Глава 7. Практикум

7.1. Предисловие к главе

Для выполнения практических заданий предлагаемых в этой главе, необходимо иметь несколько объединенных TCP/IP-сетью компьютеров с операционной системой Windows XP. Кроме того, для разработки приложений на языке C++ требуется среда разработки Microsoft Visual Studio не ниже седьмой версии и доступ к соответствующей версии MSDN.

Каждая практическая работа состоит из нескольких заданий. Задания, как правило, связаны между собой и требуют последовательного выполнения. Практическая работа считается выполненной, если успешно выполнены все ее задания.

7.2. Практическая работа № 1. Сетевые утилиты

7.2.1. Цель и задачи работы

Целью работы является ознакомление с функциональными возможностями сетевых утилит операционной системы Windows.

В результате работы студент будет уметь определять характеристики TCP/IP-сети, тестировать соединения компьютеров в сети, использовать сетевые утилиты при отладке приложений.

7.2.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены во второй главе этого пособия. В качестве дополнительной литературы рекомендуются источники [5, 6, 7, 10].

7.2.3. Утилита ipconfig

Задание 1. Получите справку о параметрах утилиты **ipconfig.**

Задание 2. Получите короткий отчет утилиты исследуйте его**.**

Задание 3. Получите полный отчет утилиты**.** Выпишите символическое имя хоста , IP-адрес, маску подсети, MAC-адрес адаптера.

Задание 4. Определите, к какому классу адресов относится выписанный IP-адрес; вычислите максимальное количество хостов, которое может быть в подсети и укажите диапазон их адресов; определите код производителя сетевого адаптера.

7.2.4. Утилита hostname

Задание 5. Определите имя NetBIOS-имя компьютера с помощью утилиты **hostname**. Сравните его с именем полученным с помощью утилиты **ipconfig**.

7.2.5. Утилита ping

Задание 6. Получите справку о параметрах утилиты **ping.**

Задание 7. С помощью **ping** проверьте работоспособность интерфейса внутренней петли компьютера.

Задание 8. С помощью утилиты **ping** проверьте доступность интерфейса какого-нибудь компьютера в локальной сети, указав в качестве параметров его IP-адрес.

Задание 9. С помощью утилиты **ping** проверьте доступность интерфейса какого-нибудь компьютера в локальной сети, указав в качестве параметров символическое имя хоста.

Задание 10. С помощью утилиты **ping** проверьте доступность интерфейса какого-нибудь компьютера в локальной сети, указав в качестве параметров символическое имя хоста и увеличив размер буфера отправки до 1000 байт

Задание 11. С помощью утилиты **ping** проверьте доступность интерфейса какого-нибудь компьютера в локальной сети, указав в качестве параметров его IP-адрес и установив количество отправляемых запросов равное 17.

Примечание. Обратите внимание на значение TTL, которое выдается в отчетах утилиты **ping.** Первоначальное значение TTL (Time To Live, время жизни) по умолчанию равно 128. Это значение записывается в заголовок каждой дейтаграммы и уменьшается на единицу после прохождения каждого маршрутизатора. Если в процессе движения дейтаграммы в сети значение TLL уменьшится до нуля, то дейтаграмма уничтожается. Такой подход гарантирует от зацикливания дейтаграмм в сети. С помощью ключа **i** утилиты ping, можно на период проверки значение TTL изменить.

7.2.5. Утилита tracert

Задание 12. Получите справку о параметрах утилиты **tracert**.

Задание 13. С помощью утилиты **tracert** определите маршрут хоста самого к себе (интерфейс внутренней петли).

Задание 14. С помощью утилиты **tracert** определите маршрут к хосту в локальной сети. Определите количество прыжков в полученном маршруте.

7.2.6. Утилита route

Задание 15. Получите справку о параметрах утилиты **route**.

Задание 16. Распечатайте на экран монитора таблицу активных маршрутов компьютера. Исследуйте полученный отчет. Определите строки таблицы, соответствующие интерфейсу внутренней петли и широковещательным адресам. Определите IP- адреса шлюзов.

7.2.7. Утилита arp

Задание 17. Получите справку о параметрах утилиты **arp**.

Задание 18. Распечатайте на экран монитора arp-таблицу. Исследуйте полученный отчет. Определите хосты, которым соответствуют строки arp-таблицы. Определите IP-адрес, которого нет в arp-таблице, но есть в локальной сети. Выполните утилиту **ping** в адрес этого хоста. Распечатайте снова arp-таблицу и объясните произошедшие изменения. Определите MAC-адреса двух хостов с ближайшими IP-адресами.

