Интерфейс сокетов

- *Интерфейс прикладной программы* (API) представляет собой просто набор функций (интерфейс), использующийся программистами для разработки прикладных программ в определенных компьютерных средах.
- В восьмидесятых годах американское правительственное агентство по поддержке исследовательских проектов (ARPA), финансировало реализацию протоколов TCP/IP для UNIX в Калифорнийском университете в г. Беркли. В ходе этого проекта группа исследователей-программистов разработала интерфейс прикладного программирования для сетевых приложений TCP/IP (TCP/IP API), этот интерфейс был назван ***сокетами TCP/IP (TCP/IP sockets).***

- Так как изначально разработчиками интерфейса сокетов были исследователи университета в г. Беркли, этот интерфейс часто называется сокетами Беркли (Berkeley sockets) или сокетами в стиле Беркли.

- Интерфейс сокетов - это API для сетей TCP/IP. Другими словами, он описывает набор программных функций или процедур, позволяющий разрабатывать приложения для использования в сетях TCP/IP. В наши дни интерфейс сокетов наиболее широко используется в сетях на базе TCP/IP.

Ввод-вывод сетевых данных и данных в файловой системе

- Интерфейс сокетов Беркли проще понять, если вы имеете хотя бы общее представление о системе ввода-вывода в UNIX. Как уже было сказано, разработчики сокетов Беркли стремились реализовать протоколы TCP/IP в рамках операционной среды UNIX. При этом они располагали уже имеющимся в UNIX набором вызовов системных функций. В результате, интерфейс сокетов использовал те же системные вызовы, что и остальные программы. Он оказался встроенным в саму операционную систему.

- Системные вызовы ввода-вывода в UNIX выглядят как последовательный цикл, состоящий из операций типа *открыть-считать-записать-закрыть.* Файл, перед тем как читать из него, должен быть *открыт.* Далее над ним выполняются операции по считыванию/записи. По окончании операций файл *закрывается.*
- Программисты в UNIX могут не знать различий между файлами и внешними устройствами. Одни и те же системные вызовы используются по отношению к принтеру, модему, дисплею и файлам. Внешние устройства и файлы отображены в UNIX на одну файловую систему. Чтобы открыть файл или получить доступ к устройству, программист должен вызвать одну и ту же системную функцию. В ответ на системный вызов UNIX возвращает так называемый *дескриптор файла* (file descriptor); его иногда называют указателем (file handler). Дескриптор файла указывает на внутрисистемную таблицу, описывающую открытый файл или устройство. Для файла в таблице могут приводиться его имя, размер, дата создания. Дескриптор файла в UNIX может указывать на файл, внешнее устройство или любой другой объект, позволяющий выполнять над ним системные функции ввода или вывода.
- На первых порах разработки интерфейса сокетов исследователи пытались заставить сетевой ввод-вывод функционировать так же, как и любой другой ввод-вывод UNIX. Другими словами, они хотели, чтобы интерфейс сокетов считывал и записывал данные в уже известном нам цикле открыть-считать-записать-закрыть.

Возникают некоторые проблемы

- В процессе разработки исследователи выяснили, что сетевой ввод-вывод значительно сложнее, чем ввод-вывод для любых других устройств. Проектируя обычные системы ввода-вывода, они столкнулись с проблемами, с которыми никогда до этого не встречались.
- Сетевые программы, как правило, создаются по модели клиент-сервер. Разработчики сокетов могли запросто реализовать такой API для системы ввода-вывода UNIX, в котором программист мог бы создать программу-клиент, активно устанавливающую сетевое соединение. Однако этот же API был бы должен позволить создавать и программы-серверы пассивно ждущие, пока к ним кто-нибудь не обратится. Поскольку обычная система ввода-вывода UNIX попросту не умеет пассивно вводить и выводить данные, разработчики были вынуждены создать набор новых функций для обработки пассивных операций ввода-вывода

