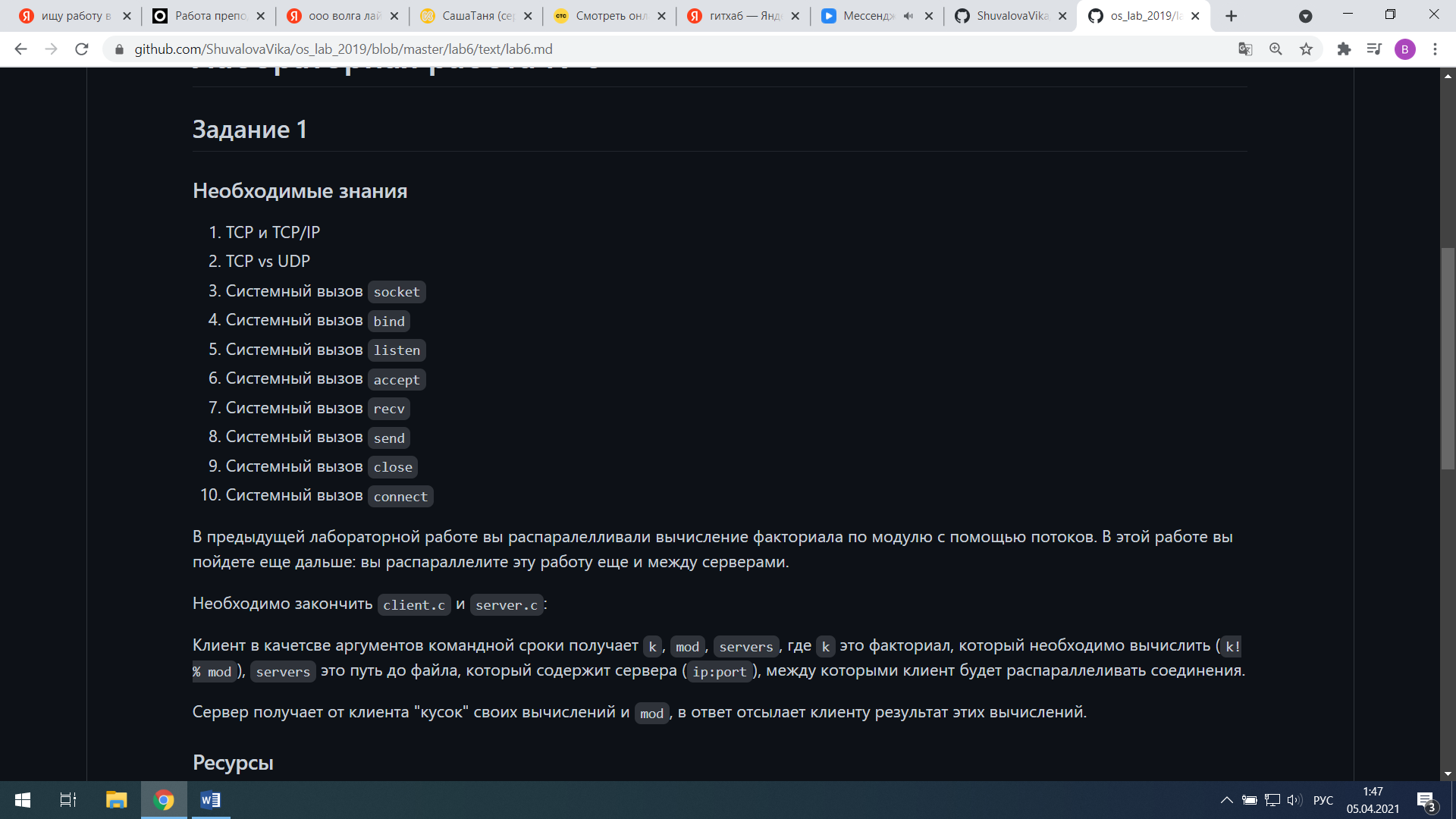
**Лабораторная работа № 6.**

Немецков Михаил, ПМ-31

**Задание 1.**



**TCP/IP** — [сетевая модель](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи).

**Уровни**

Прикладной

Хост-Хост

Межсетевой

Доступ к сети

Наборы правил, решающих задачу по передаче данных, составляют стек [протоколов передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), на которых базируется [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). Название TCP/IP происходит из двух важнейших протоколов семейства — [Transmission Control Protocol](https://ru.wikipedia.org/wiki/Transmission_Control_Protocol) (TCP) и [Internet Protocol](https://ru.wikipedia.org/wiki/Internet_Protocol) (IP).

