Лабораторная работа №8. Использование механизма сокетов для разработки распределенных приложений.

Краткое описание функций интерфейса Windows Sockets – WinSock 2

Основная цель разработки спецификации WinSock 2 – создание независимого от протоколов транспортного интерфейса.

Перед использованием сокетов необходимо загрузить корректную версию библиотеки WinSock с помощью функции инициализации

```
#include <winsock2.h>
// Функция загрузки нужной версии библиотеки WinSock
//
// FUNCTION: int WSAStartup(WORD, LPWSADATA)
//
// PARAMETERS:  [in]  wVersion       - номер версии
//              [out] lpWSAData      - структура с информацией о версии
//
// RETURN VALUE: 0                   - в случае успеха
//               WSASYSNOTREADY      - отсутствие готовности для работы
//                                     в сетевой среде
//               WSASVERNOTSUPPORTED - отсутствие поддержки заданной версии
//                                     библиотеки сокетов
//               WSAEINPROGRESS      - незавершенная блокирующая операция
//                                     библиотеки сокетов версии 1.1
//               WSAEPROCLIM         - достигнуто ограничение на количество заданий
//                                     в заданной версии библиотеки сокетов
//               WSAEFAULT           - переданы неверный параметр lpWSAData
//
// COMMENTS:  Эта функция напрямую возвращает состояние ошибки
//            Она должна быть первой в списке вызовов функций WinSock API
//
int WSAStartup(WORD wVersion, LPWSADATA lpWSAData);
```

На платформах Win32 обычно используется версия 2.2. Чтобы загрузить библиотеку этой версии нужно вызвать *WSAStartup* следующим образом.

```
#include <winsock2.h>
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "ws2_32.lib")

void main(void)
{

    WORD wRequestedLibraryVersion = 0x0202;
    WSADATA wsaData;
    if (0 != WSAStartup(wRequestedLibraryVersion, &wsaData))
    {
        printf("Невозможно загрузить Winsock 2.2\n");
        getchar();
        return EXIT_FAILURE;   // после нажатия клавиши программа завершается
```

```
    }
    printf("Библиотека инициализирована.\n");
    getchar();

    // здесь должен размещается оставшийся код, использующие сокеты
    ...

    // выгрузка библиотеки сокетов
    WSACleanup();
}
```

Для завершения работы с WinSock необходимо вызвать *WSACleanup*, которая освобождает занятые ресурсов.

```
#include <winsock2.h>
// Функция выгрузки библиотеки WinSock
//
// FUNCTION: int WSACleanup(void)
//
// PARAMETERS:  нет
//
// RETURN VALUE: 0             - в случае успеха
//               SOCKET_ERROR  - в случае ошибки
//
// COMMENTS:  Эта функция напрямую не возвращает состояние ошибки
//            Для получения кода ошибки нужно вызвать WSAGetLastError,
//            которая может возвратить следующие коды
//               WSANOTINITIALISED   - еще не вызывалась функция WSAStartup
//               WSAENETDOWN         - сбой сети
//               WSAEINPROGRESS      - незавершенная блокирующая операция
//
int WSACleanup(void);
```

Протокол TCP широко используется в Internet, он поддерживается большинством ОС и применяется в LAN и WAN. Он реализует связь с установлением логического соединения между двумя узлами, гарантирует надежную доставку данных между ними. То есть когда программы связываются по TCP, осуществляется виртуальное соединение отправителя с получателем, после чего возможен двунаправленный обмен данными. Это также напоминает известный подход «клиент-сервер».

Сервер — это процесс, который ожидает подключения клиентов для обслуживания их запросов. Сервер должен прослушивать соединение на стандартном имени, таковым в стеке TCP/IP является адрес интерфейса и номер порта. Первый шаг установления соединения — привязка сокета протокола к его имени с помощью функции *bind*. Второй — перевод сокета в режим прослушивания функцией *listen*. Третий — принятие клиента функцией *WSAAccept*.

При использовании протокола IPv4 устройствам назначается 32-разрядный IP-адрес. Для взаимодействия с сервером по TCP клиент должен указать IP-адрес сервера и номер порта. Чтобы прослушивать входящие запросы клиента, сервер

тоже должен указать IP-адрес и номер порта. В Winsock IP-адрес и порт указываются в структуре SOCKADDR_IN:

```
#include <winsock2.h>
struct sockaddr_in
{
    short sin_family;         // для IP должен использоваться AF_INET
    u_short sin_port;         // любой свободный порт из диапазона (1024, 65535)
    struct  in_addr sin_addr; // IP-адрес в 4-байтном виде
    char sin_zero[8];         // заполнитель нулями
};
```

Вспомогательная функция *inet_addr* преобразует IP-адрес из точечной нотации в беззнаковое длинное целое число, в котором байты следуют в соответствии с сетевым порядком следования:

```
#include <winsock2.h>
// Функция преобразования IPv4-адреса из точечной нотации в число
//
// FUNCTION: unsigned long inet_addr(const char*)
//
// PARAMETERS:  [in]    cCodePage  - строка с адресом в десятично-точечной нотации
//
// RETURN VALUE: число с IP-адресом - в случае успеха
//               INADDR_NONE        - в случае ошибки в адресе
//               0                  - если передана пустая строка
//
// COMMENTS:  Все возвращаемые адреса имеют сетевой порядок
//            Строка может содержать числа в 10-, 8-, 16-чных и смешанном форматах
//
unsigned long inet_addr(const char* cCodePage);
```

Существуют два специальных адреса *INADDR_ANY*, который позволяет серверу слушать клиента через любой сетевой интерфейс на несущем компьютере, и *INADDR_BROADCAST*, позволяющие широковещательно рассылать дейтаграммы по сети.

