**Лабораторная работа №6**

**Задание 1**

**Необходимые знания**

1.TCP и TCP/IP.

-Протокол управления передачей и межсетевой протокол (TCP/IP) - это стандартный набор протоколов, разработанный в конце 1970-х Агентством передовых оборонных исследовательских проектов (DARPA) как средство взаимодействия между различными типами компьютеров и компьютерными сетями. TCP/IP является движущей силой интернета и поэтому является самым популярным стеком сетевых протоколов на Земле.

-Два компонента протокола TCP/IP представляют разные аспекты компьютерного взаимодействия. Межсетевой протокол («IP» в TCP/IP) - это протокол без установления соединения, который предоставляет только маршрутизацию пакетов, используя IP пакет (datagram) как основной блок сетевой информации. IP пакет содержит заголовок с последующим сообщением. Протокол управления передачей («TCP» в TCP/IP) позволяет сетевым хостам устанавливать соединения, которые могут использоваться для передачи потоков данных. TCP также гарантирует, что данные между подключениями будут доставлены и что они придут на сетевой хост в том же порядке, как они были посланы с другого.

-Настройка протокола TCP/IP состоит из нескольких элементов, которые должны быть установлены через редактирование конфигурационных файлов, или использования таких решений как сервер DHCP, который будучи включен может быть настроен на раздачу правильных TCP/IP настроек клиентам сети автоматически.

2.TCP vs UDP.

-TCP - это протокол основанный на соединении, обеспечивающий коррекцию ошибок и гарантирующий доставку данных с использованием так называемого управления потоком. Управление потоком определяет когда передачу потока данных необходимо остановить и когда послать пакеты данных повторно, например, из-за сбоев в канале, тем самым убеждаясь в завершенности и аккуратности переданных данных. TCP обычно используется для обмена важной информации, таких как транзакции базы данных.

-С другой стороны, пользовательский пакетный протокол (UDP) - это протокол без установления соединения, который редко используется для передачи важных данных, поскольку не использует управление потоком и иные методы проверки целостности передаваемых данных. UDP обычно используется в таких приложениях, как передача аудио- и видео-потоков, поскольку он значительно быстрее TCP из-за отсутствия коррекции ошибок и управления потоком, где потеря нескольких пакетов обычно не катастрофично.

3.Системный вызов socket.

-Для создания сокета используется системный вызов socket.

s = socket(domain, type, protocol);

-Этот вызов основывается на информации о коммуникационном домене и типе сокета. Для использования особенностей Internet, значения параметров должны быть следующими:

* communication domain -

AF\_INET (Internet протоколы).

* type of the socket -

SOCK\_STREAM; Этот тип обеспечивает последовательный, надежный, ориентированный на установление двусторонней связи поток байтов.

-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.

-Для создания сокета типа stream с протоколом TCP, обеспечивающим коммуникационную поддержку, вызов функции socket должен быть следующим:

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

4.Системный вызов bind.

-Для связывания сокета с адресом и номером порта используют системный вызов bind:

bind(s, name, namelen);

-Привязываемое имя (name) это строка байт переменной длины, которая интерпретируется поддерживаемым протоколом. Интерпретация может различаться в различных коммуникационных доменах.

5.Системный вызов listen.

-Системный вызов listen() переводит TCP–сокет в пассивное (слушающее) состояние и создает очередей для порождаемых при установлении соединения присоединенных сокетов, находящихся в состоянии не полностью установленного соединения и полностью установленного соединения. Для этого вызов имеет два параметра: дескриптор TCP–сокета и число, определяющее глубину создаваемых очередей.

-Системный вызов listen требует предварительной настройки адреса сокета с помощью системного вызова bind().

6.Системный вызов accept.

-Системный вызов accept() позволяет серверу получить информацию о полностью установленных соединениях. Если очередь полностью установленных соединений не пуста, то он возвращает дескриптор для первого присоединенного сокета в этой очереди, одновременно удаляя его из очереди. Если очередь пуста, то вызов ожидает появления полностью установленного соединения.

