Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проектированию

по курсу «Основы операционных систем»

на тему «Система программ обмена сообщениями»

Выполнил:

студент группы 22ВВП1

Расторгуева К.В.

Принял:

к.т.н. доцент Егоров В.Ю.

Пенза 2024

СОДЕРЖАНИЕ

[РЕФЕРАТ 5](#_Toc184755515)

[ВВЕДЕНИЕ 6](#_Toc184755516)

[1. ПОСТАНОВКА ЗАДАЧИ 7](#_Toc184755517)

[2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 8](#_Toc184755518)

[3. ОПИСАНИЕ РАЗРАБОТКИ ПРОТОКОЛА 9](#_Toc184755519)

[4. ОПИСАНИЕ ПРОГРАММЫ 12](#_Toc184755520)

[4.1 Описание Сервера 12](#_Toc184755521)

[4.2 Описание Клиента 19](#_Toc184755522)

[5. ОТЛАДКА И ТЕСТИРОВАНИЕ 29](#_Toc184755523)

[ЗАКЛЮЧЕНИЕ 36](#_Toc184755524)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 37](#_Toc184755525)

[ПРИЛОЖЕНИE A – UML-диаграмма последовательности 38](#_Toc184755526)

[ПРИЛОЖЕНИE В – UML-диаграмма деятельности Сервера 39](#_Toc184755527)

[ПРИЛОЖЕНИE С – UML-диаграмма деятельности Клиента 40](#_Toc184755528)

[ПРИЛОЖЕНИE D – UML-диаграмма состояний Сервера 41](#_Toc184755529)

[ПРИЛОЖЕНИE E – Дополнительные UML-диаграммы 42](#_Toc184755530)

[ПРИЛОЖЕНИE H – Листинг Сервера 43](#_Toc184755531)

[ПРИЛОЖЕНИE I – Листинг Клиента Windows Forms 58](#_Toc184755532)

[ПРИЛОЖЕНИE J – Листинг Клиента DLL библиотека 68](#_Toc184755533)

# РЕФЕРАТ

Пояснительная записка 75 с., 12 рис., 1 таблица, 2 источника, 8 приложений

ПРОТОКОЛ ПЕРЕДАЧИ ДАННЫХ, СИСТЕМА ПРОГРАММ ОБМЕНА ДАННЫМИ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ПОЧТОВЫЙ ЯЩИК, ПОЧТОВЫЙ КАНАЛ

Целью проекта является создание системы программ обмена сообщениями клиент-серверной архитектуры.

В ходе работы был проведен анализ предметной области, установлены задачи проекта для достижения поставленной цели.

Результатом проведенной работы является создание функционирующего приложения для обмена сообщениями между Пользователями.

# ВВЕДЕНИЕ

Обмен сообщениями представляет собой процесс передачи данных между различными системами и компонентами. Принципы работы данных систем включают в себя как синхронные, так и асинхронные методы передачи, позволяющие реализовать различные сценарии взаимодействия.

Важно отметить, что существуют разные модели обмена сообщениями, такие как соединение точка-точка, где сообщения передаются от одного отправителя к одному получателю, и соединение точка-многоточка, позволяющий отправлять сообщения сразу нескольким получателям. Эти модели имеют свои особенности и применимы в различных ситуациях, в зависимости от требований к надежности, производительности и масштабируемости.

Клиент-серверная архитектура является одним из наиболее распространенных подходов в проектировании систем обмена сообщениями. В этой архитектуре четко разграничены роли Клиента и Сервера. Клиент выступает в роли инициатора запросов, в то время как Сервер обрабатывает эти запросы и отдает необходимые данные. Взаимодействие между Клиентом и Сервером происходит через протоколы, которые регламентируют порядок передачи сообщений.

Таким образом, изучение систем обмена сообщениями, их принципов, моделей и архитектур является важным шагом к созданию эффективных и надежных приложений, способных обеспечивать высокую степень взаимодействия в условиях динамически развивающегося технологического окружения.

# 1. ПОСТАНОВКА ЗАДАЧИ

Целью работы является реализация системы программ клиент-серверной архитектуры, взаимодействующей посредством самостоятельно разработанного протокола передачи данных.

Разрабатываемая система так же должна удовлетворять следующим требованиям:

— наличие идентификации Пользователя на Сервере посредством Клиента;

— возможность передачи текстовых сообщений между Пользователями (индивидуальных и широковещательных);

— возможность передачи файлов.

Функционал Сервера включает в себя:

— организацию взаимодействия между клиентами (передача сообщений, файлов);

— отложенную передачу данных;

— ведение учета онлайн Пользователей;

— организацию отображения уникального стиля отображения текстовых сообщений.

Функционал Клиента включает в себя:

— графический опорный интерфейс Пользователя;

— взаимодействие с Сервером.

# 2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Мэйлслот — один из механизмов межпроцессного взаимодействия, обеспечивающий однонаправленную передачу информации и позволяющий производить широковещательную рассылку сообщений по сети. Мэйлслот является клиент-серверным интерфейсом. Сервер мэйлслотов — процесс, который создаёт мэйлслот и может читать из него информацию. Мэйлслот существует до тех пор, пока не закрыты все его серверные дескрипторы. Если несколько серверных процессов внутри домена создадут мэйлслоты с одинаковым именем, то сообщения, адресованные этому мэйлслоту и посылаемые в домен, будут приниматься всеми создавшими его процессами. Клиентом мэйлслота может быть любой процесс, знающий его имя. Клиент записывает в мэйлслот сообщения для передачи их посредством датаграмм серверу. Один и тот же процесс может быть одновременно клиентом и сервером мэйлслотов.

Для создания мэйлслотов используется специальное пространство имён «\\.\*mailslot*\[путь]имя». Для записи информации в мэйлслот на локальном компьютере клиентом используется то же самое имя, что использовалось сервером для создания мэйлслота. Для записи информации в мэйлслот удалённого компьютера используется форма «\\имя компьютера\*mailslot*\[путь]имя». Для записи информации во все мэйлслоты домена указывается имя домена или звёздочка, обозначающая первичный домен системы: «\\имя домена\*mailslot*\[путь]имя» или «\\\*\*mailslot*\[путь]имя».

Для создания мэйлслота сервером используется специальная функция *Windows API CreateMailslot*. Для создания дескриптора мэйлслота клиентом, записи информации клиентом, чтения её сервером и закрытия дескриптора используются функции для работы с файлами *CreateFile*, *WriteFile*, *ReadFile*, *CloseHandle*.

# 3. ОПИСАНИЕ РАЗРАБОТКИ ПРОТОКОЛА

В ходе написания системы программ клиент-серверной архитектуры был разработан следующий протокол передачи данных, представленный на рисунке 1.

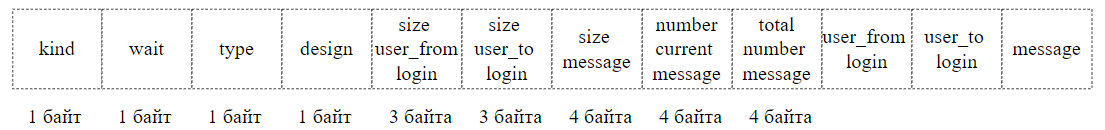


Рисунок 1 ­- Протокол передачи данных

1. *kind* – вид сообщения:

‘0’ – личное сообщение,

‘1’ – широковещательное сообщение;

1. *wait* - признак отложенного сообщения:

‘0’ – мгновенное сообщение,

‘1’ – отложенное сообщение;

1. *type* – тип сообщения:

‘0’ – текстовое сообщение,

‘1’ – файл *txt*,

‘2’ – файл *bmp*,

‘3’ – файл *jpg*,

‘4’ – файл *gif*,

‘5’ – файл *png*;

1. *design* – стиль написания текста сообщения:

‘0’ – стиль по умолчанию,

‘1’ – жирный,

‘2’ – наклонный,

‘3’ – подчеркнутый,

‘4’ – жирный + наклонный,

‘5’ – жирный + подчеркнутый,

‘6’ – наклонный + подчеркнутый,

‘7’ – жирный + наклонный + подчеркнутый;

1. *size user\_from login* – размер логина отправителя;
2. *size user\_to login* – размер логина получателя;
3. *size message* – размер сообщения;
4. *number current message* – номер текущего сообщения;
5. *total number message* – общее количество сообщений;
6. *user\_from login* – логин отправителя;
7. *user\_to login* – логин получателя;
8. *message* – сообщение.

Вид сообщения хранится в поле *kind*. При значении ‘0’ сообщение назначено для конкретного Пользователя, имя которого хранится в поле *user\_to login* (логин получателя). При значении ‘1’ сообщение широковещательное – отправляется всем Пользователям.

Признак отложенного сообщения хранится в поле *wait*. Значение ‘0’ назначается для сообщений, предназначенных для мгновенной отправки. Значение ‘1’ предназначено для сообщений, отправляемых через промежуток времени, который отсчитывается Сервером.

Поле *type* содержит тип сообщения. Значение ‘0’ присваивается текстовым сообщениям. Значения, отличные от нуля, присваиваются сообщениям, в которых передаются файловые данные. Значение при передаче файлов определяется типом передаваемого вложения.

Поле *design* определяет стиль отображения текстового сообщения на стороне получателя. По умолчанию используется ‘0’ – дефолтный стиль написания. Значения отображают различные комбинации жирного, наклонного и подчеркнутого написания.

Значение в поле *size user\_from login* (размер логина отправителя) определяет длину имени отправителя – поле *user\_from login* (логин отправителя). Значение в поле *size user\_to login* (размер логина получателя) определяет длину имени получателя – поле *user\_to login* (логин получателя). Значение в поле *size message* (размер сообщения) определяет длину информационного сообщения – поле *message* (сообщение). Данные необходимы для корректной отправки сообщения нужному адресату.

При отправке сообщения более 300 байт оно разбивается на части. Поле *number current message* содержит номер текущего отправляемого сообщения. Поле *total number message* содержит количество частей, на которые было разбито сообщение. Получение двух параметров - *number current message* и *total number message* - позволяет осуществить проверку корректного получения сообщения, разбитого на несколько частей.

Перечень кодов возможных ошибок на стороне Сервера:

501 – ошибка создания mailslot.

502 – ошибка записи в mailslot.

503 – ошибка создания таймера.

504 – ошибка установки таймера.

505 – некорректная отправка сообщения.

506 – ошибка создания объекта-мьютекса.

507 – ошибка создания объекта-события.

508 – ошибка записи в файл.

Код успешного выполнения операции = 200 OK.

Перечень кодов возможных ошибок на стороне Клиента:

401 – ошибка создания mailslot.

402 – ошибка создания или открытия файла.

403 – ошибка записи в mailslot.

406 – неприемлемый логин/пароль.

Код успешного выполнения операции = 200 OK.

# 4. ОПИСАНИЕ ПРОГРАММЫ

## 4.1 Описание Сервера

Для ведения учета Пользователей создается структура *UserList*, содержащая поля *Login*, *Password*, *NameOfMailslot*.

При запуске Сервера в основной функции *main()* создаются 3 нити:

— нить для регистрации Пользователя (*hThreadsRegistration*);

— нить для отправки списка онлайн Пользователей (*hThreadsOnlineUserList*);

— нить для отслеживания выхода Пользователя из приложения (*hThreadsDisconnection*).

Помимо нитей создается объект-событие *hEvent* и объект-мьютекс *hMutex* для дальнейшей синхронизации, позволяющей корректно выводить список онлайн Пользователей.

hEvent = CreateEvent(NULL, FALSE, FALSE, "UserRegistrationEvent");

if (hEvent == NULL) {

// Обработка ошибки

fprintf(stderr, "CreateEvent failed (%ld)\n", GetLastError());

return -1;

}

Также создается почтовый объект с именем «*\\\\.\\mailslot\\ChannelFromServer*» для чтения получаемых сообщений от всех клиентов. Затем в бесконечном цикле осуществляется проверка созданного почтового ящика и чтение сообщение при наличии. Полученное сообщение передается в функцию *ThreadFuncForSending()* для отправки сообщения предназначенному получателю.

while (1) {

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslot, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

PrintCurrentTime();

printf("Получено новое сообщение: %s", szBuf);

if (szBuf[2] == '0') {hThreads = CreateThread(NULL, 0, ThreadFuncForSending, (LPVOID)szBuf, 0, NULL);

WaitForSingleObject(hThreads, INFINITE);

}

}

}

Sleep(1);

}

Если полученное сообщения является файлом, то на Сервере формируется файл с указанным в протоколе расширением и добавляется полученная информация в файл. Затем сформированный файл открывается для чтения. Прочитанная информация из файла разделяется на части по 300 байт и отправляется адресату.

Функция *ThreadFuncForSending()* разделяет полученную строку на отдельные компоненты согласно протоколу:

— в переменную *UserName* записывается имя отправителя;

— в переменную *RecepientName* записывается имя получателя;

— в переменную *iMsgSizeCorrect* записывается размер сообщения;

— в переменную *PartProtocolDesign* записывается номер кода стиля текста;

— в переменную *Msg* записывается текст сообщения.

Проверка на признак отложенного сообщения производится по второму символу в строке. Если сообщение обладает таким признаком вызывается функция таймера *FuncForTimer()*.

Проверка на вид сообщения – личное или широковещательное – осуществляется по первому символу в строке.

При отправке личного сообщения осуществляется проход по структуре *UserList* для нахождения отправителя и получателя. Уникальное имя канала для каждого Пользователя содержится в поле *NameOfMailslot* структуры.

while (head\_copy != NULL) {

if ((strcmp(head\_copy->Login, RecepientName) == 0) || (strcmp(head\_copy->Login, UserName) == 0)) {

hMailslot = CreateFileA(head\_copy->NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

if (!WriteFile(hMailslot, Msg, strlen(Msg), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -501;

}

CloseHandle(hMailslot);

//функция записи в файл для сохранения переписки

strcpy(Msg\_copy, Msg);

FuncMailStorage(head\_copy->Login, Msg\_copy);

}

head\_copy = head\_copy->next;

}

Для широковещательной рассылки сообщений осуществляется проход по структуре *UserList* и отправление сообщения каждому Пользователю по его уникальному каналу.

Функция *hThreadsRegistration* создает почтовый ящик с именем «*\\\\.\\mailslot\\RegistrationOnServerChannel*» и открывает его для чтения. При получении сообщения в формате Login\_Password (где Login – логин, Password – пароль Пользователя, проходящего регистрацию) осуществляется проверка структуры *UserList* на наличие Пользователя с такими данными. Пользователь осуществляет регистрацию. Если указанный логин уже существует в структуре *UserList*, то Клиенту отправляется ошибка «*406\_Not\_Acceptable\_Login*». При успешной проверке логина и пароля нового Пользователя создается новый узел, добавляемый в структуру *UserList*. При входе в систему Пользователь указывает идентификационные данные, которые проходят проверку через БД.

После формирования уникального имени канала для нового Пользователя он отправляется Клиенту по почтовому каналу с именем «*\\\\.\\mailslot\\GetUnicNameFromServerChannel*»

do {

hMailslotUnicNameUser = CreateFile(TEXT("\\\\.\\mailslot\\GetUnicNameFromServerChannel"), GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

} while (hMailslotUnicNameUser == INVALID\_HANDLE\_VALUE);

if (!WriteFile(hMailslotUnicNameUser, UnicNameChannel, strlen(UnicNameChannel), &cbWritten, NULL)) {

CloseHandle(hMailslotUnicNameUser);

fprintf(stdout, "WriteFile: Error %ld\n", GetLastError());

return -502;

}

Для отправки списка онлайн Пользователей была разработана функция *SendOnlineUserList()*. Для синхронного обновления у Клиентов списка онлайн Пользователей применяется объект-событие *hEvent*. Цикл отправки списка онлайн клиентов срабатывает только в том случае, если объект-событие *hEvent* переходит в отмеченное состояние. В свою очередь объект-событие *hEvent* устанавливается в отмеченное состояние при регистрации Пользователя на Сервере и при закрытии Пользователем приложения.