Несколько замечаний относительно порядка следования байтов. В процессорах x86 многобайтные числа представляют от менее значимого байта к более значимому. В частности, IP-адрес и номер порта хранятся в памяти именно так. Это так называемый *системный порядок*. При передаче их по сети стандарты требуют, чтобы многобайтные значения представлялись от старшего байта к младшему, что называется *сетевым порядком*. Две следующие функции позволяют преобразовывать, соответственно, 4- и 2-байтовые числа из системного порядка в сетевой.

```
#include <winsock2.h>
// Функция преобразования длинного целого числа из системного порядка байтов
// в сетевой
//
// FUNCTION: u_long htonl(u_long)
//
```

```
// PARAMETERS:  [in]    uHostLong  - 32-битное число в системном порядке байтов
//
// RETURN VALUE: 32-битное число в сетевом порядке – во всех случаях
//
// COMMENTS:  нет
//
u_long htonl(u_long uHostLong);

// Функция преобразования короткого целого числа из с порядка байтов
// в сетевой
//
// FUNCTION: u_short htons(u_short)
//
// PARAMETERS:  [in]    uHostShort - 16-битное число в системном порядке байтов
//
// RETURN VALUE: 16-битное число в сетевом порядке – во всех случаях
//
// COMMENTS:  нет
//
u_short htons(u_short uHostShort);
```

Следующие две функции решают обратную задачу, т.е. переставляют байты из сетевого порядка в системный:

```
#include <winsock2.h>
// Функция преобразования длинного целого числа из сетевого порядка байтов
// в системный
//
// FUNCTION: u_long ntohl(u_long)
//
// PARAMETERS:  [in]    uNetworkLong  - 32-битное число в сетевом порядке байтов
//
// RETURN VALUE: 32-битное число в системном порядке – во всех случаях
//
// COMMENTS:  нет
//
u_long ntohl(u_long uNetworkLong);

// Функция преобразования короткого целого числа из сетевого порядка байтов
// в системный
//
// FUNCTION: u_long ntohs(u_short)
//
// PARAMETERS:  [in]    uNetworkShort  - 16-битное число в сетевом порядке байтов
//
// RETURN VALUE: 16-битное число в системном порядке – во всех случаях
//
// COMMENTS:  нет
//
u_short ntohs(u_short uNetworkShort);
```

Следующий фрагмент демонстрирует, как создается структура с адресом и номером порта при помощи описанных функций.

```
#include <winsock2.h>
```

```
SOCKADDR_IN     NetworkAddress;

INT iPortNumber = 5202;

NetworkAddress.sin_family = AF_INET;
NetworkAddress.sin_addr.s_addr = inet_addr("192.168.135.24");
NetworkAddress.sin_port = htons(iPortNumber);
```

Сокет создается функцией

```
#include <winsock2.h>
// Функция создания сокета
//
// FUNCTION: SOCKET WSASocket(int, int, int, LPWSAPROTOCOL_INFO, GROUP, DWORD)
//
// PARAMETERS:  [in]  iAddressFamily - семейство адресов протокола
//              [in]  iSocketType    - тип сокета для данного протокола
//              [in]  iProtocol      - транспорт
//              [in]  lpProtocolInfo - информация о протоколе
//              [in]  group          - зарезервировано для будущего использования
//              [in]  dwFlags        - дополнительные возможности сокета
//
// RETURN VALUE: хэндл созданного сокета – в случае успеха
//               INVALID_SOCKET          - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//            При использовании протокола TCP параметр iAddressFamily должен
//                принимать значение AF_INET
//            При использовании протокола TCP параметр iSocketType должен
//                принимать значение SOCK_STREAM
//            При использовании протокола TCP параметр iProtocol должен
//                принимать значение IPPROTO_TCP или IPPROTO_IP
//
SOCKET WSASocket(int iAddressFamily, int iSocketType, int iProtocol
                 , LPWSAPROTOCOL_INFO lpProtocolInfo
                 , GROUP group, DWORD dwFlags
                );
```

Создать сокет при помощи протокола TCP можно следующим образом.

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
SOCKET ServerSocket;      // Серверный сокет
ServerSocket = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);

// Проверяем, удалось ли создать сокет. Завершаем выполнение в случае ошибки
if (INVALID_SOCKET == ServerSocket)
{
    printf("Ошибка создания сокета с кодом = %d\n", WSAGetLastError());
    WSACleanup();
    getchar();
    return EXIT_FAILURE;
}
...
```

После создания сокета конкретного протокола его надо связать со

стандартным адресом при помощи функции

```
#include <winsock2.h>
// Функция связывания существующего сокета со стандартным адресом
//
// FUNCTION: int bind(SOCKET, const struct sockaddr*, int)
//
// PARAMETERS:  [in]  unboundSocket  - сокет, ожидающий соединения клиентов
//              [in]  addressName    - буфер со структурой, содержащей адрес
//              [in]  nameLength     - длина буфера
//
// RETURN VALUE: 0               - в случае успеха
//               SOCKET_ERROR    - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int bind(SOCKET unboundSocket, const struct sockaddr* addressName, int nameLength);
```

При возникновении ошибки функция *bind* возвращает *SOCKET_ERROR*.

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
NetworkAddress.sin_family = AF_INET;
NetworkAddress.sin_addr.s_addr = INADDR_ANY;
NetworkAddress.sin_port = htons(iPortNumber);

int BindingResult = bind(ServerSocket
                         , (struct sockaddr *)&NetworkAddress
                         , sizeof(NetworkAddress)
                        );
if (SOCKET_ERROR == BindingResult)
{
    printf("Ошибка bind() с кодом = %d.\n", WSAGetLastError());
    getchar();
    closesocket(ServerSocket);
    WSACleanup();
    return EXIT_FAILURE;
}
...
```

На следующем шаге надо перевести сокет в режим прослушивания, ведь функция *bind* только связывает сокет с заданным адресом. Для перевода сокета в состояние ожидания используется следующая функция.

```
#include <winsock2.h>
// Функция перевода связанного сокета в режим прослушивания
//
// FUNCTION: int listen(SOCKET, int)
//
// PARAMETERS:  [in]  boundSocket - связанный через bind, но неприсоединенный сокет
//              [in]  queueLength - максимальная длина очереди соединений
//
// RETURN VALUE: 0               - в случае успеха
//               SOCKET_ERROR    - в случае ошибки
```

```
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//            Если значение параметра queueLength установлено в SOMAXCONN,
//                 то он означает длину очереди, максимально возможную
//                 для заданного протокола
int listen(SOCKET boundSocket, int queueLength);
```