7.2.8. Утилита nslookup

Задание 19. **З**апустите утилиту **nslookup** в диалоговом режиме и наберите команду **help**. Ознакомьтесь с полученным отчетом, отражающим возможности утилиты **nslookup**.

Задание 20. **З**апустите утилиту **nslookup** в диалоговом режиме. Определите имя и IP-адрес хоста, на котором установлен DNS-сервер по умолчанию. Определите IP-адреса хостов по их именам (имена хостов выдаст преподаватель).

7.2.9. Утилита netstat

Задание 21. Получите справку о параметрах утилиты **netstat**.

Задание 22. Запустите утилиту **netstat -a** для отображения всех подключений и ожидающих портов. Исследуйте отчет. Выясните, какие из известных служб прослушивают порты. С какими из этих портов поддерживается внешнее соединение и по какому протоколу ? Определите имена хостов и номера портов внешних соединений .

Задание 23. Запустите утилиту **netstat -b** для отображения исполняемых файлов участвующих в создании подключений. Определите исполняемые файлы служб, прослушивающих порты, идентификаторы процессов операционной системы.

Задание 24. Запустите утилиту **netstat -ab**. Исследуйте полученный отчет. Для формирования файла отчета утилиты, перенаправьте вывод утилиты в файл с помощью команды: **netstat -ab > c:\report.txt.** Проконтролируйте наличие отчета в файле.

7.2.9. Утилита nbstat

Задание 25. Получите справку о параметрах утилиты **nbtstat**. Выполните все команды отраженные в справке. Исследуйте полученные отчеты.

7.2.9. Утилита net

Задание 26. Получите справку о параметрах утилиты **net**. Получите справку по отдельным командам утилиты с помощью команды **help.** Получите статистику рабочей станции и сервера компьютера с помощью команды **statistics**. Перешлите сообщение на соседний компьютер с помощью команды **send**. Получите список пользователей компьютера с помощью команды **user**.

7.3. Практическая работа № 2. Обмен данными по TCP-соединению

7.3.1. Цель и задачи работы

Основной целью практической работы является приобретение навыков разработки простейшего распределенного приложения архитектуры клиент-сервер, осуществляющего обмен данными в локальной сети через Windows Sockets TCP-соединение.

Результатом практической работы является разработанное распределенное приложение со схемой взаимодействия процессов, описанной в разделе 3.4 и изображенной на рисунке 3.4.2 пособия.

7.3.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены в разделах 3.2-3.11, 3.14 пособия.

7.3.3. Разработка серверной части распределенного приложения

Задание 1. Ознакомьтесь со схемой сервера, изображенной на рисунке 3.4.2 пособия. Создайте с помощью Visual Studio консольное приложение **ServerT** (наименование проекта), которое будет использовано для построения серверной части приложения (сервера). Включите необходимые директивы компилятора (указанные в разделе 3.2 пособия) для подключения динамической библиотеки **WS2_32.LIB**. Откомпилируйте приложение, убедитесь в отсутствии ошибок.

Задание 2. В рамках приложения **ServerT**, созданного в задании 1, разработайте о функцию **SetErrorMsgText,** предназначенную для обработки стандартных ошибок библиотеки **WS2_32.LIB**. Предполагается, что функция **SetErrorMsgText** будет использоваться в операторе **throw** для генерации исключения при возникновении ошибок в функциях интерфейса Winsock2. Для получения кода ошибки функций Winsock2 примените функцию **WSAGetLastError,** описание которой приводится в разделе 3.3 пособия. Там же приводится полный список кодов возврата функции **WSAGetLastError** и пример ее использования.

Задание 3. Доработайте приложение **ServerT** таким образом, чтобы оно только инициализировало библиотеку **WS2_32.LIB** и завешало работу с этой библиотекой. Для этого используйте функции **WSAStartup** и **WSACleanup**, описанные в разделах 3.5 и 3.6 пособия. Обработку ошибок осуществите с помощью конструкции **try-catch** и функции **SetErrorMsgText**, разработанной в задании 2. Используйте пример программы, приведенный в разделе 3.6 пособия. Убедитесь в работоспособности приложения.