Для исключения одновременного обращения к глобальной структуре *UserList* нескольких нитей используется объект-мьютекс *hMutex*. Перед началом работы со структурой *UserList* производится захватывание объекта-мьютекса *hMutex*, после завершения работы – освобождение.

Отправка списка онлайн Пользователей производится каждому Клиенту по уникальному имени почтового канала.

while (1) {

WaitForSingleObject(hEvent, INFINITE); // Ожидание события

char snim[5];

ListOnlineUser[0] = '\0';

WaitForSingleObject(hMutex, INFINITE); // Блокировка мьютекса

head\_copy = head;

while (head\_copy != NULL) {

char NameOfMailslot[128];

char postfix[7] = "Online";

strcpy(NameOfMailslot, head\_copy->NameOfMailslot);

strcat(NameOfMailslot, postfix);

do {

hMailslot = CreateFileA(NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

} while (hMailslot == INVALID\_HANDLE\_VALUE);

if (!WriteFile(hMailslot, ListOnlineUser, strlen(ListOnlineUser), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

CloseHandle(hMailslot);

head\_copy = head\_copy->next;

}

ReleaseMutex(hMutex);

ResetEvent(hEvent);

}

Вспомогательная функция *ClientDisconnection()* создана для корректного формирования списка онлайн Пользователей. В функции реализовано прослушивание почтового канала «*\\\\.\\mailslot\\ClientDisconnectionChannel*». При появлении сообщения, содержащего логин Пользователя, вышедшего из приложения, осуществляется поиск узла в структуре *UserList* с указанными данными. Затем найденный узел удаляется из структуры *UserList*.

hMailslotDisconnection = CreateMailslot(TEXT("\\\\.\\mailslot\\ClientDisconnectionChannel"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslotDisconnection == INVALID\_HANDLE\_VALUE) return -501;

while (1) {

fReturnCode = GetMailslotInfo(hMailslotDisconnection, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslotDisconnection, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

head\_copy = head;

previous = NULL;

while (head\_copy != NULL) {

if (strcmp(head\_copy->Login, szBuf) == 0) {

if (previous == NULL) {

head = head\_copy->next;

}

else {

previous->next = head\_copy->next;

}

free(head\_copy);

break;

}

previous = head\_copy;

head\_copy = head\_copy->next;

}

}

}

}

При отправке текстового сообщения функция *ThreadFuncForSending()* вызывает функцию *FuncMailStorage()*, которая выполняет сохранение всех отправляемых сообщений в файл. В функцию передается логин Пользователя, которому отправляется сообщение. Учитывая этот параметр, формируется путь к файлу, название которого обладает форматом *Login.txt*. Если файл с таким названием уже существует, он открывается для дополнения информации. Иначе создается новый файл с указанным именем. В файл производится запись стиль текста сообщения, время отправки сообщения и текст сообщения.

snprintf(fullFileName, sizeof(fullFileName), "%s%s", UserLogin, FileExpansion);

FILE\* file = fopen(fullFileName, "a");

if (file == NULL) {

printf("Ошибка открытия файла для записи переписки");

return 508;

}

fprintf(file, "%s", designText);

time\_t now = time(NULL);

struct tm\* local = localtime(&now);

fprintf(file, "%02d.%02d.%4d %02d:%02d:%02d ",

local->tm\_mday,

local->tm\_mon + 1,

local->tm\_year + 1900,

local->tm\_hour,

local->tm\_min,

local->tm\_sec);

fprintf(file, "%s", massage);

fclose(file);

## 4.2 Описание Клиента

При запуске программы происходит вывод на экран *Windows Forms* формы – визуальной поверхности, на которой выводится информация для Пользователя. При нажатии на кнопку «Зарегистрироваться» из *DLL* библиотеки вызывается функция *UserRegistration()*, в которую передается имя Пользователя, состоящее из логина и пароля, которые он ввел. При нажатии на кнопку «Войти» идентификационные данные Пользователя проходят проверку через БД.

Функция *UserRegistration()* открывает почтовый канал «*\\\\.\\mailslot\\RegistrationOnServerChannel*», созданный сервером, отправляет имя Пользователя для его регистрации на сервере. При успешной регистрации Сервер формирует уникальное имя канала для Пользователя и передает его Клиенту через почтовый канал «*\\\\.\\mailslot\\GetUnicNameFromServerChannel*»Сервер создает почтовый объект с уникальным каналом.

globalhMailslotOnline = CreateMailslot(lpMailslotUnicNameUserOnline, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (globalhMailslotOnline == INVALID\_HANDLE\_VALUE) MessageBox(0, "Ошибка при регистрации", "Уведомление", MB\_OK);

В случае неудачной регистрации/неудачном входе на Сервере вместо уникального имени канала возвращаются ошибки «406\_Not\_Acceptable\_Login» и «406\_Not\_Acceptable\_Password», информирующие о том, что введенный Пользователем логин уже присвоен другому Клиенту или о том, что введенный Пользователем пароль является неверным.

Кроме уникального почтового канала, через который Клиент получает сообщения, в функции создается канал для передачи списка онлайн Пользователей.

globalhMailslotOnline = CreateMailslot(lpMailslotUnicNameUserOnline, 0, MAILSLOT\_WAIT\_FOREVER, NULL);;

if (globalhMailslotOnline == INVALID\_HANDLE\_VALUE) MessageBox(0, "Ошибка при регистрации", "Уведомление", MB\_OK);

Также кнопки «Зарегистрироваться» и «Войти» запускают фоновую нить для прослушивания сообщений от сервера. Задержка в 1000 миллисекунд позволяет снизить нагрузку на процессор.

private void Form1\_Load(object sender, EventArgs e)

{

\_cancellationTokenSource = new CancellationTokenSource();

}

private async Task StartReceivingMessagesAsync(CancellationToken token)

{

string oldMessage = "";

try

{

while (!token.IsCancellationRequested)

{

string tmpMessage = await Task1Async();

if (!Equals(oldMessage, tmpMessage))

{

UpdateRichTextBox(tmpMessage);

oldMessage = tmpMessage;

}

await Task.Delay(1000);

}

}

}

private async Task<string> Task1Async()

{

return await Task.Run(() = > myfuncReceiptMessage());

}

Фоновая нить периодически вызывает из *DLL* библиотеки функцию *myfuncReceiptMessage()*, которая прослушивает уникальный канал Пользователя. Если почтовый ящик оказывается непустым и пришедшее сообщение текстовое, то функция записывает сообщение в переменную *szBuf*, указатель на которую возвращает в окно приложения *Windows Forms.* При получении сообщения, оповещающего, что был отправлен файл, то создается файл с расширением, указанным в сообщении. Также из полученного сообщения извлекается информация о номере текущего сообщения и общем количестве сообщений, которые должны прийти для корректного формирования файлы. Переданные байты из сообщения записываются в файл. При получении последнего сообщения с информацией из файла на Сервер отпрввляется сообщение о том, что файл получен.

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslot, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

char type = szBuf[0];

if (type == '0') {

globalBuf[0] = '\0';

memmove(szBuf, szBuf + 1, strlen(szBuf));

strcat(globalBuf, szBuf); }

else {

char fullFuleName[512] = "recepient.";

if (type == '1') strcat(fullFuleName, "txt");

if (type == '2') strcat(fullFuleName, "bmp");

if (type == '3') strcat(fullFuleName, "jpg");

if (type == '4') strcat(fullFuleName, "gif");

if (type == '5') strcat(fullFuleName, "png");

char bufferFile[300];

char cNumberCurrentMsg[5];

char cNumberTotalMsg[5];

char cUserNameSize[4];

char cUserName[128];

int iNumberCurrentMsg;

int iNumberTotalMsg;

int iUserNameSize;

strncpy(cNumberCurrentMsg, szBuf+1, 4); cNumberCurrentMsg[4] = '\0';

iNumberCurrentMsg = atoi(cNumberCurrentMsg);

strncpy(cNumberTotalMsg, szBuf + 5, 4);

cNumberTotalMsg[4] = '\0';

iNumberTotalMsg = atoi(cNumberTotalMsg);

strncpy(cUserNameSize, szBuf + 9, 3);

cUserNameSize[3] = '\0';

iUserNameSize = atoi(cUserNameSize);

strncpy(cUserName, szBuf + 12, 3);

cUserName[iUserNameSize] = '\0';

memmove(bufferFile, szBuf + 12 + iUserNameSize, cbRead - 12 - iUserNameSize);

FILE\* outFile = fopen(fullFuleName, "ab");

if (outFile == NULL) {

fprintf(stderr, "Error opening file for writing\n");

}

size\_t written = fwrite(bufferFile, 1, cbRead - 12 - iUserNameSize, outFile);

if (written != cbRead) {

fprintf(stderr, "Error writing data to file\n");

fclose(outFile);

}

fclose(outFile);

printf("Received data: %lu bytes\n", cbRead);

if (iNumberCurrentMsg == iNumberTotalMsg) {

MessageBox(0, "Получен новый файл", "Уведомление", MB\_OK);

char ProtocolAnswer[128] = "000000000000000000000000";

char Answer[] = "Ваш файл получен\n";

memmove(ProtocolAnswer+7, cUserNameSize, 3);

memmove(ProtocolAnswer + 10, "0017", 4);

memmove(ProtocolAnswer + 22, cUserName, iUserNameSize);

memmove(ProtocolAnswer + 22 + iUserNameSize, Answer, strlen(Answer));

myfuncSendMessage(ProtocolAnswer);}}}}

При получении нового сообщения в формате *DesignMessage* (где *Design* – бит номера кода стиля текста, *Message* – текст сообщения) в классе *UpdateRichTextBox* производится его обработка на применение определенного стиля к тексту. Для этого вызывается функция *MessageFormatting()*, применяющая определенный стиль к тексту.

FontStyle fontStyle = FontStyle.Regular;

if (message[0] == '1') fontStyle = FontStyle.Bold;

if (message[0] == '2') fontStyle = FontStyle.Italic;

if (message[0] == '3') fontStyle = FontStyle.Underline;

if (message[0] == '4') fontStyle = FontStyle.Bold | FontStyle.Italic;

if (message[0] == '5') fontStyle = FontStyle.Bold | FontStyle.Underline;

if (message[0] == '6') fontStyle = FontStyle.Italic | FontStyle.Underline;

if (message[0] == '7') fontStyle = FontStyle.Bold | FontStyle.Italic | FontStyle.Underline;

richTextBox\_forOutputMessage.SelectionFont = new Font(richTextBox\_forOutputMessage.Font, fontStyle);

После выполнения функции *MessageFormatting()* происходит вывод сообщения с добавлением текущего времени и отсечением первого байта, отведенного под стиль.

MessageFormatting(message);

message = message.Substring(1);

message = time + " " + message;

Кнопка «Зарегистрироваться» запускает фоновую нить, вызывающую из *DLL* библиотеки функцию *myfuncOnlineUserList()*, позволяющую выводить Пользователей, находящихся в сети. Для этого в функции производится чтение канала, предназначенного для получения списка онлайн Пользователей. Возвращаемым значением функции является полученный список Пользователей, находящихся в сети.

hMailslot = globalhMailslotOnline;

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslot, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

strcpy(globalBufOnline, szBuf);

}

}

Обновление списка онлайн Пользователей при выходе одного из Клиента из сети осуществляет вспомогательная функция *NotifyServerAboutDisconnection()*, которая открывает почтовый канал «*\\\\.\\mailslot\\ClientDisconnectionChannel*» и отправляет имя Пользователя, вышедшего из сети. Вызов функции *NotifyServerAboutDisconnection()* осуществляется при закрытии *Windows Forms*.

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

{

string UserName;

UserName = textBox\_forUserName.Text;

NotifyServerAboutDisconnection(UserName);

}

При нажатии на кнопку «Отправить» происходит формирование протокола отправки, в котором указывается

— вид сообщения (личное или широковещательное);

— признак отложенного сообщения;

— тип сообщения;

— стиль текста сообщения;

— размер логина получателя;

— размер логина отправителя;

— размер сообщения;

— номер текущего сообщения;

— общее количество сообщений;

— логин отправителя;

— логин получателя;

— текст сообщения.

ProtocolManager.ProtocolGeneration(sOutputMessageNew);

ProtocolManager.ProtocolGeneration(DateOfRecepient);

ProtocolManager.ProtocolGeneration(textBox\_forUserName.Text);

ProtocolManager.ProtocolGeneration(sOutputMessageNew.Length.ToString().PadLeft(4, '0'));

ProtocolManager.ProtocolGeneration(DateOfRecepient.Length.ToString().PadLeft(3, '0')); ProtocolManager.ProtocolGeneration(textBox\_forUserName.Text.Length.ToString().PadLeft(3, '0'));

ProtocolManager.ProtocolGeneration(DesignText.ToString()); ProtocolManager.ProtocolGeneration("t"); ProtocolManager.ProtocolGeneration(checkBox1\_WaitMessage.Checked?"1":"0");

ProtocolManager.ProtocolGeneration(checkBox1\_BroadcastMsg.Checked?"1":"0");

После формирования протокола из *DLL* библиотеки вызывается функция отправки сообщения на сервер *myfuncSendMessage()*. В функции открывается почтовый канала «*\\\\.\\mailslot\\ChannelFromServer*» и производится запись протокола.

hMailslot = CreateFile(lpMailslotName, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return -501;

if (!WriteFile(hMailslot, ProtocolMessage, strlen(ProtocolMessage), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

Кнопка «Загрузить историю сообщений» вызывает из *DLL* библиотеки функцию *ReadingOldMessage()*, которая принимает логин Пользователя. Функция формирует путь к файлу, название которого обладает форматом *Login.txt*. Затем происходит открытие файл с построчным чтением. Над строками файла выполняется операция конкатенации, результат сохраняется в переменную *fullBuffer*, указатель на которую является возвращаемым значением функции.

snprintf(fullFileName, sizeof(fullFileName), "%s%s", UserLogin, FileExpansion);

strcat(FilePath, fullFileName);

FILE\* file = fopen(FilePath, "r");

if (file == NULL) {

MessageBox(0, "Ошибка открытия файла для чтения", "Ошибка", MB\_OK);

//return;

}

char Buffer[512];

while (fgets(Buffer, sizeof(Buffer), file) != NULL) {

strcat(fullBuffer, Buffer);

}

fclose(file);

В основной программе полученная строка разбивается на отдельные сообщения, которые записываются в массив *ListSplitMessage*. Затем в цикле каждое сообщение, представленное форматом *DesignMessage* (где *Design* – бит номера кода стиля текста, *Message* – текст сообщения) обрабатывается функцией *MessageFormatting()*, форматируется для отсечения первого бита и выводится на экран.

string[] ListSplitMessage = ListOldMessages.Split(new[]{ '\n' }, StringSplitOptions.RemoveEmptyEntries);

string messageOutput;

foreach(var message in ListSplitMessage)

{

if (string.IsNullOrWhiteSpace(message)) continue; // Проверка на пустые сообщения

// Сохранение текущей позиции

int previousSelectionStart = richTextBox\_forOutputMessage.SelectionStart;

// Переход в конец текста

richTextBox\_forOutputMessage.SelectionStart = richTextBox\_forOutputMessage.TextLength;

richTextBox\_forOutputMessage.SelectionLength = 0; // Ничего не выделяем

// Обновление стиля перед добавлением текста

MessageFormatting(message);

messageOutput = message.Substring(1);

richTextBox\_forOutputMessage.SelectedText = messageOutput;

// Восстановление предыдущего положения курсора

richTextBox\_forOutputMessage.SelectionStart = previousSelectionStart;

richTextBox\_forOutputMessage.ScrollToCaret();

}

}

Полный код программы можно увидеть в Приложении.

# 5. ОТЛАДКА И ТЕСТИРОВАНИЕ

В качестве среды разработки была выбрана программа Microsoft Visual Studio 2022. Для отладки использовались несколько возможностей Visual Studio: точка останова, трассировка, анализ содержимого переменных. Тестирование проводилось во время разработки и также после завершения разработки.

