Task1 – общие сведения

Цель – освоить клиент-серверную модель взаимодействия процессов, основанную на сокетах.
Средство – языки Си или C++ (для тех, кто уже знаком с языком C++).

Базовая модель:

- Процесс-сервер:
  - создает "слушающий" сокет и ждет запросов на соединение от клиентов
  - приняв запрос – создает сыновний процесс, который должен обслужить клиента и завершиться, а процесс-отец продолжает принимать запросы на соединение и создавать новых сыновей.
- Процесс-сын (обслуживание клиента) – получить от процесса-клиента данные, обработать их и вернуть клиенту, после чего завершиться – возможна обработка нескольких порций данных от клиента, до получения от клиента команды на завершение
- Процесс-клиент запрашивает у пользователя данные, отправляет их серверу (адрес сервера можно задать в командной строке, или спросить у пользователя), получает от сервера ответ, выполняет преобразования (или некие действия, например, печать на экран).

Такая модель предполагает, что клиенты не общаются между собой через сервер, а взаимодействуют только с сервером (каждого клиента обслуживает отдельный серверный процесс). Реализацию подобной модели можно посмотреть на сайте http://cmcmsu.info/, раздел "второй курс, весенний семестр" в методич. пособии "Модельный SQL-интерпретатор. (2005 г.)".

Модель с межклиентским взаимодействием

- Процесс-сервер:
  - создает "слушающий" сокет и ждет запросов на соединение от клиентов
  - В бесконечном цикле – либо принимает запрос на подключение от клиента, либо обрабатывает данные полученные от клиента. Обработка данных может включать в себя в том числе отправку данных конкретному клиенту, отправку данных всем клиентам, сохранение данных и т.п.
- Процесс-клиент запрашивает у пользователя данные, отправляет их серверу (адрес сервера можно задать в командной строке, или спросить у пользователя), получает от сервера ответ, выполняет преобразования (или некие действия, например, печать на экран).

Очевидно, что в данной модели сервер выступает в качестве управляющего узла, ему необходимо знать всех подключенных клиентов (а не только одного как это делает процесс-сын в базовой модели) и уметь понимать, чего эти клиенты хотят в этом вам, поможет функция select(). Этот системный вызов подробно описывается в методическом пособии А.В. Столярова "Многопользовательский игровой сервер" (http://www.stolyarov.info/) или в разделе "Учебные пособия" кафедрального сайта АЯ (http://al.cs.msu.su/), также в книге А.В. Столярова "Программирование", том 3 (стр. 222 в http://www.stolyarov.info/books/pdf/progintro_vol3.pdf). Кроме того достаточно подробное описание этой функции и других для работы с сокетами на языке С можно найти здесь – https://rsdn.org/article/unix/sockets.xml и здесь http://citforum.ru/programming/unix/sockets/, для тех, кого не пугает английский язык, есть еще неплохая презентация https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf .

Для тестирования как базовой модели, так и расширенной модели не обязательно иметь несколько компьютеров – можно запускать клиентов из разных консолей на одном компьютере, хотя при желании Вы можете развернуть сервер где-нибудь в сети (например, на digitalocean.com) и даже подружить Ваши программы клиенты с чужими программами-серверами.

Дополнительные требования и условия к Task1

- Не забывайте про обработку сигналов как на сервере, так и на клиенте (ctrl+C и ctrl+Z), в данном задании ctrl+Z сервера может приводить к тому, что сокеты останутся занятыми и повторный запуск (сервера и клиентов) не будет соединять их между собой до полного перезапуска терминала

