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. 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
    c. 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.

Стиль написания кода.

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 команды 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.

4. **Основные ошибки в реализации**
    a. Попытка чтения всех данных из TCP сокета за один вызов может привести к тому, что будет прочитана лишь часть из них (так как другая часть могла еще не дойти). TCP гарантирует передачу потока байт, но не имеет встроенных средств для выделения границ сообщений, то есть не гарантирует прием данных в ***recv*** именно такими “порициями”, как они были “отправлены” (записаны в сокет с помощью вызова ***send***). Реализация выделения границ сообщения (окончания комманды, строки и т.д.) ложится на программиста.
    b. В случае работы с протоколом UDP, для организации надежной доставки необходимо организовывать какое-либо подтверждение. Однако реализация “в лоб” (с ожиданием подтверждения для каждого отправленного UDP пакета) будет приводить к существенному снижению производительности (см. **1.c.** и **1.d.**)
    c. Отправка неоправданно малого количества байт в одном UDP пакете приведет к существенному оверхеду, что так же не позволит выполнить требования по скорости передачи и оверхеду. Среднее значение оверхеда должно быть не более 30% за всю сессию. (**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.

)
    d. Чтение или запись данных в сокет очень малыми порциями (например по 1 байту). Это приводит сущесвенной нагрузке на CPU за счет частого переключения контекста между ядром и процессом.

Лабораторная работа №1

Знакомство с программированием сокетов.

ВАЖНО: Выбор языка программирования осуществлять с учетом своего варианта на л.б. №4

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

Команда 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 команды, которые были реализованы в предыдущей части, добавляя к этому команду передачи файла. Запрос на закачку файла на сервер или скачивание с него, должен инициировать 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.

После завершения передачи вывести битрейт (скорость передачи данных).

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 (необходимо сконфигурировать параметр сокета SO_KEEPALIVE для точного контроля интервала удержания соединения в программе или в настройках ОС -- для Windws см. Things that you may want to know about TCP Keepalives) . Если же сообщение о проблеме уже выведено, то решение о попытке восстановления должен принимать пользователь.

Сервер обязан поддерживать **восстановление докачивания/скачивания** файла. Допускаются следующие ограничения: докачка осуществляется если после восстановления 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 (и данные сессии) относящиеся к незавершенным загрузкам.

Сервер и клиент обязаны работать в рамках **одного потока**

Необязательно: Изучение внеполосного режима передачи данных.

Модификация программы:

Во время передачи данных с использованием протокола TCP, передающая сторона должна генерировать внеполосные данные (например процент переданных данных от общего количества) 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), а при получении срочных данных -- выводить их.

Вопросы:

1. Установление и разрыв соединения на уровне протокола TCP
2. Скользящее окно передачи данных в протоколе TCP: назначение, механизм функционирования
3. Механизм медленного старта
4. Алгоритм Нэгла (Nagle) преимущества и недостатки
5. Срочные данные в TCP: механизм применения, ограничения.

Лабораторная работа №2

Клиент-серверная программа для передачи файла по сети с использованием протокола UDP.

Модификация программы из Л.р. 1.

Добавить к клиенту и серверу возможность работы по протоколу UDP: передача команд и файлов. 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) или физического обрыва сети (не на аудиторных компьютерах).

После завершения передачи вывести битрейт (скорость передачи данных)

Определить оптимальный размер передаваемого буфера данных обеспечивающего наибольшую пропускную способность. Продемонстрировать производительность протокола UDP в сравнении с реализацией на TCP Л.Р. №1 (должна быть выше минимум в 1.5 раза). Объяснить полученное значение размера буфера на основании характеристик протокола UDP.

Реализовать собственные механизмы:

- подтверждения передачи пакета
- повторной передачи в случае потери пакета или ответа на пакет
- скользящего окна для предотвращения ожидания подтверждения на каждый предыдущий посланный пакет перед отправкой следующего

Вопросы:

1. Объяснить выбор оптимального размера передаваемого буфера данных на основании характеристик протокола UDP (и возможные ограничения, накладываемые протоколом Ethernet)

Лабораторная работа №3

Организация параллельной обработки запросов на сервере с помощью мультиплексирования

Реализовать в сервере “параллельное” обслуживание нескольких клиентов с помощью мультиплексирования (методы select, pselect, poll). То есть сервер должен работать в рамках одного потока, последовательно исполняя запросы клиентов и отправку/прием порций данных от них, а также подключение новых клиентов. Размер порции должен быть таким, чтобы при интерактивной работе с сервером (например с помощью **telnet**), отклик на команды не превышал **t** = ping * 10.
При обработке команд сервером, он не должен прерывать передачу/прием файлов от других клиентов.

Вопросы:

1. Объяснить преимущества механизма асинхронной передачи данных

Лабораторная работа №4
организация параллельной обработки запросов

Важно: отговорки “Мой язык программирования не поддерживает такое взаимодействие между процессами” не принимаются в качестве основания для смены варианта (см. примечание в л.р. №1)

Модификация программы из Л.р. 1 / 2. Модифицировать сервер для организации параллельного обслуживания нескольких клиентов
Выбор варианта: `номер_по_журналу mod 17`

