Лабораторная работа № 1

Клиент-серверное приложение для передачи сообщений с использованием протоколов TCP и UDP

1.1 Цель работы

Целью лабораторной работы является ознакомление студентов с основной структурой клиент-серверного приложения на примере написания программ передачи/приема текстовых сообщений с использованием сокетов в блокирующем режиме.

1.2 Задание на лабораторную работу

В первой лабораторной работе необходимо разработать 2 клиент-серверных приложения на основе протоколов TCP и UDP. Клиент должен подключиться к серверу и отправить ему текстовое сообщение; сервер, получив сообщение, должен вывести его на экран.

1.3 Методические указания

1.3.1 Понятие сокета

Сокеты (sockets) представляют собой высокоуровневый унифицированный интерфейс взаимодействия с телекоммуникационными протоколами. Приложение просто пишет данные в сокет, их дальнейшая буферизация, отправка и транспортировка осуществляется используемым стеком протоколов и сетевой аппаратурой. Чтение данных из сокета происходит аналогичным образом.

В программе сокет идентифицируется дескриптором - это просто переменная типа int(в ОС Windows тип SOCKET см. таблицу 1.1). Программа получает дескриптор от операционной системы при создании сокета, а затем передаёт его сервисам socket API для выполнения того или иного действия.

1.3.2 Обзор сокетов

Существует два вида сокетов - ***синхронные*** (блокируемые) и ***асинхронные*** (неблокируемые). ***Синхронные*** сокеты задерживают управление на время выполнения операции, а ***асинхронные*** возвращают его немедленно, продолжая выполнение в фоновом режиме.

Сокеты позволяют работать с множеством протоколов и являются удобным средством межпроцессорного взаимодействия, но в рамках данного лабораторного практикума речь будет идти только о сокетах семейства протоколов TCP/IP, использующихся для обмена данными между узлами сети Интернет.

Независимо от вида, сокеты делятся на два типа - ***потоковые*** и ***датаграммные***. ***Потоковые*** сокеты работают с установкой соединения, обеспечивая надежную идентификацию обоих сторон и гарантируют целостность и успешность доставки данных. ***Датаграмные*** сокеты работают без установки соединения и не обеспечивают контроля успешности доставки данных, зато они значительно быстрее потоковых.

Выбор того или иного типа сокетов определяется транспортным протоколом, на котором работает сервер, клиент не может по своему желанию установить с датаграммным сервером потоковое соединение.

Датаграммные сокеты опираются на протокол UDP, а потоковые - на TCP.

Для работы с библиотекой Winsock 2.x в исходный текст программы необходимо включить директиву "#include <winsock2.h>", а в командной строке линкера указать "ws2_32.lib". В Microsoft Visual Studio для этого достаточно нажать <Alt-F7>, перейти к закладке "Link" и к списку библиотек, перечисленных в строке "Object/Library modules", добавить "ws2_32.lib", отделив ее от остальных символом пробела.

Перед началом использования функций библиотеки Winsock, ее необходимо подготовить к работе вызовом функции "int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData)", передав в старшем байте слова wVersionRequested номер требуемой версии, а в младшем - номер подверсии.

Аргумент lpWSAData должен указывать на структуру WSADATA, в которую при успешной инициализации будет занесена информация о производителе библиотеки. Если инициализация проваливается, функция возвращает ненулевое значение.

Для работы с сокетами в *nix никаких дополнительных действий не требуется, достаточно подключить необходимые заголовочные файлы.

1.3.3 Атрибуты сокета

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 используется функция ***socket***, имеющая следующий прототип:

Таблица 1.1

Описание функции в *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int socket (int domain, int type, int protocol);	SOCKET socket (int domain, int type, int protocol);

Домен определяет пространство адресов, в котором располагается сокет, и множество протоколов, которые используются для передачи данных. Мы будем использовать домен Internet, задаваемый константой ***AF_INET*** (префикс AF означает "address family" - "семейство адресов"). Сокеты, размещённые в этом домене, могут использоваться для работы в любой IP-сети. Существуют и другие домены (AF_IPX для протоколов Novell, AF_INET6 для новой модификации протокола IP - IPv6 и т. д.).

