Лекция IV, Основные принципы разработки сетевых приложений

Курс читает Рогозин Николай Олегович, кафедра ИУ-7

Клиент-серверное приложение

- Типичное сетевое приложение состоит из двух частей — **клиентской** и **серверной** программ, работающих на двух различных конечных системах.
- Когда запускаются эти две программы, создаются два процесса: клиентский и серверный, которые взаимодействуют друг с другом, производя чтение и запись в сокеты.
- Таким образом, при создании сетевого приложения основная задача разработчика — написать коды как для клиентской, так и для серверной частей.

Одноранговая архитектура

- применение серверов или центров обработки сведено до минимума или вообще до нуля.
- Вместо них приложения используют непосредственное взаимодействие между парой соединенных хостов, называемых пирами

Открытые приложения

- Для открытых приложений функционирование описывается стандартами выбранного протокола, такими как RFC или другие документы: правила, определяющие операции в нем, известны всем.
- В такой реализации клиентская и серверная программы должны соответствовать правилам, продиктованным документами RFC.
- Например, клиентская программа может быть реализацией клиентской части протокола FTP, определенного в RFC 959; аналогично, серверная программа может быть частью реализации протокола FTP сервера, также описанного в документе RFC 959.

Проприетарные приложения

- Другой тип сетевых приложений представляют проприетарные (закрытые) приложения. В этом случае клиентские и серверные программы используют протокол прикладного уровня, который открыто не опубликован ни в RFC, ни в каких-либо других документах.
- Разработчик (либо команда разработчиков) создает клиентские и серверные программы и имеет полный контроль над всем, что происходит в коде.
- Но так как в кодах программы не реализован открытый протокол, другие независимые разработчики не смогут написать код, который бы взаимодействовал с этим приложением.

Сокеты Беркли

- API для разработки сетевых приложений на языке C
- Первое появление – в ОС UNIX BSD 4.1 (1982), затем в UNIX BSD 4.2 (1983)
- Представляют де-факто стандарт абстракции для сетевых сокетов
- Относятся к транспортному уровню, используют протоколы TCP и UDP
- Наиболее широко используются в сетях на базе TCP/IP

Ввод-вывод сетевых данных

- Поскольку сокеты представляют реализацию протокола TCP/IP в среде Unix, использует те же системные вызовы, что и остальные программы этой среды
- Системные вызовы выглядят как последовательный цикл, состоящий из операций типа открыть-считать-записать-закрыть
- Перед чтением файл всегда должен быть открыт. По окончании записи файл всегда закрывается.
- Одни и те же системные вызовы используются для работы с различной средой: принтером, модемом, дисплеем, файлами.

Ввод-вывод сетевых данных

- На первых порах разработки интерфейса сокетов пытались заставить сетевой ввод-вывод функционировать так же, как и любой другой ввод-вывод UNIX. Т.е. считывать и записывать данные в цикле открыть-считать-записать-закрыть.
- Однако возникла проблема: системная модель API не подходила для клиент-серверной архитектуры, т.к. не позволяла реализацию серверной части.

Ввод-вывод сетевых данных

- Обычная система ввода-вывода Unix не умеет пассивно вводить и выводить данные
- Стандартные функции ввода-вывода UNIX также плохо умеют устанавливать соединения. Как правило, они пользуются фиксированным адресом файла и устройства для обращения к нему.
- Адрес файла или устройства для каждого компьютера — постоянная величина. Не подходит для датаграмм.
- Соединение (или путь) к файлу или устройству доступно на протяжении всего цикла запись-считывание — то есть до тех пор, пока программа не закроет соединение.

Новый API

- От внесения изменений в существующую систему в итоге отказались в пользу разработки новой
- Тем не менее, ряд сходств остался: дескриптор сокетов в интерфейсе называется дескриптором файла, а информация о сокете хранится в той же таблице, что и дескрипторы файлов

