Вопросы можно задавать здесь:
https://docs.google.com/document/d/1zxOfYZkEeJ5V1yVsu5QtZ_h-n1O19iO2uXBw6Co-6lo/edit?usp=sharing

1. **Общие требования**
    a. Все лабораторные сдаются одновременно на 2+ машинах, то есть демонстрируется работа в реальной сети. Сдавать с использованием виртуальных машин возможно только в порядке индивидуального согласования.
    b. Для написания ЛР могут использоваться языки C/C++/Objective-C в любом сочетании. Использовать иные языки возможно только в случае, если они предоставляют непосредственный доступ к системным вызовам и струкутрам Berkeley Sockets. В случае спорной ситуации, необходимо будет переписать реализацию на C/C++/Objective-C.
    c. Скорость передачи файлов должна быть не менее 50% от максимальной теоретической скорости передачи данных в сети, вне зависимости от задержек.
    d. Критерии проверки скорости передачи будут учитывать как реальную проверку скорости, так и теоретическую. Параметры сети, в которых должна выдерживаться указанная в пункте **1.c** скорость:
        i. задержки в сети - 10 секунд
        ii. количество потерянных UDP пакетов - 5%, но не менее 1-го из серии.
        iii. если передаются 2 независимых пакета (не в рамках одной серии), то теряется хотя бы один из них, причем выбираться будет наихудший случай для реализованного алгоритма.
    e. Все реализации должны работать в рамках одного процесса и потока, если иное явно не указано в условиях ЛР. Если потоки автоматически создаются ОС (вызов callBack функции и обработчики сигналов), то это допустимо.
    f. Код функций должен быть коротким и содержать только суть выполняемых действий. Каждая фунция (метод класса) должна выполнять только одно понятное, простое и законченное действие, которое ясно из ее названия. При необходимости выполнить несколько действий, необходимо каждое из них вынести в отдельную функцию. Максимальная сложность функции должна быть ограничена 10-15 действиями и весь ее код должен вмещаться в 60 строк, то есть примерно на один экран, включая и пустые строки. Под действиями подразумеваются вызовы функций, условные операторы, математические выражения и т.д. Инициализация и объявление перемененных (в том числе вызовы memset, zeromem, malloc, free) дейтсвиями не считаются. Допустимо иметь несколько больших функций, только если вы сможете обосновать необходимость их создания (преимущества относительно нескольких более маленьких). Накладные расходы на вызов функий, а также плохая архитектура приложения (передача 100500 параметров) обоснованием не являются.
    g. Реализации лабораторных работ должны быть кросплатформенными. То есть один и тот же код должен компилироватся и работать как в Windows так и в одной из *nix систем на выбор: Linux, BSD, Mac OS X. Для Windows

обязательным условием язвляется использование копилятора MSVC входящего в Visual Studio 2010 или выше. Для *nix систем выбор средства разработки остается на усмотрение студента.

i. Для реализаций ЛР защищенных **до 18.10.2014** включительно требование пункта **2.a** не распространяется. То есть, последующая работа должна включать функуциональность предыдущей, работа которой гарантируется только в одной из ОС.
ii. Для эмуляции *nix сигналов на Windows, допускается создание дополнительных потоков, без какой либо обработки данных в них, а только для ожидания событий! Например ожиданий событий на сокете с использованием функциий **WSACreateEvent** и **WSAEventSelect** (или других, например **WSARecv**/**WSARecvEx**)
iii. Так как fopen/open под Windows не работает с UTF-8, то допускается работа с именами файлов представленных только в ASCII, так как цель ЛР изучение работы с сетью и сокетам. Однако реализация обертки вокруг файлов с поддержкой Unicode и вызовом функции CreateFileW (принимает UTF-16) будет большим плюсом.
iv. Стиль написания кода (об этом говорилось еще в прошлом году).
Неправильный способ написания:

```
#if WINDOWS
        WSAConnect(s);
#elif UNIX
        connect(s);
#endif
```

Правильный способ:

```
Connect(s);
/*А уже внутри функции разбираться что вызывать и когда*/
```

