**Сокеты**

**Источник: https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html**

**См.также**

* [https://ru.wikipedia.org/wiki/Сокет\_(программный\_интерфейс)](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81))
* [https://ru.wikipedia.org/wiki/Сокеты\_Беркли](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82%D1%8B_%D0%91%D0%B5%D1%80%D0%BA%D0%BB%D0%B8)
* <http://rsdn.ru/article/unix/sockets.xml>
* <http://pymotw.com/2/socket/index.html>
* <http://masandilov.ru/network/guide_to_network_programming>

Сокеты (англ. socket — разъём) — название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет — абстрактный объект, представляющий конечную точку соединения.

**Принципы сокетов**

Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы (в UNIX непривилегированные процессы не могут использовать порты меньше 1024). Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т.д.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX-адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут сообщаться любые локальные процессы путём чтения/записи из него (см. Доменный сокет Unix). Сокеты типа INET доступны из сети и требуют выделения номера порта.

Обычно клиент явно подсоединяется к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером.

**Основные функции**

|  |  |
| --- | --- |
| **Общие** |  |
| Socket | Создать новый сокет и вернуть файловый дескриптор |
| Send | Отправить данные по сети |
| Receive | Получить данные из сети |
| Close | Закрыть соединение |
|  |  |
| **Серверные** |  |
| Bind | Связать сокет с IP-адресом и портом |
| Listen | Объявить о желании принимать соединения. Слушает порт и ждет когда будет установлено соединение |
| Accept | Принять запрос на установку соединения |
|  |  |
| **Клиентские** |  |
| Connect | Установить соединение |

**socket()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?socket+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket>

Создаёт конечную точку соединения и возвращает файловый дескриптор. Принимает три аргумента:

1. **domain** указывающий семейство протоколов создаваемого сокета
   * **AF\_INET** для сетевого протокола IPv4
   * **AF\_INET6** для IPv6
   * **AF\_UNIX** для локальных сокетов (используя файл)
2. **type**
   * **SOCK\_STREAM** (надёжная потокоориентированная служба (сервис) или потоковый сокет)
   * **SOCK\_DGRAM** (служба датаграмм или датаграммный сокет)
   * **SOCK\_RAW** (Сырой сокет — сырой протокол поверх сетевого уровня).
3. **protocol**

Протоколы обозначаются символьными константами с префиксом **IPPROTO\_\*** (например, **IPPROTO\_TCP** или **IPPROTO\_UDP**). Допускается значение protocol=0 (протокол не указан), в этом случае используется значение по умолчанию для данного вида соединений.

**Примечание**

Функция возвращает −1 в случае ошибки. Иначе, она возвращает целое число, представляющее присвоенный дескриптор.

Пример на Си

#include *<sys/types.h>*

#include *<sys/socket.h>*

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

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<unistd.h>*

#include *<string.h>*

#include *<sys/types.h>*

#include *<sys/socket.h>*

#include *<netinet/in.h>*

#include *<netdb.h>*

void error(**const** char \*msg)

{

perror(msg);

exit(0);

}

int main(int argc, char \*argv[])

{

int sockfd, portno, n;

**struct** sockaddr\_in serv\_addr;

**struct** hostent \*server;

char buffer[256];

**if** (argc < 3) {

fprintf(stderr,"usage %s hostname port**\n**", argv[0]);

exit(0);

}

*// Задаем номер порта*

portno = atoi(argv[2]);

*// Создаем сокет*

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

**if** (sockfd < 0)

error("ERROR opening socket");

*// Конвертирует имя хоста в IP адрес*

server = gethostbyname(argv[1]);

**if** (server == NULL) {

fprintf(stderr,"ERROR, no such host**\n**");

exit(0);

}

*// Указываем тип сокета Интернет*

bzero((char \*) &serv\_addr, **sizeof**(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

*// Указаваем адрес IP сокета*

bcopy((char \*)server->h\_addr,

(char \*)&serv\_addr.sin\_addr.s\_addr,

server->h\_length);

*// Указываем порт сокета*

serv\_addr.sin\_port = htons(portno);