- Было обнаружено, что стандартные функции ввода-вывода UNIX также плохо умеют устанавливать соединения. Как правило, они пользуются фиксированным адресом файла и устройства для обращения к нему. То есть адрес файла или устройства для каждого компьютера — постоянная величина. Соединение (или путь) к файлу или устройству доступно на протяжении всего цикла запись-считывание — то есть до тех пор, пока программа не закроет соединение.
- Фиксированный адрес — превосходная мысль, если протокол передачи данных ориентирован на соединение. Для не ориентированных на соединение протоколов фиксированный адрес представляет проблему. Например, в датаграмме, переданной IP, присутствует адрес компьютера-получателя (IP-адрес), однако протокол не устанавливает предварительно никакого соединения. В системе ввода-вывода UNIX отсутствуют какие-либо возможности для этого.

- Разработчики отказались от внесения модификаций в существующую систему ввода-вывода UNIX, а вместо этого создали новый API (набор функций).
- Интерфейс сокетов, получившийся в конечном итоге, унаследовал дух операционной системы UNIX. Например, дескриптор сокетов в интерфейсе по-прежнему называется дескриптором файла, и информация о сокете хранится в той же таблице, что и дескрипторы файлов.

Абстракция сокетов

- Сокет можно рассматривать, как конечный пункт передачи данных по сети. Сетевое соединение — это процесс передачи данных по сети между двумя компьютерами или процессами. Сокет — конечный пункт передачи данных. Другими словами, когда программы используют сокет, для них он является абстракцией, представляющий одно из окончаний сетевого соединения. Для установления соединения в абстрактной модели сокетов необходимо, чтобы каждая из сетевых программ имела свой собственный сокет. Связь между двумя сокетами может быть ориентирована на соединение, а может быть и нет. Несмотря на то, что разработчики модифицировали системный код UNIX, интерфейс сокетов по-прежнему использует концепцию ввода-вывода данных UNIX. То есть сетевая модель интерфейса сокетов до сих пор использует цикл открыть-считать-записать-закрыть.

- Чтобы открыть или создать файл в UNIX (и в большинстве других ОС), вы должны указать его описание (например, имя файла и то, как вы будете его использовать: записывать или считывать). Затем вы запрашиваете у операционной системы дескриптор файла, соответствующий вашему описанию. Не существует каких-либо ограничений на то, когда запрашивать дескриптор. Как только вам нужен файл, запрашивайте его дескриптор. В один и тот же момент времени может быть открыто несколько устройств или файлов. В любом случае операционная система возвращает дескриптор (обычно это целое число), однозначно соответствующий указанному файлу или устройству. Интерфейс сокетов работает точно так же. Когда программе нужен сокет, она формирует его характеристики и обращается к API, запрашивая у сетевого программного обеспечения его дескриптор. Структура таблицы с описанием параметров сокета весьма незначительно отличается от структуры таблицы с описанием параметров файла. Однако это отличие важно.

Отличие дескриптора сокета от дескриптора файла

- Процессы получения дескриптора файла и сокета от операционной системы отличаются незначительно. Однако таблицы, на которые указывают дескрипторы, отличаются между собой. Тогда как 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. Тот факт, что дескриптор сокета не представляет определенную сетевую точку входа (endpoint), существенно отличает его от любого другого дескриптора стандартной системы ввода-вывода. В большинстве операционных систем дескриптор файла указывает на определенный файл, находящийся на жестком диске. Программы, работающие с сокетами, сначала образуют сокет и только потом соединяют его с точкой назначения на другом конце сетевого соединения. Если бы файловый ввод-вывод состоял из этих же шагов, приложение сначала получало бы дескриптор файла, а затем привязывало бы его к имени определенного файла на жестком диске.

Создание сокета

- Создавая программу TCP/IP, необходимо иметь возможность пользоваться как ориентированными, так и не ориентированными на соединение протоколами. Интерфейс сокетов позволяет программам использовать оба этих типа протоколов. Однако процессы создания сокета и соединения сокета с компьютером-получателем происходят раздельно. Чтобы создать сокет, программа вызывает функцию socket. Она, в свою очередь, возвращает дескриптор сокета, подобный дескриптору файла. Другими словами, дескриптор сокета указывает на таблицу, содержащую описание свойств и структуры сокета.