Задание 4. Доработайте приложение **ServerT** таким образом**,** чтобы оно создавало и закрывало сокет, предназначенный для ориентированного на поток соединения. Для этого используйте функции **socket** и **closesocket**, описанные в разделе 3.8. Обратите внимание на параметр **type** функции **socket**, указывающий тип соединения. Воспользуйтесь примером из раздела 3.7 пособия. Убедитесь в работоспособности приложения.

Задание 5. Добавьте в приложение **ServerT** вызов функций **bind** и **listen** для установки параметров сокета и перевода его в режим прослушивания. Функция **bind** описана в разделе 3.8, а функция **listen** в разделе 3.9 пособия. Используйте порт **2000**, в качестве параметра сокета. Установка параметров сокета осуществляется с помощью структуры

SOCKADDR_IN. Описание этой структуры приводится в разделе 3.8 пособия. Сверьте схему полученной программы со схемой сервера, изображенной на рисунке 3.4.2. Выполните приложение, убедитесь в его работоспособности.

Задание 6. Добавьте в приложение **ServerT** вызов функции **accept**, описание которой приводится в разделе 3.10 пособия. Следует обратить внимание на: 1) успешным результатом работы функции **accept** является новый сокет; 2) первым параметром функции **accept** является уже созданный ранее сокет; 3) второй параметр функции **accept** – указатель на структуру **SOCKADDR_IN** (не надо ее путать с уже применяемой выше), предназначенную для приема параметров, подключившегося сокета со стороны клиента сокета. Запустите приложение в режиме отладки (Debug) и убедитесь, что после выполнения функции **accept**, программа переходит в режим ожидания (зависает). Завершите приложение. Сохраните программу **ServerT** для дальнейшего применения.

7.3.4. Разработка клиентской части распределенного приложения

Задание 7. Ознакомьтесь со схемой клиента, изображенной на рисунке 3.4.2 пособия. Создайте с помощью Visual Studio новое консольное приложение **ClientT** (наименование проекта), которое будет использовано для построения клиентской части приложения (клиента). Повторите все те же действия для этого приложения, которые были сделаны в заданиях 2-4. Убедитесь в работоспособности приложения **ClientT**.

Задание 8. Добавьте в приложение **ClientT,** вызов функции **connect**. Описание функции и примера ее использования приводится в разделе 3.10 пособия. Следует обратить внимание на следующее: 1) параметры сокета сервера устанавливаются в структуре **SOCKADDR_IN**; 2) для номера порта необходимо установить значение **2000** (такой же номер, что установлен при параметризации сокета сервера а задании 5); 3) для установки номера порта используются специальные функции, описание которых приводится в разделе 3.8. Используйте в качестве IP-адреса собственный адрес компьютера **127.0.0.1** (интерфейс внутренней петли) – это даст возможность отладки приложения на одном компьютере. Запустите приложение на выполнение. Убедитесь, что функция **connect** завершилась с ошибкой и обработка ошибок осуществляется корректно. Найдите полученный код ошибки в таблице 3.3.1 пособия и проанализируйте его.

7.3.5. Обмен данными между сервером и клиентом

Задание 9. Запустите на выполнение приложение **ServerT** и убедитесь, что оно приостановилось на вызове функции **accept**. Запустите на

выполнение на этом же компьютере (используется интерфейс внутренней петли) приложение **ClientT**. Убедитесь, что сервер **ServerT**, вышел из состояния ожидания, а клиент **ClientT** завершился без ошибок.

Задание 10. Доработайте программу сервера **ServerT** таким образом, чтобы после подсоединения клиента на экран консоли **ServerT** выводился IP-адрес и порт, подсоединившегося клиента. Необходимые значения находятся в структуре **SOCKADDR_IN**, которая заполняется функцией **accept**. Используйте функции **htons** и **inet_ntoa**, описанные в разделе 3.8. Убедитесь в работоспособности распределенного приложения **ClientT-ServerT**.

Задание 11. Добавьте в программу сервера **ServerT** вызов функции **recv,** а в программу клиента вызов функции **send.** Описание этих функций приводится в разделе 3.11 пособия. Перешлите текст ***Hello from Client*** от клиента серверу. Выведите полученный сервером текст на экран консоли. Обратите внимание на то, что команда **recv** в программе сервера использует сокет, созданный функцией **accept,** а не созданный ранее с помощью функции **socket**.