Точка останова - это преднамеренная остановка или приостановка в программе, установленная в целях отладки. Точка останова - это средство получения знаний о программе во время ее выполнения. Во время прерывания есть возможность проверить тестовую среду (регистры общего назначения, память, журналы, файлы и т.д.), чтобы выяснить, функционирует ли программа должным образом.

Трассировка — процесс пошагового выполнения программы. В режиме трассировки программист видит последовательность выполнения команд и значения переменных на данном шаге выполнения программы, что позволяет легче обнаруживать ошибки. Трассировка может быть начата и окончена в любом месте программы, выполнение программы может останавливаться на каждой команде или на точках останова, трассировка может выполняться с заходом в процедуры и без заходов, а также осуществляться в обратном порядке.

На рисунке 2 представлено добавление нового Пользователя в структуру *UserList*.

Точка останова для проверки сформированного протокола передачи данных для отправки сообщения представлена на рисунке 3. На рисунке 4 представлено полученное Сервером сообщение. Точка останова для проверки полученного сообщения адресатом представлена на рисунке 5.

На рисунках 6 и 7 представлен результат программы при вводе некорректных данных. На рисунке 8 представлена работа программы при успешной регистрации Пользователя.

На рисунке 9 представлена работа программы попытке входа в систему незарегистрированного Пользователя.

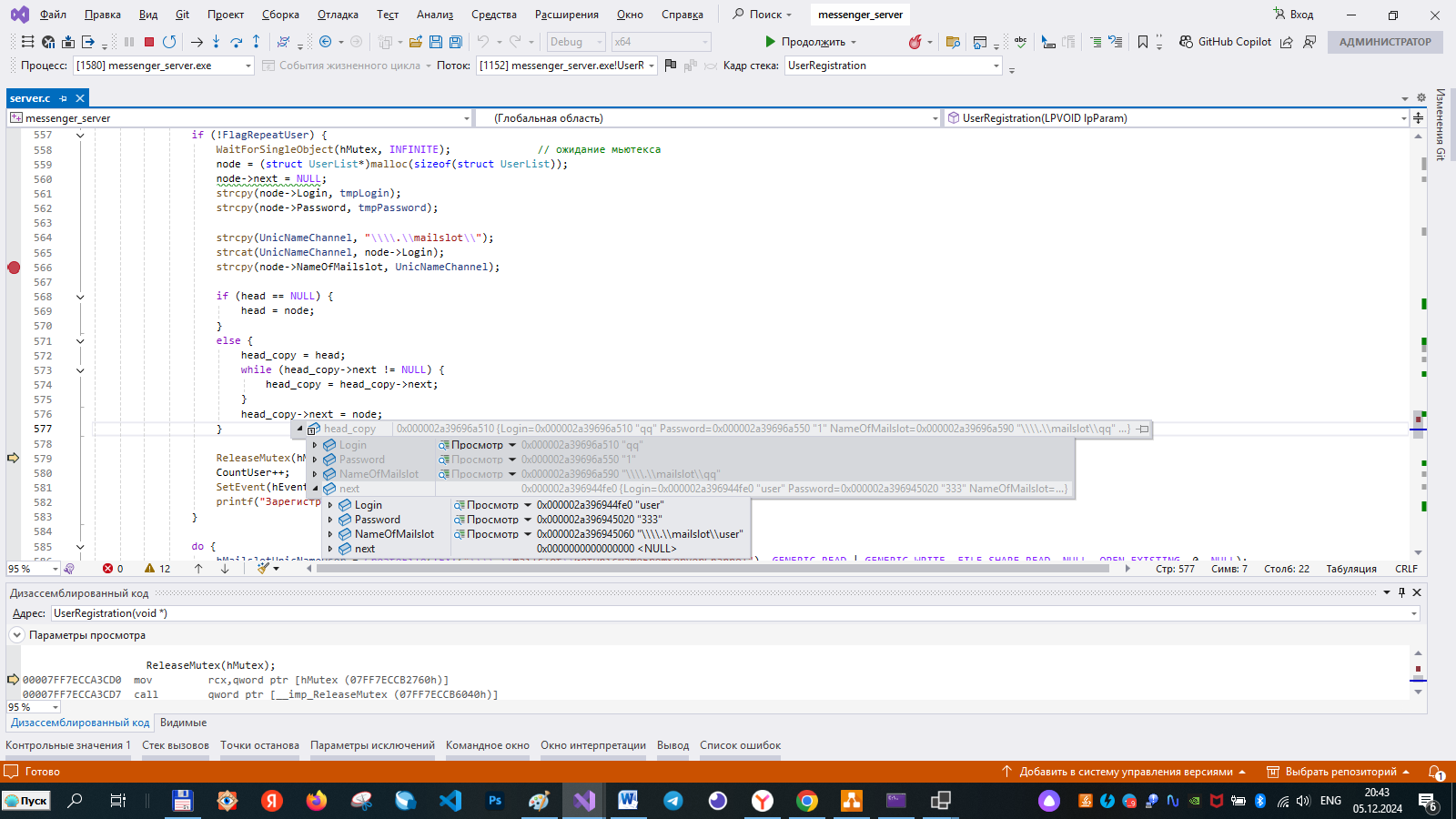


Рисунок 2 – Регистрация нового Пользователя

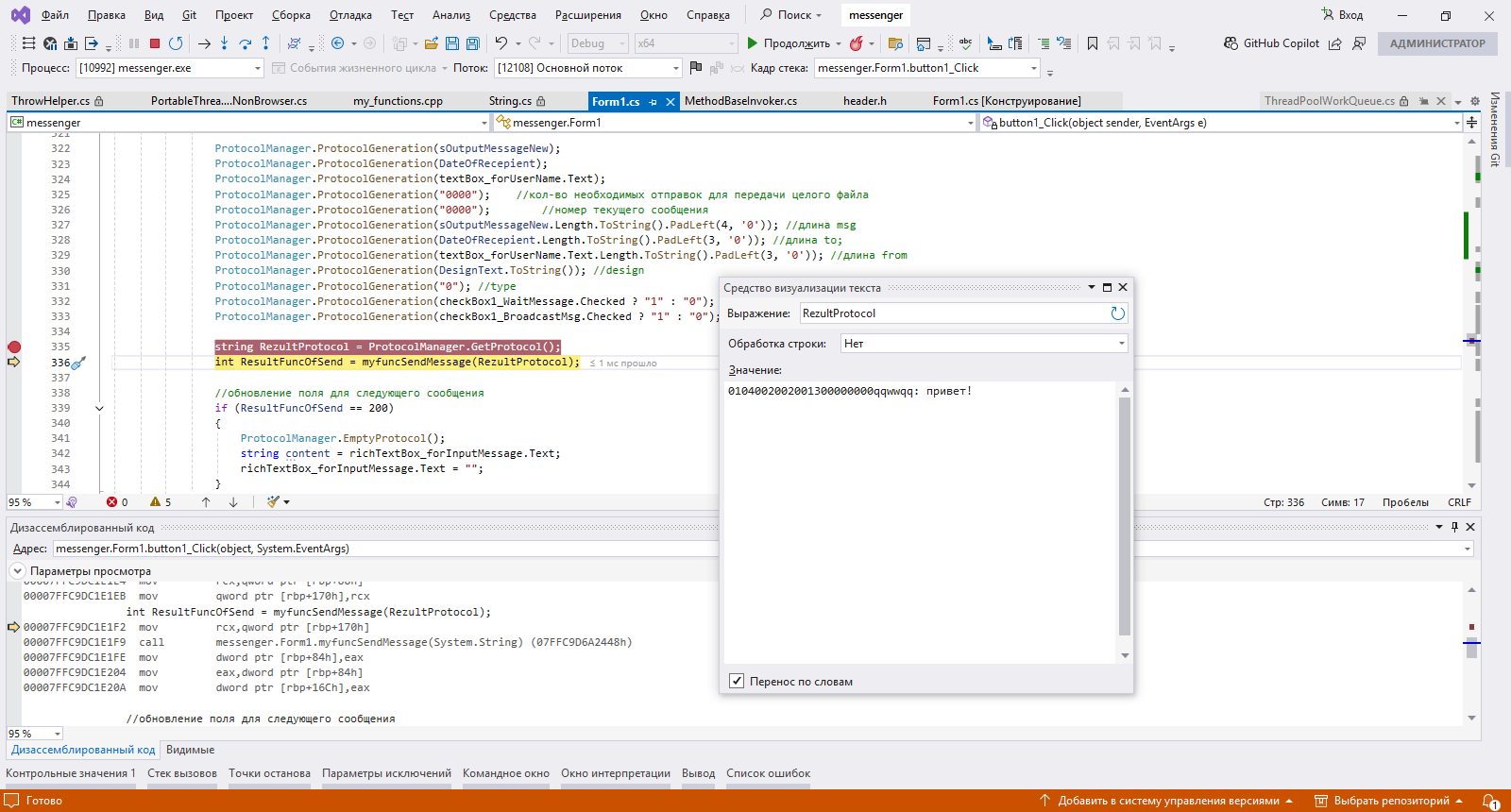


Рисунок 3 – Точка останова для проверки сформированного протокола

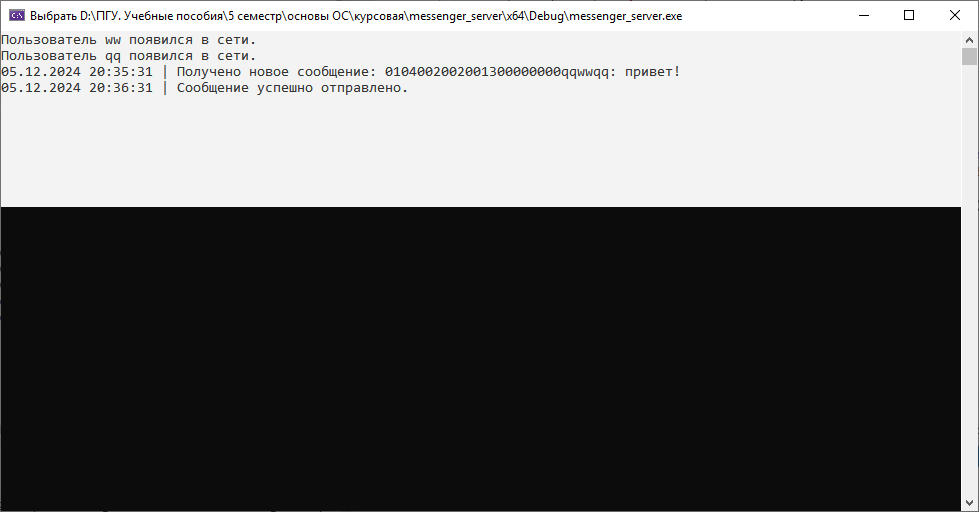


Рисунок 4 – Работа Сервера

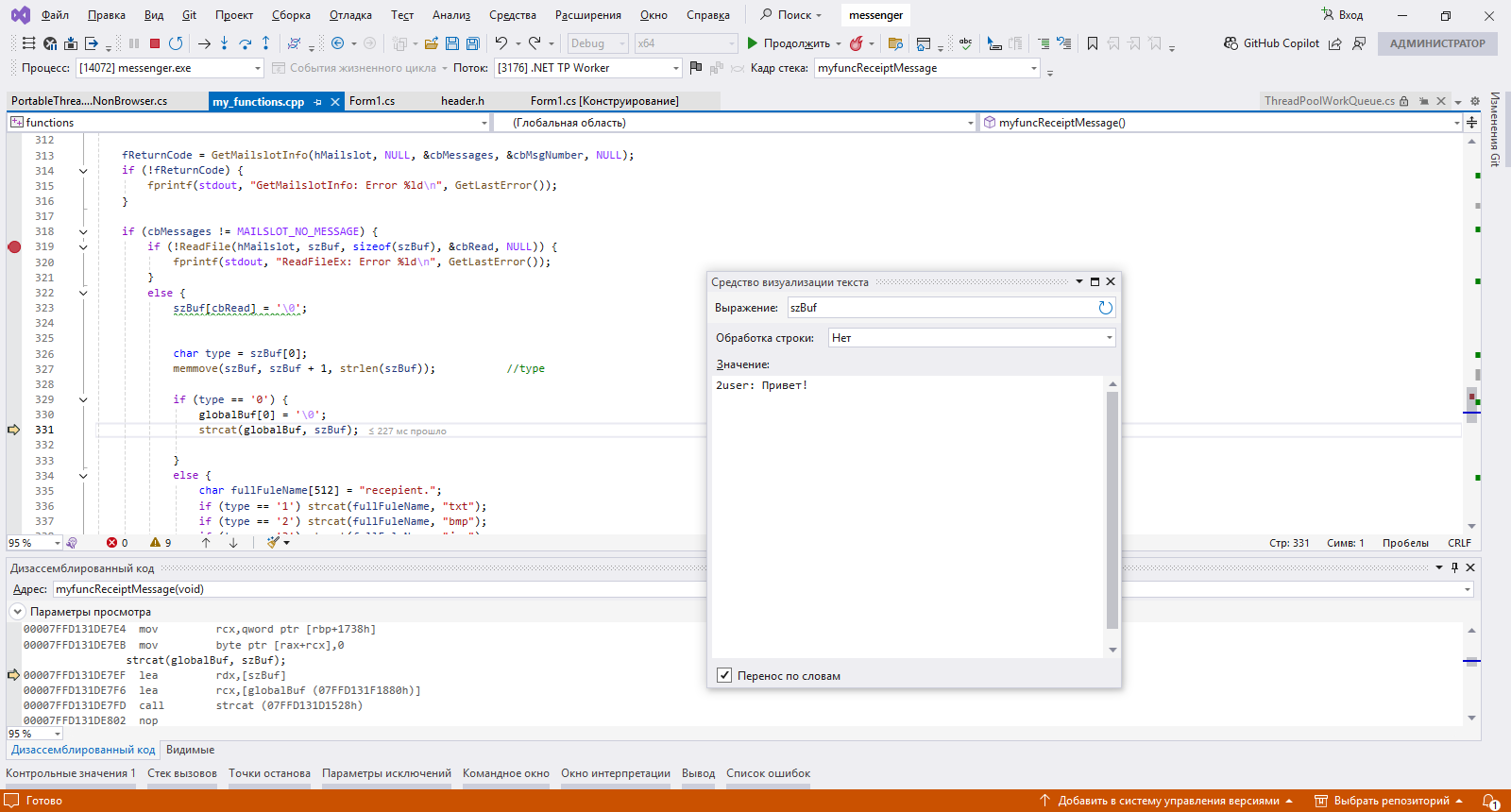


Рисунок 5 – Точка останова для проверки полученного сообщения

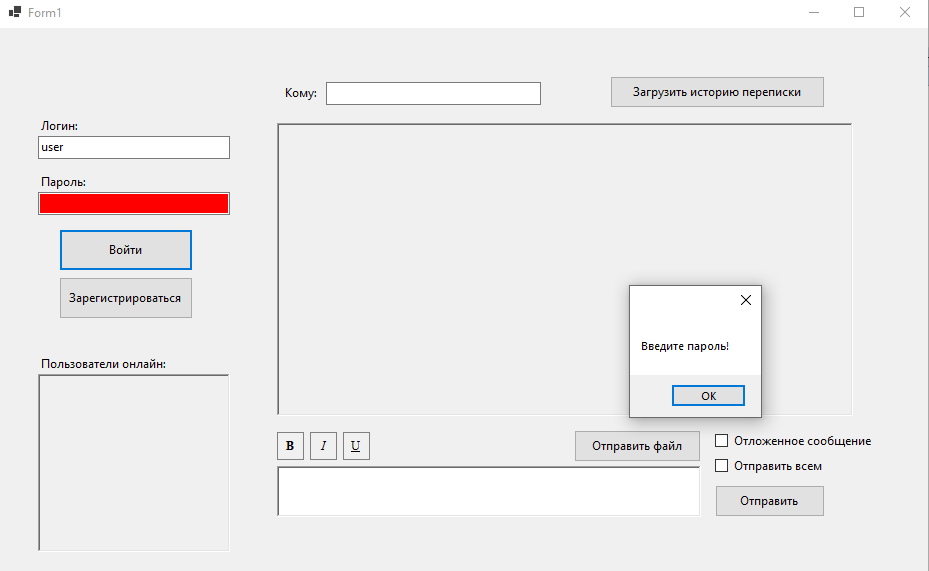


Рисунок 6 – Работа приложения при некорректном вводе исходных данных для входа/регистрации

