## 1.1 Лабораторная работа №1 Работа с Socket

### 1.1.1 Определения

Socket (гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью. При использовании протоколов TCP/IP можно говорить, что socket является средством подключения прикладной программы к порту локального узла сети.

### 1.1.2 Создание сервера

Для создания сокета в операционной системе служит системный вызов socket(). Для транспортных протоколов семейства TCP/IP существует два вида сокетов:

UDP-сокет – сокет для работы с датаграммами, и TCP сокет – для работы с каналами. Соответственно в стеке TCP/IP протокол TCP отвечает за надежную передачу потока данных, реализует он эту надежность за счет повторной отправки данных, на которые не пришли подтверждения за выделенное специальным таймером время, протокол UDP позволяет отправить дейтаграммы, но надежность передачи не обеспечивает, то есть дейтаграммы могут и не дойти.

При создании сокета необходимо точно специфицировать его тип. Эта спецификация производится с помощью трех параметров вызова socket(). Первый параметр указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства.

Создание сокета осуществляется следующим системным вызовом (Unix):

Определение функции:

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

Аргумент domain задает используемый для взаимодействия набор протоколов (вид коммуникационной области), для стека протоколов TCP/IP он должен иметь символьное значение AF\_INET или PF\_INET.

Аргумент type задает режим взаимодействия:

SOCK\_STREAM - с установлением соединения;

SOCK\_DGRAM - без установления соединения.

Аргумент protocol задает конкретный протокол транспортного уровня (из нескольких возможных в стеке протоколов). Если этот аргумент задан равным 0, то будет использован протокол "по умолчанию" (TCP для SOCK\_STREAM и UDP для SOCK\_DGRAM при использовании комплекта протоколов TCP/IP).

При удачном завершении своей работы данная функция возвращает дескриптор сокета - целое неотрицательное число, однозначно его идентифицирующее.

При обнаружении ошибки в ходе своей работы функция возвращает число "-1".

Далее текст кода для создания программы сервера или клиента будет отмечен цветом.

Указанный сокет можно соответственно подключив следующие библиотеки (POSIX):

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

Либо, в операционной системе Windows:

#include <winsock2.h>

и соответственно подключить библиотеку ws2\_32.lib.

Прежде чем воспользоваться функцией socket необходимо проинициализировать процесс библиотеки wsock32.dll вызвав функцию WSAStartup например:

WSADATA WsaData;

int err = WSAStartup (0x0101, &WsaData);

if (err == SOCKET\_ERROR) { printf ("WSAStartup() failed:

%ld\n", GetLastError ());

return 1;

}

Здесь 0х0101 версия библиотеки которую следует использовать.

Теперь объявить переменную типа SOCKET можно следующим образом:

Пример вызова:

int s = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

Далее следует задать параметры для сокета (сервера) для этого нам необходимо объявить структуру SOCKADDR\_IN sin примерно следующим образом:

SOCKADDR\_IN sin;// или struct sockaddr\_in sin;

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(80);

//преобразование из хостового в сетевой формат слова (см. также ntohs, ntohl, htonl – для одинакового восприятия числовых данных на всех архитектурах)

sin.sin\_addr.s\_addr = INADDR\_ANY;

Изучите так же функцию gethostbyname.

Структура SOCKADDR\_IN используется несколькими системными вызовами и функциями socket-интерфейса, ее определение в файле in.h выглядит следующим образом:

struct SOCKADDR\_IN { short sin\_family;

u\_short sin\_port;

struct in\_addr sin\_addr;

char sin\_zero[8];

};

Поле sin\_family определяет используемый формат адреса (набор протоколов), в нашем случае (для TCP/IP) оно должно иметь значение AF\_INET.

Поле sin\_addr содержит адрес (номер) узла сети.

Поле sin\_port содержит номер порта на узле сети.

Поле sin\_zero не используется.

Определение структуры in\_addr (из того же файла in.h) таково:

struct in\_addr { union { u\_long S\_addr;

/\* другие (не интересующие нас) члены объединения \*/ } S\_un;

#define s\_addr S\_un.S\_addr };

Структура SOCKADDR\_IN должна быть полностью заполнена перед выдачей системного вызова bind. При этом, если поле sin\_addr.s\_addr имеет значение INADDR\_ANY, то системный вызов будет привязывать к сокету c номером (адресом) локального узла сети.