Работа веб-сокетов

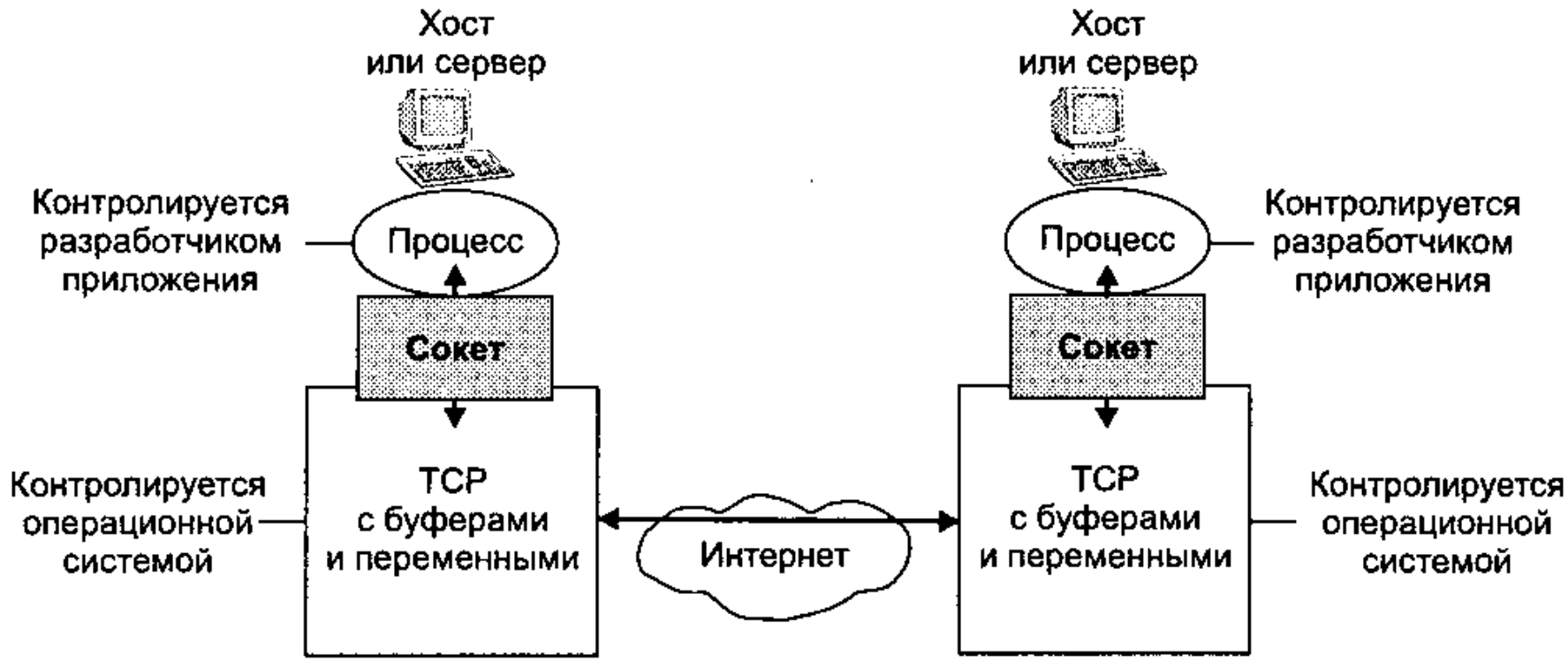


Рис. 2.18. Взаимодействие процессов при помощи TCP-сокетов

Адресация процессов

- Хост можно идентифицировать по его IP-адресу.
- IP-адрес представляет собой 32-разрядное число, которое однозначно определяет хост.
- Отправляющий процесс должен идентифицировать процесс принимающий (а точнее, принимающий сокет), запущенный на хосте-получателе.
- Наиболее популярным сетевым приложениям назначены определенные номера портов. Например, веб-сервер идентифицируется портом 80. Процесс почтового сервера, использующий протокол SMTP, использует порт 25.

Службы транспортного уровня

- Надежная передача данных
- Пропускная способность
- Время доставки
- Безопасность

Пропускная способность

- Мгновенная пропускная способность в любой момент времени — это скорость (в бит/с), с которой хост Б получает файл.
- Многие приложения, включая большинство систем в одноранговых сетях с разделением файлового доступа, отображают мгновенную пропускную способность в интерфейсе пользователя во время загрузки.
- Если файл содержит F бит, а передача занимает T секунд, то средняя пропускная способность для передачи файла равна F/T бит/с.