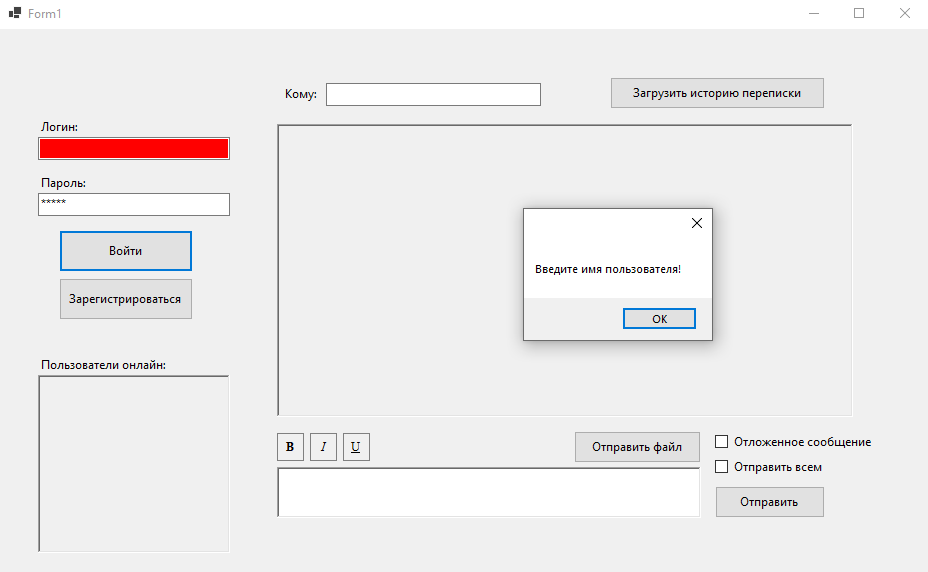


Рисунок 7 – Работа приложения при некорректном вводе исходных данных для входа/регистрации

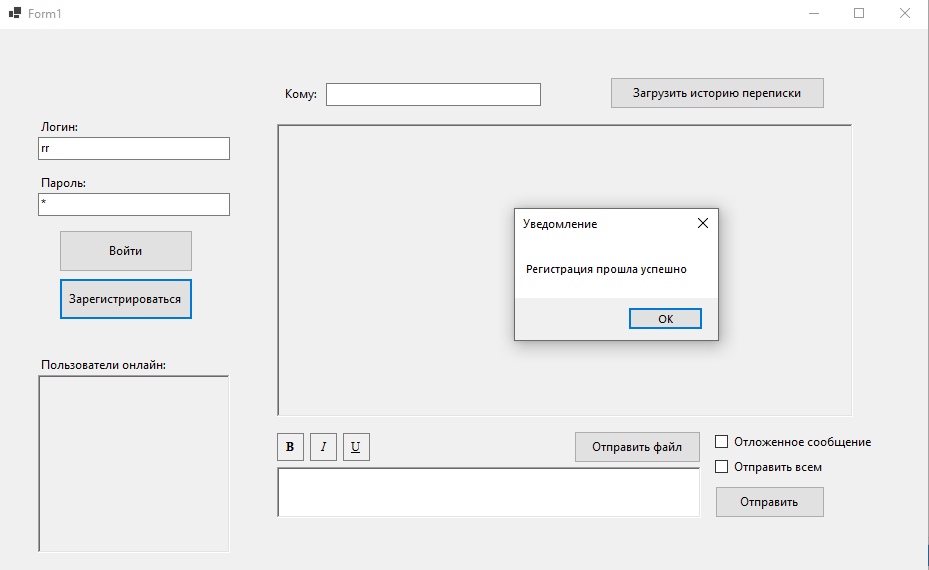


Рисунок 8 ­­– Работа программы при успешном прохождении регистрации

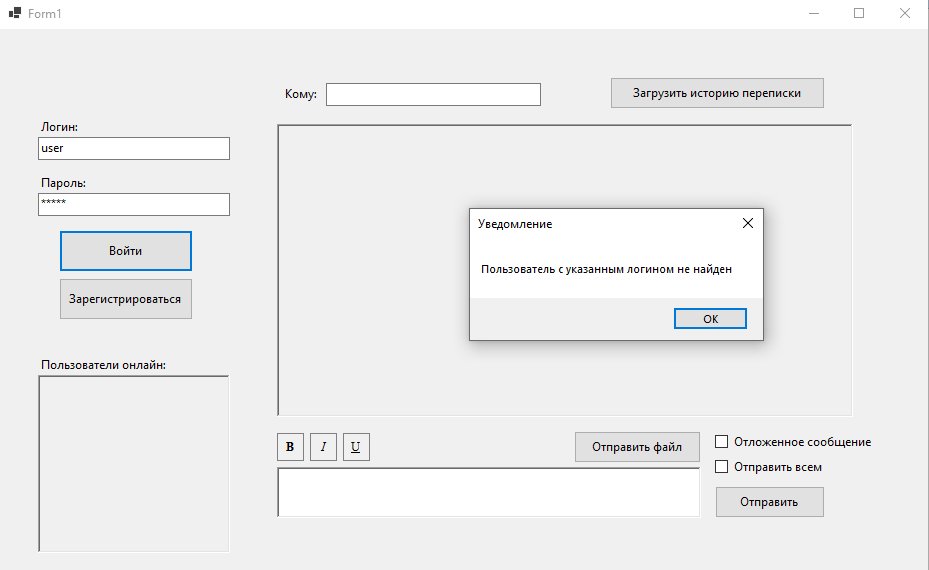


Рисунок 9 – Работа программы при вводе незарегистрированного логина

На рисунке 10 представлена работа программы при вводе логина, закрепленного за другими Пользователями. На рисунке 11 представлена работа программы попытке отправки сообщения без указания имени получателя. На рисунке 12 представлена передача сообщений между тремя Пользователями.

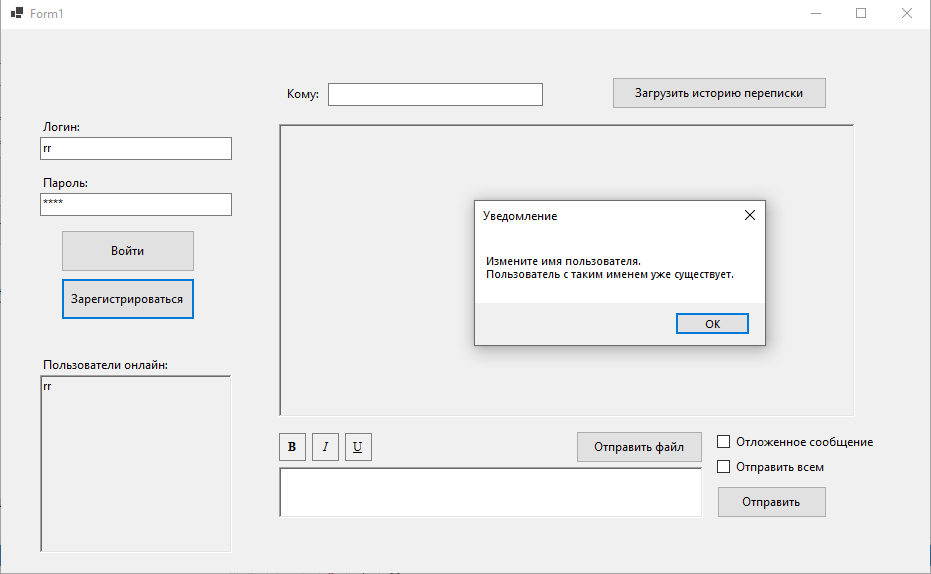


Рисунок 10 – Работа программы при вводе повторяющегося логина

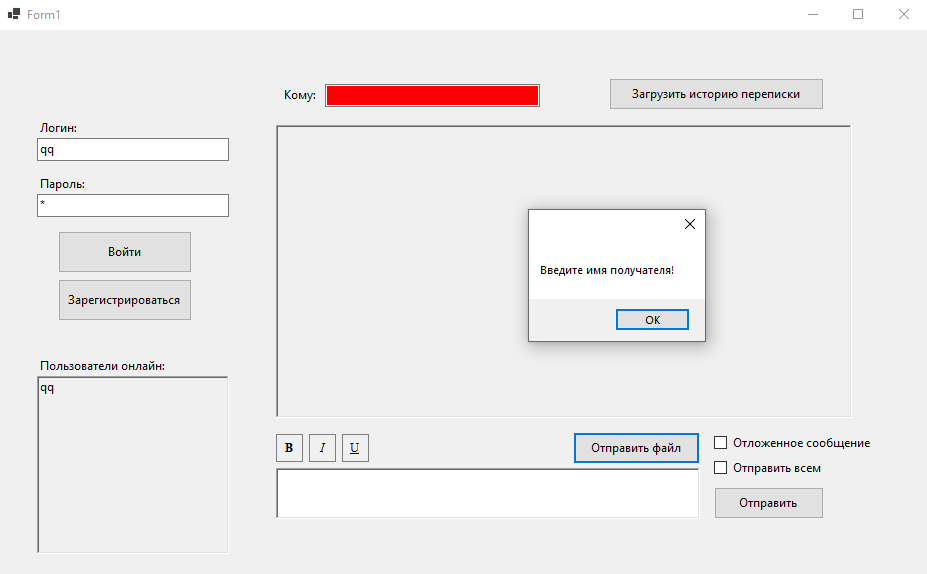


Рисунок 11 – Работа программы при пустом поле адресата

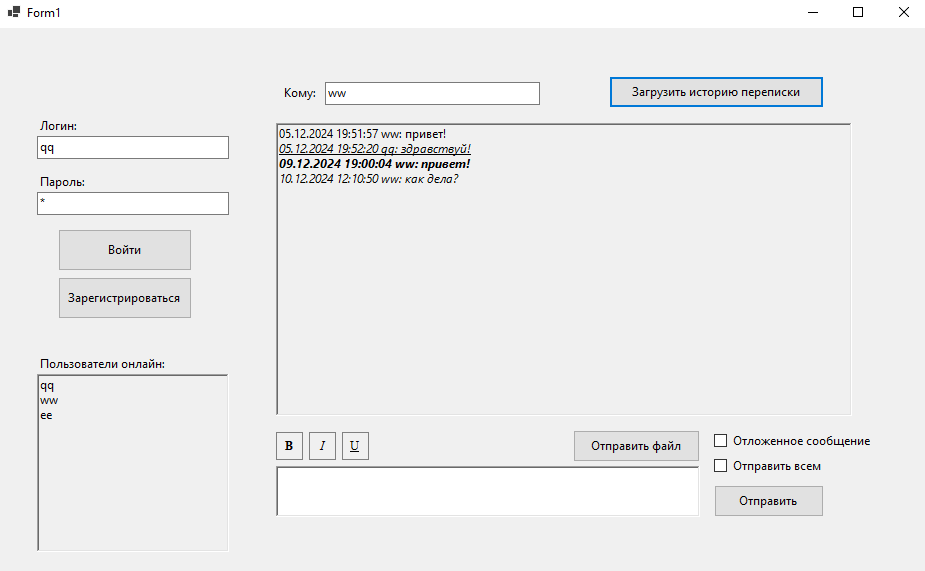


Рисунок 12 – Диалоговое окно при работе программы

Описание поведения программы при тестировании выполнено в таблице 1.

Таблица 1 – Описание поведения программы при тестировании

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Нажатие на кнопку «Войти»/ «Зарегистрироваться» при незаполненном поле «Пароль» | Отображение всплывающего окна с информационным сообщением «Введите пароль!» | Верно (рисунок 6) |
| Нажатие на кнопку «Войти»/ «Зарегистрироваться» при незаполненном поле «Логин» | Отображение всплывающего окна с информационным сообщением «Введите имя пользователя!» | Верно (рисунок 7) |
| Ввод уникального логина и пароля, не вводимых ранее | Отображение всплывающего окна с информационным сообщением «Регистрация прошла успешно» | Верно (рисунок 8) |
| Ввод уникального логина и пароля, не вводимых ранее. Нажатие на кнопку «Войти» | Отображение всплывающего окна с информационным сообщением «Пользователь с указанным логином не найден» | Верно (рисунок 9) |
| Повторный ввод одного логина. Нажатие на кнопку «Зарегистрироваться» | Отображение всплывающего окна с информационным сообщением «Измените имя Пользователя. Пользователь с таким именем уже существует» | Верно (рисунок 10) |
| Отправка сообщения нажатием на кнопку «Отправить» при незаполненном поле «Кому» | Отображение всплывающего окна с информационным сообщением «Введите имя получателя!» | Верно (рисунок 11) |

В результате тестирования программы ошибок не было выявлено.

# ЗАКЛЮЧЕНИЕ

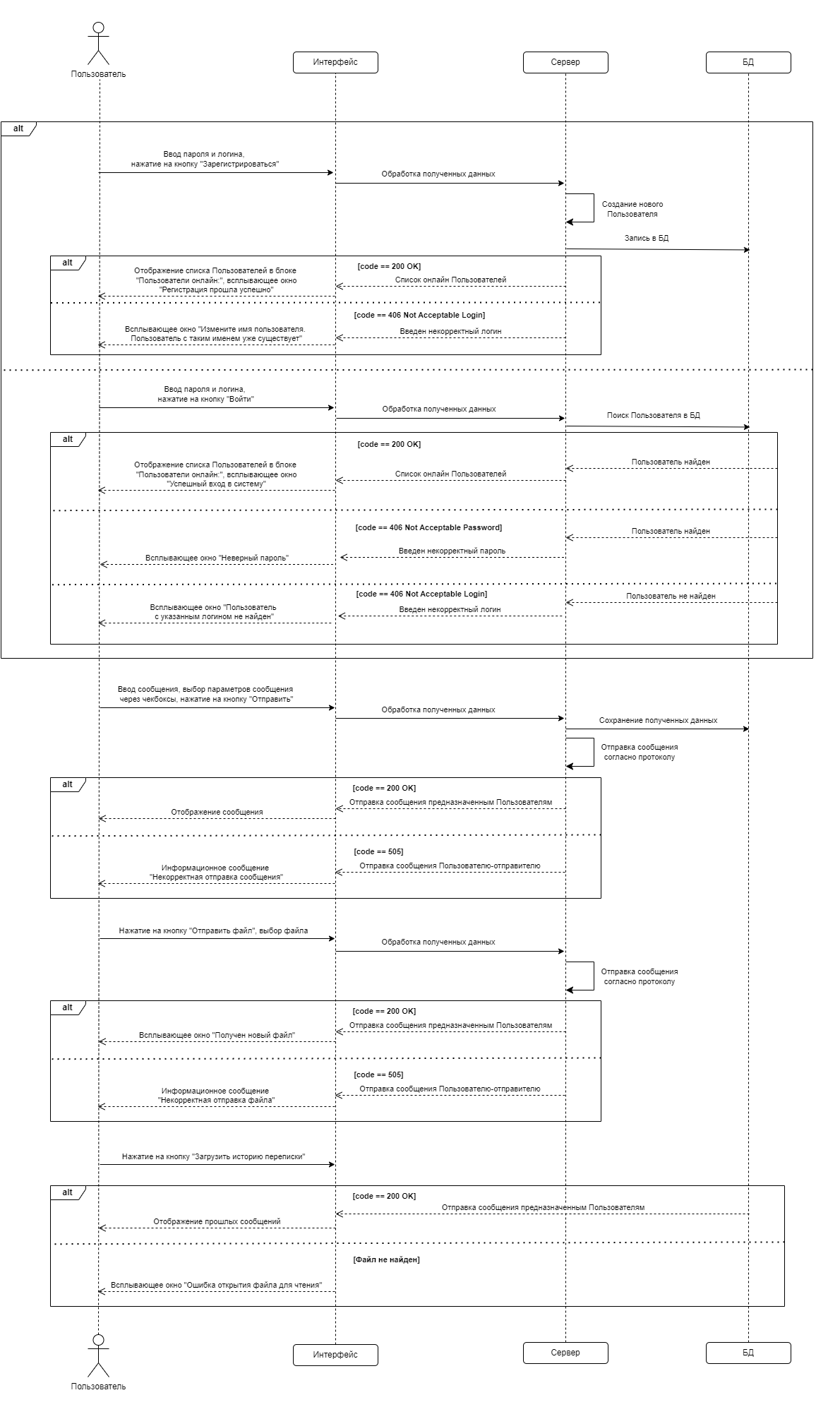
Главным результатом проведенной работы является создание функционирующей программы, которая выполняет требуемый от неё круг задач. Она предоставляет возможность передачи данных между Клиентами посредством Сервера.