Для подключения сокета к коммуникационной среде, образованной вычислительной сетью, необходимо выполнить системный вызов bind, определяющий в принятом для сети формате локальный адрес канала связи со средой. В сетях TCP/IP socket связывается с локальным портом.

Системный вызов bind имеет следующий синтаксис:

int bind (SOCKET s, SOCKADDR\_IN \*addr, int addrlen)

Пример:

err = bind( s, (LPSOCKADDR)&sin, sizeof(sin) ); // или int err = bind( s, (const struct sockaddr \*)&sin, sizeof(sin) );

Аргумент s задает дескриптор связываемого сокета.

Аргумент addr в общем случае должен указывать на структуру данных, содержащую локальный адрес, приписываемый сокету. Для сетей TCP/IP такой структурой является SOCKADDR\_IN.

Аргумент addrlen задает размер (в байтах) структуры данных, указываемой аргументом addr.

В случае успеха bind возвращает 0, в противном случае - "-1".

Для установления связи "клиент-сервер" используются системные вызовы listen и accept (на стороне сервера), а также connect (на стороне клиента). Для заполнения полей структуры sockaddr\_in, используемой в вызове connect, обычно используется библиотечная функция gethostbyname, транслирующая символическое имя узла сети в его номер (адрес).

Системный вызов listen выражает желание выдавшей его программы- сервера ожидать запросы к ней от программ-клиентов и имеет следующий вид:

int listen (SOCKET s, int n);

Пример:

err = listen(s, SOMAXCONN);

Аргумент s задает дескриптор сокета, через который программа будет ожидать запросы к ней от клиентов. Socket должен быть предварительно создан системным вызовом socket и обеспечен адресом с помощью системного вызова bind.

Аргумент n определяет максимальную длину очереди входящих запросов на установление связи. Если какой-либо клиент выдаст запрос на установление связи при полной очереди, то этот запрос будет отвергнут.

Признаком удачного завершения системного вызова listen служит нулевой код возврата.

Перед тем как воспользоваться функцией accept, необходимо объявить ещё одну переменную типа SOCKET, например s1 .

SOCKADDR\_IN from;

int fromlen=sizeof(from);

s1 = accept(s,(struct sockaddr\*)&from,&fromlen);

Это сделано для того, что бы узнать IP адрес и порт удаленного компьютера.

Для приема запросов от программ-клиентов на установление связи в программах-серверах используется системный вызов accept, имеющий следующий прототип:

int accept (SOCKET s, sockaddr\_in \*addr, int \*p\_addrlen;

Аргумент s задает дескриптор сокета, через который программа-сервер получила запрос на соединение (посредством системного запроса listen ).

Аргумент addr должен указывать на область памяти, размер которой позволял бы разместить в ней структуру данных, содержащую адрес сокета программы-клиента, сделавшей запрос на соединение. Никакой инициализации этой области не требуется.

Аргумент p\_addrlen должен указывать на область памяти в виде целого числа, задающего размер (в байтах) области памяти, указываемой аргументом addr.

Системный вызов accept извлекает из очереди, организованной системным вызовом listen, первый запрос на соединение и возвращает дескриптор нового (автоматически созданного) сокета с теми же свойствами, что и socket, задаваемый аргументом s. Этот новый дескриптор необходимо использовать во всех последующих операциях обмена данными.

Если очередь запросов на момент выполнения accept пуста, то программа переходит в состояние ожидания поступления запросов от клиентов на неопределенное время (хотя такое поведение accept можно и изменить).

Признаком неудачного завершения accept служит отрицательное возвращенное значение (дескриптор сокета отрицательным быть не может). Обычно для облуживания сервером краткосрочного взаимодействия со множеством клиентов можно реализовать «бесконечный» цикл в котором извлекается новый клиент из очереди, идет взаимодействие с ним по приему и передаче данных и последующее отключение, если же взаимодействие требует времени, или какой-то длительности по обмену сообщениями, то обслуживание клиента реализуется в новом потоке (либо организуется работа с не блокирующими сокетами (асинхронными), изучите также функцию select).

После установления соединения с клиентом можно передавать и получать данные. Для этого в операционной системе Window используются системные вызовы recv для чтения и send для записи.

Системные вызовы recv и send имеют следующие прототипы:

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

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

Возвращаемое значение:

число принятых или переданных байтов в случае успеха или -1 в случае ошибки Аргумент s задает дескриптор сокета, через который принимаются данные.