Если значение параметра *queueLength* установлено в *SOMAXCONN*, то он означает длину очереди, максимально возможную для заданного протокола. Например,

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
int ListenResult = listen(ServerSocket, SOMAXCONN);
if (SOCKET_ERROR == ListenResult)
{
    printf("Ошибка listen() с кодом = %d.\n", WSAGetLastError());
    closesocket(ServerSocket);
    WSACleanup();
    getchar();
    return EXIT_FAILURE;
}
...
```

Теперь все готово к приему соединений клиентов, нужно только вызвать функцию:

```
#include <winsock2.h>
// Функция приема соединения клиентов
//
// FUNCTION: SOCKET WSAAccept(SOCKET, struct sockaddr*
//                            , LPINT, LPCONDITIONPROC, DWORD
//                           )
//
// PARAMETERS:  [in]     listeningSocket       - связанный сокет
//                                               в состоянии прослушивания
//              [out]    socketAddress         - адрес действительной
//                                               структуры SOCKADDR_IN
//              [in out] addressLength         - указатель на длину
//                                               структуры SOCKADDR_IN
//              [in]     lpConditionProcedure  - адрес процедуры
//                                               обратного вызова
//              [in]     dwCallbackData        - параметр функции обратного вызова
//
// RETURN VALUE: хэндл сокета запросившего соединение – в случае успеха
//               INVALID_SOCKET                       - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
SOCKET WSAAccept(SOCKET listeningSocket
                 , struct sockaddr* socketAddress, LPINT addressLength
                 , LPCONDITIONPROC lpConditionProcedure, DWORD dwCallbackData
                );
```

Вызов этой функции обслуживает первый находящийся в очереди запрос на соединение. По его завершении *socketAddress* будет содержать сведения об IP-адресе клиента, отправившего запрос, а *addressLength* будет содержать размер структуры. *NULL* – это типичное значение для параметров *lpConditionProcedure* и *dwCallbackData*. Если первый из этих параметров не *NULL*, то он интерпретируется функцией *WSAAccept* как пользовательская процедура обратного вызова, которая реализуется программно. В зависимости от ее результата функция *WSAAccept* принимает или отклоняет соединение. Параметр *dwCallbackData* не имеет смысла для самой функции *WSAAccept* и передаётся без изменений в процедуру обратного вызова.

Данная функция возвращает новый дескриптор сокета, соответствующий принятому клиентскому соединению.

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
SOCKADDR_IN ClientAddress = sizeof (ClientAddress);
SOCKET ClientSocket = WSAAccept(ServerSocket
                                , (struct sockaddr *)&ClientAddr
                                , &ClientAddrSize
                                , NULL
                                , NULL
                               );
if (INVALID_SOCKET == ClientSocket)
{
    printf("Ошибка WSAAccept() с кодом = %d.\n", WSAGetLastError());
    closesocket(ServerSocket);
    WSACleanup();
    getchar();
    return EXIT_FAILURE;
}
...
```

На стороне клиента так же необходимо установление соединения. Для этого требуется три шага: создать сокет функцией *WSASocket*, разрешить имя сервера, инициировать соединение функцией *WSAConnect*.

```
#include <winsock2.h>
// Функция установки соединения с сервером
//
// FUNCTION: int WSAConnect(SOCKET, const struct sockaddr*, int
//                          , LPWSABUF, LPWSABUF, LPQOS, LPQOS)
//
// PARAMETERS:  [in]     unconnectedSocket  - существующий сокет
//              [in]     socketAddress      - адрес сокета,
//                                            к которому идет подключение
//              [in]     addressLength      - длина структуры SOCKADDR_IN
//              [out]    lpCallerData       - указатель на буфер для отсылки
//                                            при подключении
//              [out]    lpCalleeData       - указатель на буфер для приема
//                                            при подключении
```

```
//              [in]     lpSQOS             - указатель на данные
//                                            о качестве обслуживания
//              [in]     lpGQOS             - указатель на групповые данные
//
// RETURN VALUE: 0                   – в случае успеха
//               SOCKET_ERROR        - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int WSAConnect(SOCKET unconnectedSocket
               , const struct sockaddr* socketAddress, int addressLength
               , LPWSABUF lpCallerData, LPWSABUF lpCalleeData
               , LPQOS lpSQOS, LPQOS lpGQOS
              );
```

Пример использования:

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
InetAddr.sin_family = AF_INET;
InetAddr.sin_addr.s_addr = inet_addr("192.68.194.147");
InetAddr.sin_port = htons(iPortNumber);
int ConnectResult = WSAConnect(ServerSocket
                               , (struct sockaddr *) &InetAddr
                               , sizeof (InetAddr)
                               , NULL
                               , NULL
                               , NULL
                               , NULL
                              );
if (SOCKET_ERROR == ConnectResult)
{
    printf("Ошибка WSAConnect() с кодом = %d.\n", WSAGetLastError());
    closesocket(ServerSocket);
    WSACleanup();
    getchar();
return EXIT_FAILURE;
}
...
```

В сетевом программировании главное – отправлять и принимать данные. Для пересылки с установкой логического соединения используют функцию *WSASend*, а для приема – *WSARecv*. Первая из них определена так:

```
#include <winsock2.h>
// Функция отправки сообщений
//
// FUNCTION: int WSASend(SOCKET, LPWSABUF, DWORD, LPDWORD, DWORD
//                       , LPWSAOVERLAPPED, LPWSAOVERLAPPED_COMPLETION_ROUTINE
//                       )
//
// PARAMETERS:  [in]     connectedSocket     - присоединенный сокет
//              [in]     lpBuffers           - буферы с данными для отправки
//              [in]     dwBufferCount       - количество буферов для отправки
```

