# цель работы

Изучить программный интерфейс сетевых сокетов, получить навыки организации взаимодействия программ при помощи протоколов Internet и разработки прикладных сетевых сервисов.

# ХОД РАБРты

## Сетевой порядок байт

Для корректного перевода требуемых полей в сетевой порядок байт была использована функция “htons”, а после копирование при помощи “memcopy”. При попытке самостоятельной реализации “htons”, и побайтовом копировании через присваивание, корректный перевод в сетевой порядок байт не происходил из-за оптимизации компилятора и особенности архитектуры процессора, на системах “big-endian” переводить байты в прямой порядок не было нужды, и функция “htons” в таких системах представляет из себя пустой макрос.

По итогу реализованный TCP сервер будет работать как на системах с прямым порядком байт (BE), так и на системах с обратным порядком байт (LE).

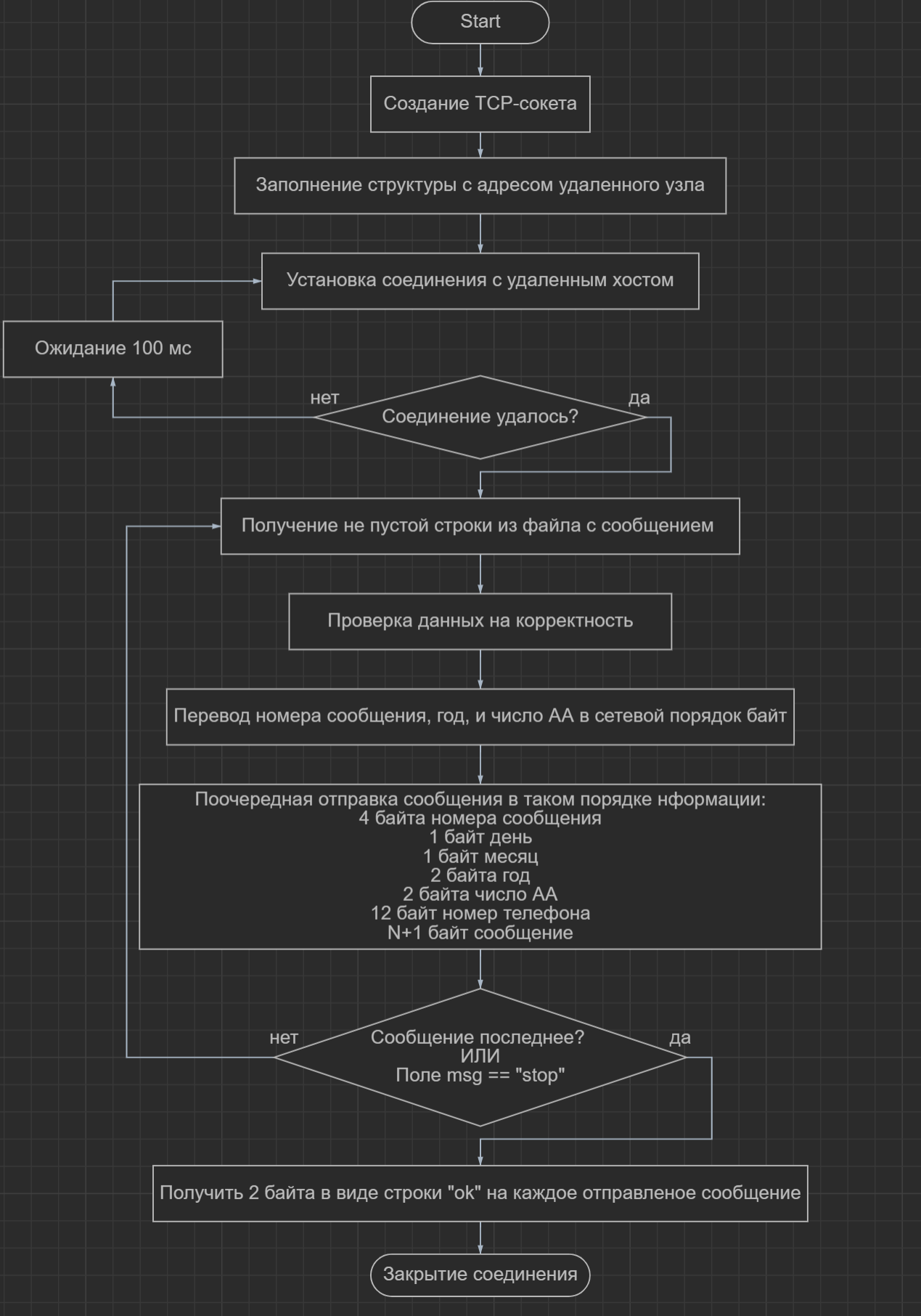
## Различия TCP и UDP

Основным различием этих сетевых моделей заключается в том, что TCP перед тем, как передавать данные устанавливает стабильное соединение и уже после этого по определенному протоколу начинается передача данных. Когда в UDP данные посылаются пакетами, а именно дейтаграммами, и такая модель имеет не стабильное соединение из-за чего некоторые пакеты могут не дойти до сервера и их придется отправлять повторно.

## TCP

### TCP client

В ходе лабораторной работы был разработан ТСП клиент, который работает по следующему алгоритму (блок схема 1).



Блок схема 1 – Алгоритм работы TCP клиента

Для хранения сообщений была использована структура “data” (рисунок 1)

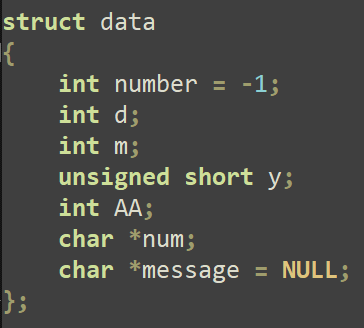
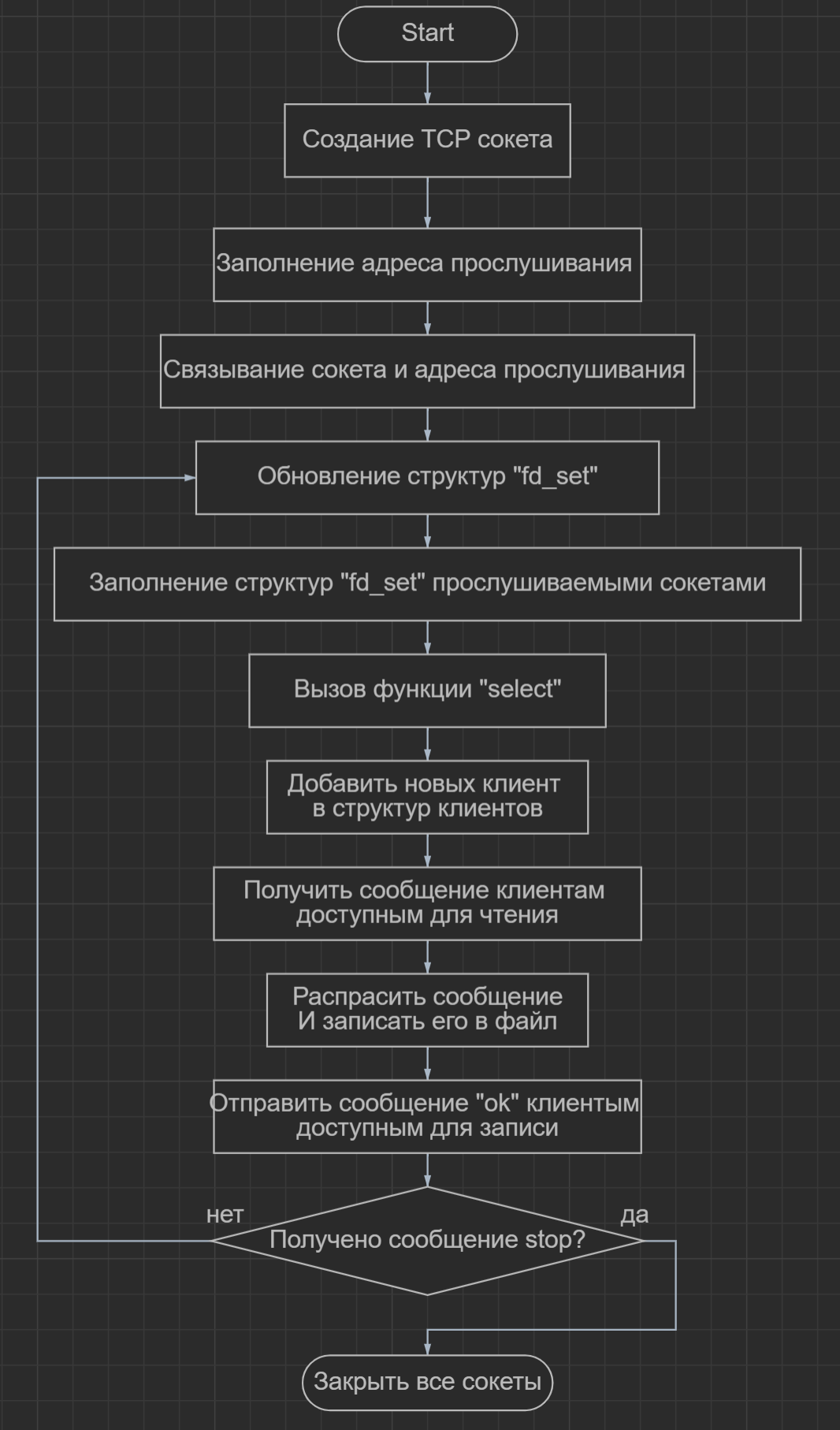


