

Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра автоматизированных систем управления (АСУ)

Работа с Socket

Отчет по лабораторной работе №1
по дисциплине «Сети и Телекоммуникации»

Выполнила студент гр. 430-2

_____Лузинсан А.А.

«____»_____2022 г.

Проверил: к.т.н. каф. АСУ

_____Суханов А.Я.

«____»_____2022 г.

Томск 2022

Содержание

Введение.....	3
2 Теория.....	4
2.1 Модель OSI.....	4
2.2 Сетевые сервисы.....	5
2.3 Классификация сетей.....	5
2.4 Интерфейсы и протоколы.....	7
2.5 Облачные сервисы.....	7
2.6 Сетевые утилиты.....	8
3 Ход работы.....	9
3.1 TCP клиент-серверное приложение на C++ в Unix системе.....	9
3.1.1 Socket.....	9
3.1.2 Bind.....	11
3.1.3 Установка соединения (сервер).....	12
3.1.4 Установка соединения (клиент).....	14
3.1.5 Обмен данными.....	14
3.1.6 Закрытие сокета.....	16
3.1.7 Обработка ошибок.....	17
3.2 TCP клиент-серверное приложение на Python в Unix системе.....	17
3.2.1 Установка соединения (сервер).....	17
3.2.2 Установка соединения (клиент).....	18
Заключение.....	19
Результаты работы программ.....	20
Листинг программ.....	21
TCP-сервер на C++.....	21
TCP-клиент на C++.....	25
TCP-сервер на Python.....	27
TCP-клиент на Python.....	27

Введение

Цель: получить практические навыки по реализации клиент-серверного приложения и реализовать прикладную задачу заключенную в задаче по варианту.

Задание: по указаниям из методического пособия реализовать ТСР клиент и сервер. Реализацию написать на языках программирования Си и Python. Обеспечить обмен сообщениями между клиентом и сервером. Выполнить задание по варианту.

Вариант №10: получить от сервера по DNS адресу набор IP-адресов. Использовать `gethostbyname()`.

2 Теория

2.1 Модель OSI

Модель OSI («Basic Reference Model Open Systems Interconnection model») - это эталонная модель взаимодействия открытых систем (стека сетевых протоколов OSI/ISO), посредством которой различные сетевые устройства могут взаимодействовать друг с другом. Блок данных информации на каждом уровне представлен в виде единиц данных протокола - PDU («Protocol Data Unit»).

Модель определяет сетевые протоколы, распределяя их на 7 логических уровней, которые представлены в таблице 2.1.

Таблица 2.1 — Уровни модели OSI их предназначение и примеры протоколов.

Уровень	Единица данных протокола (PDU)	Функции	Сетевые протоколы
7. Прикладной	Сообщения	Интерфейсы взаимодействия пользовательских процессов с сетью	HTTP, FTP, POP3, SMTP, WebSocket, TELNET
6. Представление		Представление и шифрование данных	ASCII, JPEG, MIDI, PNG, MPEG, TIFF, X.25 PAD
5. Сеансовый		Управление сеансом связи	ZIP, SMPP, L2F, L2TP, H.245, PPTP
4. Транспортный	Сегменты/Датagramмы	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SST, SCTP, DCCP
3. Сетевой	Пакеты	Определение маршрута передачи пакетов данных между клиентами и логическая адресация	IPv4, IPv6, CLNP, IPsec

2. Канальный	Биты/ Кадры/ Фреймы	Проверка и исправления ошибок передачи фреймов данных между узлами, находящихся в одной локальной сети.	ATM, CAN, Ethernet, EAPS, IEEE 802.2, MPLS, PPP
1. Физический	Биты	Передача и прием потока байтов через физическое устройство. Физическое кодирование сигналов	Bluetooth, ITU, Wi-Fi, DSL, IPDA

2.2 Сетевые сервисы

Сетевой сервис (служба) - это приложение, работающее на прикладном уровне, которое обеспечивает хранение данных, управление, представление, связь или другие возможности, которые часто реализуются с использованием архитектуры клиент-сервер.

Основная цель: увеличение вычислительной мощности предоставляемой пользователю.

Примеры сетевых сервисов:

- Веб-серверы (Протоколы HTTP/HTTPS)
- FTP-серверы для обмена файлами (Протокол FTP)
- Электронная почта (Протокол IMAP)

2.3 Классификация сетей

По количеству узлов, диаметру и структуре различают локальную, муниципальную и глобальную сети, которые представлены в таблице 2.2.