-Системный вызов также позволяет серверу узнать полный адрес клиента, установившего соединение. У вызова есть три параметра: дескриптор слушающего сокета, через который ожидается установление соединения; указатель на структуру, в которую при необходимости будет занесен полный адрес сокета клиента, установившего соединение; указатель на целую переменную, содержащую максимально допустимую длину этого адреса. Как и в случае вызова recvfrom() , последний параметр является модернизируемым, а если нас не интересует, кто с нами соединился, то вместо второго и третьего параметров можно указать значение NULL.

7.Системный вызов recv.

int recv(int s, void \*buf, size\_t len, int flags);

int recvfrom(int s, void \*buf, size\_t len, int flags, struct sockaddr \*from, socklen\_t \*fromlen);

int recvmsg(int s, struct msghdr \*msg, int flags);

-Системные вызовы recvfrom и recvmsg используются для получения сообщений из сокета, и могут использоваться для получения данных, независимо от того, является ли сокет ориентированным на соединения или нет.

-Если параметр from не равен NULL, а сокет не является ориентированным на соединения, то адрес отправителя в сообщении не заполняется. Аргумент fromlen передается по ссылке, в начале инициализируется размером буфера, связанного с from, а при возврате из функции содержит действительный размер адреса.

-Вызов recv обычно используется только на соединенном сокете (см. connect(2)) и идентичен вызову recvfrom с параметром from, установленным в NULL.

-Все три функции возвращают длину сообщения при успешном завершении. Если сообщение слишком длинное и не поместилось в предоставленный буфер, лишние байты могут быть отброшены, в зависимости от типа сокета, на котором принимаются сообщения (см. socket(2)).

8.Системный вызов send.

-Параметры системного вызова send следующие:

* s – дескриптор сокета.
* msg – указатель на структуру msghdr.
* flags – управляющая информация.

-Существует четыре системных вызова для посылки данных в сеть: write, writev, sendto и sendmsg. В данной статье будет рассмотрен только вызов sendmsg(). Оставшиеся три вызова так или иначе обращаются к функции sosend(). Библиотечные функции send, sendto и sendmsg работают только с дескрипторами сокетов, в то время как write и writev – с любым типом дескрипторов.

9.Системный вызов close.

-Параметры системного вызова close следующие:

* fp – указатель на файловую структуру.
* p – указатель на структуру proc вызывающего процесса.

-Системный вызов close закрывает (или прерывает) все существующие соединения сокета.

-Функция soo\_close() вызывает so\_close(), которая сперва проверяет, является ли закрываемый сокет слушающим (т.е. принимающим входящие соединения). Если так, то обе сокетные очереди проверяются на наличие соединений, ожидающих обработки. Для каждого такого соединения вызывается soabort(), которая, в свою очередь, вызывает tcp\_usrreq() с типом запроса PRU\_ABORT. Оператор switch передает управление функции tcp\_drop(), которая проверяет состояние сокета.

10.Системный вызов connect.

-Параметры системного вызова connect следующие:

* s – дескриптор сокета.
* name – указатель на буфер, содержащий IP-адрес и порт удаленной стороны.
* namelen – размер буфера name.

-Вызов connect обычно производится на клиентской стороне для установления соединения с сервером. Если клиентская программа не вызвала предварительно bind, то система автоматически сделает неявный bind для локального сокета.

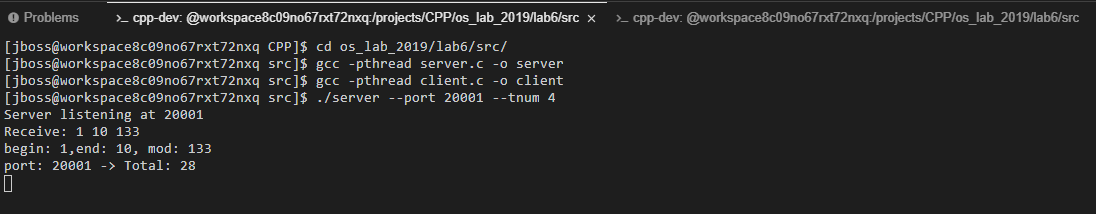
-Внутри себя системный вызов connect копирует удаленный адрес (т.е. адрес, к которому производится подключение), переданный программой, в пространство ядра и вызывает функцию soconnect(). После завершения soconnect(), connect входит в режим ожидание до тех пор, пока уровень протокола не уведомит о том, что соединение вошло в состояние ESTABLISHED, либо о том, что произошла ошибка сокета. Функция soconnect() проверяет, находится ли сокет в правильном состоянии, и вызывает pr\_usrreq() с типом запроса PRU\_CONNECT.