Аргумент buf для вызова recv указывает на область памяти, предназначенную для размещения принимаемых данных, а для вызова send - область памяти, содержащая передаваемые данные.

Аргумент len задает размер (в байтах) области buf.

Аргумент flags зависит от системы, но и UNIX, и Windows поддерживают следующие флаги:

MSG\_OOB – следует послать или принять срочные данные.

MSG\_PEEK – используется для просмотра поступивших данных без их удаления из приемного буфера. После возврата из системного вызова данные еще могут быть получены при последующем вызове recv.

MSG\_DONTROUTE – сообщает ядру, что не надо выполнять обычный алгоритм маршрутизации. Как правило, используется программами маршрутизации или для диагностических целей.

Примеры:

char buf[]= “Hello world”;

int sz = send(s, buf, sizeof(buf), 0);

При работе с протоколом TCP вам ничего больше не понадобиться. Но при работе с UDP нужны еще системные вызовы recvfrom и sendto. Они очень похожи на recv и send, но позволяют при отправке датаграммы задать адрес назначения, а при приеме – получить адрес источника.

Системные вызовы recvfrom и sendto имеют следующие прототипы:

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

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

Возвращаемое значение:

число принятых или переданных байтов в случае успеха или -1 в случае ошибки Первые четыре аргумента – s, buf, len и flags – такие же, как и в вызовах recv и send. Аргумент from в вызове recvfrom указывает на структуру, в которую ядро помещает адрес источника пришедшей датаграммы. Длина этого адреса хранится в целом числе, на которое указывает аргумент fromlen. Обратите внимание, что fromlen – это указатель на целое.

Аналогично аргумент to в вызове sendto указывает на адрес структуры, содержащей адреса назначения датаграммы, а аргумент tolen – длина этого адреса.

Заметьте, что to - это целое, а не указатель.

Для закрытия ранее созданного сокета в системе Windows используется системный вызов closesocket, а в UNIX – системный вызов close.

Прототипы системных вызовов close и closesocket имеют следующий вид:

int close(SOCKET s);

int closesocket(SOCKET s);

Аргумент s задает дескриптор ранее созданного сокета.

### 1.1.3 Создание клиента

Программа клиента делается аналогично до момента создания сокетов.

Cоздайте сокет так, как описано выше, но не пользуйтесь командой bind:

SOCKADDR\_IN anAddr;

anAddr.sin\_family = AF\_INET;

anAddr.sin\_port = htons(80);

anAddr.sin\_addr.S\_un.S\_addr = inet\_addr("127.0.0.1");

Заполнение структуры производится почти также но во время инициализации переменной anAddr необходимо указать IP-адрес сервера ( пример 127.0.0.1 ) .

Для обращения программы-клиента к серверу с запросом на установление логической соединения используется системный вызов connect, имеющий следующий прототип:

int connect(SOCKET s, const struct sockaddr \*peer, int peer\_len));

Аргумент s задает дескриптор сокета, через который программа обращается к серверу с запросом на соединение.

Аргумент addr должен указывать на структуру данных, содержащую адрес, приписанный сокету программы-сервера, к которой делается запрос на соединение. Для сетей TCP/IP такой структурой является sockaddr\_in.

Пример:

err = сonnect( s, (LPSOCKADDR)&anAddr, sizeof(sin) );

// или int err = bind( s, (const struct sockaddr \*)&anAddr, sizeof(anAddr) );

Для формирования значений полей структуры sockaddr\_in удобно использовать функцию gethostbyname.

Аргумент addrlen задает размер (в байтах) структуры данных, указываемой аргументом addr.

Для того, чтобы запрос на соединение был успешным, необходимо, по крайней мере, чтобы программа-сервер выполнила к этому моменту системный вызов listen для сокета с указанным адресом.

При успешном выполнении запроса системный вызов connect возвращает 0, в противном случае - "-1" (устанавливая код причины неуспеха в глобальной переменной errno).

Примечание. В режиме взаимодействия без установления соединения необходимости в выполнении системного вызова connect нет. Однако, его выполнение в таком режиме не является ошибкой - просто меняется смысл выполняемых при этом действий: устанавливается адрес "по умолчанию" для всех последующих посылок дейтаграмм.

### 1.1.4 Пример использования соккетов на python

Потоковый TCP соккет.