Таблица 2.2 – Виды сетей по структуре

Вычислительная сеть	Характеристика
Локальная - LAN (Local Area Network)	- это объединение компьютеров, использующих один тип среды передачи данных на расстоянии ~1-2км между узлами, являющаяся коммуникационной системой, принадлежащей одной организации. Общая протяжённость ≤10км.

Муниципальная - MAN (Metropolitan area network)	- это объединение LAN различных организаций и физических лиц в пределах города, использующих различные среды передачи данных, но содержащее один или несколько магистральных каналов с большой пропускной способностью, к которым подключаются через сопрягающие устройства (шлюзы). Протяжённость сети — 10-100 км, а подключенность — 10-100 тыс. ед.
Глобальная - WAN (Wide Area Network)	– это любые сети, пересекающие границы городов, регионов, стран и континентов, объединяющие локальные и муниципальные сети. Число узлов и расстояние между узлами неограниченно

По организационному признаку различают сети рабочих групп, отделов, кампусов и корпоративные сети, представленные в таблице 2.3.

Таблица 2.3 — Классификация вычислительных сетей по организационному признаку

Вычислительная сеть	Характеристика
Сеть рабочей группы	Включают до 10-20 компьютеров.
Сеть отделов	Используются сравнительно небольшой группой сотрудников, работающих в одном отделе предприятия. Эти сотрудники решают некоторые общие задачи, например ведут бухгалтерский учет или занимаются маркетингом. Отдел может насчитывать до 100-150 сотрудников.
Сеть кампусов CAN (Campus Area Network, «campus» - студенческий городок)	Объединяет множество сетей различных отделов одного предприятия в пределах отдельного здания или одной территории, покрывающей площадь в несколько квадратных километров. Глобальные соединения не используются. Сотрудники каждого отдела предприятия получают доступ к некоторым файлам и ресурсам сетей

	<p>других отделов.</p> <p>Обеспечивает доступ к корпоративным базам данных, независимо от того, на каких типах компьютеров они располагаются.</p>
Корпоративная сеть (EWN - "Enterprise-Wide Networks" - сеть масштаба предприятия)	<p>Объединяет большое количество компьютеров на всех территориях отдельного предприятия.</p> <p>Может быть сложно связана и способна покрывать город, регион или континент.</p> <p>Число пользователей и компьютеров — >1000, число серверов — >100.</p> <p>Иногда используют глобальные сети.</p>

2.4 Интерфейсы и протоколы

Интерфейс — совокупность аппаратных и программных средств, предоставляемая на уровень выше, для взаимодействия с программой, устройством, функцией и т.д. текущего уровня

Протокол - набор правил, соглашений, сигналов, сообщений и процедур, определяющих взаимодействие между сопрягаемыми узлами.

Стандарты разрабатываются различными организациями, для эффективного взаимодействия элементов их сетей:

- Стандарты фирм
- Стандарты комитетов и объединений
- Национальные стандарты
- Международные стандарты

2.5 Облачные сервисы

XaaS (Anything as a service) - “Всё как сервис” - все услуги, связанные с облачными вычислениями и удаленным доступом

Paas (Platform as a service) - “Платформа как сервис” - потребитель получает доступ к использованию уже настроенных информационно-

технологических платформ: ОС, СУБД и т.д

Iaas (Infrastructure as a Service) - “Инфраструктура как услуга” - потребитель получает по подписке фундаментальные информационно-технологические ресурсы: виртуальные серверы, с заданной вычислительной мощностью и размером хранилищ данных

SaaS (Software as a Service) - “Программное обеспечение как услуга” - предоставляет пользователю веб приложение (например Google Docs)

2.6 Сетевые утилиты

Ping - утилита для проверки качества соединений в сетях на основе TCP/IP. Утилита отправляет запросы протокола ICMP указанному узлу сети и фиксирует поступающие ответы. Время между отправкой запроса и получением ответа позволяет определять двусторонние задержки по маршруту и частоту потери пакетов.

Ifconfig - используется для настройки сетевых интерфейсов ядра. Если аргументы не указаны, ifconfig отображает состояние текущих активных интерфейсов.

Traceroute - Служебная компьютерная программа, предназначенная для определения маршрутов следования данных в сетях TCP/IP

3 Ход работы

3.1 TCP клиент-серверное приложение на C++ в Unix системе

3.1.1 Socket

Socket - гнездо, разъём, виртуальный коммуникационный узел в операционной системе. При системном вызове `socket()` создаётся конечная точка для коммуникации и возвращается файловый дескриптор (натуральное число - ссылка на созданный сокет), или -1 в случае ошибки.

С каждым сокетом связываются три атрибута: *домен*, *тип* и *протокол*. Эти атрибуты задаются при создании сокета и остаются неизменными на протяжении всего времени его существования.

