Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №4 по теме: «Программирование протоколов TCP/UDP с использованием библиотеки Winsock»

**Выполнил:**  
студент группы ПВ-31  
Адаменко И. И.

**Проверил:**старший преподаватель  
Федотов Е. А.

Белгород  
2015

**Цель работы:** изучить протоколы TCP/UDP, основные функции библиотеки Winsock и составить программу для приёма/передачи пакетов.

**Задание:**

1. Разработать программу «Сервер» (на языке программирования C#), которая посылает клиентам некоторое сообщение.
2. Разработать программу «Клиент» (на языке программирования C#), которая ждём от сервера сообщения и затем получает его.
3. Провести анализ функционирования разработанных программ (одновременная работа 2, 3 и т. д. приложений на 2, 3 и т. д. компьютерах ЛВС), сделать выводы.
4. Провести сравнительный анализ протоколов TCP и UDP. Сделать выводы.

# Теоретическая часть

**Протокол TCP.** TCP протокол базируется на IP для доставки пакетов, но добавляются две важные вещи: во-первых, устанавливается соединение — это позволяет ему, в отличие от IP, гарантировать доставку пакетов, во-вторых — используются порты для обмена пакетами между приложениями, а не просто узлами.

Протокол TCP предназначен для обмена данными — это «надёжный» протокол, потому что он, во-первых, обеспечивает надёжную доставку данных, так как предусматривает установления логического соединения; во-вторых, нумерует пакеты и подтверждает их приём квитанцией, а в случае потери организует повторную передачу; в-третьих, делит передаваемый поток байтов на части — сегменты — и передаёт их нижнему уровню, на приёмной стороне снова собирает их в непрерывный поток байтов.

**Протокол UDP.** Протокол UDP (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удалённым модулем UDP («бессвязный» протокол). К заголовку IP-пакета UDP добавляет поля порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля длина UDP-дейтограммы и контрольная сумма, позволяющие поддерживать целостность данных. Таким образом, если на уровне IP для определения места доставки пакета используется адрес, на уровне UDP — номер порта.

Протокол UDP ориентирован на транзакции, получение датаграмм и защита от дублирования не гарантированы. Приложения, требующие гарантированного получения потоков данных, должны использовать протокол управления пересылкой (Transmission Control Protocol — TCP).

UDP — минимальный ориентированный на обработку сообщений протокол транспортного уровня, задокументированный в RFC 768. UDP не предоставляет никаких гарантий доставки сообщения для протокола верхнего уровня и не сохраняет состояния отправленных сообщений.

UDP обеспечивает многоканальную передачу (с помощью номеров портов) и проверку целостности (с помощью контрольных сумм) заголовка и существенных данных. Надёжная передача в случае необходимости должна реализовываться пользовательским приложением.

Из-за недостатка надёжности, приложения UDP должны быть готовыми к некоторым потерям, ошибкам и дублированиям. Некоторые из них (например, TFTP) могут при необходимости добавить элементарные механизмы обеспечения надёжности на прикладном уровне.

Но чаще такие механизмы не используются UDP-приложениями и даже мешают им.

Более серьёзной потенциальной проблемой является то, что в отличие от TCP, основанные на UDP приложения не обязательно имеют хорошие механизмы контроля и избегания перегрузок. Чувствительные к перегрузкам UDP-приложения, которые потребляют значительную часть доступной пропускной способности, могут поставить под угрозу стабильность в Интернете.

Функции, необходимые для работы с Winsock в этой лабораторной работе:

* WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData) — инициализирует библиотеку Winsock. В случае успеха возвращает 0.
* WSAGetLastError (void) — возвращает код ошибки, возникшей при выполнении последней операции.
* WSACleanup (void) — осуществляет очистку памяти, занимаемой библиотекой Winsock. Функция деинициализирует библиотеку Winsock и возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR. Расширенный код ошибки можно получить при помощи функции WSAGetLastError.
* SOCKET (int af, int type, int protocol) — возвращает либо дескриптор созданного сокета, либо ошибку INVALID \_SOCKET.
* bind (SOCKET s, const struct sockaddr FAR\* name, int namelen) — привязывает сокет к какому-нибудь локальному адресу и порту.
* listen (SOCKET s, int backlog) — переводит сокет в состояние «прослушивания» (для потоковой передачи).
* connect (SOCKET s, const struct sockaddr FAR\* name, int namelen) — используется процессом-клиентом для установления связи с сервером при потоковой передаче. В случае успешного установления соединения connect возвращает 0, иначе SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* accept (SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen) — используется для принятия связи на сокет.
* sendto (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen) — осуществляет передачу данных с помощью датаграмм.
* send (SOCKET s, const char FAR \* buf, int len, int flags) — осуществляет передачу потоковых данных.
* recvfrom (SOCKET s, char FAR\* buf, int len, int flags, struct sockaddr FAR\* from, int FAR\* fromlen) — осуществляет приём данных с помощью датаграмм.
* recv (SOCKET s, char FAR\* buf, int len, int flags) — осуществляет приём потоковых данных.
* Closesocket (SOCKET s) — служит для закрытия сокета.

# Используемые функции

В этой лабораторной работе использовалась встроенная в .NET реализация Winsock, функционал которой заложен в классах Socket и EndPoint.

Используемые методы класса Socket:

* Bind — связывает объект Socket с локальной конечной точкой;
* ReceiveFrom — принимает датаграмму в буфер данных и сохраняет конечную точку;
* SendTo — посылает данные на указанную точку;
* Connect — создаёт подключение к удалённому хосту;
* Receive — возвращает данные из связанного объекта Socket в приёмный буфер;
* Listen — устанавливает объект Socket в состояние прослушивания;
* Accept — создаёт новый объект Socket для заново созданного подключения;
* Send — передаёт данные в подключённый объект Socket;
* Close — закрывает подключение Socket и освобождает все используемые ресурсы.

# Анализ функционирования программы

Было проведено тестирование программы на одном и на двух компьютерах, с использованием:

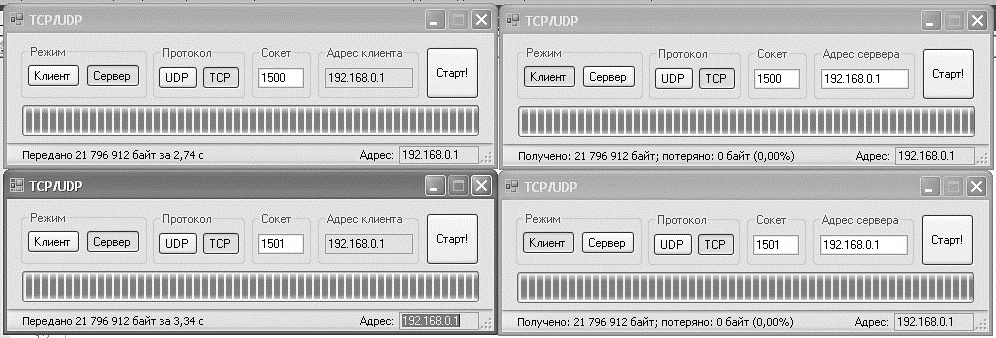
* 1 сервера и 1 клиента;
* 2 серверов и 2 клиентов;
* 3 серверов и 3 клиентов.

Результаты тестирования передачи изображения размером в 21 796 912 байт представлены в таблице ниже:

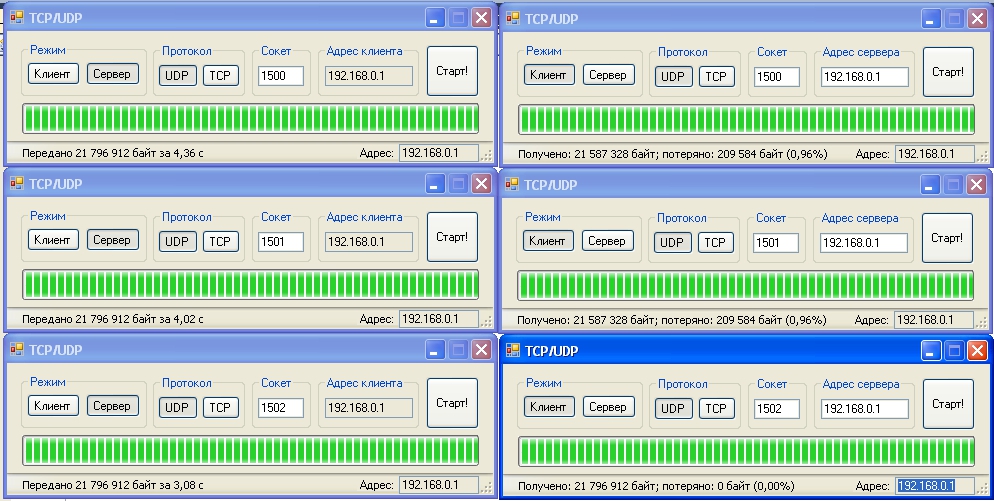
|  |  |  |
| --- | --- | --- |
| **Серверы и клиенты** | **Время** | **Потеряно** |
| **UDP, 1 компьютер** | | |
| 1 сервер, 1 клиент | 2,09 с | 660 664 байта, 3,03% |
| 2 сервера, 2 клиента | 3,21 с | 578 008 байт, 2,65% |
| 2,53 с | 606 232 байта, 2,78% |
| 3 сервера, 3 клиента | 3,93 с | 209 080 байт, 0,96% |
| 3,17 с | 2 645 416 байт, 12,14% |
| 2,53 с | 199 000 байт, 0,91% |
| **UDP, 2 компьютера** | | |
| 1 сервер, 1 клиент | 12,18 с | 126 928 байт, 0,58% |
| 2 сервера, 2 клиента | 22,63 с | 3 952 байта, 0,02% |
| 20,73 с | 45 280 байт, 0,21% |
| 3 сервера, 3 клиента | 33,34 с | 18 064 байта, 0,08% |
| 34,86 с | 16 552 байта, 0,08% |
| 35,09 с | 14 032 байта, 0,06% |
| **TCP, 1 компьютер** | | |
| 1 сервер, 1 клиент | 2,24 с | 0 байт, 0% |
| 2 сервера, 2 клиента | 2,66 с | 0 байт, 0% |
| 3,14 с | 0 байт, 0% |
| 3 сервера, 3 клиента | 4,28 с | 0 байт, 0% |
| 3,63 с | 0 байт, 0% |
| 3,17 с | 0 байт, 0% |
| **TCP, 2 компьютера** | | |
| 1 сервер, 1 клиент | 9,97 с | 0 байт, 0% |
| 2 сервера, 2 клиента | 16,86 с | 0 байт, 0% |
| 18,21 с | 0 байт, 0% |
| 3 сервера, 3 клиента | 27,06 с | 0 байт, 0% |
| 28,07 с | 0 байт, 0% |
| 28,12 с | 0 байт, 0% |

# Скриншоты приложения

Передача с помощью TCP на одной машине, с двумя парами приложений клиент-сервер:



Передача с помощью UDP на одной машине, с тремя парами приложений клиент-сервер:



# Код программы

Поскольку большая часть кода здесь аналогична коду третьей лабораторной программы (а частично и второй), то ниже представлены только дополнительные классы для работы с UDP.

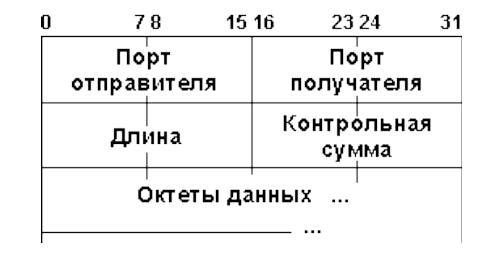
Класс для работы с клиентом UDP:

1. **using** System.Net.Sockets;
3. **namespace** Lab4
4. {
5. **class** UDPClient : Client
6. {
7. **private** Socket \_usingsocket;
9. **public** UDPClient(**string** ip, **ushort** socket) : **base**(ip, socket)
10. {
11. }
13. **public** **override** **void** Start()
14. {
15. \_usingsocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Dgram,  
     ProtocolType.Udp);
16. \_usingsocket.Bind(\_endpoint);
17. }
19. **public** **override** **byte**[] Receive(**short** length)
20. {
21. **byte**[] byteArray = **new** **byte**[length];
23. \_usingsocket.ReceiveFrom(byteArray, **ref** \_endpoint);
25. **return** byteArray;
26. }
28. **public** **override** **void** Stop()
29. {
30. \_usingsocket.Close();
31. }
32. }
33. }