Сервер

import socket

sock = socket.socket()

sock.bind(('', 9090))

sock.listen(1)

conn, addr = sock.accept()

print 'connected:', addr

while True:

data = conn.recv(1024)

if not data:

break

conn.send(data.upper())

conn.close()

Клиент

import socket

sock = socket.socket()

sock.connect(('localhost', 9090))

sock.send('hello, world!')

data = sock.recv(1024)

sock.close()

print data

Пример дейтаграммного соккета.

from socket import \*

host = 'localhost'

port = 777

addr = (host,port)

#socket - функция создания сокета

#первый параметр socket\_family может быть AF\_INET или AF\_UNIX

#второй параметр socket\_type может быть SOCK\_STREAM(для TCP) или SOCK\_DGRAM(для UDP)

udp\_socket = socket(AF\_INET, SOCK\_DGRAM)

#bind - связывает адрес и порт с сокетом

udp\_socket.bind(addr)

#Бесконечный цикл работы программы

while True:

#Если мы захотели выйти из программы

question = input('Do you want to quit? y\\n: ')

if question == 'y': break

print('wait data...')

#recvfrom - получает UDP сообщения

conn, addr = udp\_socket.recvfrom(1024)

print('client addr: ', addr)

#sendto - передача сообщения UDP

udp\_socket.sendto(b'message received by the server', addr)

udp\_socket.close()

from socket import \*

host = 'localhost'

port = 777

addr = (host,port)

#socket - функция создания сокета

#первый параметр socket\_family может быть AF\_INET или AF\_UNIX

#второй параметр socket\_type может быть SOCK\_STREAM(для TCP) или SOCK\_DGRAM(для UDP)

udp\_socket = socket(AF\_INET, SOCK\_DGRAM)

#bind - связывает адрес и порт с сокетом

udp\_socket.bind(addr)

#Бесконечный цикл работы программы

while True:

#Если мы захотели выйти из программы

question = input('Do you want to quit? y\\n: ')

if question == 'y': break

print('wait data...')

#recvfrom - получает UDP сообщения

conn, addr = udp\_socket.recvfrom(1024)

print('client addr: ', addr)

#sendto - передача сообщения UDP

udp\_socket.sendto(b'message received by the server', addr)

udp\_socket.close()

### Задание на лабораторную работу:

1. По указаниям из методички реализовать сначала сервер, затем клиент. На языке Си и на Python, обеспечить обмен сообщениями между клиентом и сервером.
2. Реализовать индивидуальное задание в соответствии со своим вариантом. Рассмотреть задание, определить, как можно реализовать приложения, используя серверные и клиентские сокеты, выработать концепцию, учесть, что приложения независимы друг от друга и могут запускаться на различных узлах. Возможно, добавление дополнительного приложения, если даже этого напрямую не указано в варианте. Можно делать на Python. Можно объединить разные приложения в одном программном проекте запуская их копии.
3. Самостоятельно рассмотреть SCTP соккеты.
4. Запустить три приложения, которые, будут передавать друг другу сообщение по кругу, первое сообщение формируется первым запущенным приложением случайным образом или вводится с клавиатуры.
5. Запустить три приложения, которые будут передавать друг другу сообщение от 1 ко 2, от 2 к 3, от 3 ко 2 и от 2 к 1 и так далее.
6. Два приложения рассылают другому приложению в случайный момент времени числа, каждая соответственно полученная пара чисел складывается. Предусмотреть буфер.
7. Приложение передает другому приложению строку, та его искажает и отправляет обратно, при получении указывается процент повреждения сообщения. И так далее, приложения обмениваются по очереди сообщениями.
8. Приложение отправляет двум приложениям два случайных числа, те два приложения определяют у кого большее число и отправляют это число четвертому приложению.
9. Есть три приложения, первое отправляет случайное число и его хэшсумму, второе получает хэш от хэша конактенации с новым случайным числом, и отправляет этот хэш, случайное число и новое случайное число на третье приложение. Третье приложение проверяет правильность полученных чисел.
10. Есть приложения, одно может только суммировать, другое вычитать, третье умножать, четвертое делить, пятое вычислять квадратный корень, реализовать получение корней квадратного уравнения.
11. Есть приложение, которое может вычислять синус, приложение которое может вычислять косинус, и приложение которое может делить два числа, реализовать приложение вычисляющее тангенс используя другие три приложения.
12. Есть приложение генерирующую отправку строки, которая проходит через два приложения, которые удаляют заданный этими приложениями символ, итоговая строка отправляется обратно первому приложению.
13. Два приложения передают друг другу по очереди случайное число через случайный промежуток времени.