Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Основы операционных систем»

на тему «Система программ обмена сообщениями»

Выполнил:

студент группы 22ВВП1

Хоссейни Нежад С.А.С.М.

Принял:

Егоров В.Ю.

Пенза 2024

# **Содержание**

[Содержание 4](#_Toc184924577)

[Введение 5](#_Toc184924578)

[1 Постановка задачи 6](#_Toc184924579)

[2 Теоретическая часть задания 7](#_Toc184924580)

[3 Описание алгоритма работы 9](#_Toc184924581)

[3.1 Описание протокола передачи данных 11](#_Toc184924582)

[4 Описание программы 13](#_Toc184924583)

[4.1 Сервер 13](#_Toc184924584)

[4.2 Клиент 13](#_Toc184924585)

[5 Описание работы программы 16](#_Toc184924586)

[Заключение 17](#_Toc184924587)

[Список используемых источников 18](#_Toc184924588)

[Приложение А – Листинг программы 19](#_Toc184924589)

[Приложение А.1 - Файл “main.cpp“ для Сервера 19](#_Toc184924590)

[Приложение А.2 - Файл “ ClientForm.cpp“ для Клиента 27](#_Toc184924591)

[Приложение А.3 - Файл “Message.h“ для Клиента 27](#_Toc184924592)

[Приложение А.4 - Файл “RegisterForm.h“ для Клиента 28](#_Toc184924593)

[Приложение А.5 - Файл “MessageForm.h“ для Клиента 35](#_Toc184924594)

[Приложение Б - UML-диаграммы 56](#_Toc184924595)

[Приложение Б.1 - UML-диаграмма последовательностей протокола взаимодействия клиента и сервера 56](#_Toc184924596)

[Приложение Б.2 - [UML-диаграмма вариантов использования приложения](#_4i7ojhp) 59](#_Toc184924597)

[Приложение Б.3 - UML-диаграмма развертывания 60](#_Toc184924598)

[Приложение Б.4 - UML-диаграмма состояний клиента 60](#_Toc184924599)

[Приложение Б.5 - UML-диаграмма состояний сервера 62](#_Toc184924600)

# **Введение**

Клиент-серверная архитектура представляет собой модель, в которой задачи и нагрузка в сети распределяются между сервисами, называемыми серверами, и их пользователями, именуемыми клиентами. В этой модели сервер и клиент выступают как программные компоненты. Чаще всего они находятся на разных устройствах и обмениваются данными через сеть с использованием протоколов связи, хотя могут размещаться и на одном устройстве.

Программы серверы предназначены для обработки запросов от клиентских приложений, предоставляя им данные или функциональные возможности, такие как обмен сообщениями или файлов. Поскольку один сервер может одновременно обслуживать множество клиентов, его обычно устанавливают на специально выделенном оборудовании с высокой производительностью. Такое оборудование настраивается для эффективной работы серверных приложений, которые нередко функционируют совместно с другими серверными программами. В связи с особой ролью такой машины в сети, ее называют сервером, а машины, на которых выполняются клиентские программы, соответственно, клиентами.

# **1 Постановка задачи**

Разработать систему программ клиент-серверной архитектуры взаимодействующей посредством вновь разработанного протокола передачи данных. Функционал системы включает в себя:

1. Идентификацию пользователя на сервере посредством клиента.
2. Передачу текстовых сообщений между пользователями (индивидуальных и широковещательных)
3. Передачу файлов

Функционал сервера:

1. Организация взаимодействия между клиентами (передача сообщений, файлов и т.д.)
2. Отложенная передача данных.
3. Накопление статистических данных

Функционал клиента:

1. Графический оконный интерфейс пользователя
2. Взаимодействие с сервером

Допускается расширение функционала системы программ в том числе: аутентификация пользователей, передачу мультимедийной информации, накопление статистических данных на сервере, протоколов работы, подключение СУБД в качестве хранилища информации о пользователях.

# **2 Теоретическая часть задания**

В сети Internet соединены миллионы компьютеров самых разных типов. Они могут функционировать под управлением различных операционных систем, однако для обмена данными обязаны поддерживать единый стандарт, основанный на протоколах стека TCP/IP (Transmission Control Protocol / Internet Protocol).

Протоколы транспортного уровня занимают ключевую позицию в современных сетях, обеспечивая надежную доставку информации адресату с сохранением правильной последовательности передаваемых данных. Каждый из них обладает уникальными особенностями, которые позволяют адаптировать их под конкретные задачи.

TCP (Transmission Control Protocol) представляет собой сетевой протокол, ориентированный на соединение. Прежде чем начнется передача данных, TCP устанавливает связь между двумя конечными узлами. Этот протокол обеспечивает высокую надежность: предотвращает утрату информации, запрашивает подтверждение о доставке, повторно отправляет утраченные пакеты и сохраняет последовательность передаваемых данных, гарантируя их упорядоченность.

Механизм работы TCP построен на надежной передаче данных через предварительное установление соединения. В случае потери пакетов он запрашивает их повторную передачу, а при получении дубликатов устраняет их. В отличие от UDP, TCP гарантирует целостность данных и уведомляет отправителя о результатах передачи.

Реализации TCP, как правило, интегрированы в ядро операционной системы, но существуют и варианты, работающие на уровне пользовательского пространства.

При взаимодействии компьютеров через Интернет TCP действует на уровне приложения, связывая конечные системы. Например, он обеспечивает обмен данными между браузером и веб-сервером. TCP гарантирует передачу потока байтов от одной программы на одном устройстве к другой программе на другом, будь то электронная почта, передача файлов или иные задачи. Этот протокол также регулирует длину сообщений, скорость передачи данных и объем сетевого трафика.

# **3 Описание алгоритма работы**

Алгоритм работы системы обмена сообщениями включает несколько ключевых этапов, которые обеспечивают функциональность клиент-серверного взаимодействия. Основной принцип работы алгоритма заключается в передаче сообщений и файлов между пользователями в сети с использованием пакетов данных. Алгоритм можно разделить на несколько этапов:

1. Запуск программы:

* Основная функция main инициализирует визуальные стили и запускает форму регистрации RegisterForm.

1. Форма регистрации (RegisterForm):

* Инициализация компонентов формы и сетевого сокета для подключения к серверу.
* Обработчики событий для регистрации и входа пользователя.
* Проверка длины введенного имени и пароля, отправка данных на сервер для регистрации или входа.
* Получение и обработка ответа от сервера.
* При успешной регистрации или входе скрытие формы регистрации и открытие формы чата.

1. Отправка сообщения:

* Когда пользователь выбирает контакт для отправки сообщения, система получает текст сообщения и инициирует передачу.
* Сообщение делится на пакеты, каждый из которых отправляется на сервер.
* Каждый пакет содержит информацию о типе сообщения (например, "message", "file", "chat\_history", "update\_contacts", "add\_contact", "disconnect", "login", "full\_chat", "load\_chat", "register"), отправителе, получателе, номере пакета, количестве пакетов, сообщение, задержку, размер сообщения.
* Сообщения отправляются через сокет на сервер, который далее передает их получателю.

1. Получение сообщения:

* Когда клиент получает сообщение, оно может быть разбито на несколько пакетов, которые необходимо собрать на стороне клиента.
* Каждый пакет содержит информацию о типе сообщения, отправителе, получателе, номере пакета, количестве пакетов, сообщение, задержку, размер сообщения.
* После получения всех пакетов, сообщение собирается в правильном порядке.

1. Отправка и получение файла:

* При отправке файла сообщение делится на несколько пакетов, каждый из которых отправляется по очереди.
* Каждый пакет содержит информацию о типе сообщения file, отправителе, получателе, номере пакета, количестве пакетов, сообщение (прочитанные данные из файла), название файла, задержку, размер сообщения.
* Получатель собирает все пакеты в правильном порядке, клиент предлагает пользователю скачать его и если пользователь соглашается, то файл скачивается.
* После завершения загрузки файла пользователю предоставляется уведомление об успешном завершении операции.

1. Обновление контактов:

* При добавлении нового пользователя в контакт список контактов обновляется.

1. История чатов:

* История чатов загружается при выборе контакта. Она отображается в текстовом поле чата.

## **3.1 Описание протокола передачи данных**

Протокол передачи данных между клиентом и сервером в данной системе реализуется с использованием сокетов TCP/IP, что обеспечивает надежное соединение и гарантированную доставку данных. Протокол состоит из нескольких типов сообщений, которые используются для различных операций: регистрация, отправка сообщений, получение сообщений, обновление списка контактов, передача файлов и другие.

1. **Структура сообщений:** Все сообщения между клиентом и сервером имеют единую структуру, определенную в типе данных StructMessage. Сообщения отправляются в виде строк. Эта структура включает в себя следующие поля:

* **type** — тип сообщения (например, "message", "file", "chat\_history", "update\_contacts", "add\_contact", "disconnect", "login", "full\_chat", "load\_chat", "register").
* **from** — имя отправителя.
* **to** — имя получателя.
* **message** — текст сообщения
* **fileName** — имя файла, если передается файл.
* **delaySeconds** — задержка в секундах, которая может быть применена к отправке сообщения.
* **packetNumber** — номер пакета в последовательности передачи.
* **totalPackets** — общее количество пакетов для передачи.
* **messageSize** — размер текущего пакета.

1. **Регистрация пользователя:** после старта приложения клиент отправляет на сервер запрос о регистрации с именем пользователя и паролем. Сервер проверяет, не зарегистрировано ли уже такое имя. Если имя уникально, сервер подтверждает успешную регистрацию, и клиент переходит к основному окну. Если имя уже занято, клиенту отправляется сообщение об ошибке, и ему предлагается выбрать другое имя.
2. **Отправка сообщений:** Клиент отправляет сообщение выбранному контакту. При этом текст сообщения разбивается на несколько пакетов (если размер превышает максимальный размер одного пакета), и каждый пакет отправляется по сети.
3. **Получение сообщений**: Получение сообщений реализовано на клиентской стороне путем прослушивания сокета. Клиент собирает отправленные пакеты от сервера в правильном порядке и отображает полное сообщение в окне чата, если открыт чат с этим пользователем и это не файл.
4. **Передача файлов**: Передача файлов осуществляется аналогично передаче сообщений, но с дополнительными параметрами, такими как имя файла. Файл разбивается на пакеты и передается через сеть. На клиентской стороне пакеты собираются в одном файле в правильном порядке, и по завершении передачи файл сохраняется, если пользователь согласиться его скачать.
5. **Обновление списка контактов**: Когда на сервере происходят изменения в списке контактов (новый контакт добавлен), сервер отправляет клиенту обновленный список. Клиент в свою очередь обновляет отображение контактов в интерфейсе.
6. **Завершение сеанса**: Когда клиент решает завершить сеанс общения и выйти из приложения, он отправляет серверу сообщение с типом "disconnect", в котором сообщается о завершении сеанса.

