ОСиС лаб 6

1. TCP и TCP/IP

**Протокол управления передачей** ( **TCP** ) является одним из основных [протоколов](https://en.wikipedia.org/wiki/Communications_protocol) в [наборе протоколов Internet](https://en.wikipedia.org/wiki/Internet_protocol_suite) . Он возник в начальной сетевой реализации, в которой он дополнял [Интернет-протокол](https://en.wikipedia.org/wiki/Internet_Protocol) (IP). Поэтому весь пакет обычно называют *TCP / IP* . TCP обеспечивает [надежную](https://en.wikipedia.org/wiki/Reliability_(computer_networking)) , упорядоченную и [проверенную на ошибки](https://en.wikipedia.org/wiki/Error_detection_and_correction) доставку потока [октетов](https://en.wikipedia.org/wiki/Octet_(computing)) (байтов) между приложениями, работающими на хостах, обменивающихся данными через IP-сеть. Основные интернет-приложения, такие как [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web" \o "Всемирная сеть) , [электронная почта](https://en.wikipedia.org/wiki/Email) , [удаленное администрирование](https://en.wikipedia.org/wiki/Remote_administration) и[передача файлов осуществляется](https://en.wikipedia.org/wiki/File_transfer) по протоколу TCP, который является частью [транспортного уровня](https://en.wikipedia.org/wiki/Transport_layer) пакета TCP / IP. [SSL / TLS](https://en.wikipedia.org/wiki/Transport_Layer_Security) часто работает поверх TCP.

1. TCP vs UDP

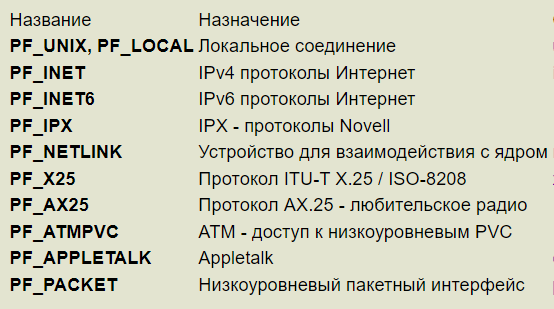
TCP [ориентирован](https://en.wikipedia.org/wiki/Connection-oriented_communication) на соединение, и соединение между клиентом и сервером устанавливается до того, как данные могут быть отправлены. Сервер должен прослушивать (пассивно открывать) запросы на соединение от клиентов, прежде чем соединение будет установлено. Трехстороннее квитирование (активное открытие), [повторная передача](https://en.wikipedia.org/wiki/Retransmission_(data_networks)) и обнаружение ошибок повышают надежность, но увеличивают [задержку](https://en.wikipedia.org/wiki/Latency_(engineering)) . Приложения, которым не требуется надежная служба [потока данных,](https://en.wikipedia.org/wiki/Data_stream) могут использовать [протокол пользовательских дейтаграмм](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP), который предоставляет службу [дейтаграмм без](https://en.wikipedia.org/wiki/Datagram) установления [соединения, в](https://en.wikipedia.org/wiki/Connectionless_communication) которой время важнее надежности. TCP использует [предотвращение перегрузки сети](https://en.wikipedia.org/wiki/TCP_congestion_control) . Однако в TCP есть уязвимости, в том числе [отказ в обслуживании](https://en.wikipedia.org/wiki/Denial-of-service_attack) , [перехват соединения](https://en.wikipedia.org/wiki/TCP_sequence_prediction_attack) , вето TCP и [атака сброса](https://en.wikipedia.org/wiki/TCP_reset_attack) .

1. Системный вызов socket

**Socket** создает конечную точку соединения и возвращает ее описатель.

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

Параметр *domain* задает домен соединения: выбирает набор протоколов, которые будут использоваться для создания соединения.



Сокет имеет тип *type*, задающий семантику коммуникации. В настоящее время определены следующие типы:

**SOCK\_STREAM**

Обеспечивает создание двусторонних надежных и последовательных потоков байтов , поддерживающих соединения. Может также поддерживаться механизм  
 внепоточных данных.

**SOCK\_DGRAM**

Поддерживает датаграммы (ненадежные сообщения с ограниченной длиной и не поддерживающие соединения).

**SOCK\_SEQPACKET**

Обеспечивает работу последовательного двустороннего канала для передачи датаграмм  
 с поддержкой соединений; датаграммы имеют ограниченную длину; от получателя требуется за один раз прочитать целый пакет.

**SOCK\_RAW**

Обеспечивает доступ к низкоуровневому сетевому протоколу.

**SOCK\_RDM**

Обеспечивает надежную доставку датаграмм без гарантии, что они будут расположены по порядку.

Параметр *protocol* задает конкретный протокол, который работает с сокетом. Обычно существует только один протокол, задающий конкретный тип сокета в определенном семействе протоколов, в этом случае *protocol* может быть определено, как 0. Однако, возможно существование нескольких таких протоколов (в этом случае и используется данный параметр).

1. Системный вызов bind

bind - привязать имя к сокету

**bind** привязывает к сокету *sockfd* локальный адрес *my\_addr* длиной *addrlen*.

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

В случае успеха возвращается ноль. При ошибке возвращается -1, а *errno* устанавливается должным образом.

1. Системный вызов listen

listen - слушать соединения на сокете

**int listen(int***s***, int***backlog***);**

Для того, чтобы принимать соединения, сначала нужно создать сокет с помощью **[socket](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=socket&category=2)**(2), затем выразить готовность принимать входящие соединения и задать размер очереди с помощью **listen**, а затем вызывать **[accept](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=accept&category=2)**(2) по мере появления новых соединений.