*// Устанавливаем соединение*

**if** (connect(sockfd, (**struct** sockaddr \*) &serv\_addr, **sizeof**(serv\_addr)) < 0)

error("ERROR connecting");

*// Вводим сообщение из консоли*

printf("Please enter the message: ");

bzero(buffer, 256);

fgets(buffer, 255, stdin);

*// Отправляем данные*

n = write(sockfd, buffer, strlen(buffer));

**if** (n < 0)

error("ERROR writing to socket");

*// Сбрасываем буфер*

bzero(buffer, 256);

*// Читаем ответ*

n = read(sockfd, buffer, 255);

**if** (n < 0)

error("ERROR reading from socket");

printf("%s**\n**", buffer);

close(sockfd);

**return** 0;

}

Пример на Python

**import** **socket**

*# Создание объекта сокета.*

sock\_obj = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM, 0)

*# AF\_INET, SOCK\_STREAM и 0 используются по умолчанию при создании сокета.*

*# Поэтому можно просто писать:*

sock\_obj = socket.socket()

**bind()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?bind+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket.bind>

Связывает сокет с конкретным адресом. Когда сокет создается при помощи socket(), он ассоциируется с некоторым семейством адресов, но не с конкретным адресом. До того как сокет сможет принять входящие соединения, он должен быть связан с адресом. bind() принимает три аргумента:

1. **sockfd** — дескриптор, представляющий сокет при привязке
2. **serv\_addr** — указатель на структуру sockaddr, представляющую адрес, к которому привязываем.
3. **addrlen** — поле socklen\_t, представляющее длину структуры sockaddr.

**Примечание**

Возвращает 0 при успехе и −1 при возникновении ошибки.

Пример на Си

#include *<sys/types.h>*

#include *<sys/socket.h>*

int bind(int sockfd, **const** **struct** sockaddr \*my\_addr, socklen\_t addrlen);

Пример на Python

server\_address = ('localhost', 8080)

sock\_obj.bind(server\_address) *# Привязка адреса и порта к сокету.*

Автоматическое получение имени хоста.

host = socket.gethostname() *# Получить имя локальной машины.*

server\_address = (host, 8080)

sock\_obj.bind(server\_address) *# Привязка адреса и порта к сокету.*

**listen()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?listen+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket.listen>

Подготавливает привязываемый сокет к принятию входящих соединений. Данная функция применима только к типам сокетов SOCK\_STREAM и SOCK\_SEQPACKET. Принимает два аргумента:

1. **sockfd** — корректный дескриптор сокета.
2. **backlog** — целое число, означающее число установленных соединений, которые могут быть обработаны в любой момент времени. Операционная система обычно ставит его равным максимальному значению.

**Примечание**

После принятия соединения оно выводится из очереди. В случае успеха возвращается 0, в случае возникновения ошибки возвращается −1.

Пример на Си

#include *<sys/socket.h>*

int listen(int sockfd, int backlog);

Пример на Python

sock\_obj.listen(5) *# Ждем соединение клиента.*

**accept()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?accept+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket.accept>

Используется для принятия запроса на установление соединения от удаленного хоста. Принимает следующие аргументы:

1. **sockfd** — дескриптор слушающего сокета на принятие соединения.
2. **cliaddr** — указатель на структуру sockaddr, для принятия информации об адресе клиента.
3. **addrlen** — указатель на socklen\_t, определяющее размер структуры, содержащей клиентский адрес и переданной в accept(). Когда accept() возвращает некоторое значение, socklen\_t указывает сколько байт структуры cliaddr использовано в данный момент.

**Примечание**

Функция возвращает дескриптор сокета, связанный с принятым соединением, или −1 в случае возникновения ошибки.

Пример на Си

#include *<sys/types.h>*

#include *<sys/socket.h>*

int accept(int sockfd, **struct** sockaddr \*cliaddr, socklen\_t \*addrlen);

Пример на Python

conn, addr = sock\_obj.accept() *# Установление соединения с клиентом.*

**connect()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?connect+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket.connect>

Устанавливает соединение с сервером.