Пропускная способность

- Транспортный протокол должен предоставлять гарантированную доступную пропускную способность, то есть доставку данных с определенной минимальной скоростью.
- Используя такую службу, приложение может запрашивать гарантированную пропускную способность r бит/с, и транспортный протокол должен будет заботиться о том, чтобы доступная скорость передачи данных была не меньше r бит/с.
- Например, если приложение IP-телефонии кодирует голосовые сообщения со скоростью 32 Кбит/с, то используется соотв. пропускная способность

Время доставки

- Протокол транспортного уровня может также обеспечивать гарантии относительно времени доставки сообщений.
- Временные гарантии тоже предоставляются в различной форме.
- Например, протокол может гарантировать, что каждый бит, отправленный передающей стороной в сокет, приходит на сокет получателя не более чем через 100 мс.

Безопасность

- Транспортный протокол может предоставлять приложениям одну или несколько служб, относящихся к безопасности.
- Транспортный протокол на передающем хосте способен шифровать все данные, отправленные процессом-источником, а затем на принимающем хосте расшифровывать их перед доставкой процессу-получателю.
- Это обеспечит конфиденциальность между двумя процессами.

Приложение	Потери данных	Пропускная способность	Чувствительность к потере данных
Передача файлов/ загрузка	Не допускаются	Эластичное	Нет
Электронная почта	Не допускаются	Эластичное	Нет
Веб-документы	Не допускаются	Эластичное (несколько Кбит/с)	Нет
IP-телефония/ Видео-конфе-ренции	Допускаются	Аудио: От нескольких Кбит/с до 1 Мбит/с Видео: От 10 Кбит/с до 5 Мбит/с	Да, сотни миллисекунд
Потоковый аудио и видеоконтент	Допускаются	См. предыдущее	Да, несколько секунд
Интерактивные игры	Допускаются	От нескольких до 10 Кбит/с	Да, сотни миллисекунд
Мгновенные сообщения	Не допускаются	Эластичное	Да и нет

Популярные приложения и используемые ими протоколы трансп. уровня

Приложение	**Протокол прикладного уровня**	**Базовый транспортный протокол**
Электронная почта	SMTP	TCP
Удаленный терминальный доступ	Telnet	TCP
Всемирная паутина	HTTP	TCP
Передача файлов	FTP	TCP
Потоковый мультимедийный контент	HTTP (например, YouTube)	TCP
IP-телефония	SIP , RTP или проприетарное (например, Skype)	UDP или TCP

Абстракция сокетов

- Сетевое соединение — это процесс передачи данных по сети между двумя компьютерами или процессами.
- Сокет — конечный пункт передачи данных, абстракция, обозначающая одно из окончаний сетевого соединения.
- Каждая из программ, устанавливающих соединение, должна иметь собственный сокет.
- Связь может быть ориентирована на соединение, либо без соединения.

Абстракция сокетов

- Когда программе нужен сокет, она формирует его характеристики и обращается к API, запрашивая у сетевого ПО его дескриптор.
- Структура таблицы с описанием параметров сокета весьма незначительно отличается от структуры таблицы с описанием параметров файла.

Дескриптор сокета и дескриптор файла

- дескриптор файла указывает на определенный файл (уже существующий или только что созданный) или устройство
- дескриптор сокета не содержит каких-либо определенных адресов или пунктов назначения сетевого соединения, в отличие от дескрипторов в большинстве ОС
- Программы, работающие с сокетами, сначала образуют сокет и только потом соединяют его с точкой назначения на другом конце сетевого соединения.

Операции сокетов Беркли

Операция	Назначение
Socket	Создать новый сокет
Bind	Связать сокет с IP-адресом и портом
Listen	Объявить о желании принимать соединения
Connect	Установить соединение
Accept	Принять запрос на установку соединения
Send	Отправить данные по сети
Receive	Получить данные из сети
Close	Закрыть соединение

Создание сокета

- Интерфейс сокетов позволяет использовать два типа протоколов: с установлением соединения и без
- Процессы создания сокета и соединения происходят раздельно
- Для создания сокета вызывается команда SOCKET, возвращающая дескриптор сокета, в котором содержится описание свойств и структуры сокета (так же как и в случае с файлом)

Создание сокета, cont.