```
//               [out]    lpSentBytes         - количество реально переданных байтов
//               [in]     dwFlags             - дополнительные флаги
//               [in]     lpOverlapped        - структура для перекрытого
//                                              ввода-вывода
//               [in]     lpCompletionRoutine - процедура для завершения
//                                              перекрытого ввода-вывода
//
// RETURN VALUE: 0                   - в случае успеха
//               SOCKET_ERROR        - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int WSASend(SOCKET connectedSocket, LPWSABUF lpBuffers, DWORD dwBufferCount
            , LPDWORD lpSentBytes, DWORD dwFlags
            , LPWSAOVERLAPPED lpOverlapped
            , LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
           );
```

Параметр *dwFlags* может принимать значения 0, *MSG_DONTROUTE*, *MSG_OOB*, или результат логического "ИЛИ" над любыми из этих параметров. При указании флага *MSG_DONTROUTE* пакеты не будут маршрутизироваться, обработка этого запроса остается на усмотрение базового протокола. Флаг *MSG_OOB* означает, что данные будут отправляться срочно (*out of band*).

При успешном выполнении *WSASend* вернет 0 (а количество реально переданных байт содержится в параметре *lpSentBytes*), иначе — ошибку *SOCKET_ERROR*.

Одна из частых ошибок *WSAECONNABORTED* происходит, когда виртуальное соединение разрывается. В этом случае сокет должен быть закрыт, так как он не может использоваться далее. То же самое надо сделать, если произошла ошибка *WSAECONNRESET*, что случается, например, при сбросе виртуального соединения, неожиданном завершении удаленного приложения или перезагрузке удаленного узла. Ошибка *WSAETIMEDOUT* происходит при обрыве соединения из-за отказа удаленной системы без предупреждения или сбоев сети.

Большинство протоколов с установлением соединения являются потоковыми, однако при чтении-записи данных через потоковый сокет нет гарантии, что будет прочитан или записан весь запрошенный объем данных. Допустим, надо отправить сообщение из буфера функцией *WSASend*.

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
char TestMessage[] = "Kukareku!!!";
DWORD  SendBytes;
WSABUF DataBuffers;
DataBuffers.len = strlen(TestMessage) + 1;
DataBuffers.buf = TestMessage;

int SendResult = WSASend(ServerSocket
```

```
                          , &DataBuffers
                          , 1
                          , &SendBytes
                          , 0
                          , NULL
                          , NULL
                         );
if (SOCKET_ERROR == SendResult)
{
    printf("Ошибка WSASend() с кодом = %d.\n", WSAGetLastError());
    getchar();
    shutdown(ServerSocket, SD_BOTH);
    closesocket(ServerSocket);
    WSACleanup();
    return EXIT_FAILURE;
}
...
```

Функция *WSARecv* – это основной инструмент приема данных по сокету и определена таким образом:

```
#include <winsock2.h>
// Функция приема сообщений
//
// FUNCTION: int WSARecv(SOCKET, LPWSABUF, DWORD, LPDWORD,  LPDWORD
//                       , LPWSAOVERLAPPED, LPWSAOVERLAPPED_COMPLETION_ROUTINE
//                       )
//
// PARAMETERS:  [in]     connectedSocket     - присоединенный сокет
//              [in out] lpBuffers           - буферы с данными для приема
//              [in]     dwBufferCount       - количество буферов для приема
//              [out]    lpReceivedBytes     - количество реально принятых байтов
//              [in out] lpFlags             - дополнительные флаги
//              [in]     lpOverlapped        - структура для перекрытого
//                                             ввода-вывода
//              [in]     lpCompletionRoutine - процедура для завершения
//                                             перекрытого ввода-вывода
//
// RETURN VALUE: 0                  - в случае успеха
//               SOCKET_ERROR       - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int WSARecv(SOCKET connectedSocket, LPWSABUF lpBuffers, DWORD dwBufferCount
            , LPDWORD lpReceivedBytes, LPDWORD lpFlags
            , LPWSAOVERLAPPED lpOverlapped
            , LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
           );
```

Параметр *lpFlags* может принимать значения 0, *MSG_PEEK*, *MSG_OOB*, или результат логического "ИЛИ" над любыми из этих параметров. При указании флага *MSG_PEEK* указывает, что доступные данные должны копироваться в принимающий буфер и при этом оставаться в системном буфере. Функция возвращает 0 в случае успешного приема данных. Пример использования этой функции

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
char buffer[MAX_BUFFER_SIZE]="\0";

WSABUF DataBuffers;
DataBuffers.len = sizeof (buffer);
DataBuffers.buf = buffer;
DWORD RecievedBytes = 0;
DWORD Flags = 0;
int RecvResult = WSARecv(ClientSocket
                         , &DataBuffers
                         , 1     // единственный буфер
                         , &RecievedBytes
                         , &Flags
                         , NULL
                         , NULL
                        );
if (SOCKET_ERROR == RecvResult)
{
    printf("Ошибка WSARecv() с ошибкой = %d.\n", WSAGetLastError());
    shutdown(ClientSocket, SD_BOTH);
    shutdown(ServerSocket, SD_BOTH);
    closesocket(ClientSocket);
    closesocket(ServerSocket);
    WSACleanup();
    getchar();
    return EXIT_FAILURE;
}
...
```

Полученное сообщение будет содержаться в *DataBuffers.buf*. По окончании работы с сокетом необходимо закрыть соединение и освободить все ресурсы, связанные с дескриптором сокета, с помощью функции *closesocket*. Рекомендуется перед ее вызовом завершить сеанс, уведомив получателя об этом с помощью функции *shutdown*.