В ходе выполнения курсового проекта были получены навыки разработки программ с использованием платформы *Windows Forms*. Усвоено построение клиент-серверного взаимодействия процессов с использованием почтовых ящиков. Также были закреплены навыки по работе с объектами-мьютексами, объектами-событиями и углублены знания языков программирования C, C++, C#.

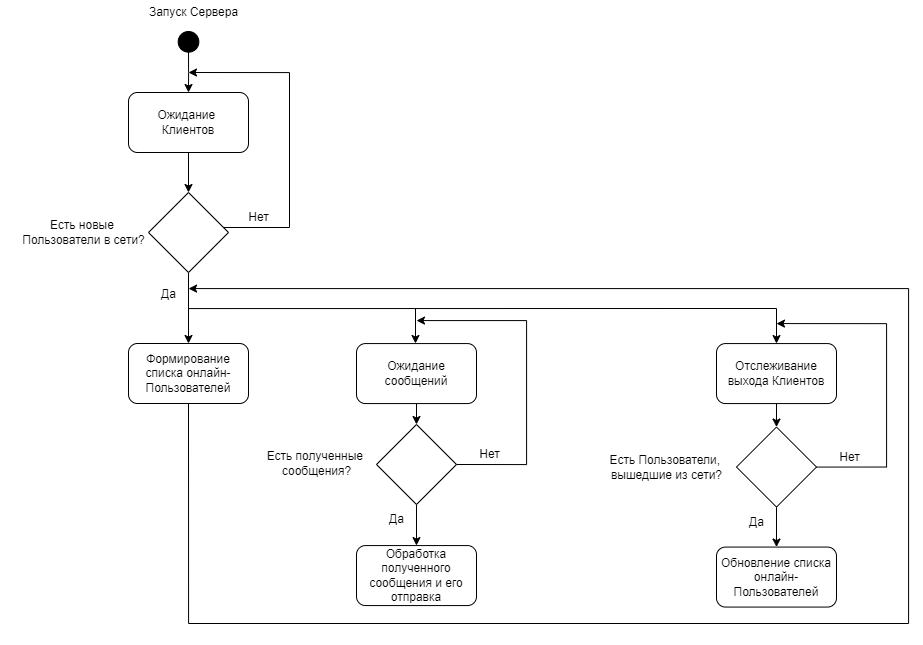
# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Пер. с англ. — 4-е изд. — Спб.: Питер; М.: Издательство «Русская Редакция»; 2008. — 720 стр.: ил.
2. Уотсон Карли, Нейгел Кристиан, Якоб Хаммер, Рид Джон Д., Скиннер, Морган Visual C# 2010: полный курс.: Пер. с англ. – М. : ООО «И.Д. Вильямс», 2011. – 960 с.:

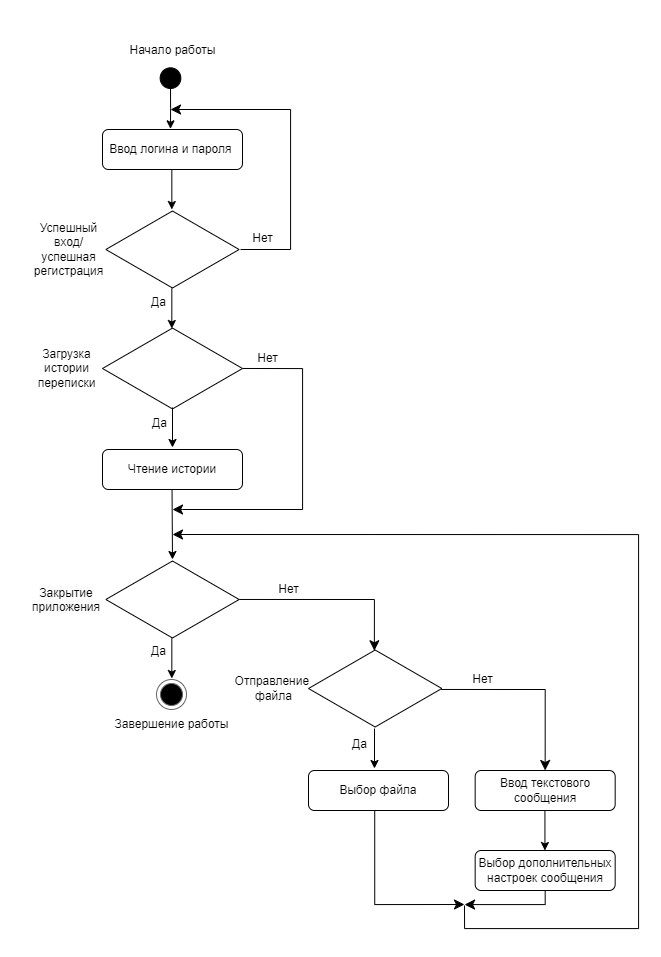
# ПРИЛОЖЕНИE A – UML-диаграмма последовательности



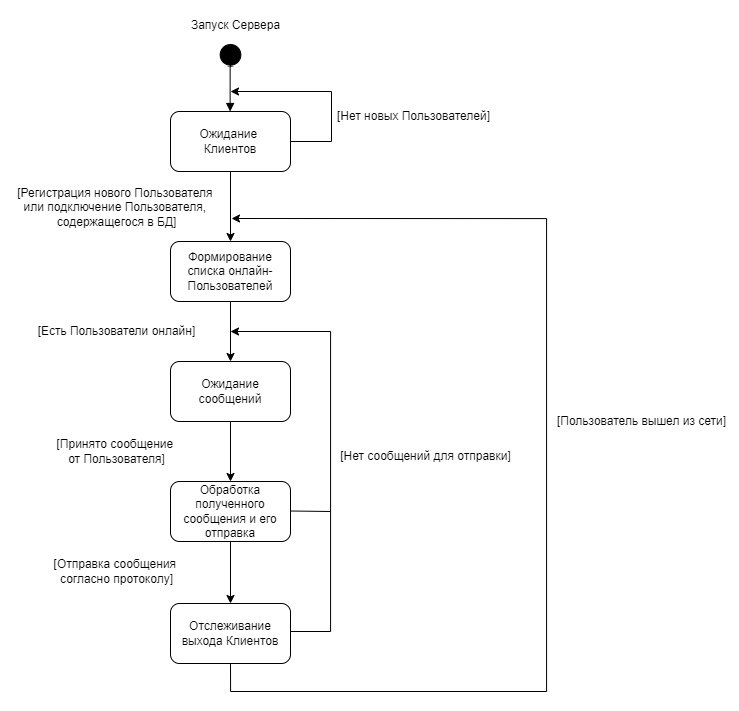
# ПРИЛОЖЕНИE В – UML-диаграмма деятельности Сервера



# ПРИЛОЖЕНИE С – UML-диаграмма деятельности Клиента



# ПРИЛОЖЕНИE D – UML-диаграмма состояний Сервера



# ПРИЛОЖЕНИE E – Дополнительные UML-диаграммы

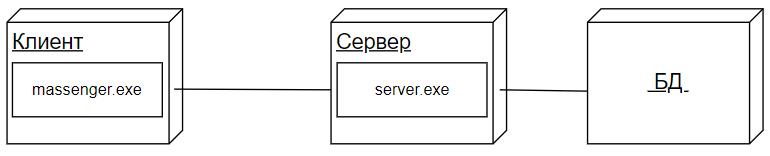


Рисунок 1 – UML-диаграмма развертывания



Рисунок 2– UML-диаграмма взаимодействия Пользователей

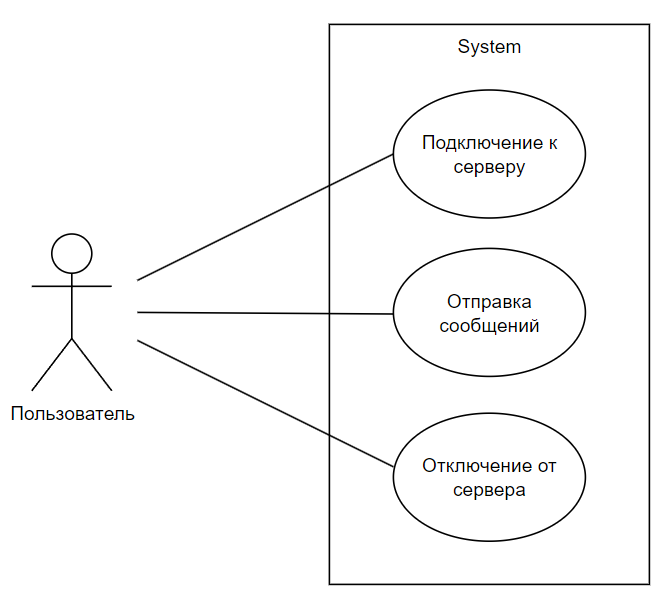


Рисунок 3– UML-диаграмма вариантов использования приложения

# ПРИЛОЖЕНИE H – Листинг Сервера

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <string.h>

#include <locale.h>

#include <time.h>

HANDLE hMutex;

HANDLE hMapFile;

HANDLE hEvent;

int CountUser = 0;

struct UserList

{

char Login[64];

char Password[64];

char NameOfMailslot[128];

struct UserList\* next;

};

struct UserList\* node,\* head\_copy,\* previous;

struct UserList\* head = NULL;

void PrintCurrentTime() {

time\_t now = time(NULL);

struct tm\* local = localtime(&now);

printf("%02d.%02d.%4d %02d:%02d:%02d | ",

local->tm\_mday,

local->tm\_mon + 1,

local->tm\_year + 1900,

local->tm\_hour,

local->tm\_min,

local->tm\_sec);

return;

}

void FuncForTimer() {

HANDLE hTimer = CreateWaitableTimer(NULL, FALSE, NULL);

if (hTimer == NULL) { return -503; }

LARGE\_INTEGER liDueTime;

liDueTime.QuadPart = -(10000000LL \* 60LL); //1 мин

if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0)) {

CloseHandle(hTimer);

return -504;

}

WaitForSingleObjectEx(hTimer, INFINITE, TRUE);

return;

}

DWORD WINAPI SendOnlineUserList(LPVOID lpParam) {

HANDLE hMailslot;

DWORD cbWritten;

char ListOnlineUser[1024];

char cUserName[128];

char cut[] = "\n";

while (1) {

WaitForSingleObject(hEvent, INFINITE); // Ожидание события

ListOnlineUser[0] = '\0';

WaitForSingleObject(hMutex, INFINITE); // Блокировка мьютекса

if (CountUser > 0) {

head\_copy = head;

while (head\_copy != NULL) {

strcpy(cUserName, head\_copy->Login);

strcat(ListOnlineUser, cUserName);

strcat(ListOnlineUser, cut);

head\_copy = head\_copy->next;

}

head\_copy = head;

while (head\_copy != NULL) {

char NameOfMailslot[128];

char postfix[7] = "Online";

strcpy(NameOfMailslot, head\_copy->NameOfMailslot);

strcat(NameOfMailslot, postfix);

do {

hMailslot = CreateFileA(NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

} while (hMailslot == INVALID\_HANDLE\_VALUE);

if (!WriteFile(hMailslot, ListOnlineUser, strlen(ListOnlineUser), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

CloseHandle(hMailslot);

head\_copy = head\_copy->next;

}

ReleaseMutex(hMutex);

ResetEvent(hEvent);

}

Sleep(200);

}

return 200;

}

int FuncMailStorage(const char\* UserLogin, char\* massage) {

char FileExpansion[] = ".txt";

char fullFileName[128];

char designText[2];

memmove(massage, massage + 1, strlen(massage)); //отсечение типа

strncpy(designText, massage, 1); //обработка стиля текста

designText[1] = '\0';

memmove(massage, massage + 1, strlen(massage));

snprintf(fullFileName, sizeof(fullFileName), "%s%s", UserLogin, FileExpansion);

FILE\* file = fopen(fullFileName, "a");

if (file == NULL) {

printf("Ошибка открытия файла для записи переписки");

return 508;

}

fprintf(file, "%s", designText);

time\_t now = time(NULL);

struct tm\* local = localtime(&now);

fprintf(file, "%02d.%02d.%4d %02d:%02d:%02d ",

local->tm\_mday,

local->tm\_mon + 1,

local->tm\_year + 1900,

local->tm\_hour,

local->tm\_min,

local->tm\_sec);

fprintf(file, "%s", massage);

fclose(file);

return 200;

}

DWORD WINAPI ThreadFuncForSending(LPVOID lpParam) {

HANDLE hMailslot;

DWORD cbWritten;

LPCSTR lpszMailslotName = NULL;

char\* buffer = (char\*)lpParam;

///

//выделениие имени получателя для открытия уникального канала

char\* copy\_ProtocolMessage = buffer;

char cRecepientNameSize[4];

char cUserNameSize[4];

char cMsgSizeCorrect[5];

char cMsgSize[5];

char cNumberCurrentMsg[5];

char cNumberTotalMsg[5];

int iRecepientNameSize;

int iUserNameSize;

int iMsgSize;

int iMsgSizeCorrect;

int iNumberCurrentMsg;

int iNumberTotalMsg;

int start\_positionFrom = 4;

int start\_positionTo = 7;

int start\_positionMsg = 10;

char RecepientName[64];

char UserName[64];

BOOL FileMsg = FALSE;

strncpy(cUserNameSize, copy\_ProtocolMessage + start\_positionFrom, 3); //выделение имени отправителя

cUserNameSize[3] = '\0';

iUserNameSize = atoi(cUserNameSize);

strncpy(UserName, copy\_ProtocolMessage + 22, iUserNameSize);

UserName[iUserNameSize] = '\0';

strncpy(cRecepientNameSize, copy\_ProtocolMessage + start\_positionTo, 3); //выделение имени получателя

cRecepientNameSize[3] = '\0';

iRecepientNameSize = atoi(cRecepientNameSize);

strncpy(RecepientName, copy\_ProtocolMessage + iUserNameSize + 22, iRecepientNameSize);

RecepientName[iRecepientNameSize] = '\0';

strncpy(cMsgSizeCorrect, copy\_ProtocolMessage + 10, 4); //размер сообщения

cMsgSizeCorrect[4] = '\0';

iMsgSizeCorrect = atoi(cMsgSizeCorrect);

strncpy(cNumberCurrentMsg, copy\_ProtocolMessage + 14, 4); //номер передаваемого сообщения (для файла)

cNumberCurrentMsg[4] = '\0';

iNumberCurrentMsg = atoi(cNumberCurrentMsg);

strncpy(cNumberTotalMsg, copy\_ProtocolMessage + 18, 4); //общее количество сообщений для файла

cNumberTotalMsg[4] = '\0';

iNumberTotalMsg = atoi(cNumberTotalMsg);