- Пример: **socket_handle = socket(protocol_family, socket_type, protocol);**
- Первый параметр: группа или семейство, к которому принадлежит протокол, например семейство TCP/IP.
- Второй параметр, тип сокета, задает режим соединения: датаграммный или ориентированный на поток байтов.
- Третий параметр определяет протокол, с которым будет работать сокет, например TCP.

Свойства протоколов и адресов

- Существует ряд символьных констант (макроопределений) для указания группы протоколов.

1) **PF_INET**, например, определяет семейство протоколов TCP/IP.
2) **PF_UNIX** определяет семейство внутренних протоколов ОС UNIX

- Подобные константы существуют для определения семейств адресов, начинающиеся на префикс “AF_”

1) **AF_INET** обознает семейство TCP/IP
2) **AF_UNIX** обозначает семейство адресов файловой системы UNIX

Свойства протоколов и адресов, cont.

- Интерфейс для работы с адресами представляет собой обобщение в соответствии с различием форматов различных сетей
- Из-за тесной связи семейств протоколов и адресов существует заблуждение о том, что это одно и то же
- Однако на сегодняшний день применение AF_INET и PF_INET имеет одинаковый результат

Тип соединения

- Соединение делится на два режима: ориентированные на установление постоянной связи и не требующие этого
- TCP использует для передачи непрерывный поток байт без деления на блоки, в то время как UDP – дейтаграммы, отдельные пакеты данных
- Второй параметр вызова функции socket определяет тип требуемого соединения : **SOCK_DGRAM** обозначает датаграммы, а **SOCK_STREAM** — поток байтов.

Выбор протокола

- Семейство TCP/IP состоит из нескольких протоколов, например IP, ICMP, TCP и UDP. Любое семейство состоит из набора протоколов, которыми пользуются сетевые программисты. Третий параметр функции socket позволяет выбрать тот протокол, который будет использоваться вместе с сокетом. Как и в случае остальных параметров, протокол задается символьной константой.
- В сетях TCP/IP все константы начинаются с префикса **IPPROTO_**. Например, протокол TCP обозначается константой **IPPROTO_TCP**. Символьная константа **IPPROTO_UDP** обозначает протокол UDP.

Выбор протокола, cont.

- Пример вызова функции socket:

  socket(PF_INET, SOCK_STREAM,IPPROTO_TCP);

- Данный вызов сообщает интерфейсу сокетов о том, что программа желает использовать семейство протоколов Интернет (**PF_INET**), протокол TCP (**IPPROTO_TCP**) для соединения, ориентированного на поток байтов (**SOCK_STREAM**).

Структура данных сокета

- Семейство протоколов
- Тип сервиса
- Локальный IP адрес
- Удалённый IP адрес
- Локальный порт протокола
- Удалённый порт протокола

Структура данных сокета

- Каждый раз, когда программа вызывает функцию-socket, реализация сокетов отводит машинную память для новой структуры данных, а затем размещает в ней семейство адресов, тип сокета и протокола.
- В таблице дескрипторов размещается указатель на эту структуру.
- Дескриптор, полученный вашей программой от функции socket, является индексом (порядковым номером) в таблице дескрипторов.

Настройка сокета

- Каждая сетевая программа вначале создает сокет, вызывая функцию socket. При помощи других функций сокет конфигурируется или настраивается так, как это нужно сетевой программе.
- Один из двух типов сетевых служб: датаграммная или ориентированная на поток байтов.
- Один из двух типов поведения программы: клиент или сервер.
- Пять блоков информации, описывающей соединение: протокол, местный и удаленный IP-адреса, номера местного и удаленного портов.

Соединение сокетов

- Ориентированная на соединение программа-клиент вызывает функцию connect, чтобы настроить сокет на сетевое соединение. Функция connect размещает информацию о локальной и удаленной конечных точках соединения в структуре данных сокета.
- Функция connect требует, чтобы были указаны: дескриптор сокета (указывающий на информацию об удаленном компьютере) и длина структуры адресных данных сокета.