Задание 12. Установите программу **ServerT** на другой компьютер локальной сети, а в программу **ClientT** внесите необходимые изменения, позволяющие ей установить связь с сервером. Убедитесь в работоспособности распределенного в локальной сети приложения **ClientT-ServerT**.

Задание 13. Внесите изменения в программы **ClientT** и **ServerT**, позволяющие 1000 раз передать сообщение типа ***Hello from Client xxx*** (***xxx*** – номер сообщения) от клиента серверу. Убедитесь в работоспособности в сети.

Задание 14. Доработайте программы **ClientT** и **ServerT** таким образом, чтобы программа **ServerT** принимала последовательность сообщений вида ***Hello from Client xxx*** от программы **ClientT** и отправляла их без изменения обратно программе **ClientT**. Программа **ClientT,** получив вернувшееся сообщение, должна увеличить в нем счетчик ***xxx*** на единицу и вновь направить в адрес **ServerT**. Количество передаваемых сообщений введите через консоль программы **ClientT**. Условием окончания работы для программы **ServerT** является получение сообщения нулевой длины. Оцените время обмена 1000 сообщениями (с помощью функций **clock**) между **ClientT** и **ServerT** через локальную сеть. Запустите утилиту **netstat** на компьютерах клиента и сервера, проанализируйте отчет и найдите информацию о приложении **ClientT- ServerT**.

Задание 15. Доработайте программу **ServerT** таким образом, чтобы отключения клиента она могла снова установить соединение с другим клиентом и продолжила свою работу.

7.4. Практическая работа № 3. Обмен данными без установки соединения

7.4.1. Цель и задачи работы

Основной целью практической работы является приобретение навыков разработки простейшего распределенного приложения архитектуры клиент-сервер, осуществляющего обмен данными в локальной сети через Windows Sockets без установки соединения (с помощью UDP-сообщений).

Результатом практической работы является разработанное распределенное приложение со схемой взаимодействия процессов, описанной в разделе 3.4 и изображенной на рисунке 3.4.1 пособия.

7.4.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены в разделах 3.2-3.12, 3.14 пособия.

7.4.3. Разработка серверной части распределенного приложения

Задание 1. Ознакомьтесь со схемой взаимодействия процессов без установки соединения в распределенном приложении, приведенной разделе 3.4 пособия (рисунок 3.4.1). Определите основные отличия этой схемы от схемы взаимодействия процессов с установкой соединения. Разработайте программу **ServerU**, реализующую блоки 1, 2 и 5 схемы сервера, изображенной на рисунке 3.4.1. Подключите функции обработки ошибок, разработанные в практической работе № 2 (с применением команд структурной обработки ошибок **try-throw-catch**). Обратите внимание 1) на параметр **type** функции **socket**; 2) на отсутствие функций **listen** и **accept**, которые применялись в приложении с соединением. Убедитесь, что разработанная программа выполняет все функции Winsock2 без ошибок.

Примечание. При разработке программ в заданиях этой практической работы рекомендуется использовать тексты программ, разработанных в предыдущей практической работе.

Задание 2. Реализуйте в программе **ServerU** блок 3 схемы сервера изображенной на рисунке 3.4.1. Используемая в блоке функция **recvfrom** описана в разделе 3.12 пособия. Установите номер серверного сокета равным **2000**. Убедитесь, что при запуске программа **ServerU**

приостанавливает свое выполнение (переходит в состояние ожидания) сразу после вызова функции **recvfrom**. Завершите программу.

7.4.4. Разработка клиентской части распределенного приложения

Задание 3. Создайте новое C++ -приложение с именем **ClientU**. Реализуйте блоки 1, 2, 3 и 5 схемы клиента, изображенной на рисунке 3.4.1. Подключите функции обработки ошибок, разработанные в практической работе № 2. В параметре **to** команды **sendto** (раздел 3.12), установите адрес структуры **SOCKADDR_IN**, содержащей IP-адрес равный **127.0.0.1** и номер порта равный **2000**. Обеспечьте пересылку сообщения ***Hello from ClientU***. Запустите на выполнение программу **ClientU** при отсутствующем сервере. Проанализируйте полученный код возврата.

7.4.5. Обмен данными между сервером и клиентом

Задание 4. Запустите на выполнение программу **ServerU** и убедитесь, что она приостановила свое выполнение. Запустите на этом же компьютере программу **ClientU** и убедитесь, что программы сервера получила сообщение и завершилась нормально.

