1. 2Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 2**

1. «МЕХАНИЗМЫ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ»
2. по дисциплине «Операционные системы»
3. Выполнил
4. студент гр. 4851003/00002 Вологдин М.В.

<*подпись*>

1. Преподаватель Крундышев В. М

<*подпись*>

1. Санкт-Петербург
2. 2022
3. **Цель работы**

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

1. **Теоретические сведения**

Transmission Control Protocol (TCP, протокол управления передачей) — один из основных протоколов передачи данных интернета. Предназначен для управления передачей данных интернета. Пакеты в TCP называются сегментами.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым целостность передаваемых данных и уведомление отправителя о результатах передачи.

UDP (User Datagram Protocol — протокол пользовательских датаграмм) — один из ключевых элементов набора сетевых протоколов для Интернета. С UDP компьютерные приложения могут посылать сообщения (датаграммы) другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных.

UDP использует простую модель передачи, без явных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа, но гарантируется, что если они придут, то в целостном состоянии. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении.

Select – универсальный механизм параллельного обслуживания клиентов. Для использования этого механизма вызывающая программа должна сохранить дескрипторы сокетов, на которых могли произойти события, в некоторую структуру fd\_set и передать ее в функцию select. При этом функция отметит среди всех переданных сокетов только те, на которых произошли события. В функцию передается 3 структуры fd\_set: первая содержит дескрипторы сокетов, которые могли стать доступны для чтения данных, вторая — для записи данных, третья — дескрипторы тех сокетов, при работе с которыми могли возникнуть ошибки. Первый параметр функции select содержит значение максимального дескриптора сокета + 1. Последний параметр функции — максимальное время ожидания события: передается в виде секунд и микросекунд.

Poll – механизм параллельного обслуживания клиентов, во многом напоминающий select. Отличие состоит в способе упаковки дескрипторов сокетов в соответствующую структуру данных. Дескриптор каждого сокета, события которого интересуют программу, сохраняется в структуре struct pollfd, которая в свою очередь содержит поля: fd – дескриптор сокета, events и revents – запрашиваемые и возникшие события, которые состоят из битовых масок. Возможные события – POLLIN ( чтение ), POLLOUT ( отправка ), POLLERR ( ошибка ) и POLLHUP ( соединение разорвано ).

1. **Ход работы**

1)TCP-client

Таблица 1 – Функции TCP-client

|  |  |
| --- | --- |
| Функция | Описание |
| init() | Инициализация сокетов |
| deinit() | Деинициализация сокетов |
| sock\_err() | Вывод ошибки в консоль |
| str\_to\_int(char\* str) | Перевод строки str в числовой формат |
| send\_info(int s, int i) | Формирование и отправление сообщения с номером i по сокету с дескриптором s |
| read\_file(char\* str) | Чтение сообщений из входного файла |
| read\_addr(char\* ip) | Отделение порта от IP-адреса |
| recv\_ok(int s) | Получение сообщения “ok” от сервера |
| receive\_mes(int s, int i, int port, char\* file) | Получение содержимого файла msg.txt от сервера, в случае входной команды get(Доп. задание) |
| int recv(int sockfd, void \*buf, int len, int flags) | Прием данных от сервера. |
| int send(int sockfd, const void \*buf, int len, int flags); | Отправка данных от клиента. |