Некоторые типы сокетов работают без установления соединения, это в основном касается UDP-сокетов. Для них соединение приобретает особое значение: цель по умолчанию для посылки и получения данных присваивается переданному адресу, позволяя использовать такие функции как send() и recv() на сокетах без установления соединения.

Загруженный сервер может отвергнуть попытку соединения, поэтому в некоторых видах программ необходимо предусмотреть повторные попытки соединения.

**Примечание**

Возвращает целое число, представляющее код ошибки: 0 означает успешное выполнение, а −1 свидетельствует об ошибке.

Пример на Си

#include *<sys/types.h>*

#include *<sys/socket.h>*

int connect(int sockfd, **const** **struct** sockaddr \*serv\_addr, socklen\_t addrlen);

Пример на Python

server\_address = ('192.168.1.100', 8080)

sock\_obj.connect(server\_address)

**Передача данных**

Для передачи данных можно пользоваться стандартными функциями чтения/записи файлов read и write, но есть специальные функции для передачи данных через сокеты:

* send
* recv
* sendto
* recvfrom
* sendmsg
* recvmsg

Нужно обратить внимание, что при использовании протокола TCP (сокеты типа SOCK\_STREAM) есть вероятность получить меньше данных, чем было передано, так как ещё не все данные были переданы, поэтому нужно либо дождаться, когда функция recv возвратит 0 байт, либо выставить флаг MSG\_WAITALL для функции recv, что заставит её дождаться окончания передачи. Для остальных типов сокетов флаг MSG\_WAITALL ничего не меняет (например, в UDP весь пакет = целое сообщение).

**send()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?send+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket.send>

**send**, **sendto** - отправка данных.

Пример на Си

#include *<sys/types.h>*

#include *<sys/socket.h>*

ssize\_t send(int s, **const** void \*buf, size\_t len, int flags);

ssize\_t sendto(int s, **const** void \*buf, size\_t len, int flags, **const** **struct** sockaddr \*to, socklen\_t tolen);

Пример на Python

IP = '192.168.1.100'

PORT = 8080

sock\_obj.send('Hello World!')

sock\_obj.sendto('Hello World!', (IP, PORT))

**resv()**

**См.также**

* <http://unixhelp.ed.ac.uk/CGI/man-cgi?recv+2>
* <https://docs.python.org/3.5/library/socket.html#socket.socket.recv>

**recv**, **recvfrom** - чтение данных из сокета.

Пример на Си

#include *<sys/types.h>*

#include *<sys/socket.h>*

ssize\_t recv(int s, void \*buf, size\_t len, int flags);

ssize\_t recvfrom(int s, void \*buf, size\_t len, int flags, **struct** sockaddr \*from, socklen\_t \*fromlen);

Пример на Python

BUFFER\_SIZE = 1024

data = conn.recv(BUFFER\_SIZE)

data, sender\_addr = conn.recvfrom(BUFFER\_SIZE)

**SOCK\_STREAM vs SOCK\_DGRAM**

**См.также**

* [UDP](http://www.soslan.ru/tcp/tcp11.html)
* [TCP](http://www.soslan.ru/tcp/tcp17.html)

| **Потоковый (SOCK\_STREAM)** | **Дейтаграммный (SOCK\_DGRAM)** |
| --- | --- |
| Устанавливает соединение | Нет |
| Гарантирует доставку данных | Нет в случае UDP |
| Гарантирует порядок доставки пакетов | Нет в случае UDP |
| Гарантирует целостность пакетов | Тоже |
| Разбивает сообщение на пакеты | Нет |
| Контролирует поток данных | Нет |

TCP гарантирует доставку пакетов, их очередность, автоматически разбивает данные на пакеты и контролирует их передачу, в отличии от UDP. Но при этом TCP работает медленнее за счет повторной передачи потерянных пакетов и большему количеству выполняемых операций над пакетами. Поэтому там где требуется гарантированная доставка (Веб-браузер, telnet, почтовый клиент) используется TCP, если же требуется передавать данные в реальном времени (многопользовательские игры, видео, звук) используют UDP.