Курс «Информационные сети и телекоммуникации»

Лабораторная работа № 3 «Неблокирующие сокеты»

Цель работы

Освоение работы с сокетами в неблокирующем режиме путем опроса готовности.

Подготовка к лабораторной работе

Завершить выполнение задания ЛР № 2 «Блокирующие потоковые сокеты».

Задание на лабораторную работу

Требуется усовершенствовать программу-сервер ЛР № 2, сделав её асинхронной, то есть способной обслуживать нескольких подключенных клиентов одновременно при помощи неблокирующих сокетов и без применения многопоточности.

1. **Изменить программу-сервер** для работы следующим образом:
    1.1. Запросить у пользователя адрес IP и порт.
    1.2. Создать сокет и выполнить его привязку к заданному адресу.
    1.3. Завести список подключенных клиентов (см. указания к выполнению).
    1.4. Циклически:
        1.4.1. Сформировать списки сокетов, на которых требуется должаться тех или иных событий из п. 1.4.2.
        1.4.2. Дождаться наступления одной или нескольких возможностей:
            - отключения ранее подключенного клиента[1];
            - подключения нового клиента;
            - принять запросы клиентов, от которых они ожидаются;
            - отправить данные клиентам, которым это необходимо.
        1.4.3. Принять входящее подключение, напечатать на экране адрес и порт клиента, добавить клиента в список подключенных.
        1.4.4. Отправить клиентам все данные, какие будет возможность передать.
        1.4.5. Принять все данные, поступившие от клиентов. Каждый целиком принятый запрос обработать, подобно тому, как делалось в ЛР № 2. Отличие: что отправку ответа нужно запланировать, но не начинать до очередной итерации цикла (см. указания к выполнению).
        1.4.6. Корректно (gracefully) завершить соединения с отключившимися клиентомами и удалить их из списка подключенных.

[1] Технически это возможность принять от клиента данные длиной 0.

2. **Проверить корректность работы программ** с файлами размером не менее сотен килобайт (например, с изображениями) при одновременном подключении не менее, чем двух клиентов (можно добавить задержку в цикл).

Указания к выполнению лабораторной работы

Архитектура приложения

С точки зрения сервера, подключенный клиент обладает:

- сокетом;
- состоянием приема:
  - массивом байт под принимаемый пакет (включая только тип и данные);
  - признаком, что уже принята длина пакета (типа и данных);
  - количеством байт, которые необходимо принять всего;
  - количеством байт, которые уже приняты;
- состоянием отправки:
  - массивом байт всех пакетов, предназначенных для отправки;
  - количеством байт, которые необходимо отправить всего;
  - количеством байт, которые уже отправлены.

Перечисленное сохраняется между итерациями цикла, индивидуально для каждого клиента и управляется только им. Целесообразно поэтому завести класс клента `Client`, в который перенести код обслуживания клиента, адаптировав его для работы с данными отдельного клиента.

Рационально из основного цикла сервера (в котором вызывается `select()`) вызывать у объекта-клиента метод для обработки наступившего события. Например, если стало возможно принять данные, вызывать метод-обработчик `onReceive()`. Псевдокод:

```
while (is_running) {
      FD_ZERO(&receivers); FD_ZERO(&senders); // …
      for (const Client& client : clients) {
            FD_SET(client.getSocket(), &receivers);
            if (client.isSending())
                  FD_SET(client.getSocket(), &senders);
      }
      // Опущена работа с сокетом-слушателем и обработка ошибок.
      select(..., &receivers, &senders, …);
      for (Client& client : clients) {
            if (FD_ISSET(client.getSocket(), &receivers))
                  client.onReceive();
            // Аналогично для других событий.
      }
}
```

Метод `isSending()` позволяет проверить, что клиенту требуется доотправить данные (то есть количество уже отправленных байт меньше, чем нужно отправить всего).

Логика работы метода-обработчика должна учитывать, что прием или отправка пакета может выполняться за несколько вызовов. Например, метод `onReceive()` может работать примерно следующим образом:

циклически:
если длина еще не принята до конца, **то** попытаться допринять её;
иначе, если не весь пакет еще принят, **то** попытаться допринять его;
если длина не была принята до конца, **и** теперь её прием окончен, **то**
приготовиться к приему типа и данных;
иначе, если пакет принят целиком, **то**
обработать его;
приготовиться к приему поля длины.

«Попытаться допринять» означает, что необходимо сделать один вызов `recv()`, чтобы принять все оставшиеся данные. Случаев неудачи может быть два: или новые данные еще не прибыли (следует выйти из функции, прием будет завершен в последующие вызовы), или произошла ошибка (следует выйти из функции и прекратить обслуживание клиента). Иных выходов из цикла можно не предусматривать. Очевидно, даже в случае успешного вызова могут быть приняты не все данные.

Подготовка к приему поля длины — это сброс счетчика принятых байт числа-длины, а также флага (**bool**), означающего, что прием длины закончен. Подготовка к приему типа и данных, соответственно, — установка упомянутого флага.

Обработка пакета включает формирование ответа и его отправку. Здесь «отправка» означает не непосредственный вызов `send()`, а добавление пакета-ответа к данным, которые необходимо (когда-то) отправить. Реальную отправку следует производить методом `onSend()`, вызываемом из основного цикла, когда это возможно.
Чтобы метод `onReceive()` не был слишком велик и сложен, можно формирование ответа вынести в отдельную функцию.

Возможный план работы

1. Выделить класс клиента и функцию (или метод) формирования ответа на запрос. Методы `onSend()` и `onReceive()` на этом этапе содержат старый блокирующий код. Завести в программе один объект-клиент и безусловно вызывать у него метод `onReceive()`; метод `onSend()` вызывать из функции формирования ответа.
2. Перевести сокет-слушатель в неблокирующий режим и ожидать подключений при помощи функции `select()`. Обслуживание клиента на данном этапе выполняется блокирующим образом.

3. Реализовать метод `onSend()` неблокирующим образом (как более простой). Перевести сокет клиента в неблокирубщий режим и обрабатывать при помощи функции `select()` *все* его события, включая прием.
   Формально на данном этапе программа некорректна, так как алгоритм приема по-прежнему блокирующий, но на практике редкие короткие запросы зачастую передаются одним фрагментом.
4. Реализовать метод `onReceive()` неблокирующим образом.
5. Завести перечень подключенных клиентов и перейти к их одновременному обслуживанию в цикле. На данном этапе приложение-сервер можно тестировать с несколикими подключенными клиентами.

Контрольные вопросы

1. Чем неблокирующий режим работы сокетов отличается от блокирующего?
2. Каковы преимущества и недостатки асинхронного режима работы сокетов, в каких случаях имеет смысл его использовать?
3. Каким образом выполняется перевод сокета в асинхронный режим работы? Возможен ли обратный перевод? Доступен ли асинхронный режим для сокетов, принимающих подключения, и для подключающихся сокетов (клиента)?
4. Что такое множество сокетов (`fd_set`), зачем оно используется и какие средства работы с ним имеются в API сокетов?
5. Как работает и интерпретирует свои параметры функция `select()`?
6. Каковы отличия в обработке ошибок для сокетов в неблокирующем режиме?