# **4 Описание программы**

## **4.1 Сервер**

1. **saveClients():** Сохраняет информацию о пользователях и их контактах в файл "clients.txt".
2. **loadClients():** Загружает информацию о пользователях и их контактах из файла "clients.txt".
3. **saveChatHistory(const string& from, const string& to, const char\* message, size\_t messageSize):** Сохраняет сообщение в файл истории чата между пользователями.
4. **loadChatHistory(const string& from, const string& to)**: Загружает историю чата между пользователями из файла.
5. **sendMessageToClients(const Message& msg)**: Отправляет сообщение указанному получателю, если его сокет доступен.
6. **sendDataMessageWithDelay(const Message& originalMsg)**: Отправляет сообщение с задержкой.
7. **handleClient(SOCKET clientSocket)**: Обрабатывает соединение с клиентом, выполняя регистрацию, вход, отправку сообщений и другие действия.

## **4.2 Клиент**

Файл **RegisterForm.h**

1. **buttonRegisterClick:** Обрабатывает нажатие кнопки "Регистрация", вызывая регистрацию пользователя на сервере.
2. **buttonLoginClick**: Обрабатывает нажатие кнопки "Вход", инициируя процесс входа пользователя в систему.
3. **checkBoxShowPasswordCheckedChanged**: Изменяет видимость пароля в зависимости от состояния флажка "Показать пароль".
4. **registerOrLogin**: Отправляет данные пользователя на сервер для регистрации или входа и отображает результат.
5. **InitializeSocket**: Инициализирует сокет и устанавливает соединение с сервером для обмена данными.

Файл **Messageform.h**

1. **buttonSendFileClick:** Обрабатывает нажатие кнопки "Отправить файл", вызывая процесс отправки файла выбранному контакту.
2. **sendFile**: Отправляет файл на сервер через сокет, используя его имя и содержимое.
3. **buttonSendClick**: Обрабатывает нажатие кнопки "Отправить сообщение", отправляя введенное сообщение выбранному контакту.
4. **getSelectedContact**: Возвращает имя выбранного контакта из списка контактов для отправки сообщения.
5. **listenForMessages**: Слушает сервер на наличие новых сообщений и обрабатывает их при получении.
6. **saveMessageToServer**: Сохраняет полученные сообщения на сервере для последующего доступа.
7. **appendChatHistory**: Добавляет новое сообщение в историю чата на клиенте, обновляя интерфейс.
8. **updateChat**: Обновляет отображение чата с новым сообщением или изменением истории.
9. **updateContactsList**: Обновляет список контактов в пользовательском интерфейсе после изменений на сервере.
10. **listBoxContactsSelectedIndexChanged**: Обрабатывает изменение выбранного контакта в списке контактов, обновляя чат.
11. **loadChatHistory**: Загружает историю сообщений для выбранного контакта с сервера.
12. **stopListening**: Останавливает прослушивание сервера для получения новых сообщений.
13. **addContact**: Добавляет нового контакта в список контактов на сервере.
14. **buttonAddContactClick**: Обрабатывает нажатие кнопки "Добавить контакт", инициируя добавление нового контакта.
15. **notifyServerAboutExit**: Уведомляет сервер о выходе пользователя из системы перед его отключением.

Файл **ClientForm.cpp**

Файл **ClientForm.cpp** запускает приложение с графическим интерфейсом, создавая и отображая форму регистрации пользователя. Основная цель этого файла — инициализация формы RegisterForm и запуск приложения Windows Forms.

Файл **Message.h**

Файл **Message.h** содержит определение структуры **StructMessage**, которая используется для представления сообщений в системе. Структура включает информацию о типе сообщения, отправителе, получателе, содержимом сообщения, имени файла (если отправляется файл), задержке, номере пакета, общем числе пакетов и размере сообщения. Этот файл позволяет организовать обмен данными между клиентом и сервером, предоставляя удобную структуру для передачи всей необходимой информации о сообщении.

# **5 Описание работы программы**

Программа представляет собой клиент-серверное приложение для обмена сообщениями и файлами. При запуске сервер загружает данные о пользователях из файла clients.txt, где хранятся логины, пароли и списки контактов, и начинает ожидать входящих соединений. Клиенты подключаются к серверу через сокеты, после чего могут зарегистрироваться или войти в систему. При регистрации сервер проверяет уникальность логина и сохраняет данные нового пользователя в файл. Если пользователь уже зарегистрирован, сервер отправляет сообщение клиенту что такой пользователь уже есть, в случае успеха пользователь может войти в систему. Если пользователь уже зарегистрирован, а ввел неправильный пароль, то сервер отправит сообщение об этом.

После входа клиент запрашивает у сервера список контактов, который загружается и отображается в интерфейсе программы. Пользователь может добавить новый контакт, и эта информация синхронизируется с сервером. Через интерфейс клиент отправляет сообщения и файлы другим пользователям. Сервер обрабатывает эти запросы, проверяет доступность адресата и пересылает данные. Сообщения могут передаваться с заданной задержкой, для чего сервер запускает отдельный поток, ожидающий указанное время перед отправкой.

История переписки между пользователями хранится на сервере в текстовых файлах, которые обновляются при отправке сообщений. Клиент может запросить полную историю чата с выбранным контактом. Сервер считывает соответствующий файл, разделяет данные на пакеты и отправляет их клиенту, где они отображаются в окне чата.

Программа завершается для клиента при разрыве соединения или закрытии приложения, о чём сервер уведомляется специальным сообщением. Сервер продолжает работу до ручного завершения и сохраняет все обновления данных для последующих запусков.

# **Заключение**

При выполнении данного курсового проекта были изучены принципы работы протоколов TCP/IP, разработка клиента и сервера.

В результате выполнения данной курсовой работы была разработана система программ клиент-серверной архитектуры, были получены следующие навыки: программирования на С++, работы с Windows Forms, проработки архитектуры программ, разработки протоколов взаимодействия, разработки визуальных частей.

Программа написана на языке C++ с использование среды программирования Visual Studio 2022.

# **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.

2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2007.– 256 с. Калиткин Н.Н. Численные методы. – М.: Наука, 1978. – 267 с.

3. Керниган Брайан У., Ритчи Деннис М. Язык программирования C, 2-е изд. СПб: ООО «Диалектика», 2020. – 288 с.

4. Подбельский В.В., Фомин С.С. Курс программирования на языке Си. Учебник. Изд. 2-е, переработанное. М: ДМК Пресс, 2018 –– 384 с.

# **Приложение А**

(Обязательное)

Листинг Программы

## **Приложение А.1**

Файл “main.cpp“ для Сервера

#include <iostream>

#include <cstring>

#include <unordered\_map>

#include <vector>

#include <fstream>

#include <sstream>

#include <winsock2.h>

#include <Windows.h>

#include <ws2tcpip.h>

#include <thread>

#include <mutex>

#include <random>

#include <cstdio>

#include <string>

#include <locale.h>

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

using namespace std;

struct Message {

char type[100];

char from[100];

char to[100];

char message[100];

char filename[100];

int delaySeconds;

int packetNumber;

int totalPackets;

size\_t messageSize;

};

struct User {

SOCKET socket;

string password;

vector<string> contacts;

};

unordered\_map<string, User> clients;

mutex fileMutex;

void saveClients() {

ofstream out("clients.txt");

for (const auto& pair : clients) {

out << pair.first << " " << pair.second.password;

for (const auto& contact : pair.second.contacts) {

out << " " << contact;

}

out << endl;

}

out.close();

}

void loadClients() {

ifstream in("clients.txt");

string line;

while (getline(in, line)) {

stringstream ss(line);

string name, password, contact;

ss >> name >> password;

clients[name] = { INVALID\_SOCKET, password };

while (ss >> contact) {

clients[name].contacts.push\_back(contact);

}

}

in.close();

}

void saveChatHistory(const string& from, const string& to, const char\* message, size\_t messageSize) {

lock\_guard<mutex> lock(fileMutex);

string filename = from + "\_" + to + "\_chat.txt";

ofstream chatLog(filename, ios\_base::app);

chatLog.write(message, messageSize);

}

string loadChatHistory(const string& from, const string& to) {

lock\_guard<mutex> lock(fileMutex);

string filename = from + "\_" + to + "\_chat.txt";

ifstream chatLog(filename);

stringstream buffer;

buffer << chatLog.rdbuf();

return buffer.str();

}

void sendMessageToClients(const Message& msg) {

if (clients.find(msg.to) != clients.end() && clients[msg.to].socket != INVALID\_SOCKET) {

int result = send(clients[msg.to].socket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

if (result == SOCKET\_ERROR) {

cerr << "Не удалось отправить сообщение на " << msg.to << ". Ошибка: " << WSAGetLastError() << endl;

}

}

}

void sendDataMessageWithDelay(const Message& originalMsg) {

this\_thread::sleep\_for(chrono::seconds(originalMsg.delaySeconds));

sendMessageToClients(originalMsg);

}

void handleClient(SOCKET clientSocket) {

Message msg;

while (recv(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0) > 0) {

if (strcmp(msg.type, "register") == 0) {

if (clients.find(msg.from) == clients.end()) {

clients[msg.from] = { clientSocket, msg.message };

saveClients();

cout << "Пользователь " << msg.from << " зарегистрировался." << endl;

strcpy\_s(msg.type, "response");

strcpy\_s(msg.message, "Registration successful");