Задание 5. Реализуйте блоки 4 в обеих программах. Перешлите полученное сервером сообщение обратно в адрес клиента и убедитесь, что сообщение получено.

Задание 6. Внесите необходимые изменения в программу **ClientU** для того, чтобы программы можно было бы расположить на разных компьютерах локальной сети. Убедитесь в работоспособности приложения.

Задание 7. Реализуйте последовательную пересылку данных от клиента к серверу и обратно по тому же принципу как это было сделано в заданиях 13, 14 практической работы № 2. Проведите измерения аналогичные оценки скорости передачи, сравните результаты.

Задание 8. Запустите сервер **ServerU** на одном из компьютеров и одновременно два клиента на двух других компьютерах локальной сети. Оцените количество сообщений, которые успел передать и получить каждый из клиентов.

Задание 9. Запустите сервер **ServerT** (разработанный в практической работе № 2) и программу клиента **ClientU**. Объясните полученный результат.

Задание 10. Запустите сервер **ServerU** и клиент **ClientT**(разработанный в практической работе № 2) . Объясните полученный результат.

7.5. Практическая работа № 4. Применение широковещательных IP-адресов

7.5.1. Цель и задачи работы

Основной целью практической работы является приобретение навыков использования широковещательных адресов распределенными в локальной сети приложениями.

Результатом практической работы являются разработанное распределенное приложение, использующее широковещательные адреса.

7.5.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены в разделах 3.2-3.12, 3.15 пособия.

7.5.3. Разработка серверной части распределенного приложения

Задание 1. Разработайте функцию **GetRequestFromClient**, описание которой представлено на рисунке 7.5.1. Функция предназначена для ожидания запроса клиентской программы. Предполагается, что правильный запрос (позывной сервера) состоит из набора символов, который указывается функции в качестве параметра **name**. Ожидание запроса в функции **GetRequestFromClient** осуществляется с помощью функции **recvfrom**. Если поступившее сообщение является позывным сервера, то функция, заполняет возвращаемую структуру **SOCKADDR_IN** (параметры **from** и **flen** функции) и завершается с кодом возврата **true.** Если поступившее сообщение не является позывным сервера, то оно игнорируется, и функция вновь переходит в состояние ожидания. Если функция **recvfrom** завершается аварийно с кодом **WSAETIMEDOUT** (таблица 3.3.1), то функция **GetRequestFromClient** должна завершиться с кодом возврата **false.** Любой другой аварийный код завершения должен приводить к исключительной ситуации (оператор **throw**), соответствующей функциям обработки ошибок разработанных в практическом занятии № 2.

Создайте новое приложение **ServerB**, вызывающее функцию **GetRequestFromClient**. Пусть позывной сервера будет ***Hello***. Запустите приложение **ServerB** и убедитесь, что программа перешла в состояние ожидания. Запустите приложение **ClientU**, разработанное в практической работе № 3. Убедитесь, что **ServerB** не реагирует на ошибочный позывной. Исправьте в приложении **ClientU** посылаемую строку на ***Hello*** и убедитесь, что **ServerB** реагирует на правильный позывной.

```
// -- обработать запрос клиента
// Назначение: функция предназначена для обработки запроса
//             клиентской программы

bool  GetRequestFromClient(
          char*             name, //[in] позывной сервера
          short             port, //[in] номер просушиваемого порта
          struct sockaddr* from, //[out] указатель на SOCKADDR_IN
          int*              flen  //[out] указатель на размер from
                            )

// Код возврата: в случае если пришел запрос клиента, то
//        функция возвращает значение true, иначе возвращается
//        значение false
// Примечание: параметр name – строка, заканчивающаяся 0x00 и
//        содержащая позывной сервера (набор символов,
//        получаемый сервером от клиента и интерпретируемый,
//        как запрос на установку соединения);
//        параметр from – содержит указатель структуры,
//        содержащей параметры сокета клиента приславшего
//        запрос
```

Рисунок 7.5.1. Описание функции GetRequestFromClient

Примечание. При разработке функции **GetRequestFromClient** и других функций в этом практическом задании целесообразно вызов функций **WSAStartup** и **WSACleanup** осуществлять вне разрабатываемых функций.