Создание сокета

- Следующий пример показывает возможную форму вызова функции socket:

  socket_handle = socket(protocol_family, socket_type, protocol);

Создание сокета

- Создавая сокет, вы указываете три параметра: группу, к которой принадлежит протокол, тип сокета и сам протокол. Первый параметр задает группу или семейство, к которому принадлежит протокол, например семейство TCP/IP. Второй параметр, тип сокета, задает режим соединения: датаграммный или ориентированный на поток байтов. Параметр “протокол” определяет протокол, с которым будет работать сокет, например TCP. В следующих разделах мы подробно обсудим различные параметры функции socket.

Параметры сокета

- Конечно, разработка API изначально ориентировалась на TCP/IP, однако и другие сети не были забыты. Различные семейства протоколов появились благодаря концепции универсальности API сокетов.

Семейства протоколов и адресов

- Для указания группы протоколов в интерфейсе сокетов определены символьные константы (макроопределения). Символьная константа **PF_INET**, например, определяет семейство протоколов TCP/IP. Константы, определяющие другие семейства, также начинаются с префикса “**PF_**”. Константа **PF_UNIX** определяет семейство внутренних протоколов ОС UNIX, а **PF_NS** — семейство протоколов фирмы Ксерокс.
- *Семейства адресов* тесно связаны с семействами протоколов. Форматы адресов различных сетей не одинаковы. Разработчики сокетов в полном соответствии с этим соображением еще больше обобщили интерфейс сокетов, реализовав для работы с различными сетями возможность обращения к различным семействам сетевых адресов. Символьные константы, указывающие на семейство адресов, начинаются с префикса “**AF_**”. Константа, обозначающая семейство адресов Интернет (TCP/IP), называется **AF_INET**. Константа AF_NS обозначает семейство адресов фирмы Ксерокс, а **AF_UNIX** — файловой системы UNIX.

Семейства протоколов и адресов

- К сожалению, ввиду тесной взаимосвязи между семействами протоколов и семействами адресов, существует ошибочное мнение, что это одно и то же.
- На сегодняшний день константы **PF_INET** и **AF_INET** имеют одинаковые значения, поэтому все равно, как их применять.

Тип соединения

- Соединения TCP/IP бывают двух режимов: ориентированные и не ориентированные на соединение. В ориентированных на соединение протоколах данные перемещаются как единый, последовательный поток байтов без какого-либо деления на блоки. В не ориентированных на соединение протоколах сетевые данные перемещаются в виде отдельных пакетов, называемых датаграммами. Как уже отмечалось, сокеты могут работать как с не ориентированными, так и с ориентированными на соединение протоколами.
- Второй параметр вызова функции socket обозначает тип соединения, который вы желаете использовать. Символьная константа **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

- Следующий оператор демонстрирует, как может выглядеть вызов функции socket:
- **socket_handle = 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**).

Упрощенная структура данных сокета

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

- Структура данных сокета включает элементы для хранения аргументов, с которыми была вызвана функция socket. Кроме того, в структуре размещены четыре адреса: локальный IP-адрес, удаленный IP-адрес, адреса локального и удаленного портов. 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. Для того чтобы передавать данные через сокет, можно воспользоваться одним из двух типов сетевых служб: датаграммной или ориентированной на поток байтов.
- Также сокет можно настроить на один из двух типов поведения программы: клиент или сервер. Ниже в таблице перечислены функции API, используемые для конфигурирования сокета.
- Любой сокет должен содержать пять блоков информации, описывающей соединение: протокол, местный и удаленный IP-адреса, номера местного и удаленного портов.

Функции интерфейса сокетов, используемые при конфигурации

Использование сокета	**Местный процесс**	**Удаленный процесс**
Ориентированный на соединение клиент	Вызов функции connect() записывает в структуру данных сокета информацию как о местном, так и об удаленном участниках соединения	
Ориентированный на соединение сервер	bind()	listen() и accept()
Не ориентированный на соединение клиент	bind()	sendto()
Не ориентированный на соединение сервер	bind()	recvfrom()

Соединение сокета

- Сокет представляет абстракцию, пользуясь которой вы можете настраивать и программировать конечные точки сетевого соединения.
- 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. Функция socket всегда вызывается до того, как устанавливается соединение. Дескриптор сокета указывает программному обеспечению, какая именно структура данных в таблице дескрипторов имеется в виду. Дескриптор также сообщает о том, куда нужно записать информацию об адресах удаленного участника соединения.

