# 3. Проектно-конструкторская часть

## 

## 3.1 Разработка структуры приложения

Для разработки приложения использовалась среда Microsoft Visual Studio, включающая в себя широкий набор функциональных инструментов.

Ввод данных в приложении осуществляется через графический интерфейс с использованием клавиатуры. При этом вводимые данные могут быть на любом языке с использованием чисел и системных символов.

После входа в приложение пользователь должен ввести своё имя или псевдоним, в соответствующее текстовое поле, а также собственный порт и порт собеседника, после чего может начать общение в чате. Ввод обоих портов является обязательным, так как наличия неадресуемого пользователя в чате недопустимо.

Написав сообщение его нужно послать, при помощи специальной кнопки отправить. Следствием правильной работы приложения является отображение отправленного сообщения в окне диалога.

Общее назначение программного средства - общение в локальной сети по средствам отправки текстовых сообщений.

**3.2 Используемые функции Win32API**

* **wWinMain -** вызывается системой как начальная точка входа, для базирующейся на Windows, прикладной программы.

int WINAPI wWinMain(

HINSTANCE hInstance, // дескриптор текущего экземпляра окна

HINSTANCE hPrevInstance, // дескриптор предыдущего экземпляра окна

LPSTR lpCmdLine, // указатель на командную строку

int nCmdShow // показывает состояние окна

);

* **DefWindowProc** - вызывается оконной процедурой по умолчанию, чтобы обеспечить обработку по умолчанию любого сообщения окна, которые приложение не обрабатывает. Эта функция гарантирует то, что обрабатывается каждое сообщение.

DefWindowProc(

HWND hWnd, // Дескриптор процедуры, которая получает сообщение

UINT Msg, // Определяет сообщение

WPARAM wParam,// Определяет дополнительную информацию о сообщении

LPARAM lParam // Определяет дополнительную информацию о сообщении

);

* **CreateWindowEx** - создает перекрывающее, выпрыгивающее или дочернее окно с расширенным стилем.

CreateWindowEx(

DWORD dwExStyle,      // улучшенный стиль окна  
LPCTSTR lpClassName,  // указатель на зарегистрированное имя класса  
LPCTSTR lpWindowName, // указатель на имя окна  
DWORD dwStyle,        // стиль окна  
int x,                // горизонтальная начальная позиция окна  
int y,                // вертикальная начальная позиция окна  
int nWidth,           // ширина окна  
int nHeight,          // высота окна  
HWND hWndParent,      // дескриптор родительского или окна владельца  
HMENU hMenu,          // дескриптор меню или ID дочернего окна  
HINSTANCE hInstance,  // дескриптор экземпляра прикладной программы  
LPVOID lpParam        // указатель на данные создания окна

);

* **GetWindowText -** копирует текст заголовка определяемого окна (если окно имеет его) в буфер. Если заданное окно является органом управления, копируется его текст.

GetWindowText(

HWND hWnd, // дескриптор окна или элемента управления с текстом

LPTSTR lpString, // адрес буфера для текста

int nMaxCount // максимальное число символов для копирования

);

* **TranslateMessage -** переводит сообщения виртуальных клавиш в символьные сообщения. Символьные сообщения помещаются в очереди сообщений вызывающего потока для прочтения.

TranslateMessage(

const MSG\* lpMsg // указатель на структуру MSG

);

* **ShowWindow -** устанавливает состояние показа определяемого окна.

ShowWindow(

HWND hWnd, // дескриптор окна

int nCmdShow // состояние показа окна

);

* **RegisterClass -** регистрирует класс окна для последующего использования при вызове функции CreateWindow или CreateWindowEx

RegisterClass(

CONST WNDCLASS\* lpWndClass // указатель на структуру WNDCLASS

);

* **DispatchMessage -**  распределяет сообщение оконной процедуре. Обычно она используется, чтобы доставить сообщение, извлеченное функцией GetMessage.

DispatchMessage(

const MSG\* lpmsg // указатель на структуру MSG, которая содержит сообщение.

);

* **SendMessage -**  отправляет заданное сообщение окну или окнам. Функция вызывает оконную процедуру для заданного окна и не возвращает значение до тех пор, пока оконная процедура не обработает сообщение.

SendMessage(

HWND hWnd, // дескриптор окна, процедура принимает сообщение

UINT Msg, // определяет сообщение, которое будет отправлено

WPARAM wParam, // определяет дополнительную информацию

LPARAM lParam // определяет дополнительную информацию

);

* **WSAStartup -** вызывается для инициации использования WS2 \_32.dll.