2. **Важные замечания по реализации ЛР**
   a. Следует обратить внимание, что каждая последующая ЛР включает в себя функциональность предыдущей.
   b. Тот процесс который принимает входящие подключения условно называется *“сервером”*. *“Сервер”,* вне зависимости от вида, должен поддерживать возможность работы с несколькими клиентами. То есть при отключении *“клиента”*, сервер должен иметь возможность принять и обслужить другого клиента без своего перезапуска. Завершение работы сервера без явной комманды на это действие будет считаться ошибкой реализации.
3. **Основные понятия**
   a. **Последовательный сервер** - сервер исполняющий комманды (запросы) клиента исключительно последовательно. Если сервер не поддерживает одновременную работу с несколькими клиентами, то их запросы на подключение он должен отвергать. Однако следует обратить внимание на пункт **2.b**.

b. **Оверхед** (overhead) - это накладные расходы которые будут затрачены на выполнение полезной работы. В частности к работе с протоколами TCP/UDP - это размер служебных данных по отношению к полезным. Под служебными данными подразумеваются:
    i. заголовки IP,UDP/TCP,
    ii. служебные пакеты ASK, SYN и т.д.
    iii. данные используемые программой для правильного выполнения, такие как: номер пакета, размер файла, ответные пакеты подтверждение приема (включая размер всех заголовков) и т.д.

4. **Основные ошибки в реализации**
    a. Попытка чтения данных из TCP сокета за одну операцию может привести к тому, что не все необходимые данные будут в итоге прочитаны. TCP гарантирует передачу потока байт, но не имеет средств для выделения границ сообщений, то есть не гарантирует прием данных именно такими “порициями”, как они были отправлены. Реализация выделения границ сообщения (окончания комманды, строки и т.д.) ложится на программиста.
    b. Ожидание подтверждения каждого отправленного UDP пакета, и отправка неоправданно малого количества байт в одном пакете. Ожидание каждого пакета приведет к невозможности выполнения требования пункта **1.c**.
    c. Чтение или отправка данных очень малыми порциями. Требование можно трактовать как то, что среднее значение оверхеда должно быть не более 4% за всю сессию. Что такое оверхед описано в пункте **3.b.**

Важная информация!

Так как количетсво часов в семестре формально расчитаны на 4 лабораторные работы, а сложность заданий (с учетом кросплатформенности) как раз равна сложности 4 лаборатных работ по программе от 2010 года, то задания будут перегруппированы в 4 лабораторные работы без изменения сути, но возможно с изменением порядка заданий.

Изменения будут отражены как в этом документе, так и выложены на сетевом диске в университете.

Для вас это будет означать изменения в допусках к занятиям по причине незащищенных работ, с соответсвующим изменением количества ведомостичек в меньшую сторону.

Лабораторная работа № 1

Знакомство с программированием сокетов.

Необходимо реализовать простейшую программу-сервер с использованием протокола TCP. Сервер должен поддерживать выполнение нескольких комманд,

определяемых на усмотрение студента, но как минимум должен поддерживать выполнение следующих (или аналогичных):

- **ECHO** (возвращает данные переданные клиентом после команды),
- **TIME** (возвращает текущее время сервера),
- **CLOSE** (EXIT/QUIT) (закрывает соединение).

Комманда может иметь параметры (например **ECHO**). Любая команда должна оканчиваться символами **\r\n** или **\n**.

В качестве клиента предполагается использование системных утилит: telnet, netcat и других. Возможно использование собственной программы клиента, но это является не обязательным.

Продемонстрировать использование утилит: nmap -- сканирование портов сервера, netstat -- список открытых сокетов на сервере, номера портов.

Клиент серверная программа для передачи файла по сети с использованием протокола TCP

Необходимо реализовать клиент и последовательный сервер, позволяющие обмениваться файлами. Сервер должен продолжать поддерживать комманды, которые были реализованы в лабораторной работе №1, добавляя к этому комманду передачи файла. Запрос на закачку файла на сервер или скачивание с него, должен инцировать клиент. Команды могут быть к примеру **UPLOAD**/**DOWNLOAD**. Решать проблемы связанные с тем что такого файла нет (например мы хотим скачть файл с сервера, но не знаем каково его имя), не нужно, достаточно вывести сообщение, что файла с запрашиваемым именем нет.

Файлы должны передаваться с помощью протокола TCP. Реализация должна учитывать возможные исключительные ситуации, связанные с проблемами сети, такие как физический или программный обрыв соединения.

Алгоритм определения разрыва соединения может быть любым, но таким, чтобы пользователь смог узнать об этом в разумное время (от 30 секунд до 2-5 минут). До вывода сообщения о наличии проблем с соединением программа должна восстанавливать передачу файла самостоятельно. Если же сообщение о проблеме уже выведено, то решение о попытке восстановления должен принимать пользователь.