//выделение сообщения и стиля форматирования текста

int iMsgPosition = 22 + iUserNameSize + iRecepientNameSize;

iMsgSize = strlen(buffer) - iMsgPosition + 1;

if (((buffer[2] == '0') && (iMsgSize == iMsgSizeCorrect + 1)) || ((buffer[2] != '0') && (iMsgSize != iMsgSizeCorrect + 1))) {

char\* Msg = malloc(iMsgSize \* sizeof(char) + 1);

char\* Msg\_copy = malloc(iMsgSize \* sizeof(char) + 1);

char\* PartProtocolDesign = malloc(iMsgSize \* sizeof(char) + 1);

char\* PartProtocolType = malloc(iMsgSize \* sizeof(char) + 10);

char\* PartNumCurrentMsg = malloc(iMsgSize \* sizeof(char) + 1);

char\* PartNumTotalMsg = malloc(iMsgSize \* sizeof(char) + 1);

memset(Msg, 0, iMsgSize + 1);

memset(Msg\_copy, 0, iMsgSize + 1);

memset(PartProtocolDesign, 0, iMsgSize + 1);

memset(PartProtocolType, 0, iMsgSize + 1);

strncpy(PartProtocolType, copy\_ProtocolMessage + 2, 1);

PartProtocolType[1] = '\0';

if (strcmp(PartProtocolType, "0") == 0) { //для текстового сообщения

strncpy(PartProtocolDesign, copy\_ProtocolMessage + 3, 1);

PartProtocolDesign[1] = '\0';

strcat(PartProtocolType, PartProtocolDesign);

}

else { //для файла

strncpy(PartNumCurrentMsg, copy\_ProtocolMessage + 14, 4);

PartNumCurrentMsg[4] = '\0';

strncpy(PartNumTotalMsg, copy\_ProtocolMessage + 18, 4);

PartNumTotalMsg[4] = '\0';

strcat(PartProtocolType, PartNumCurrentMsg);

strcat(PartProtocolType, PartNumTotalMsg);

}

strncpy(Msg, copy\_ProtocolMessage + iMsgPosition, iMsgSize);

Msg[iMsgSize] = '\0';

strcat(PartProtocolType, Msg);

strcpy(Msg, PartProtocolType);

free(PartProtocolDesign);

free(PartProtocolType);

free(PartNumCurrentMsg);

free(PartNumTotalMsg);

if (buffer[1] == '1') FuncForTimer();

if (buffer[2] != '0') FileMsg = TRUE;

head\_copy = head;

if (buffer[0] == '0') {

while (head\_copy != NULL) {

//личное сообщение - текстовое

if ((strcmp(head\_copy->Login, RecepientName) == 0) || (strcmp(head\_copy->Login, UserName) == 0)) {

if (buffer[2] == '0') {

if ((FileMsg) && (strcmp(head\_copy->Login, UserName) == 0)) {

head\_copy = head\_copy->next;

continue;

}

hMailslot = CreateFileA(head\_copy->NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

if (!WriteFile(hMailslot, Msg, strlen(Msg), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -501;

}

CloseHandle(hMailslot);

//фукция записи в файл

strcpy(Msg\_copy, Msg);

FuncMailStorage(head\_copy->Login, Msg\_copy);

}

else {

if ((FileMsg) && (strcmp(head\_copy->Login, UserName) == 0)) {

head\_copy = head\_copy->next;

continue;

}

hMailslot = CreateFileA(head\_copy->NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

char type = buffer[2];

char fullFuleName[512] = "recepient.";

if (buffer[2] == '1') strcat(fullFuleName, "txt");

if (buffer[2] == '2') strcat(fullFuleName, "bmp");

if (buffer[2] == '3') strcat(fullFuleName, "jpg");

if (buffer[2] == '4') strcat(fullFuleName, "gif");

if (buffer[2] == '5') strcat(fullFuleName, "png");

FILE\* inFile;

char bufferFile[300];

size\_t bytesRead;

inFile = fopen(fullFuleName, "rb");

if (inFile == NULL) {

fprintf(stderr, "Ошибка при открытии файла: %s\n", fullFuleName); // ошибка - открытие файла

CloseHandle(hMailslot);

return 508;

}

fseek(inFile, 0, SEEK\_END);

int sizeFile = ftell(inFile);

fseek(inFile, 0, SEEK\_SET);

int iCurrentMsg = 1;

int SendPosition = 0;

while (iNumberTotalMsg > 0) {

int chunkSize = (SendPosition + 300 <= sizeFile) ? 300 : (sizeFile - SendPosition);

bytesRead = fread(bufferFile, 1, chunkSize, inFile);

char\* dataWithPrefix = malloc((315 + iUserNameSize) \* sizeof(char) + 1);

memcpy(dataWithPrefix, &type, 1);

char cCurrentMsg[5];

sprintf(cCurrentMsg, "%04d", iCurrentMsg);

memcpy(dataWithPrefix + 1, cCurrentMsg, 4);

memcpy(dataWithPrefix + 5, cNumberTotalMsg, 4);

memcpy(dataWithPrefix + 9, cUserNameSize, 3);

memcpy(dataWithPrefix + 12, UserName, iUserNameSize);

memcpy(dataWithPrefix + 12 + iUserNameSize, bufferFile, 300);

if (!WriteFile(hMailslot, dataWithPrefix, chunkSize + 12 + iUserNameSize, &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

iNumberTotalMsg--;

iCurrentMsg++;

SendPosition += 300;

}

fclose(inFile);

CloseHandle(hMailslot);

}

}

head\_copy = head\_copy->next;

}

}

else {

while (head\_copy != NULL) { //широковещ сообщение - текстовое

if ((FileMsg) && (strcmp(head\_copy->Login, UserName) == 0)) {

head\_copy = head\_copy->next;

continue;

}

hMailslot = CreateFileA(head\_copy->NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

if (!WriteFile(hMailslot, Msg, strlen(Msg), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -501;

}

CloseHandle(hMailslot);

//фукция записи в файл

if (buffer[2] == '0') {

strcpy(Msg\_copy, Msg);

FuncMailStorage(head\_copy->Login, Msg\_copy);

}

head\_copy = head\_copy->next;

}

}

PrintCurrentTime();

printf("Сообщение успешно отправлено.\n\n");

return 200;

}

else {

char Msg[] = "Некорректная отправка сообщения.\n";

printf("%s", Msg);

while (head\_copy != NULL) {

if (strcmp(head\_copy->Login, UserName) == 0) {

hMailslot = CreateFileA(head\_copy->NameOfMailslot, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

if (!WriteFile(hMailslot, Msg, strlen(Msg), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

return 505;

}

head\_copy = head\_copy->next;

}

}

}

DWORD WINAPI ClientDisconnection(LPVOID lpParam) {

HANDLE hMailslotDisconnection;

BOOL fReturnCode;

DWORD cbMessages;

DWORD cbMsgNumber;

char szBuf[512];

DWORD cbRead;

hMailslotDisconnection = CreateMailslot(TEXT("\\\\.\\mailslot\\ClientDisconnectionChannel"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslotDisconnection == INVALID\_HANDLE\_VALUE) return -501;

while (1) {

fReturnCode = GetMailslotInfo(hMailslotDisconnection, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslotDisconnection, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

//удаление пользователя

head\_copy = head;

previous = NULL;

while (head\_copy != NULL) {

if (strcmp(head\_copy->Login, szBuf) == 0) {

if (previous == NULL) {

head = head\_copy->next;

}

else {

previous->next = head\_copy->next;

}

free(head\_copy);

break;

}

previous = head\_copy;

head\_copy = head\_copy->next;

}

SetEvent(hEvent);

printf("Пользователь %s вышел из сети.\n", szBuf);

}

}

Sleep(20);

}

}

DWORD WINAPI UserEnter(LPVOID lpParam) {

HANDLE hMailslotRegistration;

HANDLE hMailslotUnicNameUser;

BOOL fReturnCode;

DWORD cbMsgRegistration;

DWORD cbMsgRegistrationNumber;

char szBuf[512];

DWORD cbRead;

DWORD cbWritten;

char LoginError[] = "406\_Not\_Acceptable\_Login";

char PasswordError[] = "406\_Not\_Acceptable\_Password";

BOOL FlagRepeatUser = FALSE;

char tmpLogin[64];

char tmpPassword[64];

char LoginFromBase[64];

char PasswordFromBase[64];

char UnicNameChannel[128];

char\* DataPart;

char\* szBufCopy[512];

hMailslotRegistration = CreateMailslot(TEXT("\\\\.\\mailslot\\EnterChannel"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslotRegistration == INVALID\_HANDLE\_VALUE) return -501;

while (1) {

fReturnCode = GetMailslotInfo(hMailslotRegistration, NULL, &cbMsgRegistration, &cbMsgRegistrationNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMsgRegistration != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslotRegistration, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

DataPart = strtok(szBuf, " ");

strcpy(tmpLogin, DataPart);

DataPart = strtok(NULL, " ");

strcpy(tmpPassword, DataPart);

char line[128];

FILE\* file = fopen("registration.txt", "r");

if (file == NULL) {

printf("Ошибка открытия файла для чтения");

return 508;

}

FlagRepeatUser = FALSE; //проверка на наличие указанного пользователя

while (fgets(line, sizeof(line), file) != NULL) {

DataPart = strtok(line, " ");

strcpy(LoginFromBase, DataPart);

DataPart = strtok(NULL, " ");

strcpy(PasswordFromBase, DataPart);

int len = strlen(PasswordFromBase);

PasswordFromBase[len-1] = '\0';

/\*char\* space\_ptr = strchr(line, ' ');

if (space\_ptr != NULL) {

\*space\_ptr = '\0';

}\*/

if (strcmp(LoginFromBase, tmpLogin) == 0) {

if (strcmp(PasswordFromBase, tmpPassword) == 0) {

WaitForSingleObject(hMutex, INFINITE); // ожидание мьютекса

node = (struct UserList\*)malloc(sizeof(struct UserList));

node->next = NULL;

strcpy(node->Login, tmpLogin);

strcpy(node->Password, tmpPassword);

strcpy(UnicNameChannel, "\\\\.\\mailslot\\");

strcat(UnicNameChannel, node->Login);

strcpy(node->NameOfMailslot, UnicNameChannel);

if (head == NULL) {

head = node;

}

else {

head\_copy = head;

while (head\_copy->next != NULL) {

head\_copy = head\_copy->next;

}

head\_copy->next = node;

}

ReleaseMutex(hMutex);

CountUser++;

SetEvent(hEvent);

printf("Пользователь %s появился в сети.\n", szBuf);

break;

}

else {

strcpy(UnicNameChannel, PasswordError);

break;

}

}

else {

strcpy(UnicNameChannel, LoginError);

}

}

fclose(file);

do {

hMailslotUnicNameUser = CreateFile(TEXT("\\\\.\\mailslot\\GetUnicNameFromServerChannel"), GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

} while (hMailslotUnicNameUser == INVALID\_HANDLE\_VALUE);

if (!WriteFile(hMailslotUnicNameUser, UnicNameChannel, strlen(UnicNameChannel), &cbWritten, NULL)) {

CloseHandle(hMailslotUnicNameUser);

fprintf(stdout, "WriteFile: Error %ld\n", GetLastError());

return -502;

}

CloseHandle(hMailslotUnicNameUser);

}

}

Sleep(1);

}

}

DWORD WINAPI UserRegistration(LPVOID lpParam) {

HANDLE hMailslotRegistration;

HANDLE hMailslotUnicNameUser;

BOOL fReturnCode;

DWORD cbMsgRegistration;

DWORD cbMsgRegistrationNumber;

char szBuf[512];

DWORD cbRead;

DWORD cbWritten;

char LoginError[] = "406\_Not\_Acceptable\_Login";

BOOL FlagRepeatUser = FALSE;

char tmpLogin[64];

char tmpPassword[64];

char UnicNameChannel[128];

char\* DataPart;

char\* szBufCopy[512];

hMailslotRegistration = CreateMailslot(TEXT("\\\\.\\mailslot\\RegistrationOnServerChannel"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslotRegistration == INVALID\_HANDLE\_VALUE) return -501;

while (1) {

fReturnCode = GetMailslotInfo(hMailslotRegistration, NULL, &cbMsgRegistration, &cbMsgRegistrationNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMsgRegistration != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslotRegistration, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

DataPart = strtok(szBuf, " ");

strcpy(tmpLogin, DataPart);

DataPart = strtok(NULL, " ");

strcpy(tmpPassword, DataPart);

char line[128];

FILE\* file = fopen("registration.txt", "r");

if (file == NULL) {

printf("Ошибка открытия файла для чтения");

return 508;

}

FlagRepeatUser = FALSE; //проверка на наличие такого же пользователя

while (fgets(line, sizeof(line), file) != NULL) {

char\* space\_ptr = strchr(line, ' ');

if (space\_ptr != NULL) {

\*space\_ptr = '\0';

}

if (strcmp(line, tmpLogin) == 0) {

strcpy(UnicNameChannel, LoginError);

FlagRepeatUser = TRUE;

break;

}

}

fclose(file);

if(!FlagRepeatUser) {

FILE\* fileAdd = fopen("registration.txt", "a");

fprintf(fileAdd, "%s ", tmpLogin);

fprintf(fileAdd, "%s\n", tmpPassword);

fclose(file);

}

if (!FlagRepeatUser) {

WaitForSingleObject(hMutex, INFINITE); // ожидание мьютекса

node = (struct UserList\*)malloc(sizeof(struct UserList));

node->next = NULL;

strcpy(node->Login, tmpLogin);

strcpy(node->Password, tmpPassword);

strcpy(UnicNameChannel, "\\\\.\\mailslot\\");

strcat(UnicNameChannel, node->Login);

strcpy(node->NameOfMailslot, UnicNameChannel);

if (head == NULL) {

head = node;

}

else {

head\_copy = head;

while (head\_copy->next != NULL) {

head\_copy = head\_copy->next;

}

head\_copy->next = node;

}

ReleaseMutex(hMutex);

CountUser++;

SetEvent(hEvent);

printf("Зарегистрирован новый пользователь %s.\n", szBuf);

}

do {

hMailslotUnicNameUser = CreateFile(TEXT("\\\\.\\mailslot\\GetUnicNameFromServerChannel"), GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

} while (hMailslotUnicNameUser == INVALID\_HANDLE\_VALUE);

if (!WriteFile(hMailslotUnicNameUser, UnicNameChannel, strlen(UnicNameChannel), &cbWritten, NULL)) {

CloseHandle(hMailslotUnicNameUser);

fprintf(stdout, "WriteFile: Error %ld\n", GetLastError());

return -502;

}

CloseHandle(hMailslotUnicNameUser);

}

}

Sleep(1);

}

}

int main() {

setlocale(LC\_ALL, "rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

hMutex = CreateMutex(NULL, FALSE, NULL);

if (hMutex == NULL) {

fprintf(stderr, "CreateMutex (%ld)\n", GetLastError());

return 506;

}

hEvent = CreateEvent(NULL, FALSE, FALSE, "UserRegistrationEvent");

if (hEvent == NULL) {

fprintf(stderr, "CreateEvent (%ld)\n", GetLastError());

return 507;

}

HANDLE hMailslot;

BOOL fReturnCode;

DWORD cbMessages;

DWORD cbMsgNumber;

char szBuf[512];

DWORD cbRead;

HANDLE hThreads;

HANDLE hThreadsRegistration;

hThreadsRegistration = CreateThread(NULL, 0, UserRegistration, NULL, 0, NULL);

HANDLE hThreadsEnter;

hThreadsEnter = CreateThread(NULL, 0, UserEnter, NULL, 0, NULL);

HANDLE hThreadsDisconnection;

hThreadsDisconnection = CreateThread(NULL, 0, ClientDisconnection, NULL, 0, NULL);