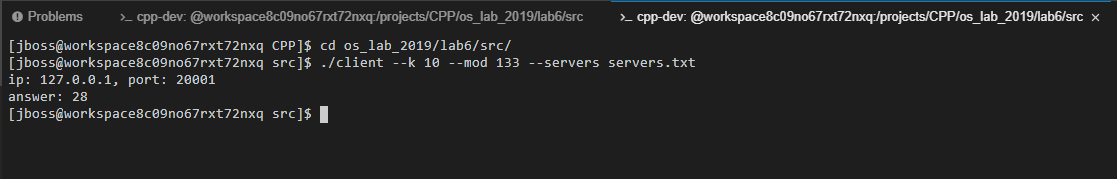
***Необходимо закончить client.c и server.c:***

***Клиент в качетсве аргументов командной сроки получает k, mod, servers, где k это факториал, который необходимо вычислить (k! % mod), servers это путь до файла, который содержит сервера (ip:port), между которыми клиент будет распараллеливать соединения.***

***Сервер получает от клиента "кусок" своих вычислений и mod, в ответ отсылает клиенту результат этих вычислений.***

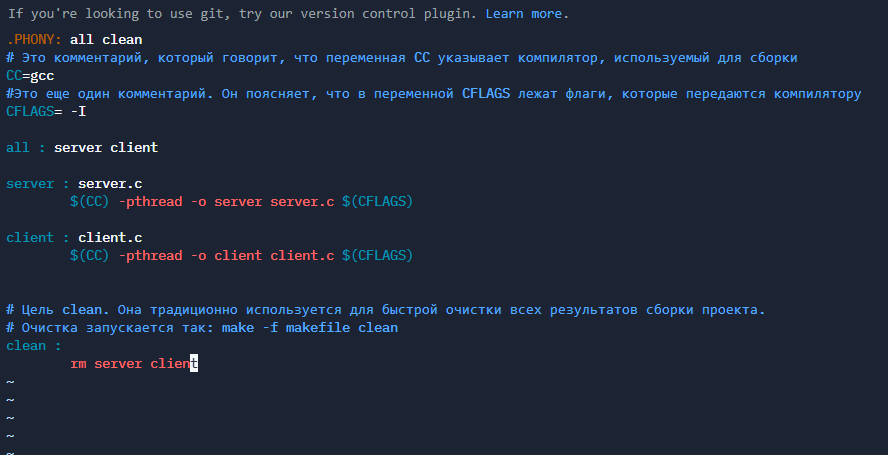
Code in github

https://github.com/lehoanggia2k/os\_lab\_2019



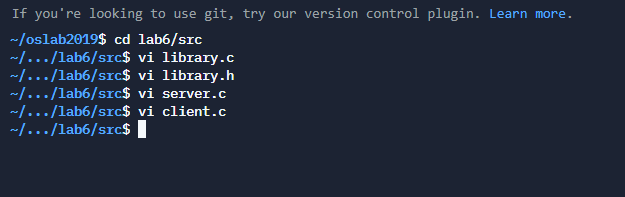
**Задание 2**

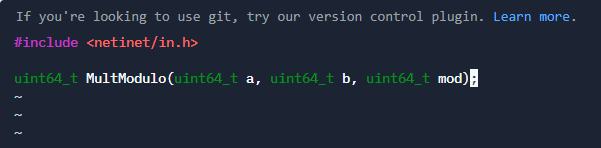
***Создать makefile для программ клиента и сервера.***

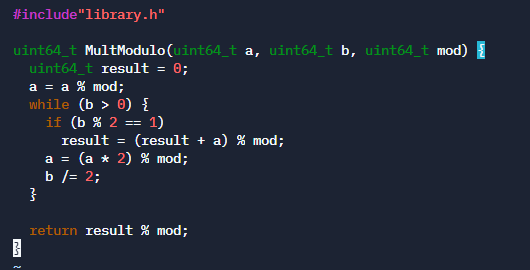
******

**Задание 3**

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







Makefile