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

}

else {

cerr << "Регистрация не удалась. Пользователь " << msg.from << " уже существует." << endl;

strcpy\_s(msg.type, "response");

strcpy\_s(msg.message, "Такой пользователь уже есть!");

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

}

}

else if (strcmp(msg.type, "login") == 0) {

if (clients.find(msg.from) != clients.end() && clients[msg.from].password == msg.message) {

clients[msg.from].socket = clientSocket;

cout << "Пользователь " << msg.from << " вошел в систему." << endl;

strcpy\_s(msg.type, "response");

strcpy\_s(msg.message, "Login successful");

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

stringstream ss;

for (const auto& contact : clients[msg.from].contacts) {

ss << contact << ",";

}

string contactsList = ss.str();

if (!contactsList.empty()) {

contactsList.pop\_back();

}

strcpy\_s(msg.type, "update\_contacts");

strncpy\_s(msg.message, contactsList.c\_str(), sizeof(msg.message) - 1);

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

}

else if (clients.find(msg.from) == clients.end()) {

cerr << "Не удалось войти в систему. Пользователь " << msg.from << " не зарегистрирован." << endl;

strcpy\_s(msg.type, "response");

strcpy\_s(msg.message, "Пользователь не зарегестрирован!");

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

}

else {

cerr << "Не удалось войти в систему для " << msg.from << ". Неверный пароль." << endl;

strcpy\_s(msg.type, "response");

strcpy\_s(msg.message, "Не верный пароль!");

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

}

}

else if (strcmp(msg.type, "message") == 0 || strcmp(msg.type, "file") == 0) {

thread(sendDataMessageWithDelay, msg).detach();

}

else if (strcmp(msg.type, "full\_chat") == 0) {

saveChatHistory(msg.from, msg.to, msg.message, msg.messageSize);

}

else if (strcmp(msg.type, "add\_contact") == 0) {

string contactName(msg.to);

if (clients.find(contactName) != clients.end()) {

if (find(clients[msg.from].contacts.begin(), clients[msg.from].contacts.end(), contactName) == clients[msg.from].contacts.end()) {

clients[msg.from].contacts.push\_back(contactName);

saveClients();

stringstream ss;

for (const auto& contact : clients[msg.from].contacts) {

ss << contact << ",";

}

string contactsList = ss.str();

if (!contactsList.empty()) {

contactsList.pop\_back();

}

strcpy\_s(msg.type, "update\_contacts");

strncpy\_s(msg.message, contactsList.c\_str(), sizeof(msg.message) - 1);

send(clientSocket, reinterpret\_cast<const char\*>(&msg), sizeof(msg), 0);

}

else {

cerr << "Контакт " << contactName << " уже существует для пользователя " << msg.from << "." << endl;

}

}

else {

cerr << "Пользователь " << contactName << " не найден." << endl;

}

}

else if (strcmp(msg.type, "load\_chat") == 0) {

string chatHistory = loadChatHistory(msg.from, msg.to);

Message chatMsg;

strcpy\_s(chatMsg.type, sizeof(chatMsg.type), "chat\_history");

strcpy\_s(chatMsg.from, sizeof(chatMsg.from), "server");

strcpy\_s(chatMsg.to, sizeof(chatMsg.to), msg.from);

size\_t pos = 0;

int totalPackets = (chatHistory.size() + sizeof(chatMsg.message) - 1) / sizeof(chatMsg.message);

chatMsg.totalPackets = totalPackets;

int packetNumber = 0;

while (pos < chatHistory.size()) {

size\_t bytesToCopy = (sizeof(chatMsg.message) < (chatHistory.size() - pos)) ? sizeof(chatMsg.message) : (chatHistory.size() - pos);

memcpy\_s(chatMsg.message, sizeof(chatMsg.message), chatHistory.data() + pos, bytesToCopy);

chatMsg.messageSize = bytesToCopy;

chatMsg.packetNumber = packetNumber++;

send(clientSocket, reinterpret\_cast<char\*>(&chatMsg), sizeof(chatMsg), 0);

pos += bytesToCopy;

}

}

else if (strcmp(msg.type, "disconnect") == 0) {

cerr << "Клиент " << msg.from << " отключился." << endl;

}

}

closesocket(clientSocket);

}

int main() {

setlocale(LC\_ALL, "Rus");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

WSAData wsaData;

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

loadClients();

SOCKET serverSocket, clientSocket;

struct sockaddr\_in serverAddr, clientAddr;

int clientAddrLen = sizeof(clientAddr);

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

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_addr.s\_addr = INADDR\_ANY;

serverAddr.sin\_port = htons(8080);

bind(serverSocket, (struct sockaddr\*)&serverAddr, sizeof(serverAddr));

listen(serverSocket, 5);

cout << "Сервер запущен. Ожидание клиентов..." << endl;

while (true) {

clientSocket = accept(serverSocket, (struct sockaddr\*)&clientAddr, &clientAddrLen);

thread(handleClient, clientSocket).detach();

}

closesocket(serverSocket);

WSACleanup();

return 0;

}

## **Приложение А.2**

Файл “ ClientForm.cpp“ для Клиента

#include "RegisterForm.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void main(array<String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

MyClient::RegisterForm form;

Application::Run(% form);

}

## **Приложение А.3**

Файл “Message.h“ для Клиента

#pragma once

struct StructMessage {

char type[100];

char from[100];

char to[100];

char message[100];

char fileName[100];

int delaySeconds;

int packetNumber;

int totalPackets;

size\_t messageSize;

};

## **Приложение А.4**

Файл “RegisterForm.h“ для Клиента

#pragma once

#include "MessageForm.h"

#include "Message.h"

#include <winsock2.h>

#include <ws2tcpip.h>

#include <msclr\marshal.h>

#include <msclr\marshal\_cppstd.h>

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

namespace MyClient {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace msclr::interop;

public ref class RegisterForm : public System::Windows::Forms::Form {

public:

RegisterForm(void) {

InitializeComponent();

InitializeSocket();

}

protected:

~RegisterForm() {

if (components) {

delete components;

}

}

private:

System::Windows::Forms::Label^ labelName;

System::Windows::Forms::Label^ labelPassword;

System::Windows::Forms::TextBox^ textBoxName;

System::Windows::Forms::TextBox^ textBoxPassword;

System::Windows::Forms::Button^ buttonRegister;

System::Windows::Forms::Button^ buttonLogin;

System::Windows::Forms::CheckBox^ checkBoxShowPassword;

SOCKET clientSocket;

System::ComponentModel::Container^ components;

void InitializeComponent(void) {

this->labelName = (gcnew System::Windows::Forms::Label());

this->labelPassword = (gcnew System::Windows::Forms::Label());

this->textBoxName = (gcnew System::Windows::Forms::TextBox());

this->textBoxPassword = (gcnew System::Windows::Forms::TextBox());

this->buttonRegister = (gcnew System::Windows::Forms::Button());

this->buttonLogin = (gcnew System::Windows::Forms::Button());

this->checkBoxShowPassword = (gcnew System::Windows::Forms::CheckBox());

this->SuspendLayout();

this->labelName->AutoSize = true;

this->labelName->Font = (gcnew System::Drawing::Font(L"Arial", 12));

this->labelName->Location = System::Drawing::Point(15, 16);

this->labelName->Margin = System::Windows::Forms::Padding(2, 0, 2, 0);

this->labelName->Name = L"labelName";

this->labelName->Size = System::Drawing::Size(106, 18);

this->labelName->TabIndex = 0;

this->labelName->Text = L"Введите имя:";

this->labelPassword->AutoSize = true;

this->labelPassword->Font = (gcnew System::Drawing::Font(L"Arial", 12));

this->labelPassword->Location = System::Drawing::Point(15, 86);

this->labelPassword->Margin = System::Windows::Forms::Padding(2, 0, 2, 0);

this->labelPassword->Name = L"labelPassword";

this->labelPassword->Size = System::Drawing::Size(130, 18);

this->labelPassword->TabIndex = 2;

this->labelPassword->Text = L"Введите пароль:";

this->textBoxName->Font = (gcnew System::Drawing::Font(L"Arial", 12));

this->textBoxName->Location = System::Drawing::Point(15, 41);

this->textBoxName->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->textBoxName->Name = L"textBoxName";

this->textBoxName->Size = System::Drawing::Size(289, 26);

this->textBoxName->TabIndex = 1;

this->textBoxPassword->Font = (gcnew System::Drawing::Font(L"Arial", 12));

this->textBoxPassword->Location = System::Drawing::Point(15, 111);

this->textBoxPassword->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->textBoxPassword->Name = L"textBoxPassword";

this->textBoxPassword->PasswordChar = '\*';

this->textBoxPassword->Size = System::Drawing::Size(289, 26);

this->textBoxPassword->TabIndex = 3;

this->buttonRegister->Font = (gcnew System::Drawing::Font(L"Arial", 12));

this->buttonRegister->Location = System::Drawing::Point(15, 193);

this->buttonRegister->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->buttonRegister->Name = L"buttonRegister";

this->buttonRegister->Size = System::Drawing::Size(130, 32);

this->buttonRegister->TabIndex = 5;

this->buttonRegister->Text = L"Регистрация";

this->buttonRegister->UseVisualStyleBackColor = true;

this->buttonRegister->Click += gcnew System::EventHandler(this, &RegisterForm::buttonRegisterClick);

this->buttonLogin->Font = (gcnew System::Drawing::Font(L"Arial", 12));

this->buttonLogin->Location = System::Drawing::Point(174, 193);

this->buttonLogin->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->buttonLogin->Name = L"buttonLogin";

this->buttonLogin->Size = System::Drawing::Size(130, 32);

this->buttonLogin->TabIndex = 6;

this->buttonLogin->Text = L"Вход";

this->buttonLogin->UseVisualStyleBackColor = true;

this->buttonLogin->Click += gcnew System::EventHandler(this, &RegisterForm::buttonLoginClick);

this->checkBoxShowPassword->AutoSize = true;

this->checkBoxShowPassword->Font = (gcnew System::Drawing::Font(L"Arial", 10));

this->checkBoxShowPassword->Location = System::Drawing::Point(15, 150);

this->checkBoxShowPassword->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->checkBoxShowPassword->Name = L"checkBoxShowPassword";

this->checkBoxShowPassword->Size = System::Drawing::Size(138, 20);

this->checkBoxShowPassword->TabIndex = 4;

this->checkBoxShowPassword->Text = L"Показать пароль";

this->checkBoxShowPassword->UseVisualStyleBackColor = true;

this->checkBoxShowPassword->CheckedChanged += gcnew System::EventHandler(this, &RegisterForm::checkBoxShowPasswordCheckedChanged);