Параметр *backlog* задает максимальную длину, до которой может расти очередь ожидающих соединений. Если приходит запрос на соединение, а очередь полна, то клиент получит ошибку **ECONNREFUSED** или, если соответствующие протокол поддерживает повторную передачу, запрос может быть игнорирован, чтобы попытаться ответить на повторный запрос. 

В случае успеха возвращается ноль. При ошибке возвращается -1, а *errno* устанавливается должным образом.

1. Системный вызов accept

accept - принять соединение на сокете 

**int accept(int***s***, struct sockaddr \****addr***, socklen\_t \****addrlen***)**

Функция **accept** используется с сокетами, ориентированными на устанавление соединения (**SOCK\_STREAM**, **SOCK\_SEQPACKET** и **SOCK\_RDM**). Эта функция извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет почти с такими же параметрами, что и у *s*, и выделяет для сокета новый файловый дескриптор, который и возвращается. Новый сокет более не находится в слушающем состоянии. Исходный сокет *s* не изменяется при этом вызове.

Аргумент *s*~-- это сокет, который был создан с помощью **[socket](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=socket&category=2)**(2), привязан к локальному адресу с помощью **[bind](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=bind&category=2)**(2), и слушает соединения после **[listen](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=listen&category=2)**(2).

Аргумент *addr*~-- это указатель на структуру **sockaddr**. В эту структуру помещается адрес другой стороны, в том виде, в каком он известен на коммуникационном уровне. Точный формат адреса, передаваемого в параметре *addr*, определяется "семейством" сокета.

Аргумент *addrlen* является параметром, передаваемым по ссылке: перед вызовом он содержит размер структуры, на которую ссылается *addr*, а после вызова~-- действительную длину адреса в байтах. Если *addr* равен **NULL**, он не заполняется.

Если в очереди нет запросов на соединение, и на сокет не установлен флаг, что он является неблокирующим, **accept** блокирует вызвавшую программу до появления соединения. Если сокет является неблокирующим, а в очереди нет запросов на соединение, то **accept** возвращает EAGAIN.

1. Системный вызов recv

recv, recvfrom, recvmsg - получить сообщение из сокета

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

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

**int recvmsg(int***s***, struct msghdr \****msg***, int***flags***);**

Системные вызовы **recvfrom** и **recvmsg** используются для получения сообщений из сокета, и могут использоваться для получения данных, независимо от того, является ли сокет ориентированным на соединения или нет.

Если параметр *from* не равен **NULL**, а сокет не является ориентированным на соединения, то адрес отправителя в сообщении не заполняется. Аргумент *fromlen* передается по ссылке, в начале инициализируется размером буфера, связанного с *from*, а при возврате из функции содержит действительный размер адреса.

Вызов **recv** обычно используется только на *соединенном* сокете (см. **[connect](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=connect&category=2)**(2)) и идентичен вызову **recvfrom** с параметром *from*, установленным в **NULL**.

Все три функции возвращают длину сообщения при успешном завершении. Если сообщение слишком длинное и не поместилось в предоставленный буфер, лишние байты могут быть отброшены, в зависимости от типа сокета, на котором принимаются сообщения (см. **[socket](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=socket&category=2)**(2)).

Если на сокете не доступно ни одного сообщения, то обсуждаемые функции ожидают их прибытия, если сокет не помечен как неблокирующий (см. **[fcntl](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=fcntl&category=2)**(2)), в противном случае возвращается значение -1, а внешняя переменная *errno* устанавливается в значение **EAGAIN**. Все эти функции обычно возвращают уже доступные данные вплоть до запрошенного объема, и не ждут, пока появятся данные полной запрошенной длины.

1. Системный вызов send

send, sendto, sendmsg - отправляет сообщения в сокет

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

**send**, **sendto**, и **sendmsg** используются для пересылки сообщений в другой сокет. **send** можно использовать, только если сокет находится в состоянии *соединения*, тогда как **sendto** и **sendmsg** можно использовать в любое время.

Адрес получателя задается параметром *to* длиной *tolen*. Длина сообщения задается параметром *len*. Если сообщение слишком длинное, чтобы быть отосланным протоколом нижнего уровня, возвращается ошибка **EMSGSIZE**, а сообщение не отсылается.

Неудачная отправка не показывается  
 с помощью   **send**. Локальные ошибки принимают значение -1.

Когда сообщение не помещается в буфер отправки сокета, **send** обычно дожидается завершения отправки, если только сокет не находится в неблокирующем режиме. Если сокет находится в неблокирующем режиме, то в этом случае возвращается **EAGAIN**. Системный вызов **[select](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=select&category=2)**(2) можно использовать для выяснения, возможно ли отправлять данные.

1. Системный вызов close

close - закрыть файловый дескриптор

**int close(int** *fd***);**

**close** закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно. Все блокировки, находящиеся на соответствующем файле, снимаются (независимо от того, был ли использован для установки блокировки именно этот файловый дескриптор).

Если *fd* является последней копией какого-либо файлового дескриптора, то ресурсы, связанные с ним, освобождаются; если дескриптор был последней ссылкой на файл, удаленный с помощью **[unlink](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=unlink&category=2)**(2), то файл окончательно удаляется.

1. Системный вызов connect

connect - инициирует соединение на сокете

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

Файловый дескриптор *sockfd* должен ссылаться на сокет. Если сокет имеет тип **SOCK\_DGRAM**, значит, адрес *serv\_addr* является адресом по умолчанию, куда посылаются датаграммы, и единственным адресом, откуда они принимаются. Если сокет имеет тип **SOCK\_STREAM** или **SOCK\_SEQPACKET**, то данный системный вызов попытается установить соединение с другим сокетом. Другой сокет задан параметром *serv\_addr*, являющийся адресом длиной *addrelen* в пространстве коммуникации сокета. Каждое пространство коммуникации интерпретирует параметр *serv\_addr* по-своему.