```
#include <winsock2.h>
// Функция завершения логического соединения
//
// FUNCTION: int shutdown(SOCKET, int)
//
// PARAMETERS:  [in]     connectedSocket     - присоединенный сокет
//              [in]     shutdownAction      - как закрыть соединение
//
// RETURN VALUE: 0                   - в случае успеха
//               SOCKET_ERROR        - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//            Значение параметра shutdownAction, равное  SD_RECEIVE, приводит
//                     к закрытию приемников, SD_SEND – передатчиков,
//                     а SD_BOTH — закрывает обе стороны
//
int shutdown(SOCKET connectedSocket, int shutdownAction);
```

```
// Функция закрытия сокета
//
// FUNCTION: int closesocket(SOCKET)
//
// PARAMETERS:  [in]     closingSocket    - сокет, который нужно закрыть
//
// RETURN VALUE: 0                    – в случае успеха
//               SOCKET_ERROR         - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int closesocket(SOCKET closingSocket);
```

Связь без установления соединения выполняется при помощи пользовательских дейтаграмм протокола UDP. Он не гарантирует надежности, однако может осуществлять передачу данных нескольким адресатам и принимать их от нескольких источников. В частности, данные, отправляемые клиентом, передаются на сервер немедленно, независимо от готовности (или неготовности) сервера. При получении данных сервер не подтверждает их прием. Данные передаются порциями, называемыми дейтаграммами.

Процесс получения данных на сокете, не требующем соединения, относительно прост. Сокет создается функцией *WSASocket:*

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
...
SOCKET ServerSocket;      // Серверный сокет
ServerSocket = WSASocket(AF_INET, SOCK_DGRAM, IPPROTO_UDP, NULL, 0, 0);

// Проверяем, удалось ли создать сокет. Завершаем выполнение в случае ошибки
if (INVALID_SOCKET == ServerSocket)
{
    printf("Ошибка создания сокета с кодом = %d\n", WSAGetLastError());
    WSACleanup();
    getchar();
    return EXIT_FAILURE;
}
...
```

Затем осуществляется привязка сокета к интерфейсу, на котором будут приниматься данные, с помощью функции *bind* (аналогично протоколам, ориентированным на соединения). После этого нужно просто ожидать приема входящих данных. Соединения нет, а значит, сокет-приемник будет получать дейтаграммы от любой станции в сети. После создания сокета конкретного протокола его надо связать со стандартным адресом при помощи функции bind, которая при возникновении ошибки возвращает *SOCKET_ERROR*.

```
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>
```

```
...
NetworkAddress.sin_family = AF_INET;
NetworkAddress.sin_addr.s_addr = INADDR_ANY;
NetworkAddress.sin_port = htons(iPortNumber);

int BindingResult = bind(ServerSocket
                         , (struct sockaddr *) &NetworkAddress
                         , sizeof(NetworkAddress)
                        );
if (SOCKET_ERROR == BindingResult)
{
    printf("Ошибка bind() с кодом = %d.\n", WSAGetLastError());
    getchar();
    closesocket(ServerSocket);
    WSACleanup();
    return EXIT_FAILURE;
}
...
```

Для пересылки без установки логического соединения используют функцию *WSASendTo*, а для приема — *WSARecvFrom*. Первая из них определена так:

```
#include <winsock2.h>
// Функция отправки сообщений
//
// FUNCTION: int WSASendTo(SOCKET, LPWSABUF, DWORD, LPDWORD, DWORD
//                         , const struct sockaddr*, int,
//                         , LPWSAOVERLAPPED, LPWSAOVERLAPPED_COMPLETION_ROUTINE
//                        )
//
// PARAMETERS:  [in]     clientSocket            - сокет для отправки данных
//              [in]     lpBuffers               - буферы для отправляемых данных
//              [in]     dwBufferCount           - количество буферов для отправки
//              [out]    lpSentBytes             - число отправляемых байтов
//              [in]     dwFlags                 - дополнительные флаги
//              [in]     lpReceivingAddress      - адрес станции-приемника
//              [in]     receivingAddressLength  - длина адреса применика
//              [in]     lpOverlapped            - структура для перекрытого
//                                                 ввода-вывода
//              [in]     lpCompletionRoutine     - процедура для завершения
//                                                 перекрытого ввода-вывода
// RETURN VALUE: 0                  - в случае успеха
//               SOCKET_ERROR       - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int WSASendTo(SOCKET clientSocket, LPWSABUF lpBuffers, DWORD dwBufferCount
              , LPDWORD lpSentBytes, DWORD dwFlags
              , const struct sockaddr* lpReceivingAddress
              , int receivingAddressLength
              , LPWSAOVERLAPPED lpOverlapped
              , LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
             );
```

При успешном выполнении *WSASendTo* вернет 0, иначе – ошибку *SOCKET_ERROR*.

Функция приема аналогична отправке, она также возвращает 0 в случае успеха. Если сокет к этому времени был уже закрыт, то возвращаемым значением является 0. Если же возникла ошибка – *SOCKET_ERROR*.

```
// Функция приема сообщений
//
// FUNCTION: int WSARecvFrom(SOCKET, LPWSABUF, DWORD, LPDWORD, LPDWORD
//                       , LPWSAOVERLAPPED, LPWSAOVERLAPPED_COMPLETION_ROUTINE
//                       )
//
// PARAMETERS:  [in]     serverSocket        - сокет для приема данных
//              [in out] lpBuffers            - буферы с данными для приема
//              [in]     dwBufferCount        - количество буферов для приема
//              [out]    lpReceivedBytes      - количество принятых байтов
//              [in out] lpFlags              - дополнительные флаги
//              [out]    lpSendinngAddress    - адрес станции-отправителя
//              [out]    sendingAddressLength - длина адреса отправителя
//              [in]     lpOverlapped         - структура для перекрытого
//                                              ввода-вывода
//              [in]     lpCompletionRoutine  - процедура для завершения
//                                              перекрытого ввода-вывода
// RETURN VALUE: 0                  - в случае успеха
//               SOCKET_ERROR       - в случае ошибки
//
// COMMENTS:  Для доступа к коду возникшей ошибки нужно вызвать WSAGetLastError
//
int WSARecvFrom(SOCKET serverSocket, LPWSABUF lpBuffers, DWORD dwBufferCount
                , LPDWORD lpReceivedBytes, LPDWORD lpFlags
                , struct sockaddr* lpSendinngAddress
                , LPINT sendingAddressLength
                , LPWSAOVERLAPPED lpOverlapped
                , LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
               );
```

По окончании работы с сокетом необходимо закрыть соединение и освободить все ресурсы с помощью функции *closesocket*.