Тип сокета определяет способ передачи данных по сети. Чаще других применяются:

- ***SOCK_STREAM.*** Передача потока данных с предварительной установкой соединения. Обеспечивается надёжный канал передачи данных, при котором фрагменты отправленного блока не теряются, не переупорядочиваются и не дублируются.

- ***SOCK_DGRAM.*** Передача данных в виде отдельных сообщений (датаграмм). Предварительная установка соединения не требуется. Обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться, дублироваться и переупорядочиваться. Допускается передача сообщения нескольким получателям (multicasting) и широковещательная передача (broadcasting).

- ***SOCK_RAW.*** Этот тип присваивается низкоуровневым (т. н. "сырым") сокетам.

Обратите внимание, что не все домены допускают задание произвольного типа сокета. Например, совместно с доменом Unix используется только тип ***SOCK_STREAM***. С другой стороны, для Internet-домена можно задавать любой из перечисленных типов. В этом случае для реализации ***SOCK_STREAM*** используется протокол TCP, для реализации ***SOCK_DGRAM*** - протокол UDP, а тип ***SOCK_RAW*** используется для низкоуровневой работы с протоколами IP, ICMP и т. д.

Наконец, последний атрибут определяет протокол, используемый для передачи данных. Протокол часто однозначно определяется по домену и типу сокета. В этом случае в качестве третьего параметра функции ***socket*** можно передать 0, что соответствует протоколу по умолчанию. Тем не менее, иногда (например, при работе с низкоуровневыми сокетами) требуется задать

протокол явно. Числовые идентификаторы протоколов зависят от выбранного домена; их можно найти в документации

1.3.4 Адреса

Прежде чем передавать данные через сокет, его необходимо связать с адресом в выбранном домене (эту процедуру называют именованием сокета). Иногда связывание осуществляется ***неявно*** (внутри функций ***connect*** и ***accept***), но выполнять его необходимо во всех случаях. Вид адреса зависит от выбранного вами домена. В Internet-домене адрес задаётся комбинацией IP-адреса и 16-битного номера порта.

Для явного связывания сокета с некоторым адресом используется функция ***bind.*** Её прототип имеет вид:

Таблица 1.2

Описание функции в *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int bind(int sockfd, struct sockaddr *addr, int addrlen);	int bind(SOCKET sockfd, struct sockaddr *addr, int addrlen);

В качестве первого параметра передаётся дескриптор сокета, который мы хотим привязать к заданному адресу. Второй параметр, ***addr***, содержит указатель на структуру с адресом, а третий - длину этой структуры. Посмотрим, что она собой представляет.

Таблица 1.3

Описание структуры в *nix	Описание структуры в Windows

struct sockaddr {
unsigned short sa_family; // Семейство адресов
char sa_data[14]; // 14 байтов для хранения адреса
};	

Поле ***sa_family*** содержит идентификатор домена, тот же, что и первый параметр функции ***socket***. В зависимости от значения этого поля по-разному интерпретируется содержимое массива ***sa_data***. Разумеется, работать с этим массивом напрямую не очень удобно, поэтому вы можете использовать вместо ***sockaddr*** структуру вида ***sockaddr_in***. При передаче в функцию ***bind*** указатель на эту структуру приводится к указателю на ***sockaddr***. Рассмотрим структуру ***sockaddr_in.***

Таблица 1.4

Описание структуры в *nix	Описание структуры в Windows

struct sockaddr_in {
short int sin_family; // Семейство адресов
unsigned short int sin_port; // Номер порта
struct in_addr sin_addr; // IP-адрес
unsigned char sin_zero[8]; // "Дополнение" до размера структуры sockaddr
};	

Здесь поле ***sin_family*** соответствует полю ***sa_family*** в ***sockaddr***, в ***sin_port*** записывается номер порта, а в ***sin_addr*** - IP-адрес хоста. Поле ***sin_addr*** само является структурой, которая имеет вид:

Таблица 1.5

Описание структуры в *nix	Описание структуры в Windows

struct in_addr {
unsigned long s_addr;
};	

Раньше ***in_addr*** представляла собой объединение, содержащее гораздо большее число полей. Сейчас, когда в ней осталось всего одно поле, она продолжает использоваться для обратной совместимости.