HANDLE hThreadsOnlineUserList;

hThreadsDisconnection = CreateThread(NULL, 0, SendOnlineUserList, NULL, 0, NULL);

hMailslot = CreateMailslot(TEXT("\\\\.\\mailslot\\ChannelFromServer"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return -501;

char buffer[300];

while (1) {

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslot, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

PrintCurrentTime();

printf("Получено новое сообщение: %s", szBuf);

if (szBuf[2] == '0') {

hThreads = CreateThread(NULL, 0, ThreadFuncForSending, (LPVOID)szBuf, 0, NULL);

WaitForSingleObject(hThreads, INFINITE);

}

else {

char type = szBuf[2];

char fullFuleName[512] = "recepient.";

if (type == '1') strcat(fullFuleName, "txt");

if (type == '2') strcat(fullFuleName, "bmp");

if (type == '3') strcat(fullFuleName, "jpg");

if (type == '4') strcat(fullFuleName, "gif");

if (type == '5') strcat(fullFuleName, "png");

size\_t totalSize = cbRead;

char\* dataForFile = malloc(cbRead \* sizeof(char) + 1);

char\* prefix = malloc(26 \* sizeof(char) + 1);

memcpy(prefix, szBuf, 26);

memcpy(szBuf, szBuf+26, cbRead);

FILE\* outFile = fopen(fullFuleName, "ab");

if (outFile == NULL) {

fprintf(stderr, "Error opening file for writing\n");

}

size\_t written = fwrite(szBuf, 1, cbRead-26, outFile);

if (written != cbRead-26) {

fprintf(stderr, "Error writing data to file\n");

fclose(outFile);

}

fclose(outFile);

printf("Received data: %lu bytes\n", cbRead-26);

char\* PartNumCurrentMsg = malloc(4 \* sizeof(char) + 1);

char\* PartNumTotalMsg = malloc(4 \* sizeof(char) + 1);

strncpy(PartNumCurrentMsg, prefix + 14, 4);

PartNumCurrentMsg[4] = '\0';

strncpy(PartNumTotalMsg, prefix + 18, 4);

PartNumTotalMsg[4] = '\0';

if (strcmp(PartNumCurrentMsg, PartNumTotalMsg) == 0) {

hThreads = CreateThread(NULL, 0, ThreadFuncForSending, (LPVOID)prefix, 0, NULL);

}

}

}

}

Sleep(1);

}

CloseHandle(hMailslot);

CloseHandle(hMutex);

return 0;

}

# ПРИЛОЖЕНИE I – Листинг Клиента Windows Forms

using Microsoft.VisualBasic;

using Microsoft.VisualBasic.ApplicationServices;

using System;

using System.Collections;

using System.Reflection.Metadata;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Runtime.InteropServices.JavaScript.JSType;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

using Timer = System.Windows.Forms.Timer;

namespace messenger

{

public partial class Form1 : Form

{

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

internal static extern int myfuncSendMessage(string ProtocolMassage);

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

internal static extern int myfuncSendFile(string filePath, string ProtocolMessage, int SendPosition, int chunkSize);

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

internal static extern int NotifyServerAboutDisconnection(string UserDate);

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

internal static extern int UserRegistration(string UserDate);

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

internal static extern int UserEnter(string UserDate);

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

[return: MarshalAs(UnmanagedType.LPStr)]

internal static extern string myfuncReceiptMessage();

private CancellationTokenSource \_cancellationTokenSource;

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

[return: MarshalAs(UnmanagedType.LPStr)]

internal static extern string myfuncOnlineUserList();

private CancellationTokenSource \_cancelTokenSource;

[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

[return: MarshalAs(UnmanagedType.LPStr)]

internal static extern string ReadingOldMessage(string UserLogin);

//[DllImport("functions.dll", CallingConvention = CallingConvention.StdCall)]

//[return: MarshalAs(UnmanagedType.LPStr)]

//internal static extern string myfuncSendFile(string filePath, int SendPosition);

public Form1()

{

InitializeComponent();

this.Load += Form2\_Load;

this.FormClosing += MainForm\_FormClosing;

}

public class ProtocolManager //генерация протокола

{

public static string myProtocol = string.Empty;

public static void EmptyProtocol()

{

myProtocol = myProtocol.Remove(0);

}

public static void ProtocolGeneration(string PartOfProtocol)

{

myProtocol = PartOfProtocol + myProtocol;

}

public static string GetProtocol()

{

return myProtocol;

}

}

private void Form1\_Load(object sender, EventArgs e) //фоновая нить - чтение сообщений с сервера

{

\_cancellationTokenSource = new CancellationTokenSource();

}

private async Task StartReceivingMessagesAsync(CancellationToken token)

{

string oldMessage = "";

try

{

while (!token.IsCancellationRequested)

{

string tmpMessage = await Task1Async();

if (!Equals(oldMessage, tmpMessage))

{

UpdateRichTextBox(tmpMessage);

oldMessage = tmpMessage;

}

await Task.Delay(1000);

}

}

catch (TaskCanceledException)

{

}

}

private async Task<string> Task1Async()

{

return await Task.Run(() => myfuncReceiptMessage());

}

private char DesignPartOfProtocol()

{

if (checkBox1\_BoldText.Checked && checkBox1\_ItalicText.Checked && checkBox1\_UnderlineText.Checked) return '7';

if (checkBox1\_BoldText.Checked && checkBox1\_ItalicText.Checked) return '4';

if (checkBox1\_BoldText.Checked && checkBox1\_UnderlineText.Checked) return '5';

if (checkBox1\_ItalicText.Checked && checkBox1\_UnderlineText.Checked) return '6';

if (checkBox1\_BoldText.Checked) return '1';

if (checkBox1\_ItalicText.Checked) return '2';

if (checkBox1\_UnderlineText.Checked) return '3';

return '0';

}

private void MessageFormatting(string message)

{

if (string.IsNullOrEmpty(message))

{

return; // Прерываем выполнение метода, если строка пустая или null

}

FontStyle fontStyle = FontStyle.Regular;

if (message[0] == '1') fontStyle = FontStyle.Bold;

if (message[0] == '2') fontStyle = FontStyle.Italic;

if (message[0] == '3') fontStyle = FontStyle.Underline;

if (message[0] == '4') fontStyle = FontStyle.Bold | FontStyle.Italic;

if (message[0] == '5') fontStyle = FontStyle.Bold | FontStyle.Underline;

if (message[0] == '6') fontStyle = FontStyle.Italic | FontStyle.Underline;

if (message[0] == '7') fontStyle = FontStyle.Bold | FontStyle.Italic | FontStyle.Underline;

if (richTextBox\_forOutputMessage.InvokeRequired)

{

richTextBox\_forOutputMessage.Invoke(new Action(() => MessageFormatting(message)));

}

else

{

if (richTextBox\_forOutputMessage == null)

{

return;

}

Font currentFont = richTextBox\_forOutputMessage.Font;

if (currentFont == null)

{

return;

}

richTextBox\_forOutputMessage.SelectionFont = new Font(currentFont, fontStyle);

}

}

private void UpdateRichTextBox(string message)

{

if (richTextBox\_forOutputMessage.InvokeRequired)

{

richTextBox\_forOutputMessage.Invoke(new Action(() => UpdateRichTextBox(message)));

}

else

{

if (message.Length > 1)

{

// сохранение текущуей позиции

int previousSelectionStart = richTextBox\_forOutputMessage.SelectionStart;

// переход в конец текста

richTextBox\_forOutputMessage.SelectionStart = richTextBox\_forOutputMessage.TextLength;

richTextBox\_forOutputMessage.SelectionLength = 0; // ничего не выделяем

// обновление стиля перед добавлением текста

MessageFormatting(message);

message = message.Substring(1);

// запись текста с учетом установленного стиля

DateTime thisDay = DateTime.Now;

string time = thisDay.ToString("yyyy.MM.dd HH:mm");

message = time + " " + message;

richTextBox\_forOutputMessage.SelectedText = message;

// восстановление предыдущего положения курсора

richTextBox\_forOutputMessage.SelectionStart = previousSelectionStart;

richTextBox\_forOutputMessage.ScrollToCaret();

}

}

}

//отмена задачи при закрытии формы

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

\_cancellationTokenSource.Cancel();

}

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e) //отключение пользователя от сервера

{

string UserName;

UserName = textBox\_forUserName.Text;

NotifyServerAboutDisconnection(UserName);

}

private async void Form2\_Load(object sender, EventArgs e) //фоновая нить - вывод онлайн участников

{

\_cancelTokenSource = new CancellationTokenSource();

await UpdatingOnlinUserList(\_cancelTokenSource.Token);

}

private async Task UpdatingOnlinUserList(CancellationToken token)

{

string oldMessage = "";

try

{

while (!token.IsCancellationRequested)

{

string tmpMessage = await Task2Async();

UpdateRichTextBoxOnlineUser(tmpMessage);

await Task.Delay(1000);

}

}

catch (TaskCanceledException)

{

}

}

private async Task<string> Task2Async()

{

return await Task.Run(() => myfuncOnlineUserList());

}

private void UpdateRichTextBoxOnlineUser(string message)

{

if (richTextBox1.InvokeRequired)

{

richTextBox1.Invoke(new Action(() => UpdateRichTextBoxOnlineUser(message)));

}

else

{

richTextBox1.Text = message;

richTextBox1.ScrollToCaret();

}

}

//отмена задачи при закрытии формы

private void Form2\_FormClosing(object sender, FormClosingEventArgs e)

{

\_cancelTokenSource.Cancel();

}

private bool InputUser()

{

if (textBox\_forUserName.Text.Length == 0) //проверка на наличие имени пользователя

{

Timer MyTimer = new System.Windows.Forms.Timer();

textBox\_forUserName.BackColor = Color.Red;

MessageBox.Show("Введите имя пользователя!");

textBox\_forUserName.BackColor = Color.White;

return false;

}

else

{

if (textBox\_forUserPassword.Text.Length == 0) //проверка на наличие пароля пользователя

{

Timer MyTimer = new System.Windows.Forms.Timer();

textBox\_forUserPassword.BackColor = Color.Red;

MessageBox.Show("Введите пароль!");

textBox\_forUserPassword.BackColor = Color.White;

return false;

}

}

return true;

}

private bool InputRecepient()

{

if (textBox\_DateOfRecipient.Text.Length == 0 && !checkBox1\_BroadcastMsg.Checked) //проверка на наличие имени получателя

{

Timer MyTimer = new System.Windows.Forms.Timer();

textBox\_DateOfRecipient.BackColor = Color.Red;

MessageBox.Show("Введите имя получателя!");

textBox\_DateOfRecipient.BackColor = Color.White;

return false;

}

return true;

}

private async void button2\_Click(object sender, EventArgs e) //регистрация пользователя на сервере

{

if (InputUser())

{

string UserDate;

UserDate = textBox\_forUserName.Text + " " + textBox\_forUserPassword.Text;

int RezultFuncOfRegistration = UserRegistration(UserDate);

await StartReceivingMessagesAsync(\_cancellationTokenSource.Token); //запуск канала чтения

}

}

private async void button5\_Click(object sender, EventArgs e) //вход уже зарегистрированного пользователя

{

if (InputUser())

{

string UserDate;

UserDate = textBox\_forUserName.Text + " " + textBox\_forUserPassword.Text;

int RezultFuncOfRegistration = UserEnter(UserDate);

await StartReceivingMessagesAsync(\_cancellationTokenSource.Token); //запуск канала чтения

}

}

private void button1\_Click(object sender, EventArgs e) //отправка сообщения на сервер

{

if (InputUser() && (InputRecepient() || checkBox1\_BroadcastMsg.Checked))

{

string sOutputMessageNew;

string sOutputMessageOld;

string DateOfRecepient = textBox\_DateOfRecipient.Text;

char DesignText = DesignPartOfProtocol();

sOutputMessageOld = richTextBox\_forOutputMessage.Text;

sOutputMessageNew = textBox\_forUserName.Text + ": " + richTextBox\_forInputMessage.Text + "\r\n";

ProtocolManager.ProtocolGeneration(sOutputMessageNew);

ProtocolManager.ProtocolGeneration(DateOfRecepient);

ProtocolManager.ProtocolGeneration(textBox\_forUserName.Text);

ProtocolManager.ProtocolGeneration("0000"); //кол-во необходимых отправок для передачи целого файла

ProtocolManager.ProtocolGeneration("0000"); //номер текущего сообщения

ProtocolManager.ProtocolGeneration(sOutputMessageNew.Length.ToString().PadLeft(4, '0')); //длина msg

ProtocolManager.ProtocolGeneration(DateOfRecepient.Length.ToString().PadLeft(3, '0')); //длина to;

ProtocolManager.ProtocolGeneration(textBox\_forUserName.Text.Length.ToString().PadLeft(3, '0')); //длина from

ProtocolManager.ProtocolGeneration(DesignText.ToString()); //design

ProtocolManager.ProtocolGeneration("0"); //type

ProtocolManager.ProtocolGeneration(checkBox1\_WaitMessage.Checked ? "1" : "0"); //отложенное сообщение

ProtocolManager.ProtocolGeneration(checkBox1\_BroadcastMsg.Checked ? "1" : "0"); //широковещательный режим рассылки

string RezultProtocol = ProtocolManager.GetProtocol();

int ResultFuncOfSend = myfuncSendMessage(RezultProtocol);

//обновление поля для следующего сообщения

if (ResultFuncOfSend == 200)

{

ProtocolManager.EmptyProtocol();

string content = richTextBox\_forInputMessage.Text;

richTextBox\_forInputMessage.Text = "";

}

}

}

private void button4\_Click(object sender, EventArgs e)

{

if (InputUser() && (InputRecepient() || checkBox1\_BroadcastMsg.Checked))

{

string DateOfRecepient = textBox\_DateOfRecipient.Text;

char DesignText = DesignPartOfProtocol();

string filePath = string.Empty;

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = "c:\\";

openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

filePath = openFileDialog.FileName;

}

}

if (!string.IsNullOrEmpty(filePath))

{

int SendPosition = 0;

System.IO.FileInfo file = new System.IO.FileInfo(filePath);

double SizeOfFile = file.Length;

string extension = Path.GetExtension(filePath);

char type = '0';

if (extension == ".txt") type = '1';

if (extension == ".bmp") type = '2';

if (extension == ".jpg") type = '3';

if (extension == ".gif") type = '4';

if (extension == ".png") type = '5';

using (FileStream inputStream = new FileStream(filePath, FileMode.Open, FileAccess.Read, FileShare.Read))

{

byte[] byteBuffer = new byte[inputStream.Length];

inputStream.Read(byteBuffer, 0, byteBuffer.Length);

List<byte> listByteBuffer = new List<byte>(byteBuffer);

double sizerezult = listByteBuffer.Count;

double NumberOfOperationForSend = Math.Ceiling(sizerezult / 300);

double NumberOfOperationForSend\_copy = NumberOfOperationForSend;

int NumberCurrentMsg = 0;

while (NumberOfOperationForSend > 0)

{

NumberCurrentMsg++;

int chunkSize = (SendPosition + 300 <= (int)sizerezult) ? 300 : ((int)sizerezult - SendPosition);

int endIndex = SendPosition + chunkSize - 1;

ProtocolManager.ProtocolGeneration(DateOfRecepient);

ProtocolManager.ProtocolGeneration(textBox\_forUserName.Text);

ProtocolManager.ProtocolGeneration(NumberOfOperationForSend\_copy.ToString().PadLeft(4, '0')); //кол-во необходимых отправок для передачи целого файла

ProtocolManager.ProtocolGeneration(NumberCurrentMsg.ToString().PadLeft(4, '0')); //номер текущего сообщения

ProtocolManager.ProtocolGeneration(chunkSize.ToString().PadLeft(4, '0')); //длина msg

ProtocolManager.ProtocolGeneration(DateOfRecepient.Length.ToString().PadLeft(3, '0')); //длина to;

ProtocolManager.ProtocolGeneration(textBox\_forUserName.Text.Length.ToString().PadLeft(3, '0')); //длина from

ProtocolManager.ProtocolGeneration(DesignText.ToString());

