Лабораторная работа № 8
Использование методов класса CSocket

8.1 Цель работы:

Целью восьмой лабораторной работы является ознакомление студентов с классом CSocket и его базовыми методами.

8.2 Задание на лабораторную работу:

В лабораторной работе необходимо разработать сервер, использующий методы класса CSocket. Сервер должен отправлять файл по запросу клиента.

8.3 Методические указания:

8.3.1 Класс CSocket

При работе с сетью можно использовать возможности, которые предоставляет среда разработки Visual C++. Классы упрощают программирование и скрывают некоторые особенности реализации протоколов и сети.

Для работы с сетью в MFC (Microsoft Foundation Classes) есть очень удобный класс — CSocket (Class Socket). В качестве предка у него выступает CAsyncSocket. Объект класса CAsyncSocket работает с сетью асинхронно, т.е. в неблокирующем режиме.

Класс CSocket как потомок класса CAsyncSocket получает все его возможности, свойства и методы. Он позволяет выполнять как синхронную, так и асинхронную передачу данных с использованием многих коммуникационных протоколов в том числе TCP, UDP, IP, ICMP и т.д.

8.3.2 Некоторые методы класса CSocket

Все методы определены в библиотеке **afxsock.h**.

1. **Create** - создает сокет и инициализирует его. Этот метод можно использовать вместо последовательности функций **socket** и **bind**.

Синтаксис:

```
BOOL Create(
        UINT nSocketPort,
        int nSocketType,
        LPCTSTR lpszSocketAddress
                );
```

Параметры:

- **nSocketPort** - порт который вы собираетесь использовать для работы или 0 для того чтобы Windows Sockets выбрала порт.
- **nSocketType** - SOCK_STREAM или SOCK_DGRAM. По умолчанию SOCK_STREAM.
- **lpszSocketAddress** - указатель на строку, содержащую сетевой адрес подключенного сокета, такой как "ftp.microsoft.com" или "128.56.22.8". По умолчанию строка для этого параметра указывает, что CSocket сам должен определять адрес клиента.

Возвращаемые значения:

- Если завершается успешно, возвращаемое значение отлично от нуля.
- Если функция завершилась ошибкой, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке вызовите функцию **GetLastError**.

2. **Listen** - устанавливает сокет в состояние прослушивания. Аналог функции **listen**.

Синтаксис:

```
BOOL Listen(
                int nConnectionBacklog
                  );
```

Параметр:

nConnectionBacklog - положительное число от 1 до 5 определяющая максимальную длину очереди запросов ожидающих соединения.

Возвращаемые значения:

- Если завершается успешно, возвращаемое значение отлично от нуля.
- Если функция завершилась ошибкой, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке вызовите функцию **GetLastError**.

3. **Accept** - принимает соединение, т.е. выбирает первый в очереди сокет на соединение, создает новый сокет и присоединяет его к **rConnectedSocket**. Аналог функции **accept**.

Синтаксис:

```
BOOL Accept(
        CSocket &rConnectedSocket,
        SOCKADDR *lpSockAddr,
        int *lpSockAddrLen
          );
```

Параметры:

- **rConnectedSocket** - ссылка на объект сокет принимающий соединение.
- ***lpSockAddr*** - указатель на структуру **SOCKADDR**, в которую будет записан адрес сокета, который соединятся. Информация для **lpSockAddr** будет взята из параметров при создании сокета. Если параметр опущен, то нет информации об удаленном адресе принимающего сокета.
- ***lpSockAddrLen*** - указатель на переменную, в которую будет записан размер структуры **SOCKADDR** в байтах. Если параметр опущен, то нет информации об удаленном адресе принимающего сокета.

Возвращаемые значения:

- Если завершается успешно, возвращаемое значение отлично от нуля.
- Если функция завершилась ошибкой, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке вызовите функцию **GetLastError**.

4. **Connect** - вызывается в клиентском процессе для установления соединения с серверным сокетом. Аналог функции **connect**.

Синтаксис:

```
void Connect(
        char address,
        int port
          );
```

Параметры:

- **address** - IP-адрес удаленного узла.
- **port** - номер порта удаленного узла.

Возвращаемые значения:

- Если завершается успешно, возвращаемое значение отлично от нуля.
- Если функция завершилась ошибкой, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке вызовите функцию **GetLastError**.

5. **Send** - отправляет данные на соединенный сокет. Аналог функции **send**.

Синтаксис:

```
int Send(
        const void* lpBuf,
        int nBufLen,
        int nFlags
          );
```

Параметры:

- **lpBuf** - указатель на участок памяти, в котором содержатся данные для отправки.
- **nBufLen** - размер в байтах отправляемых данных.
- **nFlags** - набор битовых флагов, управляющих работой функции. Если параметр опущен, то флаги не используются.

Возвращаемые значения:

- Если завершается успешно, возвращаемое значение отлично от нуля.
- Если функция завершилась ошибкой, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке вызовите функцию **GetLastError.**

6. **Receive** - принимает данные на соединенный сокет. Аналог функции **recv**.

Синтаксис:

```
int Receive(
        const void* lpBuf,
        int nBufLen,
        int nFlags
            );
```

Параметры:

- **lpBuf** - указатель на участок памяти, в который принимаются данные.
- **nBufLen** - размер в байтах принимаемых данных.
- **nFlags** - набор битовых флагов, управляющих работой функции. Если параметр опущен, то флаги не используются.

Возвращаемые значения:

- Если завершается успешно, возвращаемое значение отлично от нуля.
- Если функция завершилась ошибкой, возвращаемое значение - NULL. Чтобы получить расширенную информацию об ошибке вызовите функцию **GetLastError**.

7. **Close** - освобождается дескриптор сокета, и все дальнейшие операции с сокетом закончатся с ошибкой. Аналог функции **closesocket**.

Синтаксис:

```
int Close();
```