Достаточно легко можно организовать прием данных.

```
#include <windows.h>
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "ws2_32.lib")
...
SOCKADDR_IN  InetAddr;
INT iPortNumber = 5202;
InetAddr.sin_family = AF_INET;
InetAddr.sin_addr.s_addr = INADDR_ANY;
InetAddr.sin_port = htons(iPortNumber);

SOCKET ServerSocket;
ServerSocket = WSASocket(AF_INET, SOCK_DGRAM, IPPROTO_UDP, NULL, 0, 0);
...
SOCKADDR_IN  SenderAddr;
int SenderAddrSize = sizeof (SenderAddr);
```

```
printf("Слушаю и принимаю дейтаграммы.\n");

WSABUF DataBuf;
char buffer[256]="\0";
DataBuf.len = sizeof (buffer);
DataBuf.buf = buffer;
DWORD RecievedBytes;
DWORD dwFlags = 0;
int RecvResult = WSARecvFrom(ServerSocket
                             , &DataBuf
                             , 1
                             , &RecievedBytes
                             , &dwFlags
                             , (SOCKADDR *) &SenderAddr
                             , &SenderAddrSize
                             , NULL
                             , NULL
                            );
if (SOCKET_ERROR == RecvResult)
{
    printf("Ошибка WSARecvFrom() с кодом = %d.\n", WSAGetLastError());
    getchar();
}
else
{
    printf("От %s принято сообщение %s из %d байтов.\n"
            , inet_ntoa(SenderAddr.sin_addr)
            , DataBuf.buf
            , RecvBytes
           );
}
...
```

Не менее «сложной» будет выглядеть отправка данных.

```
#include <windows.h>
#include <winsock2.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "ws2_32.lib")
...
SOCKET ServerSocket;
ServerSocket = WSASocket(AF_INET, SOCK_DGRAM, IPPROTO_UDP, NULL, 0, 0);
    ...
char TestMessage[] = "Kukareku!!!";
DWORD SendBytes;
WSABUF DataBuf;
DataBuf.len = strlen(TestMessage) + 1;
DataBuf.buf = TestMessage;

SOCKADDR_IN receiver;
INT iPortNumber = 25202;

receiver.sin_family = AF_INET;
receiver.sin_addr.s_addr = inet_addr("192.5.4.3");
receiver.sin_port = htons(PortNum);
```

```
int SendResult = WSASendTo(ServerSocket
                           , &DataBuf
                           , 1
                           , &SendBytes
                           , 0
                           , (SOCKADDR*) &receiver
                           , sizeof (receiver)
                           , NULL
                           , NULL
                          );
if (SOCKET_ERROR == SendResult)
{
    printf("Ошибка WSASendTo() с кодом = %d.\n", WSAGetLastError());
    getchar();
    return EXIT_FAILURE;
}
printf("Послано %d байтов.\n", SendBytes);
...
```

Приложение А. Варианты заданий

Вариант 1. Клиент отсылает серверу два беззнаковых целых числа L и U, введенные пользователем, каждое из которых состоит ровно из 6 знаков. L – это нижняя граница диапазона, U – верхняя граница диапазона.

Сервер принимает значения границ диапазона, осуществляет поиск «счастливых билетов» из этого диапазона, когда а) сумма цифр на 1-3 позиции равна сумме цифр, стоящих на 4-6 позиции; б) сумма цифр на нечетных позициях равна сумме цифр на четных позициях; в) арифметический квадратный корень из суммы шести цифр номера билета есть целое число. На экран должны выводиться полученные значения, а также количество найденных «счастливых чисел».

Вариант 2. Клиент отсылает серверу введенный пользователем номер k обобщенного числа Фибоначчи (ОЧФ) и его порядок p, причем $p,\ k > 0$. Формула для вычисления числа ОЧФ заданного порядка p с заданным номером k:

$$F_k^p = \begin{cases} \sum_{i=k-1-p}^{k-1} F_i^p, \ если\, k > p+1 \\ 1, \ если\, 1 \le k \le p+1 \end{cases}$$

Сервер принимает номер и порядок числа, вычисляет по ним ОЧФ с номером k и порядком p и выводит его на экран.

Вариант 3. Клиент отсылает серверу введенную пользователем строку, хранящую целое число со знаком.

Сервер принимает строку, хранящую целое число со знаком, и выводит на экран эквивалент этого числа прописью. Например, ввод «-1211» должен приводить к выводу «минус одна тысяча двести одиннадцать», а ввод «100000101»

приводит к выводу «сто миллионов сто один». Ограничения накладываются только представлениями максимально и минимально возможного знакового целого числа используемого языка программирования.

Вариант 4. Клиент отсылает серверу введенную пользователем строку, хранящую вещественное число.

Сервер принимает строку, хранящую вещественное число в обычном или экспоненциальном форматах, и выводит на экран эквивалент этого числа прописью. Например, ввод «-12.11» должен приводить к выводу «минус двенадцать целых одиннадцать сотых», а ввод «1,2E+2» приводит к выводу «120 целых». Ограничения: 1) для обычного формата после «точки» не более двух знаков; 2) для экспоненциального формата после «точки» не более шести знаков; 3) до точки не более 15 знаков для обычного формата; 4) знак «точки» - это «.»(точка) или «,» (запятая).

Вариант 5. Клиент отсылает серверу две строки, введенные пользователем.

Сервер принимает две строки. Если обе строки хранят целые числа со знаком, то на экран выводится сумма чисел. Если были переданы вещественные числа в обычном или экспоненциальном форматах, то на экран выводится остаток от деления первого числа на второе. Во всех остальных случаях выводится результат конкатенации двух полученных строк.

Вариант 6. Клиент отсылает серверу элементы двух квадратных матриц одинакового размера. Его, а также сами элементы матриц должен вводить пользователь.

Сервер принимает две квадратные матрицы, а затем выводит на экран сумму матриц, а также значения определителей и их произведение.

Вариант 7. Клиент отсылает серверу, два введенных пользователем целых числа *A* и *B*, где *A* – сдвигаемое число, *B* — величина арифметического, логического или циклического сдвига.