**TCP** – протокол, обеспечивающий сервис, ориентированный на соединение, для пары взаимодействующих процессов, и включающий надежность, контроль трафика и исправление.

**IP** – это ненадежный, максимально обеспеченный, датаграммный пакетный протокол

Его **функции**

1. Определяет основную единицу передачи данных в интерсети.
2. Выполняет функцию маршрутизации
3. Включает правила ненадежной доставки, которые определяют, как маршрутизаторы должны обрабатывать пакеты, и при каких условиях можно уничтожать пакет
4. IP использует IP-адреса, состоящие из двух частей: адреса сети и адреса узла в сети
5. IP не ожидает от нижележащих протоколов ничего кроме возможности доставки пакетов к адресуемому узлу
6. не добавляет надежности
7. IP-пакеты могут потеряться, поменять порядок следования
8. не исправляет ошибки
9. не выполняет контроль трафика

В архитектуре TCP/IP определены два протокола уровня Хост-Хост

**Transmission Control Protocol (TCP)**

**User Datagram Protocol (UDP)**

UDP – ненадежный датаграммный протокол

1. Обеспечивает прикладным программам возможность посылать данные другим

программам с минимальными накладными расходами

1. Не добавляет надежности нижележащим уровням
2. Не выполняет контроль трафика
3. Приложения, требующие надежной доставки

потоков данных, должны использовать TCP

IP-адрес представляет собой 32-битное. Обычно адрес разбивают на 4 байта и записывают в виде 4-х чисел от 0 до 255, перечисленных через точку:

192.168.0.1

Для решения задачи маршрутизации адрес узла должен состоять из 2 частей: адрес сети и адрес узла в сети. В IP-адресе адрес сети размещается в старших битах, адрес узла в сети – в младших

**Сокет** – объект межпроцессного взаимодействия, обеспечивающий прием и передачу данных для процесса.

**Создание сокета**

int socket(int domain, int type, int protocol);

domain – коммуникационный домен

type – тип сокета

protocol – протокол транспортного уровня, если значение параметра равно 0, используется протокол по умолчанию для данного типа сокета и коммуникационного домена.

Возвращаемое значение – дескриптор сокета.

Домены -абстракции, которые подразумевают конкретную структуру адресации и множество протоколов, которое определяет различные типы сокетов внутри домена.

В Internet домене сокет - это комбинация IP адреса и номера порта, которая однозначно определяет отдельный сетевой процесс во всей глобальной сети Internet.

Функция socket создает конечную точку для коммуникаций и возвращает файловый дескриптор, ссылающийся на сокет, или -1 в случае ошибки. Данный дескриптор используется в дальнейшем для установления связи.

**Назначение адреса сокету**

int bind(int s, const struct sockaddr \*name, int namelen);

s – дескриптор сокета

name – адрес буфера, содержащего адрес сокета. Адрес представляет собой структуру struct sockaddr\_in

namelen – длина структуры, содержащей адрес

**Системный вызов****listen** используется сервером, ориентированным на установление связи путем виртуального соединения, для перевода сокета в пассивный режим и установления глубины очереди для соединений.

int listen(int sockd, int backlog);

Параметр sockd является дескриптором созданного ранее сокета, который должен быть переведен в пассивный режим, т. е. значением, которое вернул системный вызов socket() . Параметр backlog определяет максимальный размер очередей для сокетов, находящихся в состояниях полностью и не полностью установленных соединений.

**Системный вызов****accept** используется сервером, ориентированным на установление связи путем виртуального соединения, для приема полностью установленного соединения.

int accept(int sockd,

struct sockaddr \*cliaddr,

int \*clilen);

Параметр clilen содержит указатель на целую переменную, которая после возвращения из вызова будет содержать фактическую длину адреса клиента.

**Передача/прием**

ssize\_t recvfrom(int s, void \*buffer, size\_t length, int flags, struct

sockaddr \*address, socklen\_t \*address\_len);