Функция connect

- **result = connect(socket_handle, remote_socket_address, address_length);**
- **Первый параметр** функции connect, дескриптор сокета, получен ранее от функции socket.
- **Второй параметр** функции connect — адрес удаленного сокета, является указателем на структуру данных адреса сокета специального вида. Информация об адресе, хранящаяся в структуре, зависит от конкретной сети, то есть от семейства протоколов, которое мы используем.
- **Третий параметр** функции connect, длина адреса, сообщает интерфейсу длину структуры данных адресов удаленного сокета (второй параметр), измеренную в байтах

Функция bind

- Функция **bind** интерфейса сокетов позволяет программам связать локальный адрес (совокупность адресов локального компьютера и номера порта) с сокетом.
- Следующий оператор иллюстрирует вызов функции bind:

result = bind(socket_handle, local_socket_address, address_length) ;

Передача данных через сокет

- После того как сокет сконфигурирован, через него можно установить сетевое соединение. Процесс сетевого соединения подразумевает посылку и прием информации. Интерфейс сокетов включает несколько функций для выполнения этих обеих задач.
- Интерфейс сокетов Беркли обеспечивает пять функций для передачи данных через сокет. Эти функции разделены на две группы.
- Трем из них требуется указывать адрес назначения в качестве аргумента, а двум остальным — нет. Основное различие между двумя группами состоит в их ориентированности на соединение.

Функция write

- **result = write(socket_handle, message_buffer, buffer_length);**

- **Первый параметр**, дескриптор сокета - он обозначает структуру в таблице дескрипторов, содержащую информацию о данном сокете.
- **Второй параметр** функции write, буфер сообщения, указывает на буфер, то есть область памяти, в которой расположены предназначенные для передачи данные.
- **Третий параметр** вызова функции обозначает длину буфера, то есть количество данных для передачи.

Функция writev

- **result = writev(socket_handle, io_vector, vector_length);**

- **Первым параметром** требуется указывать дескриптор сокета, также как в функции write.
- **Второй параметр**, вектор ввода-вывода, указывает на массив указателей.
- **Третий параметр** функции writev определяет количество указателей в массиве указателей, заданном вектором ввода-вывода.

Функция writev, cont.

- Предположим, что данные для передачи располагаются в различных областях памяти.
- В этом случае каждый член массива указателей представляет собой указатель на одну из областей памяти, содержащей данные для передачи.
- Когда функция writev передает данные, она находит их по указанным прикладной программой в массиве указателей адресам.
- Данные высылаются в том порядке, в каком их адреса указаны в массиве указателей.

Функция send

- **result = send(socket_handle, message_buffer, buffer_length, special_flags) ;**
- **Первый параметр**: дескриптор, описывающий подключенный сокет
- **Второй параметр**: указатель на буфер передаваемых данных
- **Третий параметр**: длина, в байтах, данных в буфере на который указывает указатель во втором параметре
- **Четвертый параметр**: набор флагов, определяющих параметры вызова

Функция sendto

- Следующий оператор демонстрирует вызов функции sendto:
- **result = sendto(socket_handle, message_buffer, buffer_length, special_flags, socket_address_structure, address_structure_length) ;**
- Первые четыре те же, что и в функции send.
- Пятый параметр, структура адреса сокета, определяет адрес назначения.
- Шестой параметр, длина структуры адреса сокета, - размер этой структуры в байтах.

Функция sendmsg

- **result = sendmsg(socket_handle, message_structure, special_flags);**
- sendmsg позволяет использовать гибкую структуру данных вместо буфера, расположенного в непрерывной области памяти.
- Как и в функции writev, структура сообщения содержит указатель на массив адресов памяти.

Прием данных через сокет

- В интерфейсе сокетов есть пять функций, предназначенных для приема информации. Они называются: read, readv, recv, recvfrom, recvmsg и соответствуют функциям, использующимся для передачи данных.
- Например, функции recv и send обладают одинаковым набором параметров.
- Точно так же одинаков набор параметров и у функций writev и readv. Функция writev передает данные, а readv — принимает. И та и другая позволяют задать массив адресов памяти, где располагаются данные.
- Функции recvfrom и recvmsg соответствуют функциям sendto и sendmsg.