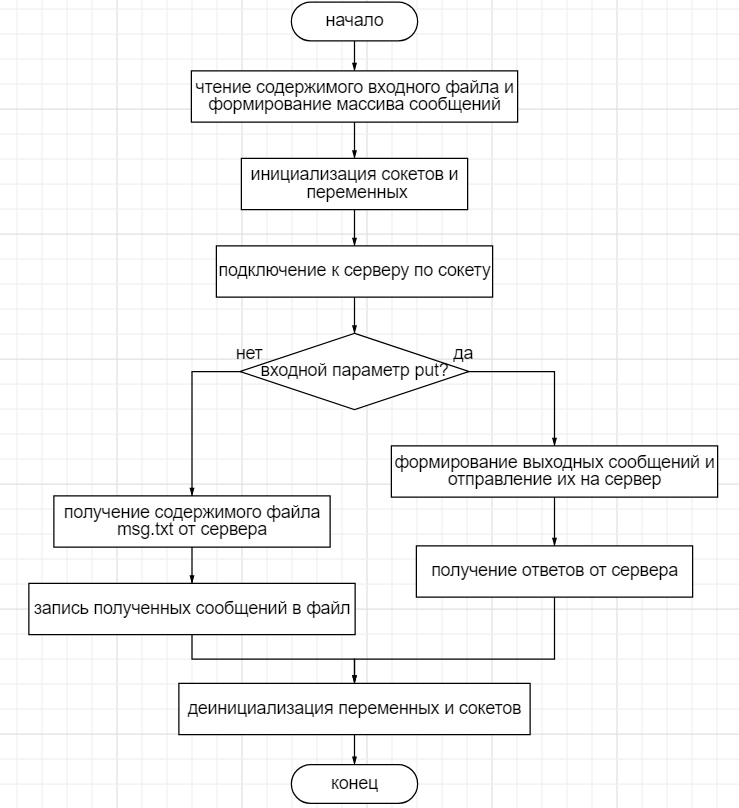


Рисунок 1 – Блок-схема работы tcp-client

2)TCP-server

Таблица 2 – Функции TCP-server

|  |  |
| --- | --- |
| Функция | Описание |
| init() | Инициализация сокетов |
| deinit() | Деинициализация сокетов |
| sock\_err() | Вывод ошибки в консоль |
| str\_to\_int(char\* str) | Перевод строки str в числовой формат |
| send\_info(int s, int i) | Формирование и отправление сообщения с номером i по сокету с дескриптором s |
| read\_file(char\* str) | Чтение сообщений из входного файла |
| set\_non\_block\_mode(int s) | Перевод сокета в неблокирующий режим |
| receive\_mes(int s, int ip, int port) | Получение содержимого файла msg.txt от сервера, в случае входной команды get(Доп. задание) |
| int recv(int sockfd, void \*buf, int len, int flags) | Прием данных от сервера. |
| int send(int sockfd, const void \*buf, int len, int flags); | Отправка данных от клиента. |

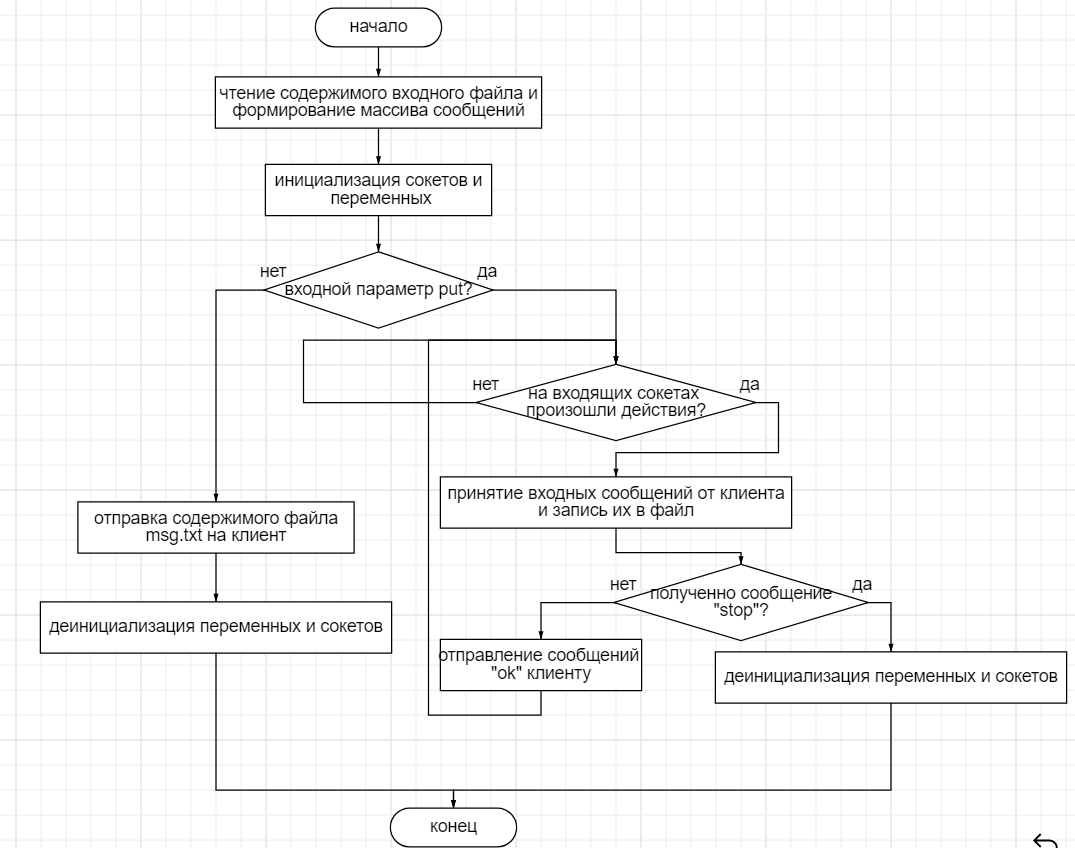




Рисунок 2 – Блок-схема работы tcp-server

3)UDP-client

Таблица 3 – Функции UDP-client

|  |  |
| --- | --- |
| Функция | Описание |
| init() | Инициализация сокетов |
| deinit() | Деинициализация сокетов |
| sock\_err() | Вывод ошибки в консоль |
| str\_to\_int(char\* str) | Перевод строки str в числовой формат |
| send\_info(int s, int i, struct sockaddr\_in addr) | Формирование и отправление сообщения с номером i по сокету с дескриптором s и адресу addr |
| read\_file(char\* str) | Чтение сообщений из входного файла |
| recv\_info(int s) | Получение информации о принятых сообщениях сервером |
| int recvfrom(int s, void \*buf, int len, int flags,  struct sockaddr \*from, int \*fromlen) | Прием данных от сервера. |
| int sendto(int sockfd, const void \*buf, int len, int flags,  const struct sockaddr \*dest\_addr, int addrlen) | Отправка данных от клиента. |