Для создания сокета в Unix системах требуется подключение следующих библиотек:

```
#include <sys/types.h>
#include <sys/socket.h>
```

Определение функции `socket` выглядит следующим образом:

```
int socket(int domain, int type, int protocol);
```

Домен (параметр `domain`) - семейство протоколов - стек протоколов - вид коммуникационной области - вся совокупность вертикальных протоколов (интерфейсов) и горизонтальных (сетевых) протоколов в сетевых системах.

Виды:

- PF_INET (Protocol family - Internet или AF_INET - Address family - Internet - синонимы) - семейство транспортных протоколов TCP/IP - IPv4 протоколы Интернет;
- PF_UNIX, PF_LOCAL - Локальное соединение - используется для связи процессов в одной машине - семейство внутренних протоколов Unix (Unix domain);

- PF_INET6 (IPv6), PF_PACKET, PF_APPLETALK, PF_ATMPVC, PF_AX25, PF_X25, PF_NETLINK и др.

Тип (параметр type) - вид интерфейса работы сокета - PDU, передающиеся по сети.

Виды:

- SOCK_STREAM (сегменты) - установление потокового двустороннего режима взаимодействия с помощью виртуального соединения, который обеспечивает последовательный и надёжный (за счёт проверки доставки сегментов от получателя) и не дублирующийся поток данных.
- SOCK_DGRAM - (Datagrams) - сохраняет границы сообщений, не поддерживают соединения. В PF_INET возможны дублирования, потеря и перестановка последовательности дейтаграмм, тогда как в PF_UNIX передача надёжна, а дейтаграммы не путаются.
- SOCK_RAW (доступ к низкоуровневу сетевому протоколу), SOCK_SEQPACKET (последовательный, двусторонний, PDU - дейтаграммы с ограниченной длиной, получатель обязан прочитать весь пакет разом, не включен в семейство AF_INET), SOCK_RDM (надёжность, непоследовательность)

Конкретный протокол (protocol) сокета.

Если 0 - используется протокол по-умолчанию, соответствующий заданному семейству и типу.

Виды:

- IPPROTO_TCP - (6) - протокол TCP (Transmission Control Protocol) - протокол управления передачей данных с трёхэтапным квитированием ("three-way handshake" - правило трёх рукопожатий - SYN(Synchronize) -> SYN-ACK(Synchronize-Acknowledge) (с выделением ресурсов) -> ACK)
- IPPROTO_UDP - (17) - протокол UDP (User Datagram protocol)

- IPPROTO_SCTP - (132) - протокол SCTP (Stream Control Transmission Protocol) - протокол передачи с управлением потоком - похож на TCP, но имеет преимущества: многопоточность, защита от DDoS-атак, синхронное соединение с четырёхэтапным квинтированием - four-way handshake (INIT -> INIT-ACK (с уникальным ключом-маркером, идент. новое соединение) -> COOKIE-ECHO (с маркером) -> COOKIE-ACK (с выделением ресурсов))
- IPPROTO_IP - (0) - dummy for IP
- IPPROTO_ICMP - (1) - control message protocol

3.1.2 Bind

Определение функции bind выглядит следующим образом:

```
int bind(int sockfd, struct sockaddr *adds, int addrlen);
```

Возвращает 0 в случае успеха, -1 в случае ошибки.

Параметры функции: сокет (sockfd) (файловый дескриптор сокета) обязательно связывают (bind) с адресом (addr) в выбранном домене — именование.

Определение структуры sockaddr:

```
struct sockaddr{
    unsigned short sa_family;    // Семейство адресов PF_xxx - тот же domain
    char          sa_data[14];  // 14 байтов для хранения адреса
                                // (для UNIX - название файла-сокета;
                                // для INET - {IP-адрес; 16-битный номер порта})
};
```

Однако можно использовать альтернативную структуру данных, позволяющую указывать адрес более удобным способом. Данной структурой является структура sockaddr_in для домена INET:

```
struct sockaddr_XX{            // (где XX-суффикс: домен "un" - UNIX; "in" - INET)
    short int      sin_family;  // Семейство адресов = sa_family
    unsigned short int sin_port; // Номер порта, определяет сокет хоста
    struct in_addr  sin_addr;   // IP-адрес, определяет хост в сети
    unsigned char   sin_zero[8]; // "Дополнение" до размера структуры sockaddr
};
```

Определяя адрес посредством этой структуры, в языке C++ далее

потребуется явная переинтерпретация типа обратно в тип `sockaddr` следующим образом:

```
reinterpret_cast<const struct sockaddr *>(&sin)
```

Как видно из определения структуры `sockaddr_in`, одним из составляющих её структуры является звено `sin_addr` структуры `in_addr`, полное представление которого представлено далее:

```
struct in_addr {  
    unsigned long s_addr;  
}
```

Важным моментом является то, что порядок хранения байтов адресов на хосте и в сети может различаться, в связи с чем требуется явно переводить числа из одного пространства в другое.