Соответствие функций приема и передачи

Функция передачи данных	Соответствующая функция приема данных
send	recv
write	read
writev	readv
sendto	recvfrom
sendmsg	recvmsg

Функция listen

- **result = listen(socket_handle, queue_length);**
- переводит сокет в пассивный режим ожидания
- подготавливает его к обработке множества одновременно поступающих запросов, организуя их в виде очереди
- Первый параметр: дескриптор сокета
- Второй параметр: длина очереди
- В настоящее время максимальная длина очереди равна пяти, однако задание длины 1 или 2 также полезно и позволит серверу не отвергнуть запрос если он не справился с обработкой

Функция accept

- позволяет серверу принять запрос на соединение, поступивший от клиента. После того как установлена входная очередь, программа-сервер вызывает функцию accept и переходит в режим ожидания (паузы), ожидая запросов.
- После того как установлена входная очередь, программа-сервер вызывает функцию accept и переходит в режим ожидания (паузы), ожидая запросов.

Функция accept, cont.

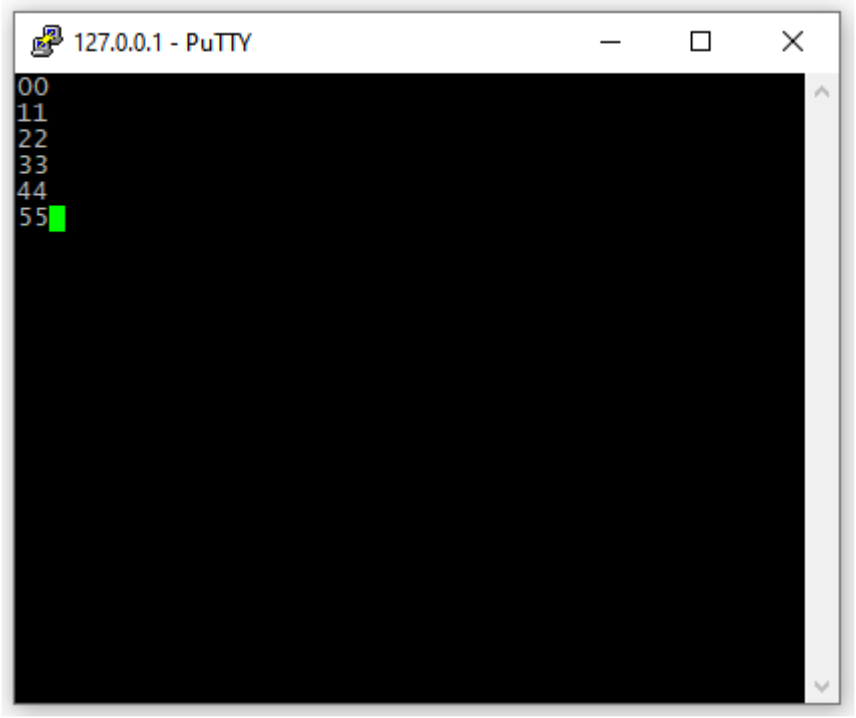
- **result = accept(socket_handle, socket_address, address_length);**
- Дескриптор сокета описывает сокет, который будет прослушиваться сервером.
- В момент появления запроса реализация сокетов заполняет структуру адреса (на которую указывает второй параметр) адресом клиента, от которого поступил запрос.
- Реализация сокетов заполняет также третий параметр, размещая в нем длину адреса.

Функция select

- позволяет одиночному процессу следить за состоянием сразу нескольких сокетов.
- **result = select(number_of_sockets, readable_sockets, writeable_sockets, error_sockets, max_time);**
- Первый параметр, количество сокетов (number of sockets), задает общее количество сокетов для наблюдения.
- Параметры readable-sockets, writeable-sockets и error-sockets являются битовыми масками, задающими тип сокетов.

Пример сокета-сервера

```python
import socket

serv_sock = socket.socket(socket.AF_INET,      #Первый сокет - слушающий
                          socket.SOCK_STREAM,
                          proto=0)

serv_sock.bind(('127.0.0.1', 8700))           #Слушающий сокет привязывается к порту

serv_sock.listen(10)

client_sock, client_addr = serv_sock.accept() #Второй сокет отвечает за прием и передачу данных

while True:
    data = client_sock.recv(1024)
    if not data:
        break
    client_sock.sendall(data)

client_sock.close()
```



Пример сокета-клиента

```
import socket
import struct
client_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_sock.connect(('127.0.0.1', 8700))

client_sock.sendall(bytes(str(123), 'utf8'))
data = client_sock.recv(1024)
strings = str(data, 'utf8')
num = int(strings)
client_sock.close()

print('Received: ' + str(num))
```