Задание 2. Разработайте функцию **PutAnswerToClient**, описание которой приводится на рисунке 7.5.2. Функция предназначена для подтверждения сервером запроса клиента на установку соединения. Функция оправляет в адрес клиента (параметры сокета клиента указываются в параметре **to**) свой позывной, что предполагает готовность сервера к дальнейшей работе с клиентом. Предполагается, что функция будет использоваться после завершения функции **GetRequestFromClient**. Внесите изменения в программу **ServerB**, чтобы сервер смог отвечать с помощью функции **PutAnswerToClient** на правильный позывной полученный от клиента. Проверьте правильность работы сервера **ServerB** с помощью программы **ClientU**.

Задание 3. Внесите изменения в программу **ServerB** таким образом, чтобы сервер отвечал на многократные запросы от разных клиентов (необходимо построить цикл с функциями **GetRequestFromClient** и **PutAnswerToClient**). Проверьте работоспособность сервера.

```
// -- ответить на запрос клиента
// Назначение: функция предназначена пересылки позывного
//             сервера программе клиента

bool  PutAnswerToClient(
          char*             name, //[in] позывной сервера
          struct sockaddr* to,    //[in] указатель на SOCKADDR_IN
          int*              lto   //[in] указатель на размер from
                        )

// Код возврата: в случае успешного завершения функция
//        возвращает значение true, иначе возвращается
//        значение false
// Примечание: параметр name – строка, заканчивающаяся 0x00 и
//        содержащая позывной сервера (набор символов,
//        получаемый сервером от клиента и интерпретируемый,
//        как запрос на установку соединения);
//        параметр to – содержит указатель структуры,
//        содержащей параметры сокета клиента
```

Рисунок 7.5.2. Описание функции PutAnswerToClient

7.5.4. Разработка клиентской части распределенного приложения

Задание 4. Разработайте функцию **GetServer**, описание которой приводится на рисунке 7.5.3. Функция предназначена для отправки широковещательного запроса в локальную сеть (всем компьютерам сегмента локальной сети) с позывным сервера. Предполагается, что на одном (или на нескольких) компьютере сети есть сервер **ServerB**, который просушивает порт с номером, указанным в параметре **port** функции **GetServer** . Для отправки широковещательного запроса, функция **GetServer**, должна использовать широковещательный IP-адрес (раздел 3.15 пособия). Использование широковещательного IP-адреса требует специального режима работы сокета, который устанавливается с помощью функции **setsockopt**, входящей в состав Winsock2. Описание этой функции и пример ее использования приводятся в разделе 3.15 пособия. После отправки широковещательного запроса с помощью функции **sendto**, функция **GetServer** должна вызвать функцию **recvfrom** для ожидания отклика сервера. При правильном отклике (отклик должен совпадать с позывным), функция формирует структуру **SOCKADDR_IN** с параметрами сокета сервера, возвращает значение **true** и завешается. Если сообщение в адрес клиента приходит, но отклик не содержит правильный позывной или функция **recvfrom** аварийно завершается с кодом **WSAETIMEDOUT**, функция должна завешаться с кодом возврата **false**. Любой другой аварийный код завершения должен приводить к исключительной ситуации

(оператор **throw**), соответствующей функциям обработки ошибок разработанных в практическом занятии № 2.

```
// -- послать запрос серверу и получить ответ
// Назначение: функция предназначена широковещательной
//             пересылки позывного серверу и приема от
//             сервера ответа.

bool  GetServer(
          char*             call, //[in] позывной сервера
          short             port, //[in] номер порта сервера
          struct sockaddr* from, //[out] указатель на SOCKADDR_IN
          int*              flen  //[out] указатель на размер from
               )
// Код возврата: в случае успешного завершения функция
//        возвращает значение true, иначе возвращается
//        значение false
// Примечание: - параметр call – строка, заканчивающаяся 0x00
//        и содержащая позывной сервера;
//        - параметр from – содержит указатель структуры,
//        которая содержит параметры сокета откликнувшегося
//        сервера
```

Рисунок 7.5.3. Описание функции GetServer

Создайте новое приложение **ClientB**, вызывающее функцию **GetServer**. Запустите сервер **ServerU,** разработанной в практической работе № 3. Убедитесь с помощью отладчика, что происходит обмен данными и функция **GetServer** завершается с возвратом **false** после получения неверного отклика.

7.5.5. Взаимодействие сервера и клиента

Задание 5. Запустите на разных компьютерах программы **ClientB** и **ServerB**. Внесите изменения в программу **ClientB** для того, чтобы она выводила на экран консоли параметры сокета сервера откликнувшегося на позывной. Внесите изменения в программу **ServerB** для того, чтобы она выводила на экран консоли параметры клиента, отправившего правильный позывной в адрес сервера.