Виды порядка хранения байтов в слове и двоичном слове:

1. Порядок хоста - host byte order;
2. Сетевой порядок - network byte order

Поэтому при указании `sin_port` (номера порта) конвертируют адреса из хостового вида в вид сетевой с помощью функций:

1. `htons()` - Host To Network Short
2. `htonl()` - Host To Network Long

Обратное преобразование обеспечивается функциями:

1. `ntohs()`
2. `ntohl()`

С IP-адресом хоста поступают аналогичным образом: `sin_addr.s_addr` инициализируют с помощью функции `inet_addr("")`, определение которой расположено в заголовочном файле `<inet.h>`

3.1.3 Установка соединения (сервер)

1. Создание сокета сервера - `socket(domain, type, protocol);`

2. Привязка сокета к локальному адресу:

- `sin_addr.s_addr` = IP-адрес, принадлежащий конкретному сетевому интерфейсу, который будет принимать соединения от клиента:

- `INADDR_ANY` для соединения с клиентами через любой сетевой интерфейс

- `sin_port` = номер порта или 0 (произвольный неиспользуемый номер порта)

3. Создание очереди запросов на соединение (сокеты переводятся в режим ожидания запросов со стороны клиента):

- `listen()` - при попытке соединения клиента с сервером (`sockfd`), сервер ставит клиента в очередь длиной `backlog`, т.к. сервер выполняет запросы последовательно. При переполнении очереди запросы игнорируются.

- возвращает 0 в случае успеха, -1 в случае ошибки.

Определение функции:

```
int listen(int sockfd, int backlog);
```

- `sockfd` - дескриптор сокета
- `backlog` - размер очереди запросов

4. Подтверждение о готовности обслужить клиентов:

- `accept()` ---> возвращает файловый дескриптор сокета клиента или -1 (в случае ошибки).

- после вызова `accept` серверный сокет (`sockfd`) остаётся в слушающем состоянии и может принимать другие соединения.

- если не интересуют 2,3-й параметры -> `NULL`

- полученный Ф.Д. клиента связан с тем же адресом, что и Ф.Д. сервера (адрес TCP-сокета не обязан быть уникальным в Internet-домене)

```
int accept(int sockfd, struct sockaddr* addr, socklen_t * addrlen);
```

- `sockfd` - Ф.Д. слушающего (серверного) сокета
- `addr` - `accept` сюда записывает адрес сокета клиента
- `addrlen` - `accept` сюда записывает использованную длину под адрес

3.1.4 Установка соединения (клиент)

1. Создание сокета клиента - `socket(domain, type, protocol);`

2. Пропуск привязки клиента к локальному адресу

• `connect()` автоматически присваивает адрес и привязывает сокет к свободному порту). Исключение: подключение к серверам `rlogind` и `rshd`, где требуется определённый порт.

3. Установление соединения клиента с сервером:

- 0 в случае успеха
- -1 в случае ошибки

```
int connect(int sockfd, const struct sockaddr *serv_addr, int addrlen);
```

- `sockfd` - Ф.Д. клиентского сокета
- `serv_addr` - адрес сервера для подключения
- `addrlen` - длина структуры с адресом сервера

Таким образом, реализация клиент-серверного приложения на языке C++ представлена на рисунке 3.1.

3.1.5 Обмен данными

3.1.5.1 `send()`

Функция `send()` отвечает за отправку данных из буфера в сокет, определение которой представлено ниже:

```
int send(int sockfd, const void* buf, int len, int flags);
```

- `sockfd` - Ф.Д. клиентского сокета, через который отправляются данные
- `buf` - указатель на буфер с данными
- `len` - длина буфера в байтах
- `flags` - набор битовых флагов, регулирующих работу функции.
(0 - передача по умолчанию)

Возвращает фактически отправленное число байтов (может быть

меньше указанного размера буфера), -1 в случае ошибки

3.1.5.2 Самописная функция для отправки всего буфера.

```
int sendall(int sockfd, const void* buf, size_t len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = send(sockfd, buf+total, len - total, flags);
        if(n == -1) { break; }
        total += n;
    }
    return (n == -1 ? -1 : total);
}
```

3.1.5.3 flags:

- MSG_OOB - (out of band data, OOB) - отправить данные как срочные. Позволяет иметь два параллельных канала данных в одном соединении. (не рекомендуется к использованию)
- MSG_DONTROUTE - запрещает маршрутизацию пакетов, при этом нижележащие транспортные слои могут проигнорировать этот флаг.