1.3.5 Порядок хранения байт

Существует два порядка хранения байтов в слове и двойном слове. Один из них называется порядком хоста (host byte order), другой - сетевым порядком (network byte order) хранения байтов. При указании IP-адреса и номера порта необходимо преобразовать число из порядка хоста в сетевой. Для этого используются функции ***htons*** (Host to Network Short) и ***htonl*** (Host to Network Long). Обратное преобразование выполняют функции ***ntohs*** и ***ntohl.***

Таблица 1.6

Описание функции *nix	Описание функции в Windows
#include <netinet/in.h>	#include <winsock2.h>
unsigned short int htons(unsigned short int hostshort);	unsigned short int htons(unsigned short int hostshort);
unsigned long int htonl(unsigned long int hostlong);	unsigned long int htonl(unsigned long int hostlong);
unsigned short int ntohs(unsigned short int netshort);	unsigned short int ntohs(unsigned short int netshort);
unsigned long int ntohl(unsigned long int netlong);	unsigned long int ntohl(unsigned long int netlong);

1.3.6 Клиент-серверные приложения с установлением соединения (сервер)

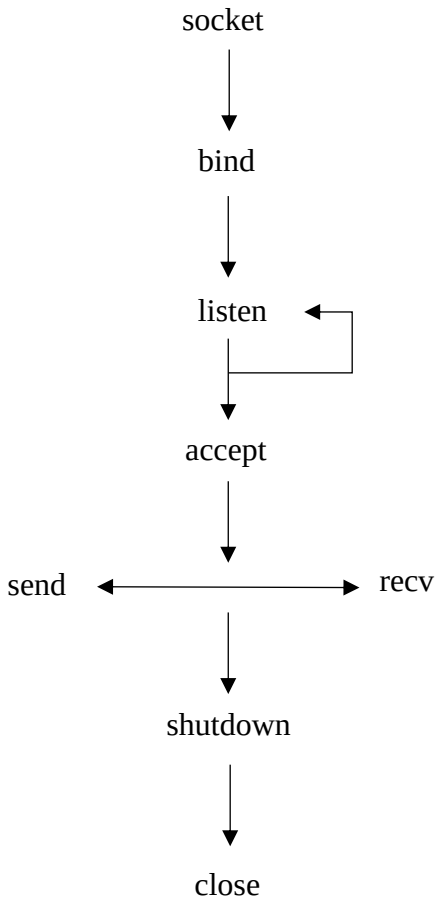

Рисунок 1.1 Клиент-серверное приложение с установлением соединения (сервер)

Установка соединения на стороне сервера состоит из следующих этапов (см. рисунок 1.1). Сначала сокет создаётся и привязывается к локальному адресу. Если компьютер имеет несколько сетевых интерфейсов с различными IP-адресами, вы можете принимать соединения только с одного из них, передав его адрес функции ***bind***. Если же вы готовы соединяться с клиентами через любой интерфейс, задайте в качестве адреса константу ***INADDR_ANY***. Что касается номера порта, вы можете задать конкретный номер или 0 (в этом случае система сама выберет произвольный неиспользуемый в данный момент номер порта).