Примечание. Разработанные функции **GetRequestFromClient** и **GetServer** имеют существенный недостаток. После вызова функции **recvfrom** они переводят поток в режим ожидания. Выход из этого состояния возможен лишь в том случае, если в адрес сокета поступило сообщение или будет исчерпан допустимый интервал ожидания. Такой алгоритм работы делает эти функции мало применимыми. Сохраните тексты этих функций, они будут дорабатываться в следующих практических работах.

Задание 6. Измените программу **ServerB** таким образом, чтобы при запуске она проверяла наличие в локальной сети еще одного такого же сервера (точнее сервера с тем же позывным) и выдавала на экран консоли предупредительное сообщение о количестве существующих серверов и их IP-адресах.

7.6. Практическая работа № 5. Использование символических имен компьютеров

7.6.1. Цель и задачи работы

Основной целью практической работы является приобретение навыков применения символических имен компьютеров при разработке распределенного в локальной сети приложения.

Результатом практической работы являются разработанное распределенное приложение, использующее символические имена компьютеров.

7.6.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены в разделах 2.8.1, 2.8.3, 3.16 пособия.

7.6.3. Определение адреса компьютера по его символическому имени

Задание 1. Разработайте функцию **GetServerByName**, описание которой приводится на рисунке 7.6.1. Функция предназначена для поиска сервера по его символическому имени и позывному. При этом предполагается, что в локальной сети работает одна из систем (DNS, NetBIOS over TCP/IP), разрешающих символические имена компьютеров. Функция **GetServerByName** является, в некотором смысле, альтернативой функции **GetServer (**практическая работа № 4**)** и должна использоваться в том случае, если известно символическое имя компьютера, на котором запущен сервер. Для поиска сервера фунция **GetServerByName** должна применить функцию **gethostbyname**, описание которой приводится в разделе 3.16. Там же имеется описание структуры **hostent**, которая используется этой функцией для хранения результата работы функции **gethostbyname**. После того, как IP-адрес сервера определен, необходимо установить необходимый номер порта и послать позывной в адрес сокета сервера. В остальном функция **GetServerByName** должна работать по тому же принципу, что и функция **GetServer**. Создайте новое приложение **ClentS**, вызывающее функцию **GetServerByName**. Проверьте работоспособность приложения при работе с программой **ServerB**.

Примечание. Функция **GetServerByName** имеет те же недостатки, что и функция **GetServer.** Сохраните текст этой функции, она будет дорабатываться в следующих практических работах

```
// -- послать запрос серверу, заданному символическим именем
// Назначение: функция предназначена пересылки позывного
//             серверу, адрес которого задан в виде
//             символического имени компьютера.

bool  GetServerByName(
       char*             name, //[in] имя компьютера в сети
       char*             call, //[in] позывной
       struct sockaddr* from, //[in,out] указатель на SOCKADDR_IN
       int*              flen  //[in,out] указатель на размер from
                    )
// Код возврата: в случае успешного завершения (сервер
//        откликнулся на позывной) возвращает значение true,
//        иначе возвращается  значение false
// Примечание: - параметр name – строка, заканчивающаяся 0x00
//        и содержащая  символическое имя компьютера;
//        - параметр call – строка, заканчивающаяся 0x00 и
//        содержащая позывной сервера;
//        - параметр from – содержит указатель структуры
//        SOCKADDR_IN, которая содержит параметры сокета
//        откликнувшегося сервера, перед вызовом функции поле
//        sin_port должно  быть заполнено; если после вызова
//        функции, код возврата равен true, то структура
//        SOCKADDR_IN содержит все параметры сокета сервера
```

Рисунок 7.6.1. Описание функции GetServerByName

7.6.4. Определение имени компьютера по его сетевому адресу

Задание 2. Доработайте программу **ServerB** таким образом, чтобы она распечатывала на экран консоли символическое имя собственного компьютера и символические имена компьютеров клиентов, которые подключаются к серверу. Программа **ServerB** должна использовать функции **gethostname** и **gethostbyaddr**, описание которых приводится в разделе 3.16 пособия.

7.7. Практическая работа № 6. Работа с интерфейсом Named Pipe

7.7.1. Цель и задачи работы

Основной целью практической работы является приобретение навыков использования интерфейса Named Pipe для разработки распределенного приложения.