3.1.5.4 recv()

Функция recv обеспечивает получение данных из сокета в буфер:

```
int recv(int sockfd, void *buf, size_t len, int flags);
```

- sockfd - Ф.Д. клиентского сокета, с которого получаем данные
- buf - указатель на буфер, куда записать данные
- len - длина буфера в байтах
- flags - набор битовых флагов, регулирующих работу функции.
(0 - получение по умолчанию)

Возвращает:

- фактически прочитанное с сокета число байтов в буфер (может быть меньше указанного размера буфера),
- -1 в случае ошибки
- 0 в случае разрыва соединения

3.1.5.5 Самописная функция для получения всех данных из сокета в буфер.

```
int recvall(int sockfd, void *buf, size_t len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = recv(sockfd, buf + total, len - total, flags);
        if(n == -1 || n == 0) { break; }
        total += n;
    }
    return (n == -1 ? -1 : total);
}
```

3.1.5.6 flags:

- MSG_OOB - (out of band data, OOB) - получить данные как срочные? При этом позволяет иметь два параллельных канала данных в одном соединении (не рекомендуется к использованию);
- MSG_PEEK - позволяет "подсмотреть" данные, полученные от удалённого хоста, не удаляя их из системного буфера. Означает, что при следующем обращении к recv вы получите те же самые данные.

3.1.6 Заккрытие сокета

Заккрытие сокета обеспечивается функцией close, определение которой находится в заголовочном файле «unistd.h». Эта операция осуществляется для того, чтобы освободить связанные с сокетом fd системные ресурсы.

```
#include <unistd.h>
```



```
int close(int fd);
```

Чтобы запретить передачу данных в каком-то одном направлении, существует функция shutdown:

```
int shutdown(int sockfd, int how);
```

- how:
 - 0 - запретить чтение из сокета
 - 1 - запретить запись в сокет
 - 2 - запретить и то и другое -> close(sockfd)

3.1.7 Обработка ошибок

- errno - глобальная переменная из <errno.h>, куда записываются код ошибки, если что-то пошло не так;
- perror() - функция для вывода диагностических ошибок;
- exit() - функция для принудительного завершения программы с указанием кода ошибки.

3.2 TCP клиент-серверное приложение на Python в Unix системе

3.2.1 Установка соединения (сервер)

По аналогии с реализацией клиент-серверного приложения на C++, составим программу с решением задачи по варианту на языке Python. Для этого требуется:

- подключить модуль socket;
- инициализировать слушающий сокет методом socket.socket параметрами socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP;
- связать только что созданный сокет с адресом «localhost» с портом 50330;
- создать очередь клиентов с помощью метода listen();
- запустить цикл, который будет прослушивать входящие соединения, и как только такое поступит, вернуть сокет клиента и его адрес в соответствующие структуры;

- далее получаем от клиента данные из сокета, предварительно декодируя их в стандартную строку формата «utf-8» В данном случае должно поступить имя хоста, которое необходимо сопоставить с IPv4-адресом;
- используя метод `socket.gethostbyname()` получаем IPv4-адрес;
- отправляем адрес, предварительно закодировав его обратно в двоичный вид для передачи по сети;
- освобождаем память, выделенную под сокет клиента;
- возвращаемся в начало цикла и снова начинаем ждать входящее соединение.

3.2.2 Установка соединения (клиент)

Реализация клиентского приложения ещё проще. Для этого требуется:

- подключить модуль `socket`;
- инициализировать клиентский сокет методом `socket.socket` параметрами `socket.AF_INET`, `socket.SOCK_STREAM`, `socket.IPPROTO_TCP`;
- подключиться к соединению, указав адрес сервера «localhost» и его конкретный порт, в данном случае — 50330;
- после установления соединения, отправляется интересующее нас имя хоста с помощью метода `sendall()`, предварительно кодируя его в битовый вид методом `encode(«utf-8»)`;
- далее ждём ответа от сервера, который возвратит нам битовый вид IPv4-адреса соответствующего имени хоста, который мы ранее отправляли;
- освобождаем память, выделенную под сокет клиента и завершаем работу программы

Результаты работы клиент-серверного приложения на Python представлены на рисунке 3.2.

Заключение

В ходе выполнения данной лабораторной работы были получены практические навыки по реализации ТСП клиент-серверного приложения. Реализация написана на языках программирования C++ и Python. Обеспечен обмен сообщениями между клиентом и сервером. А также выполнена прикладная задача по варианту, а именно были получены от сервера по DNS адресу набор IP-адресов, используя метод `gethostbyname()`.