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(326, 249);

this->Controls->Add(this->buttonLogin);

this->Controls->Add(this->buttonRegister);

this->Controls->Add(this->checkBoxShowPassword);

this->Controls->Add(this->textBoxPassword);

this->Controls->Add(this->labelPassword);

this->Controls->Add(this->textBoxName);

this->Controls->Add(this->labelName);

this->Margin = System::Windows::Forms::Padding(2, 2, 2, 2);

this->Name = L"RegisterForm";

this->Text = L"Вход/Регистрация";

this->ResumeLayout(false);

this->PerformLayout();

}

void InitializeSocket() {

WSAData wsaData;

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

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

sockaddr\_in server\_addr;

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

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(8080);

connect(clientSocket, (sockaddr\*)&server\_addr, sizeof(server\_addr));

}

void buttonRegisterClick(System::Object^ sender, System::EventArgs^ e) {

registerOrLogin("register");

}

void buttonLoginClick(System::Object^ sender, System::EventArgs^ e) {

registerOrLogin("login");

}

void checkBoxShowPasswordCheckedChanged(System::Object^ sender, System::EventArgs^ e) {

if (checkBoxShowPassword->Checked) {

textBoxPassword->PasswordChar = '\0';

}

else {

textBoxPassword->PasswordChar = '\*';

}

}

void registerOrLogin(const char\* type) {

String^ name = textBoxName->Text;

String^ password = textBoxPassword->Text;

marshal\_context context;

const char\* charName = context.marshal\_as<const char\*>(name);

const char\* charPassword = context.marshal\_as<const char\*>(password);

size\_t maxSizeOfFrom = sizeof(StructMessage::from);

size\_t maxSizeOfMessage = sizeof(StructMessage::message);

if (strlen(charName) >= maxSizeOfFrom) {

MessageBox::Show("Имя слишком длинное!");

return;

}

if (strlen(charPassword) >= maxSizeOfMessage) {

MessageBox::Show("Пароль слишком длинный!");

return;

}

StructMessage msg;

strcpy(msg.type, type);

strncpy\_s(msg.from, charName, maxSizeOfFrom - 1);

msg.from[maxSizeOfFrom - 1] = '\0';

strcpy(msg.to, "");

strncpy\_s(msg.message, charPassword, maxSizeOfMessage - 1);

msg.message[maxSizeOfMessage - 1] = '\0';

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

// Ждем ответа от сервера

StructMessage response;

recv(clientSocket, reinterpret\_cast<char\*>(&response), sizeof(response), 0);

// Обработка ответа

String^ serverMessage = gcnew String(response.message);

if (serverMessage == "Registration successful") {

MessageBox::Show("Регистрация прошла успешно!");

}

else if (serverMessage == "Login successful") {

MessageBox::Show("Вход выполнен успешно!");

this->Hide();

MessageForm^ messageForm = gcnew MessageForm(clientSocket, name);

messageForm->ShowDialog();

this->Close();

}

else {

MessageBox::Show(serverMessage);

}

}

};

}

## **Приложение А.5**

Файл “MessageForm.h“ для Клиента

#pragma once

#include "Message.h"

#include <winsock2.h>

#include <ws2tcpip.h>

#include <string>

#include <msclr\marshal.h>

#include <msclr\marshal\_cppstd.h>

#include <thread>

#include <mutex>

#include <map>

#include <Windows.h>

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

namespace MyClient {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace msclr::interop;

using namespace System::Threading;

public ref class MessageForm : public Form {

public:

MessageForm(SOCKET socket, String^ name) {

InitializeComponent();

clientSocket = socket;

clientName = name;

labelUserName->Text = "Пользователь: " + clientName;

cancellationSource = gcnew CancellationTokenSource();

listenThread = gcnew Thread(gcnew ThreadStart(this, &MessageForm::listenForMessages));

listenThread->Start();

}

protected:

~MessageForm() {

if (components) {

delete components;

}

stopListening();

}

private:

TextBox^ textBoxTo;

TextBox^ textBoxMessage;

Button^ buttonSend;

TextBox^ textBoxChat;

SOCKET clientSocket;

String^ clientName;

String^ incomingFrom;

String^ incomingMessage;

Thread^ listenThread;

CancellationTokenSource^ cancellationSource;

System::ComponentModel::Container^ components;

Label^ labelTo;

Label^ labelChat;

Label^ labelMessage;

Label^ labelUserName;

Label^ labelAddContact;

Button^ buttonSendFile;

Button^ buttonAddContact;

Label^ labelDelay;

NumericUpDown^ numericUpDownDelay;

ListBox^ listBoxContacts;

String^ currentContact;

void InitializeComponent(void) {

this->components = gcnew System::ComponentModel::Container();

this->labelTo = (gcnew Label());

this->labelAddContact = (gcnew Label());

this->labelChat = (gcnew Label());

this->labelMessage = (gcnew Label());

this->labelDelay = (gcnew Label());

this->textBoxTo = (gcnew TextBox());

this->textBoxMessage = (gcnew TextBox());

this->buttonSend = (gcnew Button());

this->buttonSendFile = (gcnew Button());

this->buttonAddContact = (gcnew Button());

this->textBoxChat = (gcnew TextBox());

this->listBoxContacts = (gcnew ListBox());

this->numericUpDownDelay = (gcnew NumericUpDown());

this->labelUserName = (gcnew Label());

this->SuspendLayout();

this->labelTo->AutoSize = true;

this->labelTo->Location = System::Drawing::Point(12, 55);

this->labelTo->Name = L"labelTo";

this->labelTo->Size = System::Drawing::Size(84, 13);

this->labelTo->TabIndex = 0;

this->labelTo->Text = L"Контакты:";

this->labelTo->Anchor = AnchorStyles::Top | AnchorStyles::Left;

this->labelAddContact->AutoSize = true;

this->labelAddContact->Location = System::Drawing::Point(170, 10);

this->labelAddContact->Name = L"labelAddContact";

this->labelAddContact->Size = System::Drawing::Size(165, 13);

this->labelAddContact->TabIndex = 0;

this->labelAddContact->Text = L"Введите кого добавить в контакты:";

this->labelAddContact->Anchor = AnchorStyles::Top | AnchorStyles::Left;

this->textBoxTo->Location = System::Drawing::Point(170, 28);

this->textBoxTo->Name = L"textBoxTo";

this->textBoxTo->Size = System::Drawing::Size(590, 20);

this->textBoxTo->TabIndex = 1;

this->textBoxTo->Anchor = AnchorStyles::Top | AnchorStyles::Left | AnchorStyles::Right;

this->buttonAddContact->Location = System::Drawing::Point(770, 26);

this->buttonAddContact->Name = L"buttonAddContact";

this->buttonAddContact->Size = System::Drawing::Size(100, 25);

this->buttonAddContact->TabIndex = 6;

this->buttonAddContact->Text = L"Добавить";

this->buttonAddContact->UseVisualStyleBackColor = true;

this->buttonAddContact->Click += gcnew System::EventHandler(this, &MessageForm::buttonAddContactClick);

this->buttonAddContact->Anchor = AnchorStyles::Top | AnchorStyles::Right;

this->labelChat->AutoSize = true;

this->labelChat->Location = System::Drawing::Point(170, 54);

this->labelChat->Name = L"labelChat";

this->labelChat->Size = System::Drawing::Size(29, 13);

this->labelChat->TabIndex = 2;

this->labelChat->Text = L"Чат";

this->labelChat->Anchor = AnchorStyles::Top | AnchorStyles::Left;

this->textBoxChat->Location = System::Drawing::Point(170, 70);

this->textBoxChat->Multiline = true;

this->textBoxChat->Name = L"textBoxChat";

this->textBoxChat->Size = System::Drawing::Size(590, 450);

this->textBoxChat->TabIndex = 3;

this->textBoxChat->ReadOnly = true;

this->textBoxChat->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;

this->textBoxChat->Anchor = AnchorStyles::Top | AnchorStyles::Left | AnchorStyles::Right | AnchorStyles::Bottom;

this->listBoxContacts->Location = System::Drawing::Point(12, 70);

this->listBoxContacts->Name = L"listBoxContacts";

this->listBoxContacts->Size = System::Drawing::Size(150, 450);

this->listBoxContacts->TabIndex = 10;

this->listBoxContacts->SelectionMode = System::Windows::Forms::SelectionMode::MultiExtended;

this->listBoxContacts->SelectedIndexChanged += gcnew System::EventHandler(this, &MessageForm::listBoxContactsSelectedIndexChanged);

this->listBoxContacts->Anchor = AnchorStyles::Top | AnchorStyles::Left | AnchorStyles::Bottom;

this->labelDelay->AutoSize = true;

this->labelDelay->Location = System::Drawing::Point(170, 530);

this->labelDelay->Name = L"labelDelay";

this->labelDelay->Size = System::Drawing::Size(150, 13);

this->labelDelay->TabIndex = 8;

this->labelDelay->Text = L"Введите задержку в секундах";

this->labelDelay->Anchor = AnchorStyles::Left | AnchorStyles::Bottom;

this->numericUpDownDelay->Location = System::Drawing::Point(170, 550);

this->numericUpDownDelay->Maximum = System::Decimal(3600);

this->numericUpDownDelay->Name = L"numericUpDownDelay";

this->numericUpDownDelay->Size = System::Drawing::Size(120, 20);

this->numericUpDownDelay->TabIndex = 9;

this->numericUpDownDelay->Anchor = AnchorStyles::Left | AnchorStyles::Bottom;

this->labelMessage->AutoSize = true;

this->labelMessage->Location = System::Drawing::Point(170, 580);

this->labelMessage->Name = L"labelMessage";