Вар	Прот.	Порождение	Механизм защиты accept / сокета ** или примечание
1	TCP	Процессы порождаются по запросу	-
2	TCP	Потоки порождаются по запросу	-
3	UDP	Потоки порождаются по запросу, каждый поток обрабатывает один запрос	-
4	UDP	Потоки порождаются по запросу, каждый поток выполняет взаимодействие с одним клиентом до завершения сессии	Запросы считанные из сокета, но отправленные от клиента не связанного с текущей сессией не должны теряться (см. опцию MSG_PEEK)
5	TCP	Пул потоков*.	Параллельный вызов
6	TCP	Пул потоков*.	Блокировка файла
7	TCP	Пул потоков*.	Взаимное исключение
8	TCP	Пул потоков*.	Передача дескриптора от главного процесса
9	UDP***	Пул потоков*.	Конкурентный доступ к сокету
10	UDP***	Пул потоков*.	Блокировка файла
11	UDP***	Пул потоков*.	Взаимное исключение
12	TCP	Пул процессов*.	Параллельный вызов
13	TCP	Пул процессов*.	Блокировка файла

Мультисервисные сервера			
14	TCP	Процессы порождаются по запросу	Каждый порт связан с определенным видом сервиса
15	TCP	Потоки порождаются по запросу	Каждый порт связан с определенным видом сервиса
16	TCP	Взаимодействие с клиентом осуществляется внешней программой-обработчиком (отдельный исполняемый файл)	Каждый порт связан с определенным видом сервиса
17	UDP	Потоки порождаются по запросу	Каждый порт связан с определенным видом сервиса

* -- Поток(процесс)-менеджер следит только за числом свободных потоков (процессов). Пул потоков инициализируется начальным числом потоков Nmin. При возрастании числа запросов пул должен динамически расширяться. 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.

** -- accept - для TCP приложения, общий сокет - для UDP

*** -- каждый поток обрабатывает по одному запросу

Вопросы:

1. Преимущества и недостатки использования порождения процессов в параллельных серверах (см. варианты мультисервисных серверов)?
2. Преимущества и недостатки порождения процессов/потоков по запросу?
3. Для чего защищать accept при параллельном к нему обращении?
4. В каких случаях необходимо использовать именно файл в качестве защиты вызова accept?
5. Объяснить варианты мультисервисных серверов
6. Объяснить механизм работы суперсервера inetd
7. Для чего нужна опция MSG_PEEK

Лабораторная работа №5
изучение протокола ICMP и особенностей его программирования + изучение протокола IP

Реализация программы параллельного ping нескольких хостов. Каждый поток выполняет одного хоста. Чтобы потоки не забирали из входного буфера ответы предназначенные другим потокам, использовать опцию MSG_PEEG для предотвращения удаления пакета из буфера. Окончательно удалять пакет из буфера только после проверки соответствия ответа адресата данному потоку.

Добавить возможность определения пути до каждого узла назначения -- механизм аналогичный программе traceroute. Отметки времени отправки высылать при отправке ping-запроса в теле пакета. При приеме ответа вычислять время передачи на основании полученной метки. Различные обработчики ситуаций “время жизни истекло” и “пинг-ответ” или “хост недостижим”.

Преобразовать утилиту ping для выполнения атаки Smurf (в качестве адреса источника указывается адрес атакуемого узла в заголовке IP-пакета). Продемонстрировать на атакуемом узле по средствам Wireshark.

Вопросы:

1. Объяснить работу программы traceroute
2. Объяснить механизм действия smurf атаки. Какие угрозы она несет? Способы защиты?
3. Объяснить назначение полей заголовка протокола IP

Лабораторная работа №6
изучение широковещательного режима передачи данных + режим многоадресной передачи

Разработать одноранговую программу чат с использованием протокола UDP или IP, передающей и принимающей сообщения с помощью широковещательного и многоадресного режима передачи данных.

Для многоадресной передачи предусмотреть возможность самостоятельного выхода из группы и принудительного игнорирования хоста другими участниками.

Предусмотреть нахождение и вывод списка IP-адресов запущенных приложений. Программа должна автоматически определить сетевые параметры интерфейса: ip адрес, сетевую маску и адрес широковещательной передачи данных, нахождение и вывод списка IP-адресов запущенных приложений.

Вопросы:

1. В чем отличие многоадресной и широковещательной передачи?
2. Как формируется широковещательный адрес?
3. Какие диапазоны адресов многоадресной передачи существуют? Каково их назначение?
4. Какие есть механизмы ограничения области распространения многоадресных пакетов?
5. Какова область распространения широковещательных пакетов?

Лабораторная работа №7
MPI: Изучение парных коммуникаций, коллективных операций, групп и коммуникаторов.

Программа для умножения матриц (размер должен быть достаточно большим, чтобы процесс занимал ~10..50 сек.) с использованием парных операций.
Реализовать 2 варианта с использованием блокирующего и неблокирующего режимов передачи. Замерить время. Неблокирующий режим должен демонстрировать заметный прирост производительности (пример организации программы см. аналогично CUDA Streams cudaMemcpyAsync).

Программы запускаются минимум на 3-х компьютерах.

Вопросы:

1. Что такое MPI_Com_World?
2. Что такое rank?
3. Какими вызовами должна начинаться и завершаться MPI программа?
4. Объяснить преимущество асинхронных операций

Лабораторная работа №8
MPI:групповые и файловые операции.

Добавить к программе из л.р. №7 возможность использования коллективных операций. Программа должна создавать произвольное количество групп (задается из командной строки) и включить в них случайное количество процессов. Каждая группа должна умножить матрицы.

Замерить время вычисления в каждой группе и вывести на экран. Сравнить со временем выполнения на парных операциях

Групповые операции с файлами.
Исходные данные для умножения задаются из 2х файлов доступных всем узлам по сети. Процессы одновременно читают свою порцию данных из файла основываясь на своей координате или rank-е.
Каждый процесс выводит результаты умножения матриц в файл, принадлежащий соответствующей группе.

Не обязательно:
Запуск приложения на кластере с помощью PBS
Запустить приложение из л.р. в кластерной среде с помощью механизмов PBS Torque.