Сервер принимает два числа, после этого осуществляется арифметический, логический и циклический сдвиг числа *A* на величину *B* (отрицательное значение — сдвиг вправо, неотрицательное — влево), и оба результата выводятся на экран.

Вариант 8. Клиент принимает от пользователя некое слово, после чего проверяет его на корректность (отсутствие знаков препинания, цифр, смешения латинских и кириллических символов) и отсылает его серверу.

Сервер принимает слово, проверяет, является ли введенное слово палиндромом, и информирует пользователя о результате проверки.

Вариант 9. Клиент отсылает серверу элементы одномерного строкового массива, введенные пользователем, а также один или несколько символов, означающих способ преобразования массива (например, символ *A* – упорядочение по возрастанию, *D* – упорядочение по убыванию, *U* – разупорядочение).

Сервер принимает массив и способ его преобразования, выполняет его и выводит результат на экран. Для упорядочения воспользоваться любым из так называемых «улучшенных алгоритмов» сортировки массивов (при этом исключено использование следующих алгоритмов: *пузырьковая* сортировка, *шейкерная* сортировка, сортировка *прямым выбором* и сортировка *прямыми включениями/вставками*).

Вариант 10. Клиент отсылает серверу элементы введенной пользователем квадратной матрицы.

Сервер принимает матрицу , выводит ее на экран, вычисляет обратную ей матрицу, которую затем выводит на экран. Обеспечить проверку возможности вычисления обратной матрицы как на стороне клиента, так и на стороне сервера.

Вариант 11. Клиент отсылает серверу элементы введенной пользователем квадратной матрицы, а также два введенных им числа – номер столбца *N* и номер строки *R*.

Сервер принимает матрицу и номер столбца *N* и номер строки *R*, выводит ее на экран, затем «вычеркивает» из матрицы столбец с номером *N* и строку с номером *R*, выводит измененную матрицу на экран, вычисляет и выводит на экран значение ее определителя.

Вариант 12. Клиент принимает от пользователя две даты – строки вида ЦЦ.ЦЦ.ЦЦЦЦ, где Ц – это любая цифра из диапазона [0-9] и отсылает серверу.

Сервер принимает обе даты, вычисляет полное количество дней, прошедших между ними, и выводит его на экран. И клиент, и сервер должны принимать во внимание некорректные даты, например, 37.06.2006 или 01.18.2006, а также такие особенности Григорианского календаря, как переход на него в России в 1918 (отсутствие дней с 1 по 13 февраля в этот год).

Вариант 13. Клиент принимает от пользователя два значения времени суток – строки вида ЦЦ.ЦЦ.ЦЦ, где Ц – это любая цифра из диапазона [0-9] и отсылает их серверу.

Сервер принимает обе строки, вычисляет полное количество секунд, прошедших между двумя значениями времени двумя способами, и выводит их на экран. И сервер, и клиент должны принимать во внимание некорректные значения, например, 28.00.06, или 01.99.20, или 08.08.65.

Вариант 14. Клиент отсылает серверу две строки, введенные пользователем.

Сервер принимает две строки, осуществляет поиск вхождения второй строки в первую любым известным методом, кроме прямого (линейного, грубой силы), и выводит на экран значение индекса элемента первой строки, с которого началось совпадение, или строку «*Нет вхождений*» в противном случае.

Вариант 15. Клиент отсылает серверу две строки, введенные пользователем.

Сервер принимает две строки, осуществляет поиск количества вхождений второй строки в первую любым известным методом, кроме прямого (линейного) поиска, и выводит на экран полученное значение.

Вариант 16. Клиент принимает от пользователя дату – строку вида ЦЦ.ЦЦ.ЦЦЦЦ, где Ц – это любая цифра из диапазона [0-9] и отсылает серверу.

Сервер принимает дату и выводит на экран число, месяц и год прописью (например, ввод «29.02.2012» приводит к выводу «Двадцать девятое февраля две тысячи двенадцатого года»). И сервер, и клиент должны принимать во внимание некорректные даты, например, 37.06.2013 или 01.18.2016, а также такие особенности Григорианского календаря, как переход на него в России в 1918 (отсутствие дней с 1 по 13 февраля в этот год).

Вариант 17. Клиент принимает от пользователя значение времени суток – строку вида ЦЦ.ЦЦ.ЦЦ,Ц или ЦЦ.ЦЦ.ЦЦ,ЦЦ, где Ц – это любая цифра из диапазона [0-9] и отсылает серверу.

Сервер принимает значение времени суток и выводит на экран часы, минуты, секунды и их долей прописью (например, ввод «12.01.20,15» приводит к выводу «двенадцать часов одна минута двадцать и пятнадцать сотых секунды», для «00.01.12,3» - «ноль часов одна минута двенадцать и три десятых секунды»). И сервер, и клиент должны принимать во внимание некорректные значения, например, 28.00.06,14; 01.99.20,15; 08.08.65,16.

Вариант 18. Клиент принимает от пользователя беззнаковое десятичное целое число N – основание системы счисления ($1 < N < 17$ или $N = 1$) и последовательность цифр в соответствии с заданной системой счисления, отсылает серверу.

Сервер принимает основание системы счисления и число в этой системе, выводит его на экран, переводит его в десятичную систему, выводит на экран, дополняет его (инвертирует) до максимальной цифры в заданной системе счисления, выводит на экран значение инвертированной последовательности, переводит ее в число в десятичной системе и выводит полученное число на экран.

Для случая $N = 1$, когда используется унарная система счисления, результатом является только перевод в десятичную систему и его вывод на экран.

Вариант 19. Клиент принимает от пользователя беззнаковое десятичное целое число N – основание системы счисления ($1 < N < 17$ или $N = 1$) и последовательность цифр в соответствии с заданной системой счисления, отсылает серверу.

Сервер принимает основание системы счисления и число в этой системе, выводит его на экран, переводит его в десятичную систему, выводит на экран, осуществляет его реверс (меняет порядок следования знаков на обратный), выводит на экран значение измененной последовательности, переводит ее в число в десятичной системе и выводит его на экран. Для случая $N = 1$, когда используется унарная система счисления, результатом является только перевод в десятичную систему и его вывод на экран.