Сервер обязан поддерживать **восстановление докачивания/скачивания** файла но с ограничениями: после обрыва соединения **подключился тот же клиент** и пытается докачать/скачать **тот же файл**, что и в прошлую сессию. Если успел подключится другой клиент, или сервер был перезапущен, то сервер имеет полное право удалить файлы относящиеся к незавершенным загрузкам.

Сервер и клиент обязаны работать в рамках одного потока (см. пункт **1.e**)

Лабораторная работа № 2

Изучение внеполосного режима передачи данных.

Модификация программы из Л.р. 1: во время передачи данных с использованием протокола TCP, передающая сторона должна генерировать внеполосные данные и выводить на экран общее количество переданных байт данных (не включая срочные),

принимающая сторона должна выводить на экран общее количество принятых байт (не включая срочные) при получении срочных данных.

Клиентсерверная программа для передачи файла по сети с использованием протокола UDP.

Добавить к клиенту и серверу возможность передачи файла с помощью протокола UDP. Уделить внимание обработке исключительных ситуаций, например физического или программного обрыва соединения. Проверять можно с помощью включения файерволла с отбрасыванием пакетов без уведомления (правило DROP) и с отбрасыванием пакетов с уведомлением (правило REJECT).

Лабораторная работа № 3

Организация параллельной обработки запросов на сервере с помощью мультиплексирования

Модификация программы из Л.р. 2. Модифицировать TCP сервер для организации параллельного обслуживания нескольких клиентов с помощью мультиплексирования (select, pselect, poll).

Лабораторная работа № 4

организация параллельной обработки запросов

Модификация программы из Л.р. 1 / 2. Модифицировать сервер для организации параллельного обслуживания нескольких клиентов

В случае предварительного создания пул процессов/потоков одлжен быть динамическм: при занятости всех существующих обработкой запросов, добавляюстя новые процессы/потоки в очередь ожидания. Максимальное количество динамически выделенных потоков/процессов должно быть ограничено числом Nmax, при котором сервер может функционировать стабильно. При завершении обработки лишние ожидающие процессы/потоки должны завершаться после определенного таймаута. Минимальное число ожидающих процессов/потоков должно быть ограничено числом Nmin.

Параллельный сервер без установления логического соединения
Ведущий поток:
1 Создать сокет и привязать его к общепринятому адресу службы
2 В цикле считывать запросы с помощью recvfrom и создавать новые ведомые потоки (процессы) для формирования ответа

Ведомый поток:
1 Работа потока начинается с получения конкретного запроса от ведущего потока (процесса), а также доступа к сокету
2 Сформировать ответ согласно прикладному протоколу и отправить его клиенту с использованием функции sendto
3 Завершить работу потока (процесса)

Параллельный сервер с установлением логического соединения
Ведущий поток:
1 Создать сокет и привязать его к общепринятому адресу службы
2 Перевести сокет в пассивный режим
3 Вызвать в цикле функцию accept для получения очередного запроса от клиента и создавть новый ведомый поток или процесс для формирования ответа
Ведомый поток:
1 Работа начинается с получения доступа к соединению, полученному от ведущего потока
2 Выполнять в цикле работу с клиентом через соединение
3 Закрыть соединение и завершить работу.

Вар.	Описание
1	Параллельный сервер с установлением логического соединения (TCP) и созданием дочерних процессов
2	Параллельный сервер с установлением логического соединения (TCP) и созданием дочерних потоков
3	Сервер с предварительным созданием дочерних процессов с параллельным вызовом accept
4	Сервер с предварительным созданием дочерних процессов с блокировкой с помощью файла для защиты accept
5	Сервер с предварительным созданием дочерних процессов с использованием взаимного исключения (семафор, мьютекс, критические секции windows) для защиты accept
6	Сервер с предварительным созданием дочерних процессов с последующей передачей дескриптора сокета дочерним процессам
7	Сервер с предварительным созданием потоков с использованием взаимного исключения для защиты accept
8	Сервер с предварительным созданием потоков, главный поток вызывает accept
9	Мультисервисный сервер (суперсервер), прослушивание нескольких портов и при запросе на любой из них передавать упраление соединением соответствующему процессу службы (для примера изучить функционирование сервера xinetd)
10	Организация параллельного обслуживания нескольких клиентов с помощью

	мультиплексирования (select, pselect, poll) в рамках одного потока на сервере