На следующем шаге создаётся очередь для ожидания запросов на соединение. При этом сокет переводится в режим ожидания запросов со стороны клиентов. Всё это выполняет функция ***listen.***

Таблица 1.7

Описание в функции *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int listen(int sockfd, int size);	int listen(SOCKET sockfd, int size);

Первый параметр - дескриптор сокета, а второй задаёт размер очереди запросов. Каждый раз, когда очередной клиент пытается соединиться с сервером, его запрос ставится в очередь, так как сервер может быть занят

обработкой других запросов. Если очередь заполнена, все последующие запросы будут игнорироваться. Когда сервер готов обслужить очередной запрос, он использует функцию ***accept.***

Таблица 1.8

Описание в функции *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int accept(int sockfd, void *addr, int *addrlen);	SOCKET accept(SOCKET sockfd, struct sockaddr *addr, int *addrlen);

Функция ***accept*** создаёт для общения с клиентом новый сокет и возвращает его дескриптор. Параметр sockfd задаёт слушающий сокет. После вызова он остаётся в слушающем состоянии и может принимать другие соединения. В структуру, на которую ссылается addr, ***записывается адрес сокета клиента***, который установил соединение с сервером. В переменную, адресуемую указателем ***addrlen***, изначально записывается размер структуры; функция accept записывает туда длину, которая реально была использована. Если вас не интересует адрес клиента, вы можете просто передать NULL в качестве второго и третьего параметров.

Обратите внимание, что полученный от ***accept*** новый сокет связан с тем же самым адресом, что и слушающий сокет. Сначала это может показаться странным. Но дело в том, что адрес TCP-сокета не обязан быть уникальным в Internet-домене. Уникальными должны быть только соединения, для идентификации которых используются два адреса сокетов, между которыми происходит обмен данными.

1.3.7 Клиент-серверные приложения с установлением соединения (клиент)

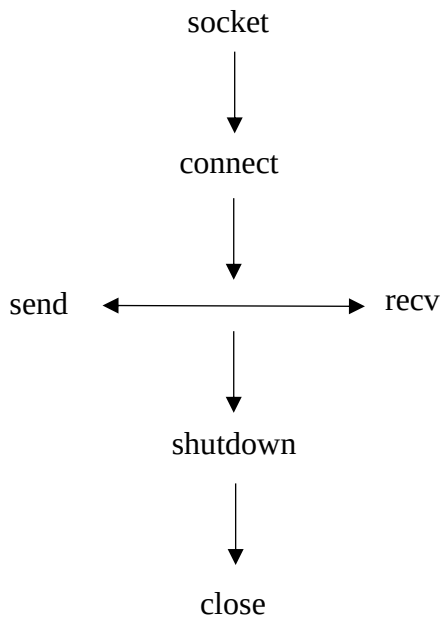

Рисунок 1.2 Клиент-серверное приложение с установлением соединения (клиент)

На стороне клиента для установления соединения используется функция ***connect,*** которая имеет следующий прототип.

Таблица 1.9

Описание в функции *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);	int connect(SOCKET sockfd, struct sockaddr * serv_addr, int addrlen);

Здесь ***sockfd*** - сокет, который будет использоваться для обмена данными с сервером, ***serv_addr*** содержит указатель на структуру с адресом сервера, а ***addrlen*** - длину этой структуры. Обычно ***сокет не требуется предварительно привязывать к локальному адресу, так как функция connect сделает это*** за вас, подобрав подходящий свободный порт. Вы можете принудительно назначить клиентскому сокету некоторый номер порта, используя ***bind*** перед вызовом ***connect***. Делать это следует в случае, когда сервер соединяется только с клиентами, использующими

определённый порт. В остальных случаях проще и надёжнее предоставить системе выбрать порт за вас.

1.3.8 Обмен данными

После того как соединение установлено, можно начинать обмен данными. Для этого используются функции ***send*** и ***recv***. В Unix для работы с сокетами можно использовать также файловые функции ***read*** и ***write***, но они обладают меньшими возможностями, а кроме того не будут работать на других платформах (например, под Windows), поэтому ими лучше не пользоваться.