- **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. Во второй части книги вы узнаете больше об этой структуре. На данный момент нам важно знать, что структура данных сокета содержит семейство адресов, порт протокола и адрес сетевого компьютера. Функция connect записывает эту информацию в таблицу дескрииторов сокетов, на которую указывает соответствующий дескриптор сокета (первый параметр функции connect).
- До того как вызвать функцию connect, информацию об адресах удаленного компьютера нужно занести в структуру данных сокета. Другими словами, функция connect должна знать сетевой адрес и номер порта удаленного компьютера. Местный IP-адрес, однако, можно не указывать.

Функция connect

- **Третий параметр** функции connect, длина адреса, сообщает интерфейсу длину структуры данных адресов удаленного сокета (второй параметр), измеренную в байтах. Содержимое (и длина) этой структуры зависит от конкретной сети. Зная длину структуры, интерфейс сокетов представляет, сколько памяти отведено для хранения этой структуры. Когда реализация сокетов выполняет функцию connect, она извлекает количество байтов, указанное третьим параметром из буфера данных, на который указывает параметр “адрес удаленного сокета”.

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 интерфейс сокетов	Описание
send	Передает данные через соединенный сокет. Использует некоторые флаги для управления поведением сокета.
write	Передает данные через соединенный сокет. Для передачи используется буфер данных.
writev	Передает данные через соединенный сокет. В качестве буфера используется раздельно расположенные блоки памяти.
sendto	Передает данные через не соединенный сокет. Использует буфер данных.
sendmsg	Передает данные через не соединенный сокет. В качестве буфера используется гибкая структура сообщения.

- Следующий оператор демонстрирует типичный вызов функции 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 вызывается так, как показано ниже:
- **result = writev(socket_handle, io_vector, vector_length) ;**

- Так же, как и в случае write, функция writev требует, чтобы первым параметром указывался дескриптор сокета.
- Второй параметр, вектор ввода-вывода, указывает на массив указателей. 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.
- Третий параметр функции writev определяет количество указателей в массиве указателей, заданном вектором ввода-вывода.

- Следующий оператор является образцом вызова функции send:
- **result = send(socket_handle, message_buffer, buffer_length, special_flags) ;**
- Основное преимущество send состоит в том, что приложение может задать некоторые флаги для управления передачей данных.

- Три вышеописанные функции (write, writev и send) возвращают целое число в качестве результата. Если не произошла ошибка, результат будет равен количеству переданных байтов. В случае ошибки, возвращаемое значение результата будет равно -1.
- **Все три функции относятся к передаче данных через соединенный сокет**

Передача данных через не соединенный сокет

- Для того чтобы послать данные через не соединенный сокет, требуется вызвать одну из двух следующих функций: sendto или sendmsg. Они обеспечиваются интерфейсом сокетов именно для этих целей. Функция sendto требует шесть параметров в качестве аргументов. 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. Следующий оператор демонстрирует вызов функции sendto:
- **result = sendto(socket_handle, message_buffer, buffer_length, special_flags, socket_address_structure, address_structure_length) ;**

- Функция sendmsg позволяет использовать гибкую структуру данных вместо буфера, расположенного в непрерывной области памяти. Следующий оператор демонстрирует вызов sendmsg. В качестве аргументов указываются дескриптор сокета, указатель на структуру данных и дополнительные флаги:
- **result = sendmsg(socket_handle, message_structure, special_flags);**
- **Структура сообщения позволяет программе гибко размещать длинные списки параметров сообщения в единой структуре данных. Функция sendmsg похожа на writev в том, что прикладная программа может разместить свои данные в нескольких раздельно расположенных блоках памяти. Другими словами, как и в функции writev, структура сообщения содержит указатель на массив адресов памяти.**