WSAStartup(

WORD wVersionRequired, // указать версию интерфейса Windows Sockets

LPWSADATA lpWSAData // должен содержать указатель на структуру типа WSADATA

);

* **inet\_addr -** принимает аргумент типа char \*, представляющий собой указатель на нуль-терминированную строку, содержащую строковое представление ip-адреса.

inet\_addr(

const char \*cp);

* **htons** - преобразует u\_short из хоста в сетевой порядок байтов TCP / IP (который является прямым порядком байтов).

htons(

u\_short hostshort

);

* **bind** - связываем адрес IP с сокетом

bind(

SOCKET s, // дескриптор, идентифицирующий несвязанный сокет.

const sockaddr \*addr, // указатель на  sockaddr локального адреса

Int namelen // Длина в байтах значения, на которое указывает name

);

* **sendto -** отправляет данные на подключенный сокет.

sendto(

SOCKET s,

const char \*buf,

int len,

int flags

const sockaddr \*to,

int tolen

);

* **recvfrom -** получает данные из подключенного сокета или привязанного сокета без установления соединения.

recvfrom (

SOCKET s,

char \*buf,

int len,

int flags

);

## 3.3 Разработка архитектуры приложения

Архитектура приложения клиент - клиент (каждый клиент является одновременно клиентом и сервером).

Классическая архитектура одноранговой сети - тип, в которой все рабочие станции имеют равные возможности и права. Для решения задачи создается одноранговая (peer-to-peer) вычислительная среда, которая позволяет отдельным элементам сети взаимодействовать без помощи серверов. Каждый участвующий компьютер вносит свой вклад в виде файлов, дискового пространства, процессорного времени. Пользователи Сети могут оказывать друг другу услуги на основе кооперации без жесткой зависимости от централизованных служб.

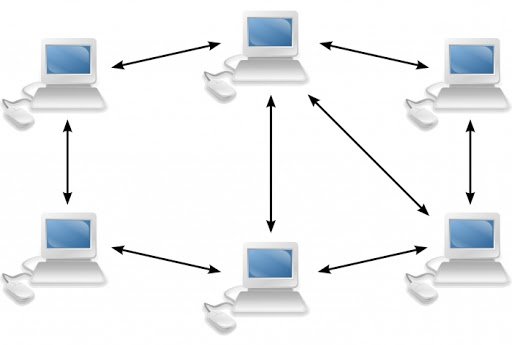


Рисунок.9 Обобщенная схема однорангового взаимодействия

UDP — User Datagram Protocol

UDP — очень быстрый протокол, поскольку в нем определен самый минимальный механизм, необходимый для передачи данных. Конечно, он имеет некоторые недостатки. Сообщения поступают в любом порядке, и то, которое отправлено первым, может быть получено последним.

Доставка сообщений UDP вовсе не гарантируется, сообщение может потеряться, и могут быть получены две копии одного и того же сообщения. Последний случай возникает, если для отправки сообщений в один адрес использовать два разных маршрута.

UDP не требует открывать соединение, и данные могут быть отправлены сразу же, как только они подготовлены. UDP не отправляет подтверждающие сообщения, поэтому данные могут быть получены или потеряны. Если при использовании UDP требуется надежная передача данных, ее следует реализовать в протоколе более высокого уровня.