this->labelMessage->Size = System::Drawing::Size(120, 13);

this->labelMessage->TabIndex = 4;

this->labelMessage->Text = L"Введите сообщение";

this->labelMessage->Anchor = AnchorStyles::Left | AnchorStyles::Bottom;

this->textBoxMessage->Location = System::Drawing::Point(170, 600);

this->textBoxMessage->Multiline = true;

this->textBoxMessage->Name = L"textBoxMessage";

this->textBoxMessage->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;

this->textBoxMessage->Size = System::Drawing::Size(590, 60);

this->textBoxMessage->TabIndex = 5;

this->textBoxMessage->Anchor = AnchorStyles::Left | AnchorStyles::Right | AnchorStyles::Bottom;

this->buttonSend->Location = System::Drawing::Point(770, 600);

this->buttonSend->Name = L"buttonSend";

this->buttonSend->Size = System::Drawing::Size(100, 27);

this->buttonSend->TabIndex = 6;

this->buttonSend->Text = L"Отправить";

this->buttonSend->UseVisualStyleBackColor = true;

this->buttonSend->Click += gcnew System::EventHandler(this, &MessageForm::buttonSendClick);

this->buttonSend->Anchor = AnchorStyles::Right | AnchorStyles::Bottom;

this->buttonSendFile->Location = System::Drawing::Point(770, 633);

this->buttonSendFile->Name = L"buttonSendFile";

this->buttonSendFile->Size = System::Drawing::Size(100, 27);

this->buttonSendFile->TabIndex = 7;

this->buttonSendFile->Text = L"Отправить файл";

this->buttonSendFile->UseVisualStyleBackColor = true;

this->buttonSendFile->Click += gcnew System::EventHandler(this, &MessageForm::buttonSendFileClick);

this->buttonSendFile->Anchor = AnchorStyles::Right | AnchorStyles::Bottom;

this->labelUserName->AutoSize = true;

this->labelUserName->Location = System::Drawing::Point(10, 10);

this->labelUserName->Name = L"labelUserName";

this->labelUserName->Size = System::Drawing::Size(90, 13);

this->labelUserName->TabIndex = 0;

this->labelUserName->Text = L"Вы: " + clientName;

this->labelUserName->MaximumSize = System::Drawing::Size(90, 0);

this->labelUserName->AutoEllipsis = true;

this->labelUserName->Anchor = AnchorStyles::Top | AnchorStyles::Left | AnchorStyles::Right;

// MessageForm

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(900, 700);

this->Controls->Add(this->listBoxContacts);

this->Controls->Add(this->labelDelay);

this->Controls->Add(this->numericUpDownDelay);

this->Controls->Add(this->buttonSendFile);

this->Controls->Add(this->labelAddContact);

this->Controls->Add(this->buttonSend);

this->Controls->Add(this->textBoxMessage);

this->Controls->Add(this->textBoxChat);

this->Controls->Add(this->buttonAddContact);

this->Controls->Add(this->textBoxTo);

this->Controls->Add(this->labelTo);

this->Controls->Add(this->labelChat);

this->Controls->Add(this->labelMessage);

this->Controls->Add(this->labelUserName);

this->Name = L"MessageForm";

this->Text = L"Message";

this->ResumeLayout(false);

this->PerformLayout();

}

void buttonSendFileClick(System::Object^ sender, System::EventArgs^ e) {

OpenFileDialog^ openFileDialog = gcnew OpenFileDialog();

if (openFileDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK) {

String^ filePath = openFileDialog->FileName;

sendFile(filePath);

}

}

void sendFile(String^ filePath) {

marshal\_context context;

auto selectedContacts = listBoxContacts->SelectedItems;

if (listBoxContacts->SelectedItem != nullptr) {

String^ to = listBoxContacts->SelectedItem->ToString();

const char\* charTo = context.marshal\_as<const char\*>(to);

const char\* fileName = context.marshal\_as<const char\*>(filePath);

std::string fileNameStr(fileName);

size\_t lastSlash = fileNameStr.find\_last\_of("\\/");

std::string shortFileName = fileNameStr.substr(lastSlash + 1);

FILE\* file;

fopen\_s(&file, fileName, "rb");

if (file == nullptr) {

MessageBox::Show("Ошибка открытия файла!");

return;

}

fseek(file, 0, SEEK\_END);

long fileSize = ftell(file);

fseek(file, 0, SEEK\_SET);

int totalPackets = (fileSize + sizeof(((StructMessage\*)0)->message) - 2) / (sizeof(((StructMessage\*)0)->message) - 1);

char buffer[100];

StructMessage msg;

strcpy\_s(msg.type, "file");

strcpy\_s(msg.from, context.marshal\_as<const char\*>(clientName));

strcpy\_s(msg.fileName, shortFileName.c\_str());

msg.totalPackets = totalPackets;

msg.delaySeconds = static\_cast<int>(numericUpDownDelay->Value);

int packetNumber = 0;

while (fileSize > 0) {

int bytesRead = fread(buffer, 1, sizeof(buffer) - 1, file);

msg.messageSize = bytesRead;

memcpy\_s(msg.message, sizeof(msg.message), buffer, bytesRead);

msg.packetNumber = packetNumber++;

for each(String ^ to in selectedContacts) {

const char\* charTo = context.marshal\_as<const char\*>(to);

strcpy\_s(msg.to, charTo);

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

}

fileSize -= bytesRead;

}

fclose(file);

}

else {

MessageBox::Show("Пожалуйста, выберите контакт для отправки сообщения!");

}

}

void buttonSendClick(System::Object^ sender, System::EventArgs^ e) {

auto selectedContacts = listBoxContacts->SelectedItems;

if (listBoxContacts->SelectedItem != nullptr) {

String^ to = listBoxContacts->SelectedItem->ToString();

String^ message = textBoxMessage->Text;

marshal\_context context;

const char\* charTo = context.marshal\_as<const char\*>(to);

std::string charMessage = context.marshal\_as<std::string>(message);

StructMessage msg;

strcpy\_s(msg.type, "message");

strcpy\_s(msg.from, context.marshal\_as<const char\*>(clientName));

msg.delaySeconds = static\_cast<int>(numericUpDownDelay->Value);

size\_t pos = 0;

int totalPackets = (charMessage.size() + sizeof(msg.message) - 1) / sizeof(msg.message);

msg.totalPackets = totalPackets;

int packetNumber = 0;

while (pos < charMessage.size()) {

size\_t bytesToCopy = (sizeof(msg.message) < (charMessage.size() - pos)) ? sizeof(msg.message) : (charMessage.size() - pos);

memcpy\_s(msg.message, sizeof(msg.message), charMessage.data() + pos, bytesToCopy);

msg.messageSize = bytesToCopy;

msg.packetNumber = packetNumber++;

for each(String ^ to in selectedContacts) {

const char\* charTo = context.marshal\_as<const char\*>(to);

strcpy\_s(msg.to, charTo);

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

}

pos += bytesToCopy;

}

textBoxChat->AppendText("Вы: " + message + Environment::NewLine);

textBoxMessage->Clear();

}

else {

MessageBox::Show("Пожалуйста, выберите контакт для отправки сообщения!");

}

}

String^ getSelectedContact() {

if (listBoxContacts->InvokeRequired) {

return (String^)listBoxContacts->Invoke(gcnew Func<String^>(this, &MessageForm::getSelectedContact));

}

else {

return listBoxContacts->SelectedItem != nullptr ? listBoxContacts->SelectedItem->ToString() : nullptr;

}

}

void listenForMessages() {

StructMessage msg;

FILE\* file = nullptr;

std::map<int, std::vector<char>> packetBuffer;

marshal\_context context;

while (!cancellationSource->Token.IsCancellationRequested) {

int result = recv(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

if (result > 0) {

if (strcmp(msg.type, "file") == 0) {

std::vector<char> packet(msg.message, msg.message + msg.messageSize);

packetBuffer[msg.packetNumber] = packet;

if (packetBuffer.size() == msg.totalPackets) {

System::Windows::Forms::DialogResult result = System::Windows::Forms::MessageBox::Show("Получен файл. Скачать?", "Уведомление", System::Windows::Forms::MessageBoxButtons::YesNo);

if (result == System::Windows::Forms::DialogResult::Yes) {

if (file == nullptr) {

fopen\_s(&file, msg.fileName, "wb");

if (file == nullptr) {

MessageBox::Show("Ошибка создания файла!");

continue;

}

}

for (const auto& packet : packetBuffer) {

fwrite(packet.second.data(), 1, packet.second.size(), file);

}

fclose(file);

file = nullptr;

MessageBox::Show("Файл успешно скачан и сохранен.");

}

packetBuffer.clear();

}

}

else if (strcmp(msg.type, "chat\_history") == 0) {

std::vector<char> packet(msg.message, msg.message + msg.messageSize);

packetBuffer[msg.packetNumber] = packet;

if (packetBuffer.size() == msg.totalPackets) {

std::vector<char> completeMessage;

for (const auto& packet : packetBuffer) {

completeMessage.insert(completeMessage.end(), packet.second.begin(), packet.second.end());

}

String^ chatHistory = gcnew String(completeMessage.data(), 0, completeMessage.size());

if (this->IsHandleCreated) {

this->Invoke(gcnew Action<String^>(this, &MessageForm::appendChatHistory), chatHistory);

}

packetBuffer.clear();

}

}

else if (strcmp(msg.type, "update\_contacts") == 0) {

String^ contactsList = gcnew String(msg.message);

if (this->IsHandleCreated) {

this->Invoke(gcnew Action<String^>(this, &MessageForm::updateContactsList), contactsList);

}

}

else {

std::vector<char> packet(msg.message, msg.message + msg.messageSize);

packetBuffer[msg.packetNumber] = packet;

if (packetBuffer.size() == msg.totalPackets) {

String^ selectedContact = getSelectedContact();

marshal\_context context;

if (selectedContact != nullptr) {

std::string contactStr = context.marshal\_as<std::string>(selectedContact);

if (contactStr == msg.from) {

std::vector<char> completeMessage;

for (const auto& packet : packetBuffer) {

completeMessage.insert(completeMessage.end(), packet.second.begin(), packet.second.end());