Формат структуры сообщения, используемый функцией sendmsg

---32 бита---

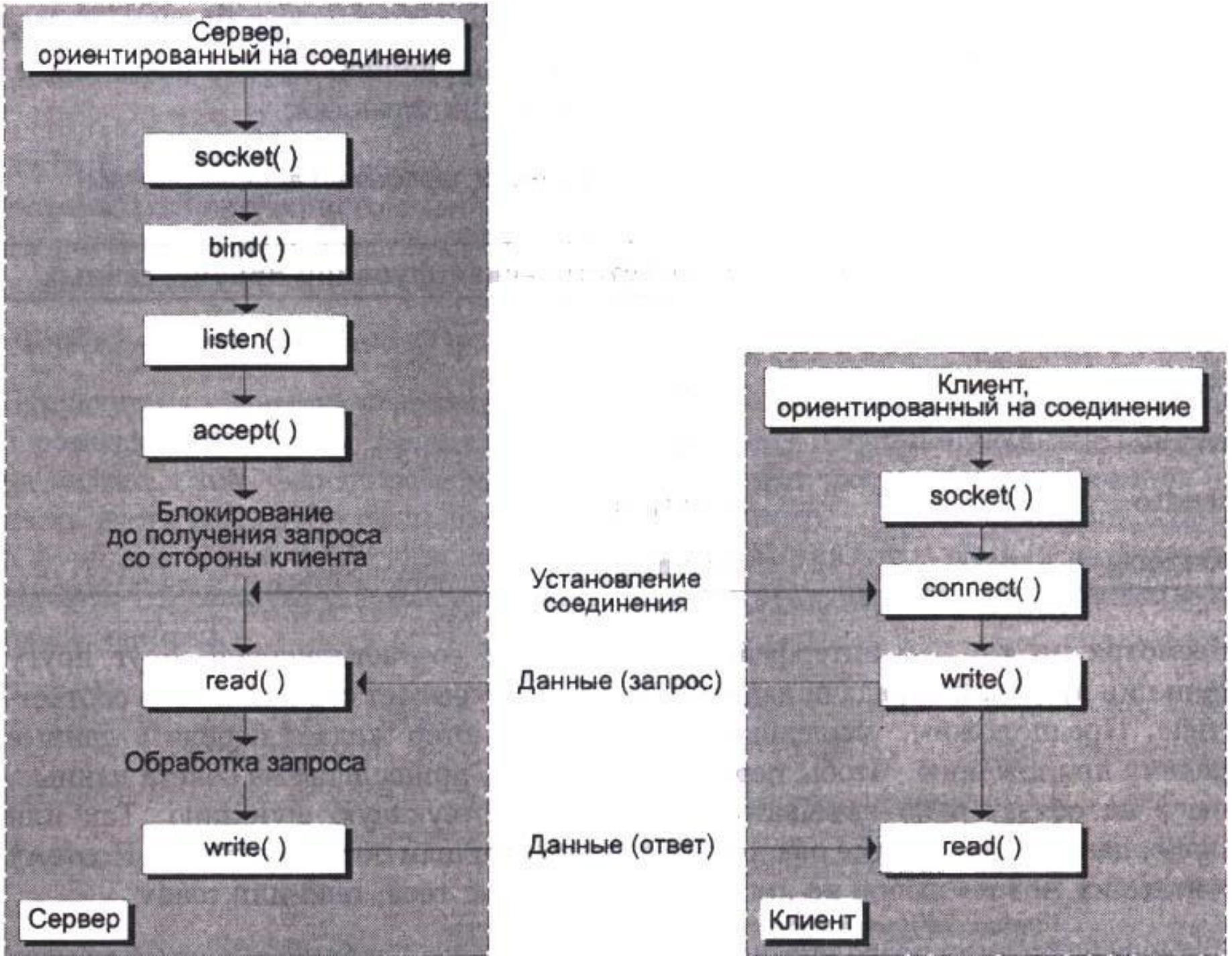
- Указатель на структуру данных сокета
- Длина структуры данных сокета
- Указатель на список векторов ввода/вывода
- Длина списка векторов ввода/вывода
- Указатель на список прав доступа
- Длина списка прав доступа