- Обращайте внимание на точность ввода команд в Ваших программах, если у вас реализована команда \help, то команда \helpppp не должна восприниматься как команда.
- Пустое сообщение – состоящее только из пробельных символов, не является сообщением, также, как и пустой ник (в случае чата) не является корректным
- Длинные сообщения (превышающие ваш буфер) должны обрабатываться одним из трех способов:
    1. Вывод сообщения об ошибке
    2. Передача сообщения по частям и склейка частей в исходное состояние
    3. “Правильное” разбиение текста (например, по пробелу, слова не должны разбиваться по полам), передача частями, вывод отдельных частей без склейки, причем так чтобы сообщения других пользователей не вклинивались в части большого сообщения.
- **Во всех вариантах** при запуске сервера указывается номер порта, на котором сервер будет ожидать подключения клиентов. При запуске клиента указывается имя компьютера(адрес) и номер порта для подсоединения к серверу.
- Общие для всех вариантов правила форматирования сообщений:
    - приватное сообщение начинается с *
    - оповещение о событии на канале, начинается с *** (например, пришел или ушел пользователь)
    - служебное сообщение начинается с ### (например, завершение работы сервера)
- Сообщения, отправляемые программой-клиентом, могут быть двух типов:
    - команда, начинается с \ и имеет следующий вид \<команда> <параметр> ... <параметр>, при этом лишние пробельные символы между параметрами игнорируются, **если команда не распознана, такое сообщение считаем обычным**
    - сообщение (реплика) чата
- Клиент может завершать свои строки любой из комбинаций: \n, \r, \n\r, \r\n (сервер поддерживает разные типы клиентских программ)
- Сервер начинает обработку сообщения от клиента, когда получит символ(ы) конца строки. До этого момента сообщение (строка) накапливается во временном буфере. Максимальная длина одной строки, получаемой сервером, может быть, как фиксированной, так и динамической (помните про обработку длинных сообщений):
- Начальные и завершающие пробельные символы (пробел, табуляция) игнорируются (т.е., строка " hi ppl " будет обработана в точности также, как "hi ppl")
- Когда сервер завершает свою работу, он должен прежде попрощаться со всеми клиентами, например, ### server is shutting down, thanks to everyone.

Для организации взаимодействия процессов необходимо использовать сокеты коммуникационного домена PF_INET с установлением соединения (SOCK_STREAM).

Студенты, выполняющие одинаковый вариант, могут объединяться в "консорциумы" для выработки единых стандартов (правил) общения сервера и клиента. Если придерживаться выработанных правил, клиент одного студента, сможет общаться с сервером другого студента и наоборот. Будет происходить взаимное тестирование.

Заметим, что "консорциум" **не означает командное программирование(!)**, каждый студент выполняет задание самостоятельно, зная только "правила взаимодействия" клиентов и серверов, и не имея представления о внутреннем устройстве чужой программы.

Такое взаимодействие возможно не только для чатов, но и для "простого" задания. Например, сервер одного студента увеличивает число на 1, другого -- на 5, третьего -- на 10. Тогда клиент может воспользоваться услугами разных серверов, в зависимости от того, на сколько нужно увеличить число. Если на 12, то один раз нужно обраться к третьему серверу и два раза -- к первому.

Набор задач является переработкой заданий, предлагаемых в разные годы Ю.С. Коруховой, П.Г. Сутыриным, А.А. Вылитком. Текст заданий дополнен К.М. Ивановым.

Варианты задания (сложность указана звездочками):

Вариант 1 (*) – Калькулятор 1:
Процесс-сервер – создает "слушающий" сокет и ждет запросов на соединение от клиентов. Приняв запрос, процесс-сервер создает сыновний процесс, который должен обслужить клиента и завершиться, а процесс-отец продолжает принимать запросы на соединение и создавать новых сыновей.

Задача сына (обслуживание) – возможные команды от клиента:

1) \+ <число> - установить число на которое сервер будет увеличивать числа для данного клиента (по-умолчанию 1), вернуть клиенту “Ok”
2) <число> - запрос на увеличение числа от клиента, задача – увеличить данное число на заданное и вернуть клиенту
3) \? – получить от сервера число, на которое тот увеличивает числа для текущего клиента
4) \- – сообщить серверу о завершении работы, при этом сервер-сын завершается

Программа-клиент:

1) запрашивает у пользователя очередную команду, отправляет ее серверу (адрес сервера можно задать в командной строке - argv, или спросить у пользователя первым действием),
2) получает от сервера ответ и печатает его на экран.

В качестве программы клиента возможно использование telnet.