## 3.4 Разработка и реализация алгоритмов приложения

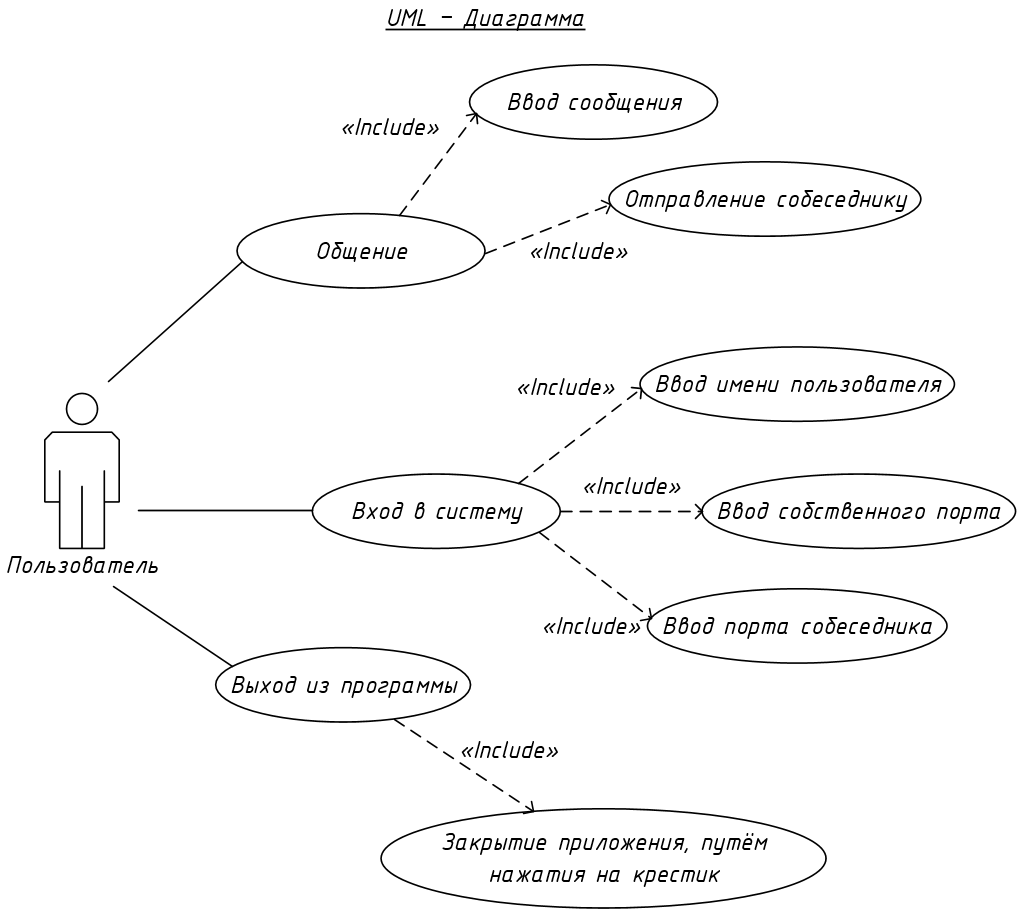


Рисунок.10 Сценарий использования (Use case) программы.

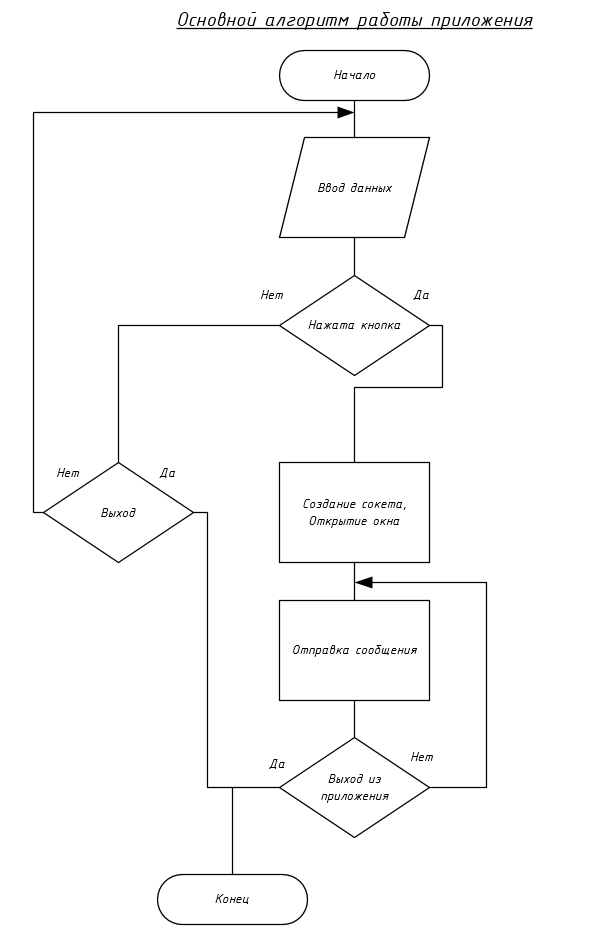


Рисунок.11 Основная блок-схема, описывающая работу приложения в целом.

