Лабораторная работа №3
Тема: Разработка программ TCP-клиент и TCP-сервер.

Студента группы МФ-21
Чистякова Артема

1.

Разработать программы TCP эхо-сервер и TCP клиент. Суть работы TCP эхо-сервера заключается в следующем: сервер принимает строку клиента выводит ее на экран с указанием IP-адреса компьютера клиента и отправляет ее назад клиенту.

Усовершенствовать программу эхо-сервера сделав ее многопоточной.

Решение

Прикреплено.

2.

Описать логику взаимодействия TCP-клиента и TCP-сервера (общая схема с указанием функций взаимодействия).

Решение

Первым действием следует «поднять» сервер.

Вызовем функцию **socket** на сервере, которая принимает базисный протокол (IPv4…) и протокол верхнего уровня (TCP…), и возвращает дескриптор файла, который ссылается на конечную точку соединения. После вызова функции socket к созданному сокету следует «привязать» IP адрес и порт. Вызовем функцию **bind**, которая принимает сокет и некоторую структуру, несущую адрес и порт, и ассоциирует информацию структуры с этим сокетом.

Сокет создан. Вызовем функцию **listen**, которая наделит переданный сокет возможностью принимать запросы на подключение.

Теперь подключим клиента.

На стороне клиента нужно создать сокет. Вызовем функцию socket с точно такими же параметрами, что и на сервере. Далее, вместо функций bind и listen вызовем функцию **connect**, в которую передадим созданный сокет и структуру, хранящую IP адрес и порт сервера. Данная функция отправит запрос на подключение клиента к серверу.

Чтобы установить соединение, сервер должен обработать запрос клиента на подключение. Для этого используем функцию **accept**, которая принимает сокет сервера, блокирует его, и при входящем подключении возвращает сокет подключаемого клиента. Теперь соединение установлено.

Чтобы начать «общение» по образовавшейся сети, следует использовать функции send и recv. **Send** принимает сокет назначения, буфер информации и длину этого буфера. Сообщение, хранящееся в буфере, будет отправлено «по сети» на сокет назначения. Если размер сообщения превосходит размер буфера сокета, то сокет блокируется. Чтобы принять сообщения на «другой» стороне, воспользуемся функцией **recv**, которая принимает сокет из которого нужно прочесть сообщение, буфер для хранения данных и его длину. Если на момент вызова функции recv буфер сокета пустой (нет входящих сообщений), то сокет блокируется. Функция возвращает фактическое количество принятых байт, так как не всегда возможно заполнить буфер полностью.

Чтобы «закрыть» соединения, вызовем функцию **close**, принимающую сокет для разрыва соединения.

3.

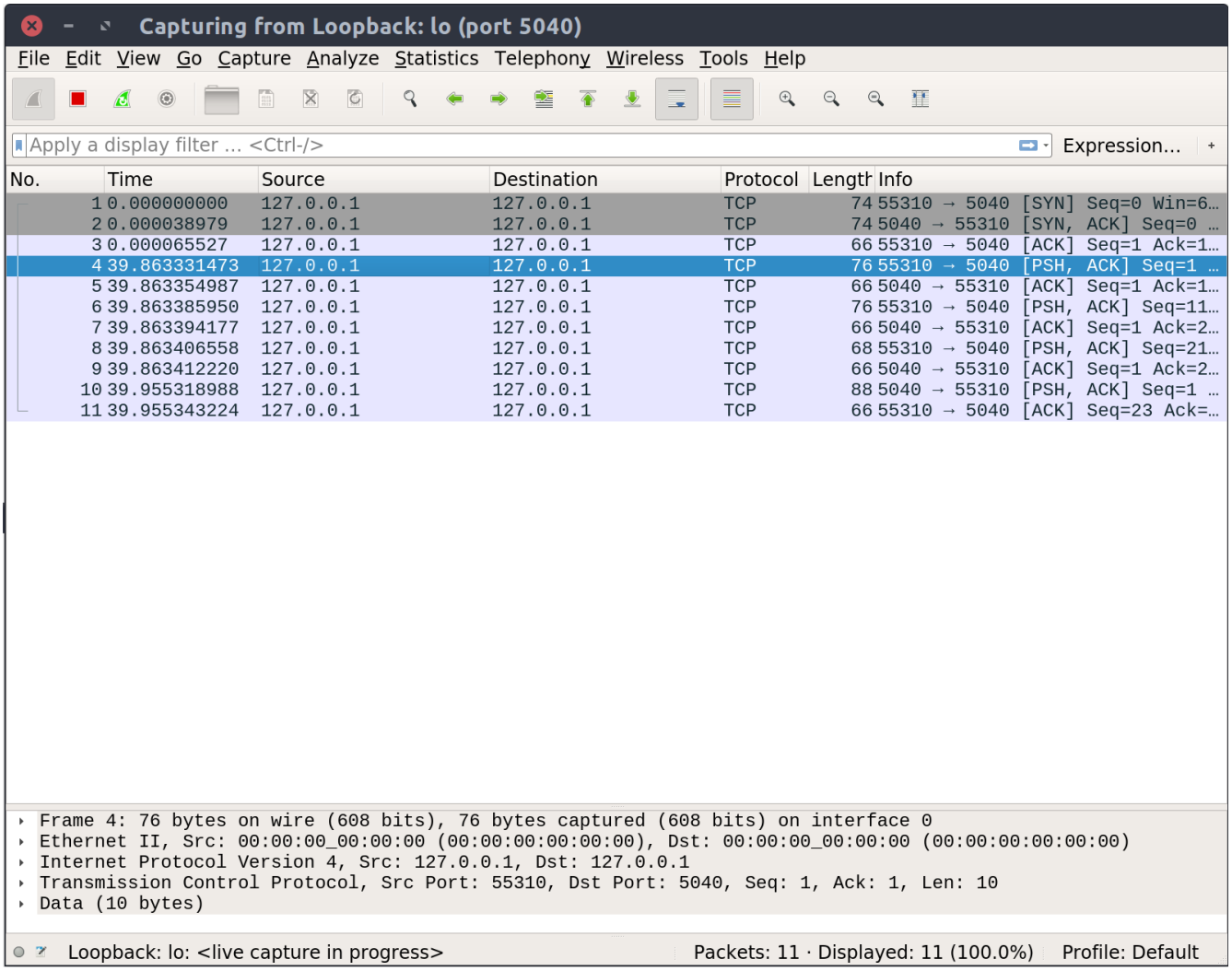
Описать зависимости между буфером клиента, отправляемой строкой и количеством отправляемых пакетов.