Функция ***send*** используется для отправки данных и имеет следующий прототип:

Таблица 1.10

Описание в функции *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int send(int sockfd, const char *msg, int len, int flags);	int send(SOCKET sockfd, const char *msg, int len, int flags);

Здесь ***sockfd*** - это дескриптор сокета, через который мы отправляем данные, ***msg*** - указатель на буфер с данными, ***len*** - длина буфера в байтах, а ***flags*** - набор битовых флагов, управляющих работой функции (если флаги не используются, передайте функции 0). Вот некоторые из них (полный список можно найти в документации):

- **MSG_OOB**. Предписывает отправить данные как срочные (out of band data, OOB). Концепция срочных данных позволяет иметь два параллельных канала данных в одном соединении. Иногда это бывает удобно. Например, Telnet использует срочные данные для передачи команд типа Ctrl+C. В настоящее время использовать их не рекомендуется из-за проблем с совместимостью (существует два разных стандарта их использования, описанные в RFC793 и RFC1122). Безопаснее просто создать для срочных данных отдельное соединение.
- **MSG_DONTROUTE**. Запрещает маршрутизацию пакетов. Нижележащие транспортные слои могут проигнорировать этот флаг.

Функция ***send*** возвращает число байтов, которое на самом деле было отправлено (или -1 в случае ошибки, для *nix и ***SOCKET_ERROR*** в

Windows). Это число может быть меньше указанного размера буфера. Если вы хотите отправить весь буфер целиком, вам придётся написать свою функцию и вызывать в ней ***send***, пока все данные не будут отправлены.

Для чтения данных из сокета используется функция ***recv***.

Таблица 1.11

Описание функции в *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int recv(int sockfd, char *buf, int len, int flags);	int recv(SOCKET sockfd, char *buf, int len, int flags);

В целом её использование аналогично ***send***. Она точно так же принимает дескриптор сокета, указатель на буфер и набор флагов. Флаг ***MSG_OOB*** используется для приёма срочных данных, а ***MSG_PEEK*** позволяет "подсмотреть" данные, полученные от удалённого хоста, не удаляя их из системного буфера (это означает, что при следующем обращении к recv вы получите те же самые данные). Полный список флагов можно найти в документации. По аналогии с ***send*** функция ***recv*** возвращает количество прочитанных байтов, которое может быть меньше размера буфера. Существует ещё один особый случай, при котором ***recv*** возвращает 0. Это означает, что соединение было разорвано.

1.3.9 Закрытие сокета

Правильно написанное приложение уведомляет получателя об окончании отправки данных. Такое поведение называется корректным завершением сеанса и осуществляется с помощью функции ***shutdown:***

Таблица 1.12

Описание функции в *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int shutdown(int sockfd, int how,);	int shutdown(SOCKET sockfd, int how,);

Параметр ***how*** может принимать одно из следующих значений:
0 - запретить чтение из сокета
1 - запретить запись в сокет
2 - запретить и то и другое

Хотя после вызова ***shutdown*** с параметром ***how***, равным 2, вы больше не сможете использовать сокет для обмена данными, вам всё равно потребуется вызвать ***close***, чтобы освободить связанные с ним системные ресурсы.

Закончив обмен данными, закройте сокет с помощью функции ***close***. Это приведёт к разрыву соединения.

Таблица 1.13

Описание функции в *nix	Описание функции в Windows
#include <unistd.h>	#include <winsock2.h>
int close(int sockfd);	int closesocket(SOCKET sockfd);

Вызов функции ***close (closesocket)*** освобождается дескриптор сокета, и все дальнейшие операции с сокетом закончатся с ошибкой.