Класс для работы с сервером UDP:

1. **using** System.Net.Sockets;
3. **namespace** Lab4
4. {
5. **class** UDPServer : Server
6. {
7. **private** Socket \_usingsocket;
9. **public** UDPServer(**string** ip, **ushort** socket) : **base**(ip, socket)
10. {
11. }
13. **public** **override** **void** Start()
14. {
15. \_usingsocket = **new** Socket(AddressFamily.InterNetwork, SocketType.Dgram,  
     ProtocolType.UDP);
16. }
18. **public** **override** **void** Send(**byte**[] buf)
19. {
20. \_usingsocket.SendTo(buf, \_endpoint);
21. }
23. **public** **override** **void** Stop()
24. {
25. \_usingsocket.Close();
26. }
27. }
28. }

# Ответы на вопросы

1. **Что представляет собой протокол TCP? Как он работает?**Transmission Control Protocol (TCP) (протокол управления передачей) — один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня модели OSI.  
     
   TCP — это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи. Реализация TCP, как правило, встроена в ядро ОС, хотя есть и реализации TCP в контексте приложения.  
     
   Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. Также TCP осуществляет надёжную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере. Программы для электронной почты и обмена файлами используют TCP. TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.
2. **Порядок установление TCP-соединения**TCP соединение начинается с т.н. “рукопожатия”: узел A посылает узлу B специальный пакет SYN — приглашение к соединению; B отвечает пакетом SYN-ACK — согласием об установлении соединения; A посылает пакет ACK — подтверждение, что согласие получено.  
     
   После этого TCP соединение считается установленным, и приложения, работающие в этих узлах, могут посылать друг другу пакеты с данными.  
     
   «Соединение» означает, что узлы помнят друг о друге, нумеруют все пакеты, идущие в обе стороны, посылают подтверждения о получении каждого пакета и перепосылают потерявшиеся по дороге пакеты. Для узла A это соединение называется исходящим, а для узла B — входящим. Любое установленное TCP соединение симметрично, и пакеты с данными по нему всегда идут в обе стороны.
3. **В чём состоит отличие протокола UDP от TCP?**В отличие от традиционной альтернативы — UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:
4. Установка соединения.
5. Передача данных.
6. Завершение соединения.
7. **Формат заголовка пакета UDP**Заголовок UDP состоит из четырёх полей, каждое по 2 байта (16 бит). Два из них необязательны к использованию в IPv4, в то время как в IPv6 необязателен только порт отправителя.  
     
   В поле **Порт** **отправителя** указывается номер порта отправителя. Предполагается, что это значение задаёт порт, на который при необходимости будет посылаться ответ. В противном же случае, значение должно быть равным 0.  
     
   Поле **Порт получателя** обязательно и содержит порт получателя.   
     
   Поле **Длина датаграммы** задаёт длину всей датаграммы (заголовка и данных) в байтах. Минимальная длина равна длине заголовка — 8 байт. Теоретически, максимальный размер поля — 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные). Фактический предел для длины данных при использовании IPv4 — 65507 (помимо 8 байт на UDP-заголовок требуется ещё 20 на IP-заголовок).

**Поле контрольной суммы** используется для проверки заголовка и данных на ошибки. Если сумма не сгенерирована передатчиком, то поле заполняется нулями. Поле является обязательным для IPv6.

1. **Опишите работу функций sendto и send библиотеки Winsocket**
   * sendto (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen) — осуществляет передачу данных по протоколу UDP.
   * send (SOCKET s, const char FAR \* buf, int len, int flags) — осуществляет передачу данных по протоколу TCP.
2. **В каких случаях предпочтительнее использовать протокол UDP?**Потоковые медиа, многопользовательские игры в реальном времени и VoIP — примеры приложений, часто использующих протокол UDP. В этих конкретных приложениях потеря пакетов обычно не является большой проблемой. Если приложению необходим высокий уровень надёжности, то можно использовать другой протокол.

# Блок-схемы работы программы