Вариант 2 (**) – Калькулятор 2:
Клиентам доступны те же команды, что и в Варианте 1, с некоторыми изменениями:

1. Все клиенты обслуживаются единым сервером (а не отдельными сыновьями).
2. Команда +<число> устанавливает глобальное для всех клиентов число, на которое будет увеличен запрос
3. Сервер ждет пока к нему не придет **два числа** от одного и того же или нескольких клиентов (с помощью команды <число>) складывает эти два числа, прибавляет к ним то, что установлено глобально и возвращает **только тем клиентам, которые учувствовали в операции** (прислали эти два числа)

Здесь и далее сервер завершает свою работу только в случае ввода (если не указано иного):

- **команды \exit на самом сервере (команды от клиентов не принимаются)**
- **Сигнала ctrl+C**
- **Сигнала ctrl+Z**

Вариант 3 (**) – Чат 1:
Процесс-сервер:
1. создает "слушающий" сокет и ждет запросов на соединение от клиентов
2. при поступлении запроса, устанавливается соединение с очередным клиентом, от клиента сервер получает имя (ник) вошедшего в "комнату для разговоров", клиент заносится в список присутствующих
3. всем присутствующим рассылается сообщение, что в комнату вошел такой-то (ник).
4. от разных клиентов могут поступать реплики – получив реплику от клиента, сервер рассылает ее всем "присутствующим" (включая самого автора!) **с указанием автора реплики**.

5. при разрыве связи (команда \quit) с клиентом сервер сообщает всем, что-такой-то (ник) нас покинул (ушел) и выводит его прощальное сообщение

Необходима реализация команд:

1. \users получить от сервера список всех пользователей (ники), которые сейчас онлайн
2. \quit <message> - выход из чата с прощальным сообщением

В качестве программы клиента возможно использование telnet. Имена пользователям могут выдаваться автоматически, но они должны быть **уникальными**. Иной подход – сервер запрашивает имя клиента.

Вариант 4 (***) – Чат 2:

Условия аналогичны Варианту 2 с дополнительными командами:

1. \private <nickname> <message> приватное сообщение пользователю с ником <nickname>, если пользователя-адресата на сервере нет, то выдается сообщение об ошибке
2. \privates – получить от сервера имена пользователей которым Вы отправляли приватные сообщения
3. \help вывод допустимых команд

При запуске программы-клиента указывается имя компьютера (адрес) и номер порта для подсоединения к серверу, а также **никнейм** пользователя – **проверка на уникальность обязательна**!

В данном и всех последующих вариантах telnet может выступать только в качестве дополнительного средства взаимодействия с сервером – наличие программы клиента обязательно

Вариант 5 (***) – Автосалон:

Программа-сервер имитирует деятельность автосалона, который продает и покупает подержанные автомобили у программ-клиентов

Программа-клиент ожидает ввода сообщений со стандартного потока ввода и одновременно ожидает поступления сообщения от сервера:

- При поступлении сообщения от сервера оно печатается на стандартный поток вывода
- При вводе сообщения с клавиатуры оно пересылается на сервер.

Со стандартного потока ввода читаются сообщения в следующем формате:

- BUY <марка машины>
  - Когда сервер получает очередное сообщение BUY … от какого-либо клиента, он удаляет автомобиль из своей базы, если такой автомобиль есть, и рассылает всем клиентам сообщение о том, что машина куплена. Если таких машин несколько, то удаляется первая
  - Если успешно найдена, то клиенту, который ее выставил на продажу отправляется **дополнительное** сообщение с поздравлением
  - Если запрошенной машины нет – отправителю BUY сервер сообщает об ошибке
- SELL <марка машины>
  - При получении запроса SELL … сервер добавляет машину к своей базе данных и рассылает всем клиентам сообщение о появлении нового автомобиля в салоне

При попытке ввода сообщений, не соответствующих формату, программа-клиент должна сообщать об ошибке.