}

incomingFrom = gcnew String(msg.from);

incomingMessage = gcnew String(completeMessage.data(), 0, completeMessage.size());

if (this->IsHandleCreated) {

this->Invoke(gcnew MethodInvoker(this, &MessageForm::updateChat));

}

}

}

saveMessageToServer(packetBuffer, msg.from, msg.to);

saveMessageToServer(packetBuffer, msg.to, msg.from);

packetBuffer.clear();

}

}

}

}

}

void saveMessageToServer(const std::map<int, std::vector<char>>& packetBuffer, const char\* from, const char\* to) {

StructMessage msgTo;

strcpy\_s(msgTo.type, "full\_chat");

strcpy\_s(msgTo.from, to);

strcpy\_s(msgTo.to, from);

msgTo.delaySeconds = 0;

marshal\_context context;

std::string combinedString;

if (context.marshal\_as<std::string>(clientName) == to) {

combinedString = std::string(msgTo.to) + ": ";

}

else {

combinedString = "Вы: ";

}

strcpy\_s(msgTo.message, combinedString.c\_str());

msgTo.messageSize = combinedString.size();

send(clientSocket, reinterpret\_cast<char\*>(&msgTo), sizeof(msgTo), 0);

StructMessage msg;

strcpy\_s(msg.type, "full\_chat");

strcpy\_s(msg.from, to);

strcpy\_s(msg.to, from);

msg.delaySeconds = 0;

msg.totalPackets = packetBuffer.size();

int packetNumber = 0;

for (const auto& packet : packetBuffer) {

memcpy\_s(msg.message, sizeof(msg.message), packet.second.data(), packet.second.size());

msg.messageSize = packet.second.size();

msg.packetNumber = packetNumber++;

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

}

StructMessage msgToN;

strcpy\_s(msgToN.type, "full\_chat");

strcpy\_s(msgToN.from, to);

strcpy\_s(msgToN.to, from);

msgToN.delaySeconds = 0;

msgToN.message[0] = '\n';

msgToN.messageSize = 1;

send(clientSocket, reinterpret\_cast<char\*>(&msgToN), sizeof(msgToN), 0);

}

void appendChatHistory(String^ chatHistory) {

if (this->IsHandleCreated) {

textBoxChat->Clear();

array<String^>^ lines = chatHistory->Split('\n');

for each(String ^ line in lines) {

textBoxChat->AppendText(line + Environment::NewLine);

}

}

}

void updateChat() {

textBoxChat->AppendText(incomingFrom + ": " + incomingMessage + Environment::NewLine);

}

void updateContactsList(String^ contactsList) {

listBoxContacts->Items->Clear();

array<String^>^ contacts = contactsList->Split(',');

for each(String ^ contact in contacts) {

listBoxContacts->Items->Add(contact);

}

}

void listBoxContactsSelectedIndexChanged(Object^ sender, EventArgs^ e) {

if (listBoxContacts->SelectedItem != nullptr) {

currentContact = listBoxContacts->SelectedItem->ToString();

textBoxChat->Clear();

loadChatHistory(currentContact);

}

}

void loadChatHistory(String^ contact) {

StructMessage msg;

strcpy\_s(msg.type, "load\_chat");

marshal\_context context;

strcpy\_s(msg.from, context.marshal\_as<const char\*>(clientName));

strcpy\_s(msg.to, context.marshal\_as<const char\*>(contact));

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

}

void stopListening() {

cancellationSource->Cancel();

closesocket(clientSocket);

if (listenThread != nullptr) {

listenThread->Join();

}

}

void addContact(String^ contactName) {

marshal\_context context;

const char\* charContactName = context.marshal\_as<const char\*>(contactName);

size\_t maxSizeOfTo = sizeof(StructMessage::to);

if (strlen(charContactName) >= maxSizeOfTo) {

MessageBox::Show("Имя слишком длинное!");

return;

}

textBoxTo->Clear();

StructMessage msg;

strcpy\_s(msg.type, "add\_contact");

strcpy\_s(msg.from, context.marshal\_as<const char\*>(clientName));

strncpy\_s(msg.to, charContactName, maxSizeOfTo - 1);

msg.to[maxSizeOfTo - 1] = '\0';

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

}

void buttonAddContactClick(Object^ sender, EventArgs^ e) {

addContact(textBoxTo->Text);

}

void notifyServerAboutExit() {

StructMessage msg;

strcpy\_s(msg.type, "disconnect");

marshal\_context context;

const char\* charName = context.marshal\_as<const char\*>(clientName);

strncpy\_s(msg.from, charName, sizeof(msg.from) - 1);

msg.from[sizeof(msg.from) - 1] = '\0';

send(clientSocket, reinterpret\_cast<char\*>(&msg), sizeof(msg), 0);

}

protected:

void OnFormClosing(System::Windows::Forms::FormClosingEventArgs^ e) override {

notifyServerAboutExit();

stopListening();

Form::OnFormClosing(e);

}

};

}

# **Приложение Б**

(Обязательное)

UML-диаграммы

## **Приложение Б.1**

UML-диаграмма последовательностей протокола взаимодействия клиента и сервера

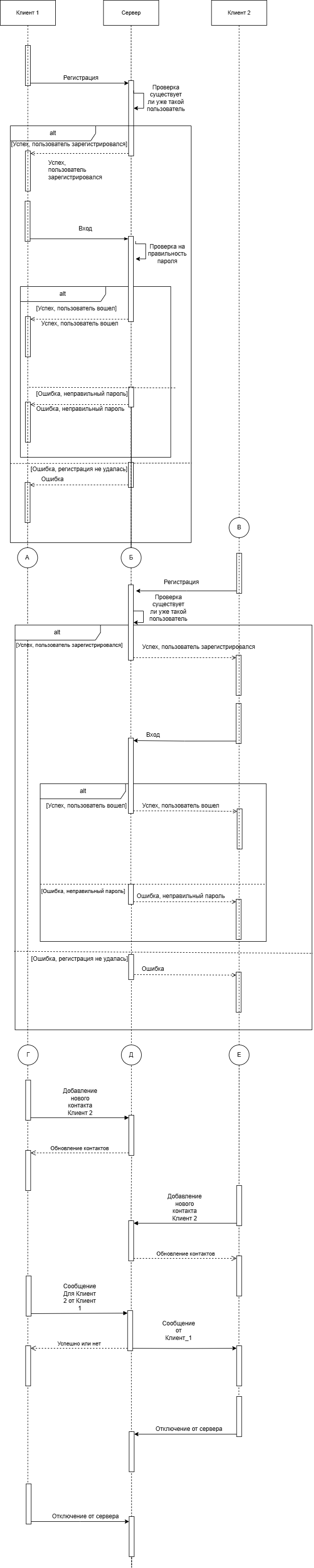


Рисунок 1 - UML-диаграмма последовательностей протокола взаимодействия клиента и сервера часть 1

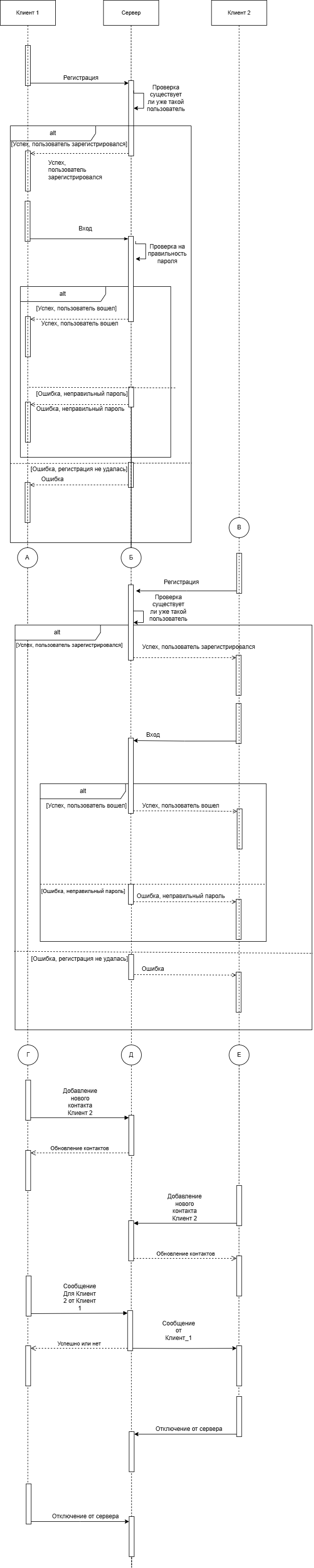


Рисунок 2 - UML-диаграмма последовательностей протокола взаимодействия клиента и сервера часть 2

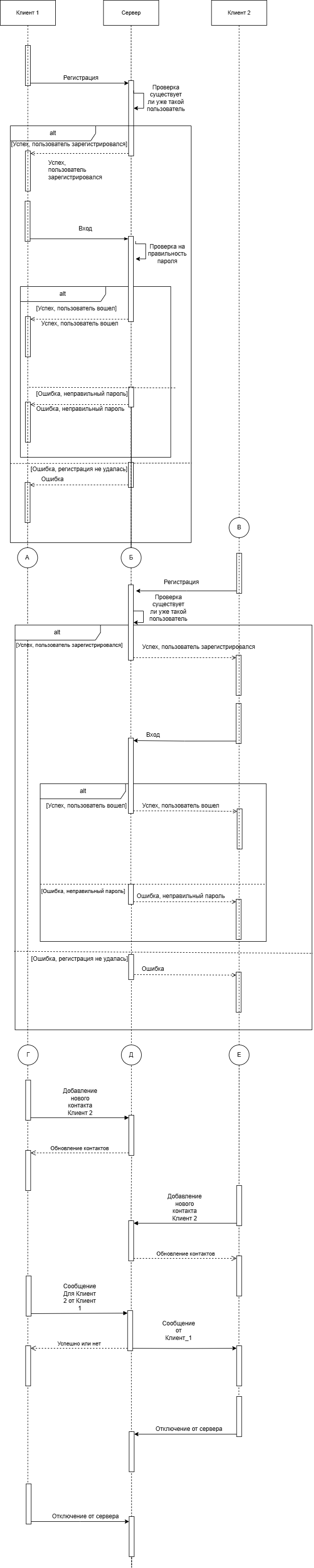
****

Рисунок 3 - UML-диаграмма последовательностей протокола взаимодействия клиента и сервера часть 3