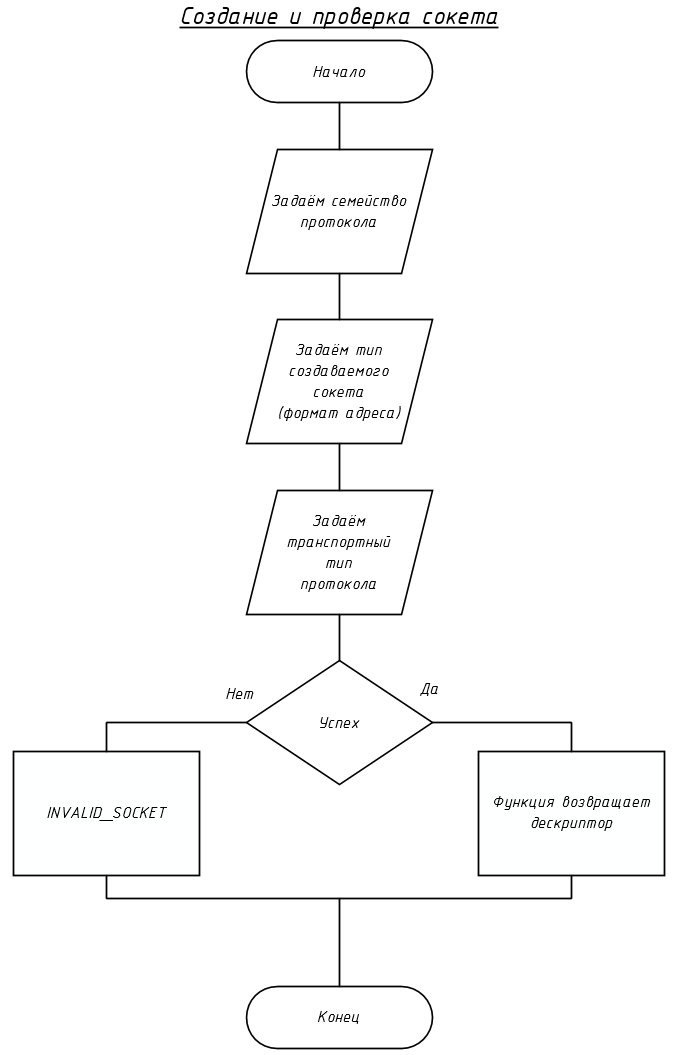


Рисунок.12 Создание и проверка сокета.

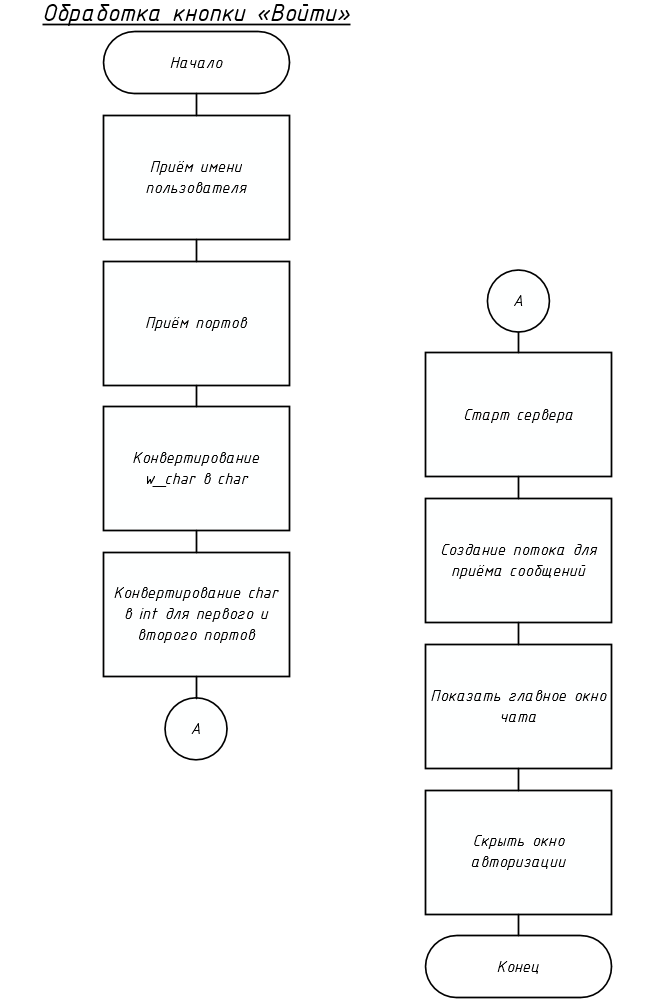


Рисунок.13 Обработка кнопки «Войти».