Вариант 6 (****) – Чат 3:

Расширение Варианта 4:

Появляется новый тип пользователей – админы, для того чтобы получить доступ к функционалу администраторов пользователь должен выполнить команду \admin, после чего сервер запрашивает пароль, при вводе корректного пароля пользователь получает доступ к дополнительным командам:

- \ban <nickname> <message> - отключает текущего пользователя с заданным именем и запрещает новые подключения к серверу с таким именем, при этом пользователю выводится причина его блокировки, указанная в <message>

- \kick <nickname> <message> - отключает пользователя с заданным именем, при этом пользователю выводится причина его блокировки, указанная в <message>
- \nick <oldnickname> <newnickname> - принудительно меняет имя пользователю
- \shutdown <meassage> – завершает работу сервера, а всем пользователям отправляется заданное сообщение

(!) Админы не могут влиять на других админов

Обычные пользователи также получают возможность менять свой никнейм командой \nick <newnickname>, если такой пользователь уже есть, должно выдаваться сообщение об ошибке тому, кто пытается занять это имя. Можно выдавать сообщение и тому клиенту, чьё имя пытаются занять, с указанием IP-адреса или никнейма второго клиента.

Вариант 7 (*****) – Чат 4:

Расширение варианта 6, вводится поддержка нескольких каналов:

- Прежде чем начать писать сообщения клиент должен выбрать канал(комнату) командой /room <название_канала> и подключиться к каналу, если он есть. Если пользователь на этом канале заблокирован, выдается сообщение, содержащие причину блокировки. Если канала нет, то таковой создается, и пользователь становится его создателем, а также его **главным администратором** (при этом сервер запрашивает пароль, который будут использоваться для всех администраторов канала)
- Все функции администраторов, описанные в Варианте 5, теперь действуют в рамках канала (кроме \shutdown – она более недоступна простым администраторам)
- Команда \users без параметров действует в рамках канала, команда \users <chanel> позволяет узнать список пользователей канала
- Администраторам канала становится доступна новая команда:
  - \topic <newname> - позволяет менять имя канала
- Главному администратору становится доступны команды:
  - \setadmin <nickname> - высылает заданному пользователю пароль от функций администратора
  - \banadmin <nickname> – запрещает пользователю использовать функционал администратора, даже если тот имеет пароль
  - \unbanadmin <nickname> – разрешает пользователю быть администратором
  - \delete – удаляет канал
  - \passwd <password> – позволяет изменить пароль администратора, при этом он высылается всем не забаненым администраторам
- Обычным пользователям становится доступна команда \leave, позволяющая покинуть канал
- Команды \private и \privates продолжают действовать в рамках всей системы
- Список доступных каналов можно узнать командой \rooms
- Вводится новый тип пользователя server admin (\serveradmin) – не может воздействовать на каналы, но может завершить работу сервера (\shutdown), а также перезагрузить его (\restart <message>) Сервер должен завершать и начинать свою работу так, чтобы между запусками сервера сохранялись (Нужно придумать формат файла, читать и писать в него по необходимости):
  - имена, пароли и статусы (создатель, оператор) всех пользователей
  - каналы, их темы, списки заблокированных пользователей с причинами.
  - отложенные сообщения (если реализованы)
- Поддержка так называемых отложенных (offline) сообщений. Если адресатом в команде / private указан пользователь, зарегистрированный на сервере, но не подключенный в данный момент, то сообщение кладется в очередь (не более 10 для одного получателя), а

отправителю посылается объяснение (user is offline right now, your message will be delivered upon his/her connection). Соответственно, при подключении адресата ему первым делом доставляются личные сообщения из очереди. В этом случае адресат должен видеть, когда они были отправлены (абсолютное время или относительное: "5 hours 45 minutes 3 seconds ago")

В данном варианте допустима частичная реализация функционала, при этом в README.md должно быть четко зафиксировано, что реализовано, а что нет.

Вариант 8 (******) – GUI Чат:

Расширение Варианта 7 – визуальный пользовательский интерфейс для чата, что можно реализовать:

- одновременное присутствие на нескольких каналах (при желании каждый канал можно реализовать на отдельном сервере-сыне)
- одновременное ведение нескольких приватных бесед (беседа как последовательность приватных сообщений, начинается первым сообщением, фактически является каналом без администраторов)
- отдельное окно для набора своего сообщения, не прерываемого потоком канала
- отдельное окно для списка пользователей на канале
- и т.д.

Для реализации оконного клиента можно воспользоваться библиотекой ncurses, в машинном зале она установлена.

Для крайне заинтересованных студентов: можно воспользоваться qt на языке C++ или MFC и аналогами.