**Системный вызов send()** используются для пересылки сообщений в другой сокет.

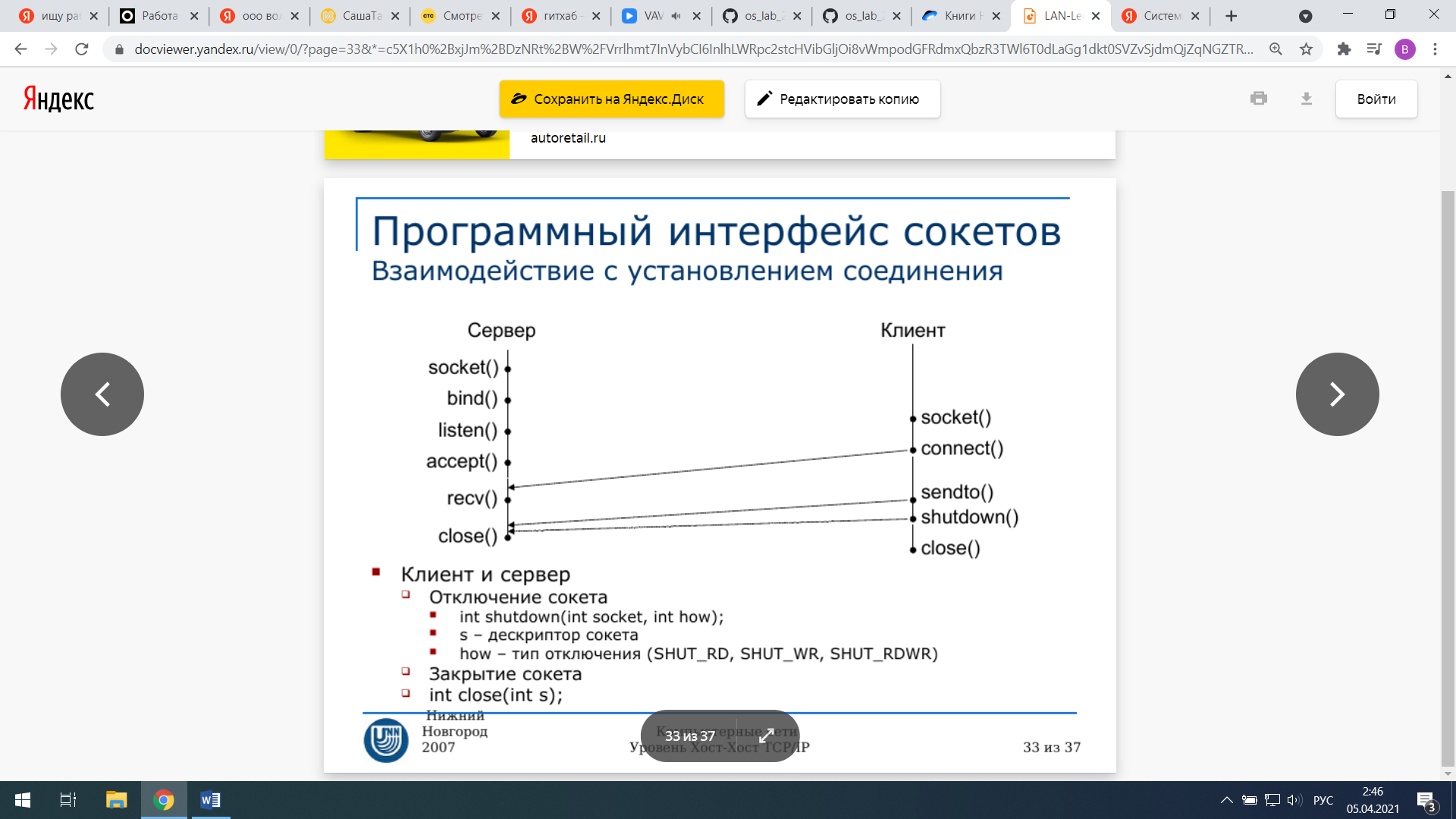
Вызов send() можно использовать, только если сокет находится в состоянии соединения (то есть известен получатель).

**Закрытие сокета**

int close(int s);

s – дескриптор сокета

**Системный вызов****connect** служит для организации связи клиента с сервером.



См. Гитхаб (код программы для распараллеливания)

**Результат:**

Сервер:

~/lab6/src$ ./server.exe --port 2555 --tnum 4

Server listening at 2555

Receive: 4 10 2

Total: 0

Клиет:

~/lab6/src$ ./client.exe --k 10 --mod 2 --servers servers.txt

answer: 0

Сервер:

~/lab6/src$ ./server.exe --port 2555 --tnum 4

Server listening at 2555

Receive: 4 2 3

Total: 2

Клиет:

~/lab6/src$ ./client.exe --k 2 --mod 3 --servers servers.txt

answer: 2

**Задание 2.**

Создать makefile для программ клиента и сервера

См.Гитхаб

**Задание 3.**

Найти дублирующийся код в двух приложениях и вынести его в библиотеку. Добавить изменения в makefile.

На гитхабе теперь все не в одном файле, а в разных. То, что не использовалось в server.c или в client.c вынесли в файлы разные, чтобы одно не мешалшось другому.