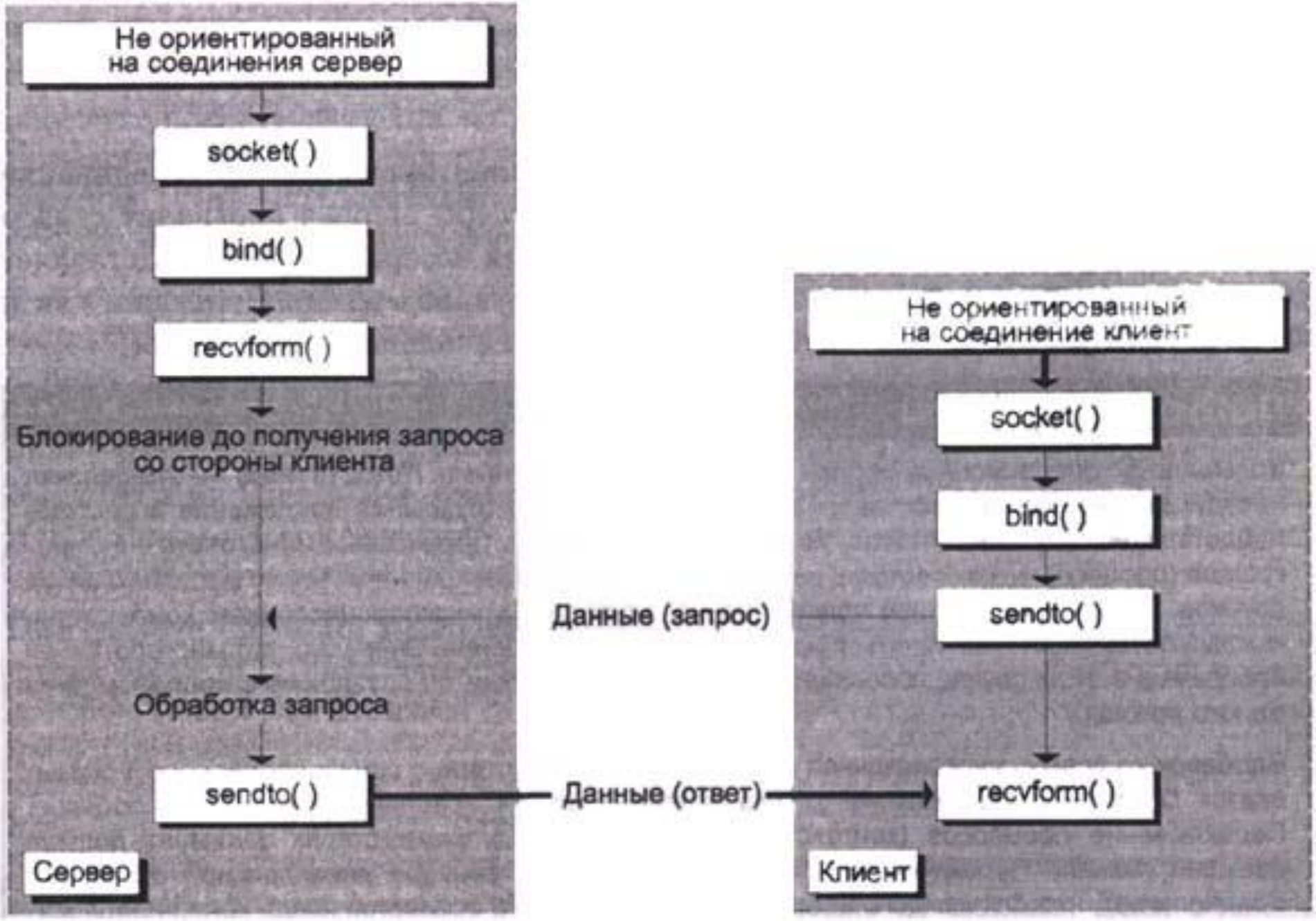
Прием данных через сокет

- 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. Функция recv принимает данные, а send — отправляет. 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.