## **Приложение Б.2**

UML-диаграмма вариантов использования приложения

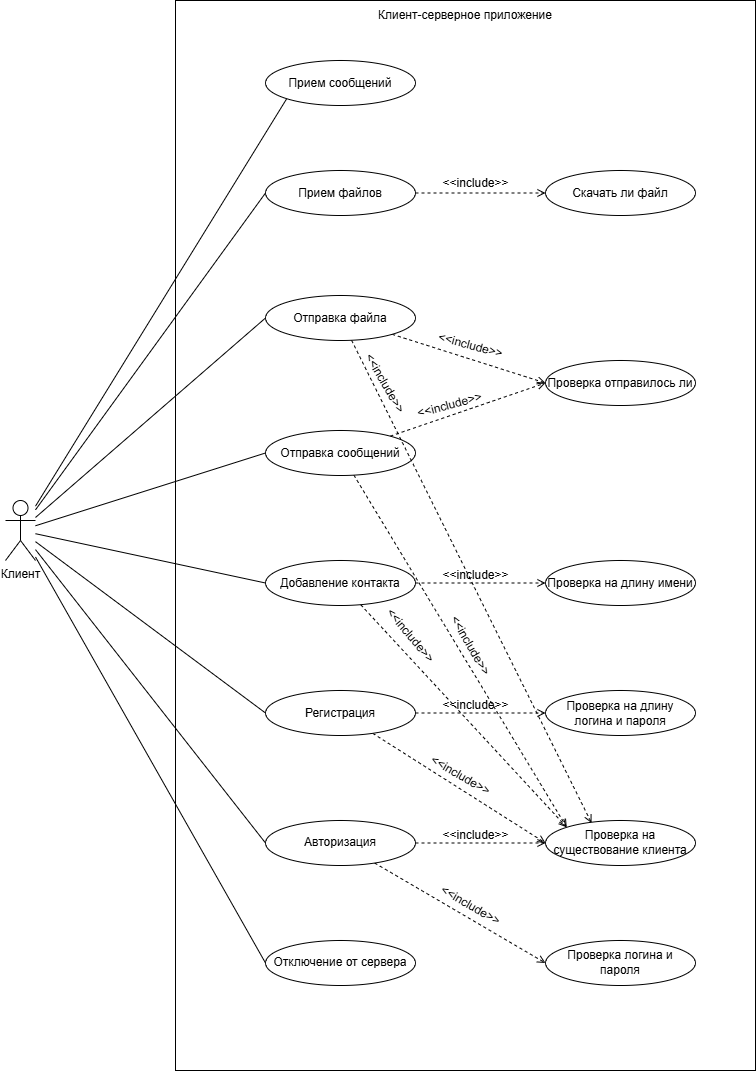


Рисунок 4 - UML-диаграмма вариантов использования приложения

## **Приложение Б.3**

UML-диаграмма развертывания



Рисунок 5 - UML-диаграмма развертывания

## **Приложение Б.4**

UML-диаграмма состояний клиента

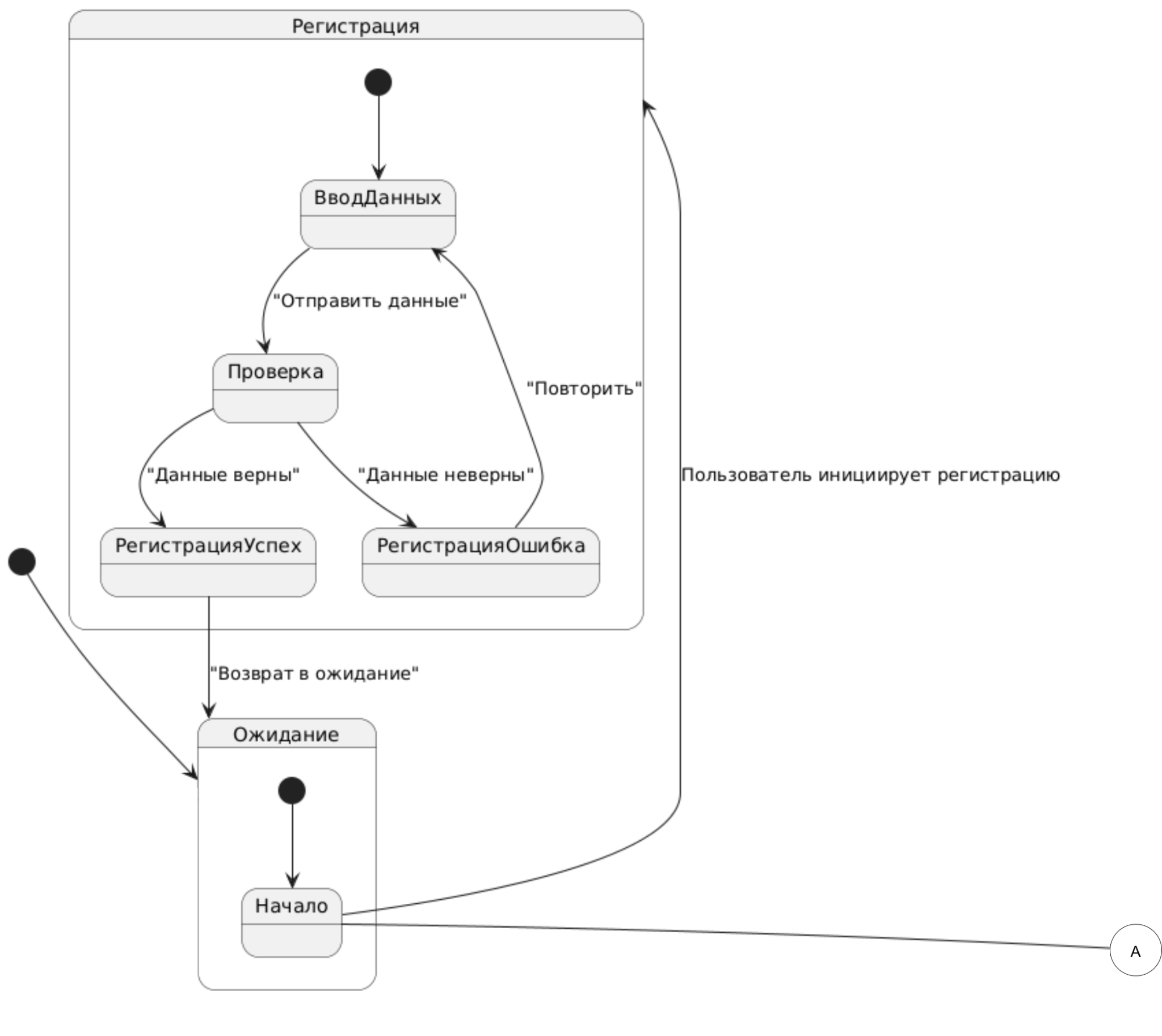


Рисунок 6 - UML-диаграмма состояний клиента часть 1