Решение

Размер входящего буфер сервера = **1000** байт.
Размер исходящего буфера сервера = **размер сообщения**.

Размер входящего буфер клиента = **10** байт.
Размер исходящего буфера клиента = **10** байт.

Сообщение: **12345678909876543 21**



Первые **3** пакета – установка соединения или «тройное рукопожатие».

Далее идут **6** чередующихся пакетов с информацией от клиента и подтверждением о получении сообщения от сервера.

2 последних пакета – «эхо» сервера и подтверждение клиента о получении сообщения.

Все сообщения в программе организованы следующим образом:

Самая первая строка сообщения хранит размер передаваемой информации и символ новой строки «\n». После располагается сама информация.

Таким образом размер передаваемого сообщения = длина информации + длина числа, которое описывает длину информации, + 1 байт на символ новой строки.

Или **msgLen = info.length() + to_string(info.length()).length() + 1** байт.

Если размер буфера = **size**, то количество пакетов для доставки сообщения будет составлять:

total = (msgLen + size - 1) / size * 2.

4.

Обосновать необходимость использования многопоточного TCP-сервера.

Решение

Необходимость многопоточного сервера обусловлена тем, что используемые нами сокеты объявлены как **блокирующиеся**. То есть, как было сказано ранее, такие функции как: **accept**, **send**, **recv** по разным причинам могут блокировать сокет на некоторое неопределенное время, что соответственно блокирует и поток в котором обрабатывается данные функции.

Рассмотрим ситуацию, когда сервер в одном потоке обрабатывает **15** клиентов. Допустим, что у клиента **6** возникли неполадки с соединением. Плохое подключение к интернету влечет за собой нестабильную отправку пакетов, что может привести к блокировке сокета (потока) на стороне сервера, а это значит, что остальные **9** клиентов не будут своевременно обработаны.

Решением данной проблемы как раз и есть комбинирование функции **select**, структуры **fd_set** и **многопоточности**.

5.

Что изменится во взаимодействии клиента и сервера, если буфер на клиенте и буфер на сервере будут иметь разные размеры? Почему?

Решение

На самом деле, на работу клиента и сервера размеры буферов никак не должны влиять: правильно написанные приложения должны следовать этому инварианту. Но размеры буферов могут повлиять на скорость работы приложения, а точнее, на скорость передачи данных по сети.

6.

Почему функцию accept не рекомендуется помещать внутрь потока взаимодействия с клиентом на многопоточном сервере?

Решение

Функция accept может заблокировать переданный сокет и, соответственно, поток в котором эта функция была вызвана. Если поместить обработку клиента в поток обработки функции accept, то может возникнуть ситуация, когда клиент из-за блокировки не будет своевременно обработан.

7.

Всегда ли необходимо закрывать сокет? Почему?

Решение

Сокет необходимо закрывать всегда. Сокет – это ресурс, а ресурсы требуют освобождения после завершения работы с ними. Закрытие сокета влечет за собой: освобождение памяти, выделенной под дескриптор сокета, разблокировку всех потоков, заблокированных на данном сокете, освобождение внутреннего буфера сокета.