Вариант 20. Клиент отсылает серверу три строки, введенных пользователем (первая и третья строки – это правильные рациональные или десятичные дроби вида «1/3» или «0,5», вторая строка – это знак арифметической операции вида «+», «-», «*», «/» либо операции сравнения «<», «>», «=», «!=», «>=», «<=»).

Сервер принимает три строки, выполняет требуемую операцию над полученными операндами, и выводит результат на экран. Обеспечить также сокращение дроби при необходимости. Если оба операнда арифметической операции являются рациональными дробями, результатом тоже должна быть рациональная дробь. Для операций сравнения достаточно результата «*Истина*» или «*Ложь*».

Вариант 21. Клиент принимает от пользователя элементы целочисленного одномерного массива, а также значение элемента для поиска, и отсылает серверу.

Сервер принимает массив и значение элемента для поиска, осуществляет его любым известным методом, кроме прямого (линейного) поиска, и выводит результат на экран.

Вариант 22. Клиент принимает от пользователя две строки символов и отсылает серверу.

Сервер принимает обе строки, осуществляет в первой строке замену латинских букв на их аналоги из кириллицы, а во второй строке заменяет символы кириллицы латинскими буквами. Затем результирующие строки выводятся на экран. Учесть тот факт, что некоторые латинские литеры могут не иметь однобуквенных кириллических эквивалентов, а некоторые символы кириллицы

вообще не представимы латинскими буквами. Возможно использование транслитерации.

Вариант 23. Клиент принимает от пользователя последовательность символьных строк и отсылает серверу.

Сервер принимает последовательность символьных строк, осуществляет смену регистра всех букв английского и русского алфавитов, а также замену символов табуляции четырьмя пробелами, удаляет лидирующие пробелы, выполняя эти операции везде, за исключением подстрок, заключенных в кавычки или апострофы, и выводит результат на экран.

Вариант 24. Клиент принимает от пользователя два целых числа и отсылает серверу.

Сервер принимает два числа. Если первое число является степенью второго числа, то на экран выводится показатель степени, и сообщение «___ не является степенью числа ___» в противном случае (на месте прочерков выводятся конкретные числа). Например, при вводе «81 -3» должно выводиться «4», а ввод «81 4» – «81 не является степенью числа 4». Учесть областей определения и значений степенной функции целочисленного переменного.

Вариант 25. Клиент принимает от пользователя коэффициенты квадратного уравнения ($ax^2+bx+c=0$), кубического уравнения ($ax^3+bx^2+cx+d=0$) или биквадратного уравнения ($ax^4+bx^3+cx^1+dx+e=0$), каждый из которых не равен нулю или единице, и отсылает их серверу.

Сервер принимает коэффициенты квадратного, кубического или биквадратного уравнения и выводит на экран выводит на экран разложение уравнения на множители. Ограничение: для кубического и квадратного уравнения исключить комплексные корни.

Вариант 26. Клиент принимает принимает от пользователя две или три строки. При вводе трех строк первая и третья – это комплексные числа вида «**±**1.2**.±***i*5.4» (моделируется программно двумя вещественными и/или целыми числами), причем первый знак необязателен, вторая строка – это знак операции вида «+», «-», «*», «/». При вводе двух строк первая — комплексное число, вторая — операция, обозначенная литерами *S* (комплексно-сопряженное число), *R* (вещественная часть), *I* (мнимая часть). Затем строки отсылаются серверу.

Сервер принимает комплексные числа и знак операции, выполняет требуемую операцию с полученными операндами, и выводит результат на экран.

Вариант 27. Клиент принимает от пользователя целочисленную квадратную матрицу и отсылает ее серверу.

Сервер принимает целочисленную квадратную матрицу, после этого заменяет строку с минимальным элементом нулями, а затем удаляет (путем перемещения соответствующих элементов матрицы) столбец с максимальным элементом. На экран выводится результирующая матрица, номера модифицированной строки и удаленного столбца.

Вариант 28. Клиент принимает от пользователя целочисленную матрицу и отправляет ее серверу.

Сервер принимает матрицу, упорядочивает ее строки в порядке убывания суммы модулей их элементов, а затем выводит на экран результирующую матрицу и значения сумм модулей элементов каждой строки.

Вариант 29. Клиент принимает от пользователя строку, содержащую дату в формате ДД.ММ.ГГГГ и отсылает ее серверу

Сервер принимает дату и выводит на экран название дня недели, соответствующего введенной дате (для 03.11.2013 — «воскресенье»). И сервер, и клиент должны учитывать ввод некорректных значений (например, 36.13.2006 или 04.02.1918).

Вариант 30. Клиент принимает от пользователя строку, содержащую дату в формате ДД.ММ.ГГГГ и отсылает ее серверу.

Сервер принимает дату и выводит на экран 6-недельный календарь (колонки — дни недели), содержащий введенную дату. Эта дата должна быть особым образом выделена среди окружающих ее дат и не должна располагаться на заголовочной, первой и последней строке календаря при горизонтальном расположении дней. И сервер, и клиент должны учитывать ввод некорректных значений.

Вариант 31. Клиент принимает от пользователя три строки: 1) дата в формате ДД.ММ.ГГ; 2) местное время (Красноярск, GMT +7) в формате ЧЧ.ММ; 3) Часовой пояс, в котором необходимо узнать время и дату в формате *N*, где *N* – число от -12 до 13. Затем три строки отсылаются серверу.

Сервер принимает строки и выводит на экран дату и время в выбранном часовом поясе. Необходимо учесть переход в следующие сутки (например, когда в Красноярске 01.03.2010, 08.15 в Мехико 28.02.2010, 19.15), а также ввод некорректных значений (например, 36.13.2006, 25.15).

Вариант 32. Клиент принимает от пользователя две строки и отсылает их серверу.

Сервер принимает две строки и выводит минимальное количество стираний, замен и добавлений символов, преобразующих одну строку в другую.