Рисунок 1 – Структура для хранения

### TCP server

В ходе лабораторной работы был реализован TCP сервер с параллельным обслуживанием клиентов, при помощи функции “select”, реализованный сервер работает по следующему алгоритму (блок схема 2).



Блок схема 2 – Алгоритм работы TCP сервера

#### Алгоритм при работе с функцией “select”

Данный метод параллельного обслуживания состоит из двух частей структура “*fd\_set”* и сама функция “select”. Первым делом структура “*fd\_set*” заполняется дескрипторами сокетов, которые могли стать доступны для чтения или отправки данных. После формирования структуры “fd\_set”, вызывается функция “select” с пятью аргументами: максимальное значение сокета; структура “fs\_set” с сокетами, которые могли стать доступны для чтения данных; вторая — для записи данных; третья — дескрипторы тех сокетов, в которых могла произойти ошибка; и время ожидания, которое функция будет ожидать до тайм аута

#### Хранение клиентов в памяти

Для хранения клиентов был использован односвязный список (рисунок 2), который содержит информации о сокете клиентов, ip, port, режим работы клиента (получение или отправка сообщений), и количество уже принятых сообщений.

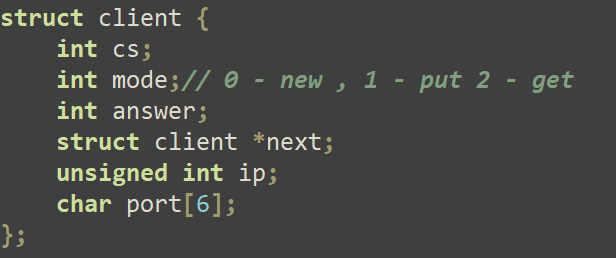
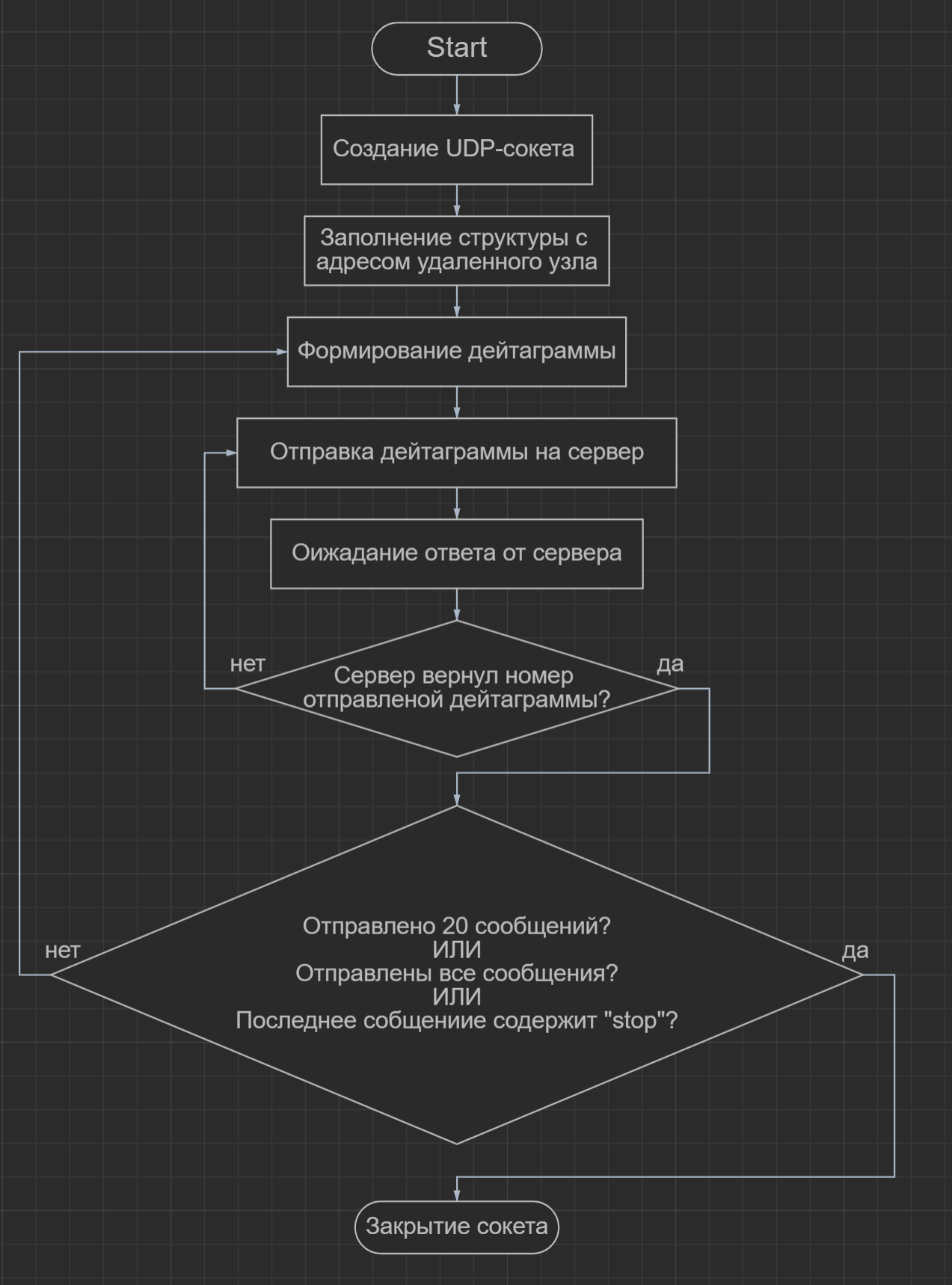