Результаты работы программ

The screenshot shows a Visual Studio Code editor with two C++ files: `server.cpp` and `client.cpp`. The `server.cpp` file contains code to listen for incoming connections, accept them, and print the host name and IP address. The `client.cpp` file contains code to connect to the server, send a host name, and receive the IP address. The terminal window at the bottom shows the output of both programs. The server output shows it accepted connections from `vk.ru` and `sdo.tusur.ru`. The client output shows it connected to the server and received the IP address.

```
server > C Socket.h > Socket.h > cycle()
103 struct hostent *host_info = gethostbyname(name_host);
104 char **addresses_net = host_info->h_addr_list;
105 char **addresses_acci = new char*[host_info->h_length];
106 struct in_addr* address;
107 int num_addr;
108 for (num_addr = 0; host_info->h_addr_list[num_addr] != 0; num_addr++)
109 {
110     addresses_acci[num_addr] = new char[16];
111     address = (struct in_addr*)(host_info->h_addr_list[num_addr]);
112     //std::cout << address << "\n";
113     std::cout << inet_ntoa(*address) << "\n";
114     strcpy(addresses_acci[num_addr], inet_ntoa(*address));
115 }
116
```

```
client > client.cpp > main()
43 char host[1024] = "sdo.tusur.ru";
44 std::cout << "\nName of host: ";
45 std::cout << host << "\n";
46 //std::cin >> host;
47 char* name_host = new char*[4];
48
49 sendall(sock, host, 1024, 0);
50 recvall(sock, name_host);
51 for(int i = NUM_ADDR-1; i > 0; i--)
52 {
53     printf("%s", name_host[i]);
54     std::cout << std::endl;
55     //std::cout << name_host[i] << "\n";
56     delete[] name_host[i];
57 }
```

Accepted with client socket: 4
socked address: 127.0.0.1
socket port: 40494

name of host: vk.ru
87.240.132.72
93.186.225.194
87.240.132.67
87.240.137.164
87.240.132.78
87.240.129.133

Accepted with client socket: 4
socked address: 127.0.0.1
socket port: 58314

name of host: sdo.tusur.ru
212.192.127.40

Client socket: 3
socked address: 127.0.0.1
socket port: 50332
Name of host: sdo.tusur.ru

212.192.127.40
[1] + Done
"/usr/bin/gdb" --interpreter=mi --tty=\$(DbgTerm) 0<"
/tmp/Microsoft-MIEngine-In-wes3sux5.Syg I>"/tmp/Microsoft-MIEngine-Out-wgr0poto.name"
luzinsan@luzinsan-NB0-WX9:~/VisualCode/SIT/ClientServerCpp\$

Рисунок 3.1 — результат работы клиент-серверного приложения на C++

The screenshot shows a Visual Studio Code editor with two Python files: `client.py` and `server.py`. The `client.py` file contains code to connect to the server, send a host name, and receive the IP address. The `server.py` file contains code to listen for incoming connections, accept them, and print the host name and IP address. The terminal window at the bottom shows the output of both programs. The server output shows it accepted connections from `google.com`, `yandex.ru`, `vk.ru`, and `sdo.tusur.ru`. The client output shows it connected to the server and received the IP address.

```
client.py
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 s.connect(('localhost', 50330)) # Подключаемся к нашему серверу.
5 name_host = 'sdo.tusur.ru'
6 print(f'Name of host: {name_host}')
7 s.sendall(name_host.encode('utf-8')) # Отправляем имя хоста.
8 data = s.recv(1024) # Получаем данные из сокета.
9 print(f'ip: {data.decode("utf-8")}')
10 s.close()
11
```

```
server.py
1 import socket
2
3 serv_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
4 serv_sock.bind(('localhost', 50330)) # Привязываем серверный сокет к localhost и 50330 порту.
5 serv_sock.listen(1) # Начинаем прослушивать входящие соединения.
6 while True:
7     conn, addr = serv_sock.accept() # Метод который принимает входящее соединение.
8     data = conn.recv(1024) # Получаем данные из сокета.
9     name_host = data.decode('utf-8')
10     print(f'Name of host: {name_host}')
11     ip_host = socket.gethostbyname(name_host)
12     print(f'ip: {ip_host}')
13     conn.send(ip_host.encode('utf-8'))
14     conn.close()
```

client

/usr/bin/python3.10 /home/luzinsan/PycharmProjects/SIT/ClientServerPython/client/client.py
name of host: sdo.tusur.ru
ip: 212.192.127.40
Process finished with `exit` code 0

server

/usr/bin/python3.10 /home/luzinsan/PycharmProjects/SIT/ClientServerPython/server/server.py
name of host: google.com
ip: 216.239.38.120
name of host: yandex.ru
ip: 213.180.193.56
name of host: vk.ru
ip: 87.240.137.164
name of host: sdo.tusur.ru
ip: 212.192.127.40