Примечание: *При использовании WinSocket, перед выходом из программы необходимо вызвать функцию* ***"int WSACleanup (void)"*** *для деинициализации библиотеки WINSOCK и освобождения используемых этим приложением ресурсов.*

1.3.10 Клиент-серверные приложения без установления соединения (датаграммные приложения)

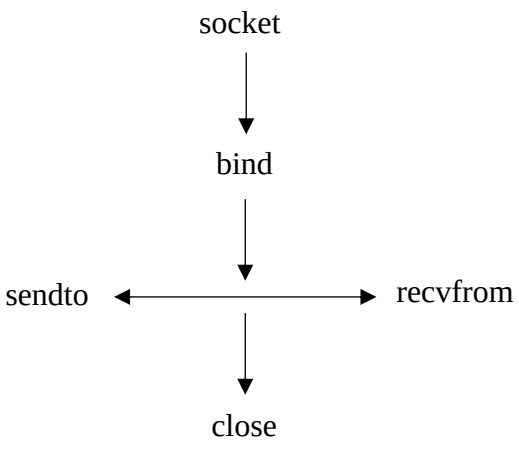

Рисунок 1.3 Обмен датаграммами

В датаграммных клиент-серверных приложениях не осуществляется установления соединения. При этом обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться, дублироваться и переупорядочиваться. Допускается передача сообщения нескольким получателям (multicasting) и широковещательная передача (broadcasting). Иногда датаграммы

оказываются полезны. Например, их удобно использовать при транслировании звука или видео по сети в реальном времени, особенно при широковещательном транслировании. Поскольку для обмена датаграммами не нужно устанавливать соединение, использовать их гораздо проще. Создав сокет с помощью ***socket*** и ***bind***, вы можете тут же использовать его для отправки или получения данных. Для этого вам понадобятся функции ***sendto*** и ***recvfrom***. (рисунок 1.3)

Таблица 1.14

Описание функции в *nix	Описание функции в Windows
#include <sys/types.h> #include <sys/socket.h>	#include <winsock2.h>
int sendto(int sockfd, const char *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);	int sendto(SOCKET sockfd, const char *msg, int len, unsigned int flags, const struct sockaddr *to, int len);
int recvfrom(int sockfd, char *buf, int len, unsigned int flags, const struct sockaddr *from, int fromlen);	int recvfrom(SOCKET sockfd, char *buf, int len, unsigned int flags, const struct sockaddr *from, int fromlen);

Функция ***sendto*** очень похожа на ***send***. Два дополнительных параметра ***to*** и ***tolen*** используются для указания адреса получателя. Для задания адреса используется структура ***sockaddr***, как и в случае с функцией ***connect***. Функция ***recvfrom*** работает аналогично ***recv***. Получив очередное сообщение, она записывает его адрес в структуру, на которую ссылается ***from***, а записанное количество байт - в переменную, адресуемую указателем ***fromlen***. Как мы знаем, аналогичным образом работает функция ***accept***.

Для сокета с типом ***SOCK_DGRAM*** тоже можно вызвать функцию ***connect***, а затем использовать ***send*** и ***recv*** для обмена данными.

1.3.11 Обработка ошибок

В процессе работы с сокетами часто возникают ошибки. Для обработки ошибок существуют функции:

Таблица 1.15

Описание функции в *nix	Описание функции в Windows
include <stdio.h>	#include <winsock2.h>
void perror(const char* prefix); Если параметр prefix не NULL, то perror сначала печатает prefix, двоеточие и пробел перед описанием ошибки. Затем печатает результат errno в stderr, завершаемый символом новой строки.	int WSAGetLastError(void); Функция WSAGetLastError позволяет определить код ошибки при неудачном завершении практически всех функций интерфейса Windows Sockets(код ошибки запишется в переменную SOCKET_ERROR). Вы должны вызывать ее сразу вслед за функцией, завершившейся неудачно.

Соответственно, вы можете проанализировать значение соответствующей переменной и предпринять действия по восстановлению нормальной работы программы, не прерывая её выполнения. А можете просто выдать диагностическое сообщение, а затем завершить программу.

1.4 Контрольные вопросы и задания

1. Описать процесс создания сокета.
2. Описать назначение структур sockaddr, sockaddr_in, in_addr. и их полей.
3. Описать процесс установление соединения.
4. Объяснить различия между обменом пакетами в ориентированных на соединение протоколах и не ориентированных на соединение протоколах.
5. Пояснить назначение функции htonl и ntohl.
6. Объяснить различия между работой с сокетами в Windows и Linux.