Результатом практической работы являются разработанное распределенное приложение, применяющее интерфейс Named Pipe .

7.7.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены в четвертой главе пособия.

7.7.3. Разработка серверной части распределенного приложения

Задание 1. Ознакомьтесь со схемой сервера, изображенной на рисунке 4.2.1. Создайте с помощью Visual Studio новое консольное приложение **ServerNP** (наименование проекта), которое будет использовано для построения серверной части распределенного приложения (сервера). Реализуйте блоки 1 и 4 сервера. В блоке 1 используются функции **CreateNamedPipe** и **ConnectNamedPipe**, описание которых приводится в разделе 4.3 пособия. Там же приведены примеры использования этих функций. Следует обратить внимание на формат имени канала, используемый функции **CreateNamedPipe –** он должен быть локальным. Пусть имя создаваемого именного канала будет ***Tube***. В блоке 4 используется функция **DisconnectNamedPipe** для разрыва соединения. Описание функции приводится в разделе 4.3 пособия. Разработайте функции обработки ошибок интерфейса Named Pipe, работающие по тому же принципу, что и функции обработки ошибок Winsock2, используемые в предыдущих практических работах. Запустите приложение **ServerNP** на выполнение и убедитесь, что поток приостановился для ожидания соединения после вызова функции **ConnectNamedPipe**.

7.7.4. Разработка клиентской части распределенного приложения

Задание 2. Ознакомьтесь со схемой клиента, изображенной на рисунке 4.2.1. Создайте с помощью Visual Studio новое консольное приложение **ClientNP**, которое будет использовано для построения клиентской части распределенного приложения (клиента). Реализуйте блоки 1 и 4 клиента. В блоке 1 применяется функция **CreateFile**, описание которой приводится в разделе 4.4 пособия. Установите в параметрах вызова функции **CreateFile**, такое же имя именованного канала, что и для сервера (задание 1). Запустите программу **ClientNP** отдельно (без сервера) и проверьте работоспособность функций обработки ошибок.

7.7.5. Обмен данными между клиентом и сервером

Задание 3. Запустите на выполнение программу **ServerNP**. После того, как программа **ServerNP** перейдет в состояние ожидания, запустите на этом же компьютере программу **ClientNP**. Убедитесь, что программа

ServerNP вышла из состояния ожидания и успешно завершилась. Программа **ClientNP** тоже должна завершиться без ошибок.

Задание 4. Реализуйте блоки 2 и 3 программ **ServerNP** и **ClientNP**. Добейтесь, чтобы программы обменялись сообщениями с помощью функций **WriteFile** и **ReadFile**, описание которых приводится в разделе 4.5 пособия.

Задание 5. Внесите изменения в программы **ServerNP** и **ClientNP** таким образом, чтобы программы взаимодействовали также, как и программы **ServerT** и **ClientT** в заданиях 14 и 15 практической работы № 2.

Задание 6. Внесите изменения в программу **ClientNP** и добейтесь взаимодействие программ клиента и сервера в случае расположения на разных компьютерах локальной сети. Следует иметь в виду, что при вызове функции **CreateFile**, теперь следует использовать сетевой формат имени именованного канала, описанный в разделе 4.3 пособия. В сетевом формате используется символическое имя серверного компьютера. Напомним, что это имя можно получить с помощью утилиты **hostname**, описанной в разделе 2.9.

Задание 7. Разработайте новую программу **ClientNPt**, использующую функцию **TransactNamedPipe** (раздел 4.6 пособия) вместо пары функций **WriteFile** и **ReadFile**. Добейтесь взаимодействия программы **ClientNPt** с сервером **ServerNP**.

Задание 8. Разработайте еще одну новую программу **ClientNPct**, использующую функцию **CallNamedPipe** (раздел 4.6 пособия). Добейтесь взаимодействия программы **ClientNPct** с сервером **ServerNP**.

7.8. Практическая работа № 7. Работа с интерфейсом Mailslots

7.8.1. Цель и задачи работы

Основной целью практической работы является приобретение навыков использования интерфейса Mailslots для разработки распределенного приложения.

Результатом практической работы являются разработанное распределенное приложение, применяющее интерфейс Mailslots .

7.8.2. Теоретические сведения

Теоретические сведения необходимые для выполнения практической работы изложены в пятой главе пособия.