ProtocolManager.ProtocolGeneration(type.ToString()); //type

ProtocolManager.ProtocolGeneration(checkBox1\_WaitMessage.Checked ? "1" : "0"); //отложенное сообщение

ProtocolManager.ProtocolGeneration(checkBox1\_BroadcastMsg.Checked ? "1" : "0"); //широковещательный режим рассылки

string RezultProtocol = ProtocolManager.GetProtocol();

int ResultFuncOfSendFile = myfuncSendFile(filePath, RezultProtocol, SendPosition, chunkSize);

if (ResultFuncOfSendFile != 200) MessageBox.Show("Ошибка при отправке файла");

SendPosition += chunkSize;

NumberOfOperationForSend--;

ProtocolManager.EmptyProtocol();

}

}

}

}

}

private void button3\_Click(object sender, EventArgs e) //загрузка прошлых сообщений

{

string UserLogin = textBox\_forUserName.Text;

string ListOldMessages = ReadingOldMessage(UserLogin);

string[] ListSplitMessage = ListOldMessages.Split(new[] { '\n' }, StringSplitOptions.RemoveEmptyEntries);

string messageOutput;

foreach (var message in ListSplitMessage)

{

if (string.IsNullOrWhiteSpace(message)) continue; // Проверка на пустые сообщения

// сохранение текущей позиции

int previousSelectionStart = richTextBox\_forOutputMessage.SelectionStart;

// переход в конец текста

richTextBox\_forOutputMessage.SelectionStart = richTextBox\_forOutputMessage.TextLength;

richTextBox\_forOutputMessage.SelectionLength = 0; // Ничего не выделяем

// обновление стиля перед добавлением текста

MessageFormatting(message);

messageOutput = message.Substring(1);

richTextBox\_forOutputMessage.SelectedText = messageOutput;

// восстановление предыдущего положения курсора

richTextBox\_forOutputMessage.SelectionStart = previousSelectionStart;

richTextBox\_forOutputMessage.ScrollToCaret();

}

}

private void checkBox1\_CheckedChanged\_1(object sender, EventArgs e)

{

}

private void checkBox1\_ItalicText\_CheckedChanged(object sender, EventArgs e)

{

}

private void checkBox1\_UnderlineText\_CheckedChanged(object sender, EventArgs e)

{

}

private void richTextBox1\_TextChanged(object sender, EventArgs e)

{

}

private void richTextBox2\_TextChanged(object sender, EventArgs e)

{

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void label1\_Click\_1(object sender, EventArgs e)

{

}

private void label2\_Click(object sender, EventArgs e)

{

}

private void textBox1\_TextChanged\_1(object sender, EventArgs e)

{

}

private void textBox\_DateOfRecipient\_TextChanged(object sender, EventArgs e)

{

}

private void label4\_Click(object sender, EventArgs e)

{

}

private void checkBox1\_BroadcastMsg\_CheckedChanged(object sender, EventArgs e)

{

}

private void richTextBox1\_TextChanged\_1(object sender, EventArgs e)

{

}

}

}

# ПРИЛОЖЕНИE J – Листинг Клиента DLL библиотека

#include "header.h"

#include <iostream>

#include <fstream>

#include <cstring>

#include <cstddef>

#include <string>

#include <sstream>

#include <string>

#include <iostream>

#include <fstream>

#include <vector>

#include <iterator>

HANDLE globalhMailslot;

HANDLE globalhMailslotOnline;

extern "C" \_\_declspec(dllexport) char\* \_\_stdcall ReadingOldMessage(char\* UserLogin) {

char FileExpansion[] = ".txt";

char fullFileName[128];

char FilePath[256] = "d:\\ПГУ. Учебные пособия\\5 семестр\\основы ОС\\курсовая\\messenger\_server\\messenger\_server\\";

char fullBuffer[2024] = "";

snprintf(fullFileName, sizeof(fullFileName), "%s%s", UserLogin, FileExpansion);

strcat(FilePath, fullFileName);

FILE\* file = fopen(FilePath, "r");

if (file == NULL) {

MessageBox(0, "Ошибка открытия файла для чтения", "Ошибка", MB\_OK);

}

char Buffer[512];

while (fgets(Buffer, sizeof(Buffer), file) != NULL) {

strcat(fullBuffer, Buffer);

}

fclose(file);

size\_t stSize = strlen(fullBuffer) + sizeof(char);

char\* pszReturn = NULL;

pszReturn = (char\*)::CoTaskMemAlloc(stSize);

strcpy\_s(pszReturn, stSize, fullBuffer);

return pszReturn;

}

extern "C" \_\_declspec(dllexport) int \_\_stdcall UserEnter(char\* UserDate) {

HANDLE hMailslot;

DWORD cbWritten;

LPCSTR lpMailslotUnicNameUser;

char\* copy\_UserData = UserDate;

char prefixMailslotName[128] = "\\\\.\\mailslot\\";

hMailslot = CreateFile(TEXT("\\\\.\\mailslot\\EnterChannel"), GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return 402;

if (!WriteFile(hMailslot, UserDate, strlen(UserDate), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return 403;

}

HANDLE hMailslotUnicName;

BOOL fReturnCode;

DWORD cbMessages = 0;

DWORD cbMsgNumber = 0;

char szBuf[512];

DWORD cbRead;

hMailslotUnicName = CreateMailslot(TEXT("\\\\.\\mailslot\\GetUnicNameFromServerChannel"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslotUnicName == INVALID\_HANDLE\_VALUE) return 401;

while (1) {

fReturnCode = GetMailslotInfo(hMailslotUnicName, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslotUnicName, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

break;

}

}

Sleep(1);

}

CloseHandle(hMailslotUnicName);

char LoginError[] = "406\_Not\_Acceptable\_Login";

char PasswordError[] = "406\_Not\_Acceptable\_Password";

if (strcmp(szBuf, LoginError) == 0) {

MessageBox(0, "Пользователь с указанным логином не найден", "Уведомление", MB\_OK);

return 406;

}

if (strcmp(szBuf, PasswordError) == 0) {

MessageBox(0, "Неверный пароль", "Уведомление", MB\_OK);

return 406;

}

lpMailslotUnicNameUser = szBuf;

globalhMailslot = CreateMailslot(lpMailslotUnicNameUser, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) MessageBox(0, "Ошибка при регистрации", "Уведомление", MB\_OK);

else MessageBox(0, "Успешеный вход в систему", "Уведомление", MB\_OK);

char postfix[7] = "Online";

strcat(szBuf, postfix);

LPCSTR lpMailslotUnicNameUserOnline = szBuf;

globalhMailslotOnline = CreateMailslot(lpMailslotUnicNameUserOnline, 0, MAILSLOT\_WAIT\_FOREVER, NULL);;

if (globalhMailslotOnline == INVALID\_HANDLE\_VALUE) MessageBox(0, "44Ошибка при регистрации", "Уведомление", MB\_OK);

return 200;

}

extern "C" \_\_declspec(dllexport) int \_\_stdcall UserRegistration(char\* UserDate) {

HANDLE hMailslot;

DWORD cbWritten;

LPCSTR lpMailslotUnicNameUser;

char\* copy\_UserData = UserDate;

char prefixMailslotName[128] = "\\\\.\\mailslot\\";

hMailslot = CreateFile(TEXT("\\\\.\\mailslot\\RegistrationOnServerChannel"), GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return 402;

if (!WriteFile(hMailslot, UserDate, strlen(UserDate), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return 403;

}

HANDLE hMailslotUnicName;

BOOL fReturnCode;

DWORD cbMessages =0;

DWORD cbMsgNumber =0;

char szBuf[512];

DWORD cbRead;

hMailslotUnicName = CreateMailslot(TEXT("\\\\.\\mailslot\\GetUnicNameFromServerChannel"), 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslotUnicName == INVALID\_HANDLE\_VALUE) return 401;

while (1) {

fReturnCode = GetMailslotInfo(hMailslotUnicName, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslotUnicName, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

break;

}

}

Sleep(1);

}

CloseHandle(hMailslotUnicName);

char LoginError[] = "406\_Not\_Acceptable\_Login";

if (strcmp(szBuf, LoginError) == 0) {

MessageBox(0, "Измените имя пользователя. \nПользователь с таким именем уже существует.", "Уведомление", MB\_OK);

return 406;

}

lpMailslotUnicNameUser = szBuf;

globalhMailslot = CreateMailslot(lpMailslotUnicNameUser, 0, MAILSLOT\_WAIT\_FOREVER, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) MessageBox(0, "Ошибка при регистрации", "Уведомление", MB\_OK);

else MessageBox(0, "Регистрация прошла успешно", "Уведомление", MB\_OK);

char postfix[7] = "Online";

strcat(szBuf, postfix);

LPCSTR lpMailslotUnicNameUserOnline = szBuf;

globalhMailslotOnline = CreateMailslot(lpMailslotUnicNameUserOnline, 0, MAILSLOT\_WAIT\_FOREVER, NULL);;

if (globalhMailslotOnline == INVALID\_HANDLE\_VALUE) MessageBox(0, "44Ошибка при регистрации", "Уведомление", MB\_OK);

return 200;

}

extern "C" \_\_declspec(dllexport) int \_\_stdcall NotifyServerAboutDisconnection(char\* UserName) {

HANDLE hMailslot;

LPCSTR lpMailslotName = TEXT("\\\\.\\mailslot\\ClientDisconnectionChannel");

DWORD cbWritten;

hMailslot = CreateFile(lpMailslotName, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return 402;

if (!WriteFile(hMailslot, UserName, strlen(UserName), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return 403;

}

CloseHandle(hMailslot);

return 200;

}

extern "C" \_\_declspec(dllexport) int \_\_stdcall myfuncSendMessage(char\* ProtocolMessage) {

HANDLE hMailslot;

LPCSTR lpMailslotName = TEXT("\\\\.\\mailslot\\ChannelFromServer");

DWORD cbWritten;

hMailslot = CreateFile(lpMailslotName, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return -401;

if (!WriteFile(hMailslot, ProtocolMessage, strlen(ProtocolMessage), &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

CloseHandle(hMailslot);

return 200;

}

extern "C" \_\_declspec(dllexport) int \_\_stdcall myfuncSendFile(char\* filePath, char\* ProtocolMessage, int SendPosition, int chunkSize) {

HANDLE hMailslot;

LPCSTR lpMailslotName = TEXT("\\\\.\\mailslot\\ChannelFromServer");

DWORD cbWritten;

char buffer[300];

hMailslot = CreateFile(lpMailslotName, GENERIC\_READ | GENERIC\_WRITE, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hMailslot == INVALID\_HANDLE\_VALUE) return 401;

std::ifstream inFile(filePath, std::ios::binary);

if (!inFile) {

return 402; //ошибка открытия файла

}

inFile.seekg(SendPosition, std::ios::beg);

inFile.read(buffer, chunkSize);

size\_t totalSize = 26 + chunkSize;

char\* dataWithPrefix = new char[totalSize];

memcpy(dataWithPrefix, ProtocolMessage, 26);

memcpy(dataWithPrefix + 26, buffer, chunkSize);

if (!WriteFile(hMailslot, dataWithPrefix, totalSize, &cbWritten, NULL)) {

CloseHandle(hMailslot);

return -502;

}

inFile.close();

CloseHandle(hMailslot);

return 200;

}

extern "C" \_\_declspec(dllexport) char\* \_\_stdcall myfuncOnlineUserList() {

HANDLE hMailslot;

BOOL fReturnCode;

DWORD cbMessages;

DWORD cbMsgNumber;

char szBuf[2048];

DWORD cbRead;

hMailslot = globalhMailslotOnline;

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslot, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

strcpy(globalBufOnline, szBuf);

}

}

size\_t stSize = strlen(globalBufOnline) + sizeof(char);

char\* pszReturn = NULL;

pszReturn = (char\*)::CoTaskMemAlloc(stSize);

strcpy\_s(pszReturn, stSize, globalBufOnline);

return pszReturn;

}

extern "C" \_\_declspec(dllexport) char\* \_\_stdcall myfuncReceiptMessage() {

HANDLE hMailslot;

BOOL fReturnCode;

DWORD cbMessages;

DWORD cbMsgNumber;

char szBuf[2048];

DWORD cbRead;

char design[2];

int NumMsg;

int tmpNumMsg = 0;

char globalBuf\_copy[512];

hMailslot = globalhMailslot;

strcpy(globalBuf\_copy, globalBuf);

fReturnCode = GetMailslotInfo(hMailslot, NULL, &cbMessages, &cbMsgNumber, NULL);

if (!fReturnCode) {

fprintf(stdout, "GetMailslotInfo: Error %ld\n", GetLastError());

}

if (cbMessages != MAILSLOT\_NO\_MESSAGE) {

if (!ReadFile(hMailslot, szBuf, sizeof(szBuf), &cbRead, NULL)) {

fprintf(stdout, "ReadFileEx: Error %ld\n", GetLastError());

}

else {

szBuf[cbRead] = '\0';

char type = szBuf[0];

if (type == '0') {

globalBuf[0] = '\0';

memmove(szBuf, szBuf + 1, strlen(szBuf)); //отсечение type

strcat(globalBuf, szBuf);

}

else {

char fullFuleName[512] = "recepient.";

if (type == '1') strcat(fullFuleName, "txt");

if (type == '2') strcat(fullFuleName, "bmp");

if (type == '3') strcat(fullFuleName, "jpg");

if (type == '4') strcat(fullFuleName, "gif");

if (type == '5') strcat(fullFuleName, "png");

char bufferFile[300];

char cNumberCurrentMsg[5];

char cNumberTotalMsg[5];

char cUserNameSize[4];

char cUserName[128];

int iNumberCurrentMsg;

int iNumberTotalMsg;

int iUserNameSize;

strncpy(cNumberCurrentMsg, szBuf+1, 4); //номер передаваемого сообщения

cNumberCurrentMsg[4] = '\0';

iNumberCurrentMsg = atoi(cNumberCurrentMsg);

strncpy(cNumberTotalMsg, szBuf + 5, 4); //общее количество сообщений

cNumberTotalMsg[4] = '\0';

iNumberTotalMsg = atoi(cNumberTotalMsg);

strncpy(cUserNameSize, szBuf + 9, 3); //имя отправителя

cUserNameSize[3] = '\0';

iUserNameSize = atoi(cUserNameSize);

strncpy(cUserName, szBuf + 12, 3);

cUserName[iUserNameSize] = '\0';

memmove(bufferFile, szBuf + 12 + iUserNameSize, cbRead - 12 - iUserNameSize);

FILE\* outFile = fopen(fullFuleName, "ab");

if (outFile == NULL) {

fprintf(stderr, "Error opening file for writing\n");

}

size\_t written = fwrite(bufferFile, 1, cbRead - 12 - iUserNameSize, outFile);

if (written != cbRead) {

fprintf(stderr, "Error writing data to file\n");

fclose(outFile);

}

fclose(outFile);

printf("Received data: %lu bytes\n", cbRead);

if (iNumberCurrentMsg == iNumberTotalMsg) {

MessageBox(0, "Получен новый файл", "Уведомление", MB\_OK);

char ProtocolAnswer[128] = "000000000000000000000000";

char Answer[] = "Ваш файл получен\n";

memmove(ProtocolAnswer+7, cUserNameSize, 3);

memmove(ProtocolAnswer + 10, "0017", 4);

memmove(ProtocolAnswer + 22, cUserName, iUserNameSize);

memmove(ProtocolAnswer + 22 + iUserNameSize, Answer, strlen(Answer));

myfuncSendMessage(ProtocolAnswer);

}

}

}

}

if (strcmp(globalBuf, globalBuf\_copy) == 0) globalBuf[0] = '\0';

size\_t stSize = strlen(globalBuf) + sizeof(char);

char\* pszReturn = NULL;

pszReturn = (char\*)::CoTaskMemAlloc(stSize);

strcpy\_s(pszReturn, stSize, globalBuf);

return pszReturn;

}