizeof(errorMesage), 0);

closesocket(connectingSocketClient);

}

else

{

EnterCriticalSection(&critSec);

for (int i = 0; i < clientSockets.size(); i++)

{

if (!clientSockets[i].chatSocket)

{

clientSockets[i].chatSocket = connectingSocketClient;

CreateThread(NULL, NULL, ThreadClientListen, reinterpret\_cast<LPVOID>(i), NULL, NULL);

if (!teacherConnect) {

SOCKET connectingSocketClient1 = NULL;

sockaddr\_in clientAddrInfo1 = { 0 };

int sizeClientAddrInfo1 = sizeof(clientAddrInfo1);

connectingSocketClient1 = accept(videoSocket,

reinterpret\_cast<sockaddr\*>(&clientAddrInfo1),

&sizeClientAddrInfo1);

clientSockets[i].videoSocket = connectingSocketClient1;

SOCKET connectingSocketClient2 = NULL;

sockaddr\_in clientAddrInfo2 = { 0 };

int sizeClientAddrInfo2 = sizeof(clientAddrInfo2);

connectingSocketClient2 = accept(voiceSocket,

reinterpret\_cast<sockaddr\*>(&clientAddrInfo2),

&sizeClientAddrInfo2);

clientSockets[i].voiceSocket = connectingSocketClient2;

}

else {

teacherConnect = false;

}

break;

}

}

connectedClientsCount++;

LeaveCriticalSection(&critSec);

}

}

catch (std::exception& ex)

{

MessageBoxA(NULL, ex.what(), NULL, MB\_ICONERROR);

}

catch (...)

{

MessageBoxA(NULL, "Unhandled client exception", NULL, MB\_ICONERROR);

}

}

DeleteCriticalSection(&critSec);

closesocket(servChatSocket);

WSACleanup();

}

DWORD WINAPI ThreadScreenShot(LPVOID \_param)

{

do

{

int len = TakeScreenShot();

InvalidateRect((HWND)Hrw, NULL, false);

for (int i = 1; i < clientSockets.size(); i++)

{

if (clientSockets[i].videoSocket && clientSockets[i].videoSocket != SOCKET\_ERROR)

{

int sizeSent = send(clientSockets[i].videoSocket, (const char\*)bmpBuffer, len, 0);

int err = WSAGetLastError();

if (sizeSent == SOCKET\_ERROR)

{

closesocket(clientSockets[i].videoSocket);

clientSockets[i].videoSocket = SOCKET\_ERROR;

}

}

}

Sleep(SCREENSHOT\_MS);

} while (!isEndOfWebinar);

return 0;

}

DWORD WINAPI ThreadFuncVideoListener(LPVOID \_param)

{

int bytesToRead = 4 \* SCREEN\_WIDTH \* SCREEN\_HEIGHT;

int nbytes;

sockaddr\_in sender\_addr;

int address\_size = sizeof(struct sockaddr\_in);

while (!isEndOfWebinar) {

nbytes = recv(videoSocket, videoRecvBuff + (4 \* SCREEN\_WIDTH \* SCREEN\_HEIGHT - bytesToRead), bytesToRead, 0);

if (nbytes == SOCKET\_ERROR) {

closesocket(videoSocket);

break;

}

bytesToRead = bytesToRead - nbytes;

if (bytesToRead == 0) {

DeleteObject(hbmp);

hbmp = CreateBitmap(SCREEN\_WIDTH, SCREEN\_HEIGHT, 1, 32, videoRecvBuff);

InvalidateRect((HWND)Hrw, NULL, false);

bytesToRead = 4 \* SCREEN\_WIDTH \* SCREEN\_HEIGHT;

}

}

return 0;

}

DWORD WINAPI ThreadSoundRecorder(LPVOID \_param)

{

while (true)

{

HRESULT hr = CoInitializeEx(NULL, COINIT\_APARTMENTTHREADED | COINIT\_DISABLE\_OLE1DDE);

RecordAudioStream(L"temp\_send.wav");

FILE\* temp;

fopen\_s(&temp, "temp\_send.wav", "rb");

int len = fread(&voiceBuf, 1, 6400000, temp);

for (int i = 1; i < clientSockets.size(); i++)

{

if (clientSockets[i].voiceSocket && clientSockets[i].voiceSocket != SOCKET\_ERROR)

{

int sizeSent = send(clientSockets[i].voiceSocket, (const char\*)voiceBuf, len, 0);

if (sizeSent == SOCKET\_ERROR)

{

int j = WSAGetLastError();

closesocket(clientSockets[i].voiceSocket);

clientSockets[i].voiceSocket = SOCKET\_ERROR;

}

}

}

fclose(temp);

CoUninitialize();

}

return 0;

}

DWORD WINAPI ThreadFuncVoiceListener(LPVOID \_param)

{

int nbytes;

sockaddr\_in sender\_addr;

int address\_size = sizeof(struct sockaddr\_in);

while (true) {

nbytes = recv(voiceSocket, voiceBuf, 8294400,0);

if (nbytes == SOCKET\_ERROR) {

closesocket(voiceSocket);

break;

}

FILE\* temp;

errno\_t err = fopen\_s(&temp, "temp\_recv.wav", "wb");

if (!err) {

int len = fwrite(&voiceBuf, 1, nbytes, temp);

fclose(temp);

PlaySound(L"temp\_recv.wav", NULL, SND\_ASYNC | SND\_NODEFAULT);

}

}

return 0;

}

DWORD WINAPI ThreadClientListen(LPVOID \_param)

{

int index = reinterpret\_cast<int>(\_param);;

try

{

while (clientSockets[index].chatSocket

!=

SOCKET\_ERROR)

{

std::string messageBuffer(BUFFER\_SIZE, '\0');

int receivedSize = recv(clientSockets[index].chatSocket,

&messageBuffer[0],

messageBuffer.size(),

0);

if (receivedSize != SOCKET\_ERROR && clientSockets[index].chatSocket != SOCKET\_ERROR)

{

if (!clientSockets[index].name.size())

{

clientSockets[index].name = messageBuffer;

clientSockets[index].name.resize(strlen(&messageBuffer[0]));

char connectMsg[] = "Подключился: ";

EnterCriticalSection(&critSec);

Brodcast(connectMsg + messageBuffer);

}

else

{

EnterCriticalSection(&critSec);

std::string longMessageCreator = clientSockets[index].name + ": " + messageBuffer;

Brodcast(longMessageCreator);

}

LeaveCriticalSection(&critSec);

}

else

{

EnterCriticalSection(&critSec);

closesocket(clientSockets[index].chatSocket);

clientSockets[index].chatSocket = SOCKET\_ERROR;

LeaveCriticalSection(&critSec);

}

}

}

catch (std::exception& ex)

{

std::cout << ex.what() << std::endl;

}

EnterCriticalSection(&critSec);

closesocket(clientSockets[index].chatSocket);

clientSockets[index].chatSocket = NULL;

Brodcast("Отключилсся: " + clientSockets[index].name);

clientSockets[index].name.resize(0);

connectedClientsCount--;

LeaveCriticalSection(&critSec);

return 0;

}

}