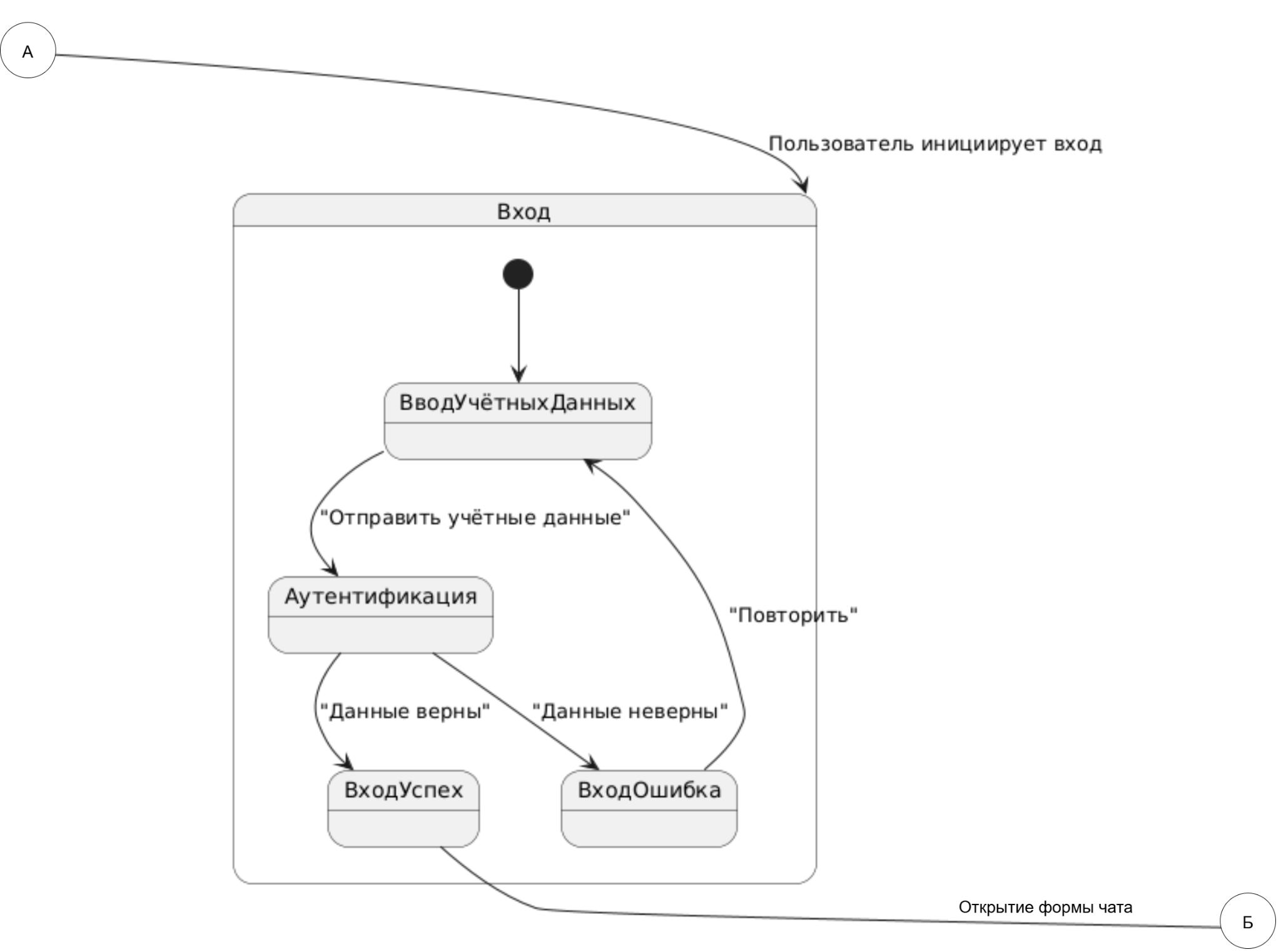


Рисунок 7 - UML-диаграмма состояний клиента часть 2

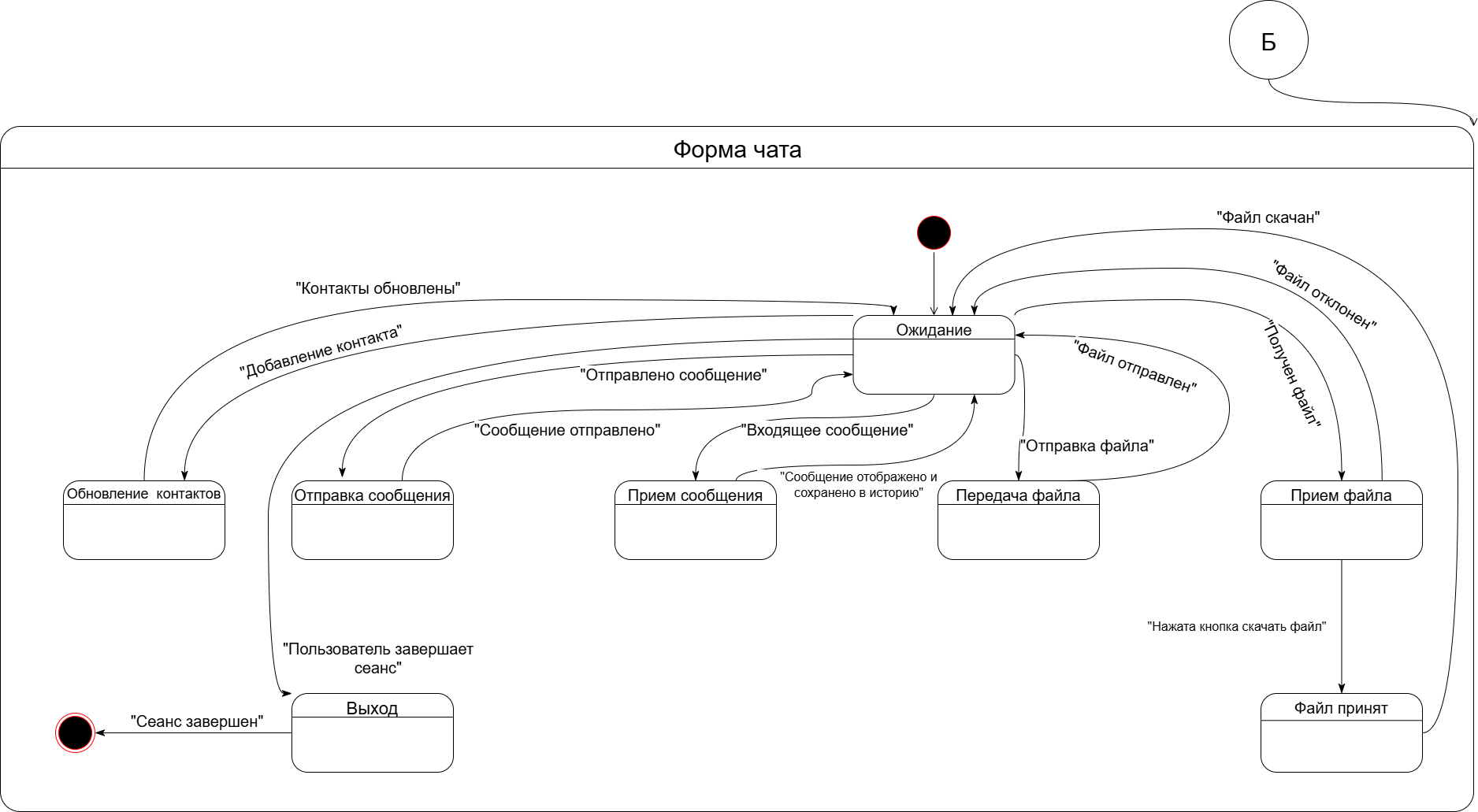


Рисунок 8 - UML-диаграмма состояний клиента часть 3

## **Приложение Б.5**

UML-диаграмма состояний сервера

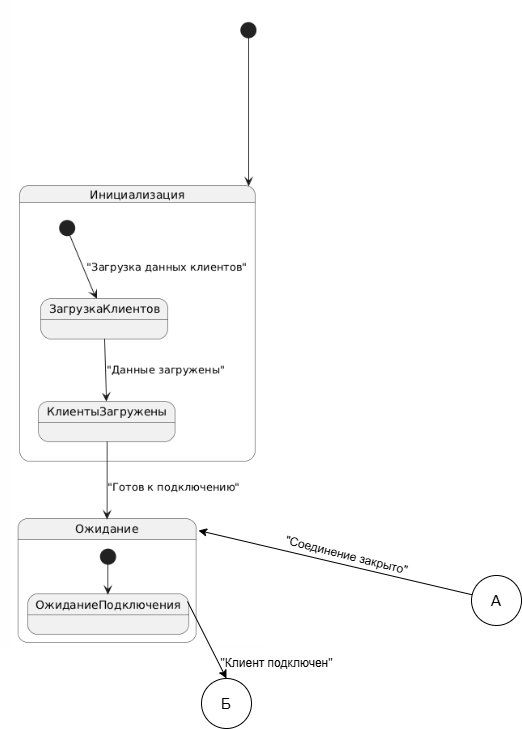


Рисунок 9 - UML-диаграмма состояний сервера часть 1

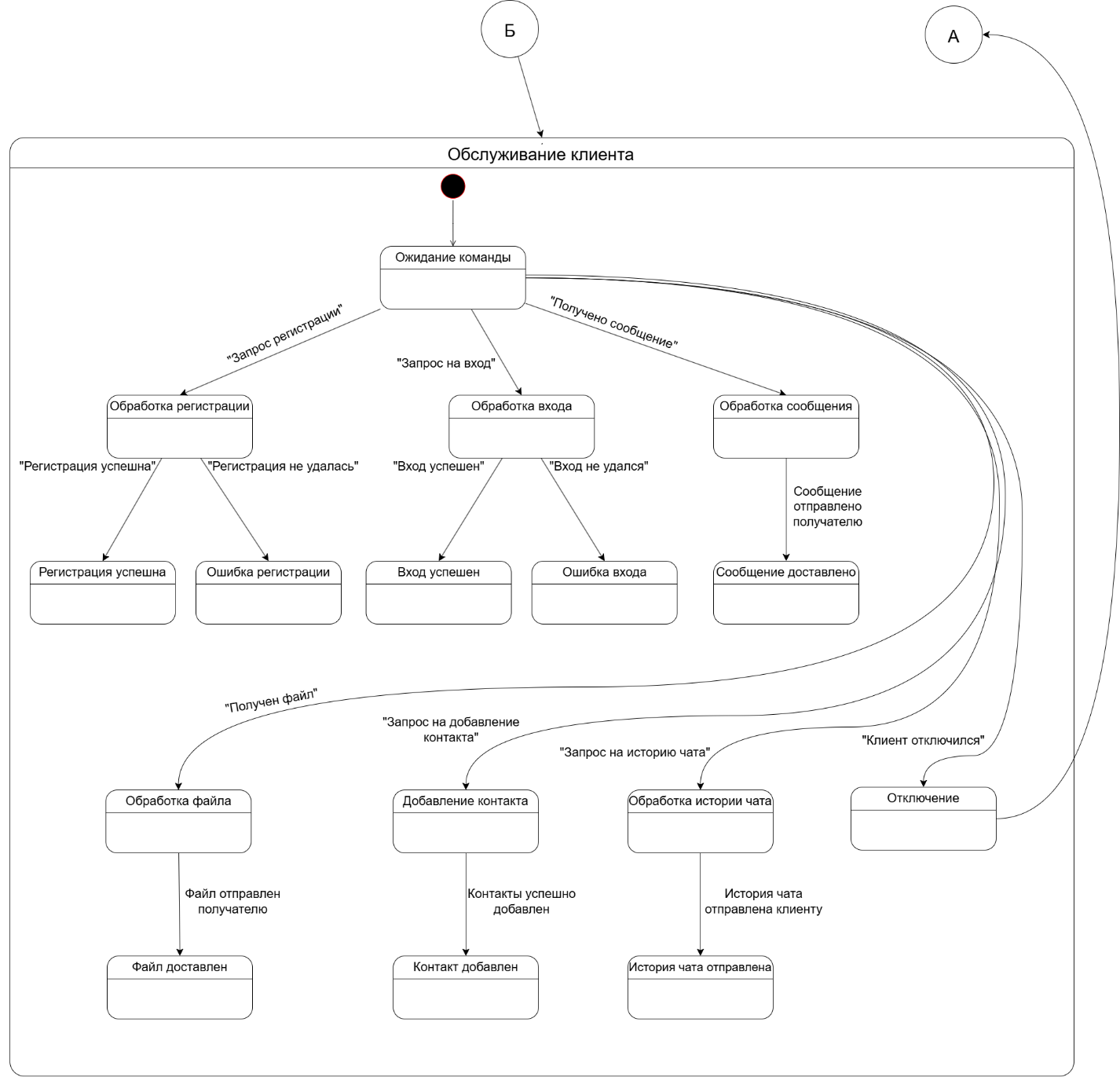


Рисунок 10 - UML-диаграмма состояний сервера часть 2