Рисунок 2 – Структура клиентов на TCP сервере

## UDP

### UDP client

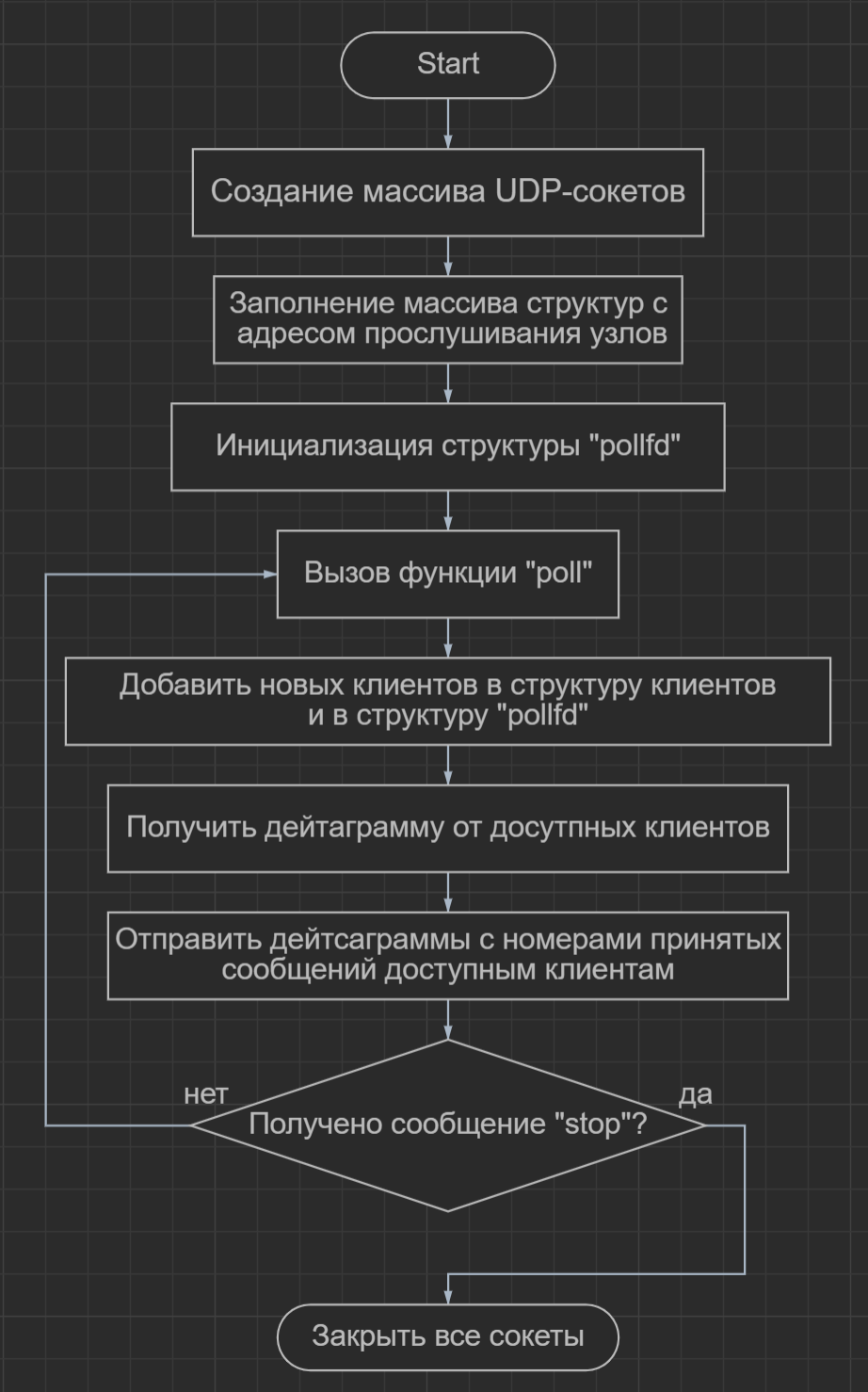
В ходе лабораторной работы был разработан UDP клиент, который работает по следующему алгоритму (блок схема 3).



Блок схема 3 – Алгоритм работы UDP клиента

### UDP server

В ходе лабораторной работы был реализован UDP сервер с параллельным обслуживанием клиентов, при помощи функции “poll”, реализованный сервер работает по следующему алгоритму (блок схема 4). Так же сервер прослушивает не один порт, а диапазон.



Блок схема 4 – Алгоритм работы UDP сервера

#### Алгоритм при работе с функцией “poll”

Алгоритм работы очень схож с алгоритмом “select”. Отличием является только то что, используемую структуру “pollfd”, не нужно инициализировать перед каждым вызовом функции “poll”, достаточно сделать это один раз и по мере добавления новых пользователей, добавлять их в эту структуру.

#### Хранение клиентов в памяти

Для хранения клиентов был использован односвязный список (рисунок 3), который содержит информации о клиентах: их id, имеющий вид ip:port, номера принятых сообщений, количество ожидающих дейтаграмм.

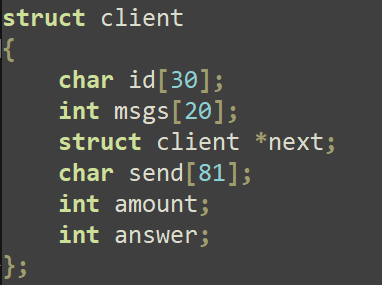


Рисунок 2 – Структура клиентов на UDP сервере