Рисунок 3.2 — результат работы клиент-серверного приложения на Python

Листинг программ TCP-сервер на C++

server.cpp:

```
#include <iostream>
#include "Socket.h"
#define DOMAIN AF_INET
#define TYPE SOCK_STREAM
#define PROTOCOL IPPROTO_TCP
#define PORT 50331
int main()
{
    Socket server(PORT);
    try
    {
        server.cycle();
    }
    catch(std::exception err)
    {
        server.~Socket();
    };
    return 0;
}
```

Socket.h:

```
#include <iostream>
// linux
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>

typedef int FD_DESCRIPTOR;

#define LISTENQ SIZE_MAX
#define ERROR -1
#define SOCK_FAIL 1
#define BIND_FAIL 2
#define LISTEN_FAIL 2
#define ACCEPT_FAIL 3
#define SEND_FAIL 4
```

```

class Socket
{
private:
    FDESCRIPTOR server_sock, client_sock;
    int domain;
    int type;
    int protocol;
    unsigned short port;
    in_addr_t ip;
    struct sockaddr_in sin, clin;
    socklen_t len_ssock, len_clsock;
public:

    Socket(unsigned short __port, char* __ip="0.0.0.0",
           int __domain=AF_INET, int __type=SOCK_STREAM, int
__protocol=IPPROTO_TCP)
        :Socket(__domain, __type, __protocol)
    {
        sin.sin_family = domain;

        inet_pton(__domain, __ip, &ip);
        sin.sin_addr.s_addr = htonl(ip);
        port = __port;
        sin.sin_port = htons(port);
        len_ssock = sizeof(sin);
        if(bind(server_sock, reinterpret_cast<struct sockaddr *>(&sin), len_ssock) ==
ERROR)
        {
            perror("bind failure");
            exit(BIND_FAIL);
        }
        else
            std::cout << "Socket binded successful!: " << server_sock
            << "\nAddress family: " << sin.sin_family
            << "\nIP address: " << inet_ntoa(sin.sin_addr)
            << "\nPORT: " << ntohs(sin.sin_port)
            << std::endl;
        if (listen(server_sock, LISTENQ) == ERROR)
        {
            perror( "Can't start to listen to." );
            exit(LISTEN_FAIL);
        }
        else
            std::cout << "Listening..." << std::endl;
    }
}

```

```

Socket(int __domain, int __type, int __protocol)
:domain{__domain}, type{__type}, protocol{__protocol},
server_sock{socket(__domain, __type, __protocol)}
{
    if(errno == ERROR)
    {
        perror("Socket error");
        exit(SOCK_FAIL);
    }
}

~Socket(){ close(server_sock);}

void cycle()
{
    std::cout << "Cycle\n";
    len_clsock = sizeof(struct sockaddr_in);

    while(1)
    {
        client_sock = accept(server_sock, reinterpret_cast<struct sockaddr *>(&clin),
&len_clsock);
        if(client_sock == ERROR)
        {
            printf("accept");
            exit(ACCEPT_FAIL);
        }
        else
        {
            std::cout << "\nAccepted with client socket: " << client_sock
                << "\nsocked address: " << inet_ntoa(clin.sin_addr)
                << "\nsocket port: " << ntohs(clin.sin_port) << std::endl;

            char *name_host = new char[1024];
            int bytes_read = recvall(client_sock, name_host, 1024, 0);
            if(bytes_read < 0) { close(client_sock); continue; }
            std::cout << "\nname of host: " << name_host << "\n";

            struct hostent *host_info = gethostbyname(name_host);
            char **addresses_net = host_info->h_addr_list;
            char **addresses_acci = new char*[host_info->h_length];
            struct in_addr* address;
            int num_addr;
            for (num_addr = 0; host_info->h_addr_list[num_addr] != 0; num_addr++)
            {
                addresses_acci[num_addr] = new char[16];
                address = (struct in_addr *)(host_info->h_addr_list[num_addr]);
                std::cout << address << "\n";
            }
        }
    }
}