. Соответствующие друг другу функции передачи и приема данных интерфейса сокетов

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


Сервер, ориентированный на соединение
socket( )
bind( )
listen( )
accept( )
Блокирование до получения запроса со стороны клиента
read( )
Обработка запроса
write( )
Сервер
Установление соединения
Данные (запрос)
Данные (ответ)
Клиент, ориентированный на соединение
socket( )
connect( )
write( )
read( )
Клиент


Не ориентированный на соединения сервер
socket( )
bind( )
recvform( )
Блокирование до получения запроса со стороны клиента
Обработка запроса
sendto( )
Сервер
Данные (запрос)
Данные (ответ)
Не ориентированный на соединение клиент
socket( )
bind( )
sendto( )
recvform( )
Клиент

Функция listen

- Функция listen не только переводит сокет в пассивный режим ожидания, но и подготавливает его к обработке множества одновременно поступающих запросов. Другими словами, в системе организуется очередь поступивших запросов, и все запросы, ожидающие обработки сервером, помещаются в нее, пока освободившийся сервер не выберет его.
- При вызове функции listen указываются два параметра: дескриптор сокета и длина очереди. Длина очереди обозначает максимальное количество запросов, которое может поместиться в ней. Следующий оператор — образец вызова функции listen:
- **result = listen(socket_handle, queue_length);**

Функция listen

- В настоящее время максимальная длина очереди равна пяти. Если вы попытаетесь указать большее число, то получите сообщение об ошибке. Если очередь при поступлении нового запроса окажется переполненной, сокет отвергнет соединение и программа-клиент получит сообщение об ошибке.
- В случае последовательного сервера, задавать длину очереди, равную одному или двум, тоже полезно. Это позволит серверу, если он не справился с обработкой за минимальное назначенное время, все-таки не отвергнуть новый запрос, а выбрать его из входной очереди.

Функция accept

- Как следует из названия, 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. Для того чтобы понять смысл всех операций сервера, мы должны подробно рассмотреть, как функционирует accept.
- При вызове accept требуется указывать три параметра: дескриптор сокета, его адрес и длину адреса. 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 accept

- Следующий оператор демонстрирует, как можно вызывать функцию accept:
- **result = accept(socket_handle, socket_address, address_length);**

- После того как реализация сокетов поместит информацию об адресе клиента в область памяти, заданную параметром функции, она выполнит операции, необходимые для того, чтобы сервер заработал. Первым делом, получив запрос соединения на сокет, обслуживаемый функцией accept, реализация образует новый сокет. Новый сокет связывается с адресом процесса-передатчика запроса.
- Коротко весь процесс можно описать следующим образом: когда на сокете, контролируемом функцией accept, появляется очередной запрос клиента, программное обеспечение-реализация сокетов автоматически создает новый сокет и немедленно соединяет его с процессом-клиентом. Сокет, на который поступил запрос, освобождается и продолжает работу в режиме ожидания запросов от любого сетевого компьютера.

Функция select

- Функция select позволяет одиночному процессу следить за состоянием сразу нескольких сокетов. При вызове select указываются пять параметров. 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.

Функция select

- Приведенный ниже оператор демонстрирует, как можно вызвать функцию select:
- result = select(number_of„sockets, readable_sockets,
- writeable_sockets, error_sockets, max_time);

- Сокет для чтения (readable socket) содержит принятые данные, которые извлекаются программой при помощи стандартных вызовов recv или recvfrom. Сокет для записи (writable socket) — это сокет, установивший соединение. Через него программа может передавать данные, используя стандартные функции типа send или sendto.

- В случае сетевой ошибки, функция select обозначает сокет, в котором это произошло, как ошибочный (exception). Ситуация ошибки требует дальнейшей программной обработки. В любом случае select определяет состояние только тех сокетов, которые были отмечены в каждой битовой маске.
- Интерфейс сокетов использует битовые маски для определения набора сокетов, за которыми будет установлено наблюдение.
- При выходе в вызывающую программу select возвращает количество сокетов, готовых к операциям ввода-вывода.