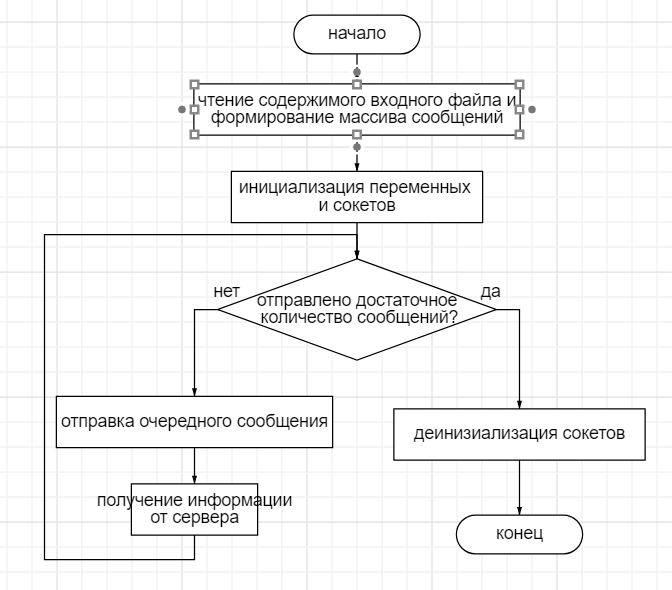


Рисунок 3 – Блок-схема работы UDP-client

4)UDP-server

Таблица 4 – Функции UDP-server

|  |  |
| --- | --- |
| Функция | Описание |
| init() | Инициализация сокетов |
| deinit() | Деинициализация сокетов |
| sock\_err() | Вывод ошибки в консоль |
| str\_to\_int(char\* str) | Перевод строки str в числовой формат |
| send\_resp(int s, int id) | Отправление информации о пользователе с индексом id по сокету s |
|  |  |
| recv\_info(int s) | Получение информации о принятых сообщениях сервером |
| int recvfrom(int s, void \*buf, int len, int flags,  struct sockaddr \*from, int \*fromlen) | Прием данных от сервера. |
| int sendto(int sockfd, const void \*buf, int len, int flags,  const struct sockaddr \*dest\_addr, int addrlen) | Отправка данных от клиента. |

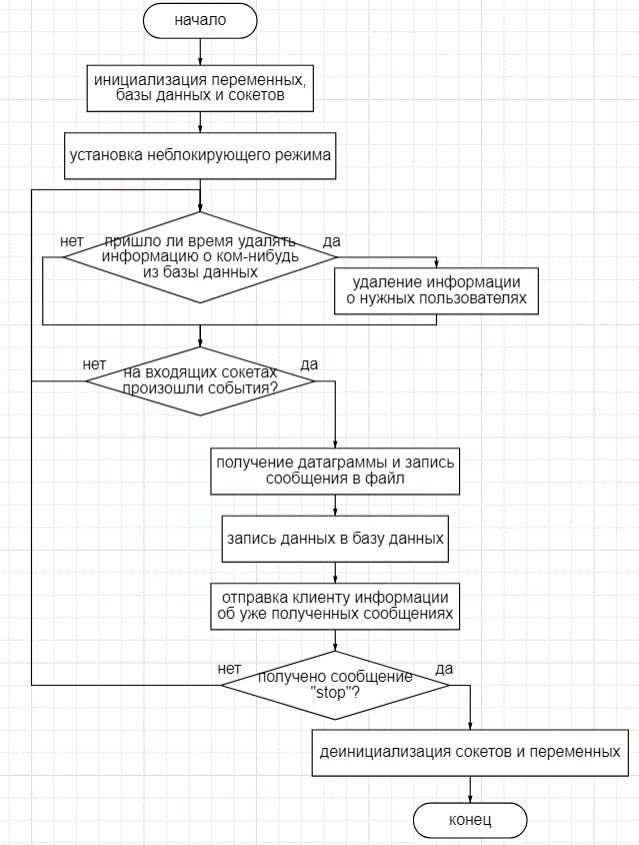


Рисунок 4 – Блок-схема работы UDP-server

1. **Результаты тестирования**

В результате проверки написанных программ были пройдены тесты:

TCP-client – не прошли тесты 6, 7, 9, 10, 11 из-за слишком большого размера входных сообщений, а также из-за большого количества самих сообщений

TCP-server – не прошли тесты из-за большого количества пользователей, а также из-за большого размера посылаемого сообщения

UDP-client – не прошли тесты 14, 15 из-за большого количества входных сообщений

Исправить это можно с помощью увеличения максимального количества хранимых в массиве сообщений, увеличения размера контейнера, который хранит сами сообщения, а также увеличения массива, хранящего данные клиентов

1. **Выводы**

В результате работы было смоделировано два типа интернет взаимодействия: по TCP и UDP протоколу. Для каждого типа соединения были написаны две программы – клиент и сервер. В ходе работы было большое количество трудностей. Например, в случае tcp, сложно было настроить правильное кодирование и корректную отправку сообщений, а в случае udp, со стороны сервера было трудно понять по какому ip-адресу нужно отправлять ответное сообщение.