```

```

        std::cout << inet_ntoa(*address) << "\n";
        strcpy(addresses_acci[num_addr], inet_ntoa(*address));
    }

    if(sendall(client_sock, addresses_acci, num_addr, 16) == 0)
    {
        perror("sendall");
        exit(SEND_FAIL);
    }
    close(client_sock);
}

int sendall(int sockfd, char** buf, size_t h_count, size_t len=16, int flags=0)
{
    int succ = h_count;
    for(int i = 0; i < h_count; i++)
        if(sendall(sockfd, buf[i], len, flags) == ERROR)
            succ -= 1;

    return succ;
}

int sendall(int sockfd, char* buf, size_t len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = send(sockfd, buf+total, len - total, flags);
        if(n == -1) { break; }
        total += n;
    }
    return (n == -1 ? -1 : total);
}

int recvall(int sockfd, char *buf, size_t len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = recv(sockfd, buf + total, len - total, flags);
        if(n == -1 || n == 0) { break;}
        total += n;
    }
    return (n == -1 ? -1 : total);
}

```



```
}  
};
```

ТСР-клиент на C++

client.cpp:

```
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <netdb.h>  
#include <unistd.h>  
#include <iostream>  
#define PORT 50331  
#define NUM_ADDR 6  
#define LEN_STR 16  
  
int sendall(int sockfd, const char* buf, size_t len, int flags);  
int recvall(int sockfd, char *buf, size_t len, int flags);  
int recvall(int sockfd, char** buf, size_t h_count=NUM_ADDR, size_t len=LEN_STR, int  
flags=0);  
  
int main()  
{  
    int sock;  
    struct sockaddr_in addr;  
  
    sock = socket(AF_INET, SOCK_STREAM, 0);  
    if(sock < 0)  
    {  
        perror("socket");  
        exit(1);  
    }  
  
    addr.sin_family = AF_INET;  
    addr.sin_port = htons(PORT); // или любой другой порт...  
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);  
    //inet_pton(PF_INET, "127.0.0.3", &addr.sin_addr.s_addr);  
    if(connect(sock, reinterpret_cast<const struct sockaddr *>(&addr), sizeof(addr)) < 0)  
    {  
        perror("connect");  
        exit(2);  
    }  
    else  
        std::cout << "Client socket: " << sock  
                    << "\nsocked address: " << inet_ntoa(addr.sin_addr)  
                    << "\nsocket port: " << ntohs(addr.sin_port);
```

```

char host[1024] = "vk.ru";
std::cout << "\nName of host: ";
std::cout << host << "\n";
//std::cin >> host;
char** name_host = new char*[4];

sendall(sock, host, 1024, 0);
recvall(sock, name_host);
for(int i = NUM_ADDR-1; i > 0; i--)
{
    printf("%s", name_host[i]);
    std::cout << std::endl;
    //std::cout << name_host[i] << "\n";
    delete[] name_host[i];
}
printf("%s", *name_host);
std::cout << std::endl;
delete[] name_host;

close(sock);
return 0;
}

int sendall(int sockfd, const char* buf, size_t len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = send(sockfd, buf+total, len - total, flags);
        if(n == -1) { break;}
        total += n;
    }
    return (n == -1 ? -1 : total);
}

int recvall(int sockfd, char** buf, size_t h_count, size_t len, int flags)
{
    int succ = h_count;
    for(int i = 0; i < h_count; i++)
    {
        buf[i] = new char[len];
        if(recvall(sockfd, buf[i], len, flags) == -1)
            succ -= 1;
    }
    return succ;
}

```

```

int recvall(int sockfd, char *buf, size_t len, int flags)
{
    int total = 0;
    int n;
    while(total < len)
    {
        n = recv(sockfd, buf + total, len - total, flags);
        //std::cout << buf;
        if(n == -1 || n == 0) { break; }
        total += n;
    }
    return (n == -1 ? -1 : total);
}

```

TCP-сервер на Python

server.py

```

import socket
serv_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM,
socket.IPPROTO_TCP)
serv_sock.bind(('localhost', 50330)) # Привязываем серверный сокет к localhost и 3030
порту.
serv_sock.listen(1) # Начинаем прослушивать входящие соединения.
while True:
    conn, addr = serv_sock.accept() # Метод который принимает входящее соединение.
    data = conn.recv(1024) # Получаем данные из сокета.
    name_host = data.decode('utf-8')
    print(f'name of host: {name_host}')
    ip_host = socket.gethostbyname(name_host)
    print(f'ip: {ip_host}')
    conn.send(ip_host.encode('utf-8'))
    conn.close()

```

TCP-клиент на Python

client.py

```

import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('localhost', 50330)) # Подключаемся к нашему серверу.
name_host = 'google.com'
print(f'name of host: {name_host}')
s.sendall(name_host.encode('utf-8')) # Отправляем имя хоста.
data = s.recv(1024) # Получаем данные из сокета.
print(f'ip: {data.decode("utf-8")}')
s.close()

```