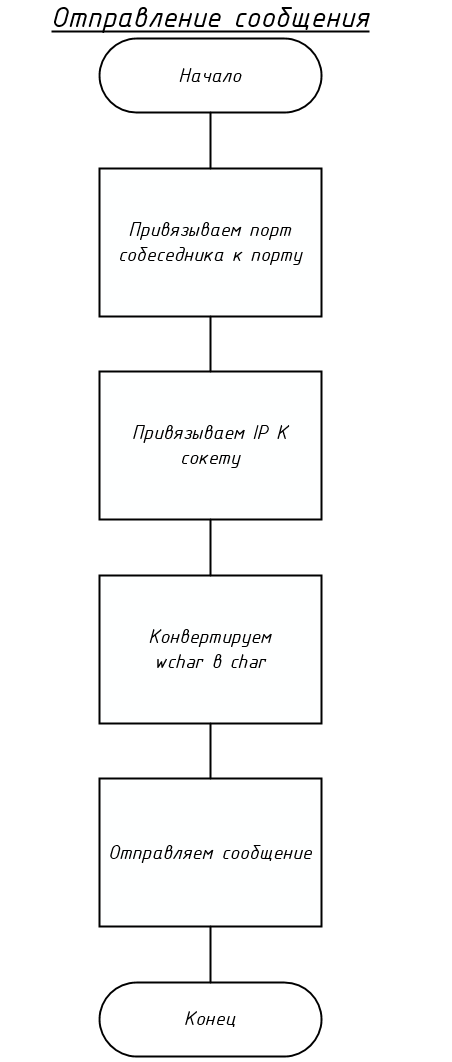


Рисунок.14 Процесс отправления сообщения.

## 3.5 Разработка интерфейса взаимодействия пользователя с системой

Окно программы должно состоять из трёх текстовых полей для ввода информации: в первое поле необходимо записать имя клиента, во второе поле записать собственный порт, в третье же поле порт собеседника. Помимо этого, должна быть кнопка «Войти», при нажатии на которую открывается второе, новое окно – чат, в котором и происходит общение между двумя пользователями.

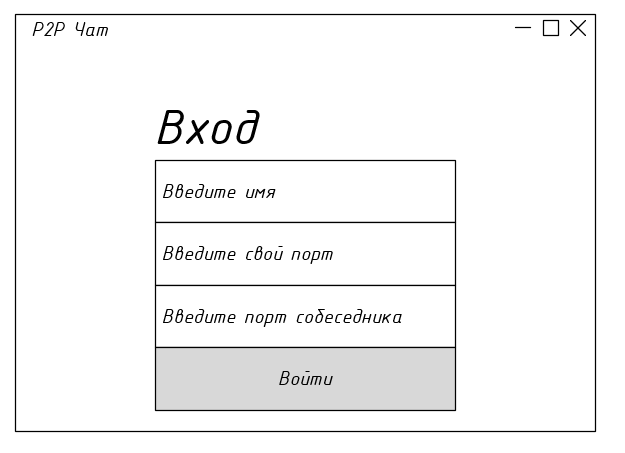


Рисунок.15 Макет окна входа.

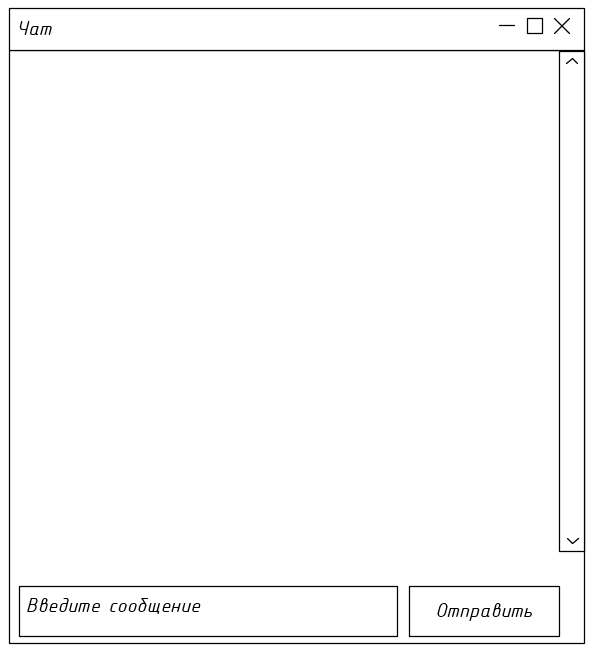


Рисунок.16 Макет окна чата.