Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Программирование сетевых приложений

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

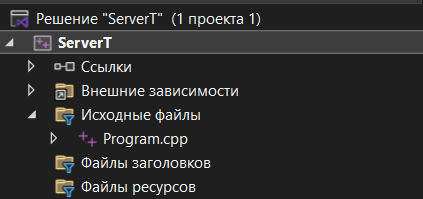
Преподаватель: Некрасова А.П.

Минск 2024

**Лабораторная работа №2. Обмен данными по TCP-соединению**

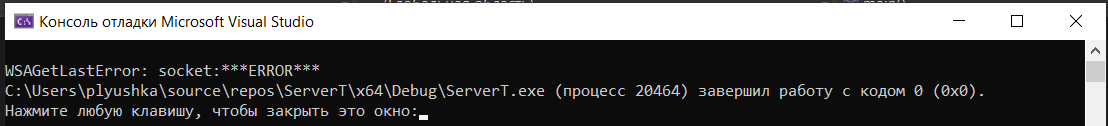
**Цель:** Приобретение навыков разработки простейшего распределенного приложения архитектуры клиент-сервер, осуществляющего обмен данными в локальной сети через Windows Sockets TCP-соединение.

**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.



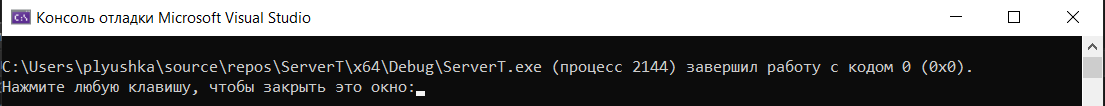
**Задание 2.** В рамках приложения **ServerT**, созданного в задании 1, разработайте о функцию **SetErrorMsgText,** предназначенную для обработки стандартных ошибок библиотеки **WS2\_32.LIB**. Предполагается, что функция **SetErrorMsgText**  будет использоваться в операторе **throw** для генерации исключения 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.

|  |
| --- |
| #include "Winsock2.h"  #pragma comment(lib, "WS2\_32.lib")  #include <ws2tcpip.h>  #include <iostream>  #include <string>  using namespace std;  string GetErrorMsgText(int code)  {  string msgText;  switch (code)  {  case WSAEINTR: msgText = "WSAEINTR"; break;  case WSAEACCES: msgText = "WSAEACCES"; break;    case WSASYSCALLFAILURE: msgText = "WSASYSCALLFAILURE"; break;  default: msgText = "\*\*\*ERROR\*\*\*"; break;  };  return msgText;  };  string SetErrorMsgText(string msgText, int code)  {  return msgText + GetErrorMsgText(code);  };  int main()  {  SOCKET sS;  try  {  if ((sS = socket(AF\_INET, SOCK\_STREAM, NULL)) == INVALID\_SOCKET)  throw SetErrorMsgText("socket:", WSAGetLastError());  }  catch (string errorMsgText)  {  cout << endl << "WSAGetLastError: " << errorMsgText;  }  return 0;  } |



**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.

|  |
| --- |
| int main()  {  WSADATA wsaData;  try  {  if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0)  throw SetErrorMsgText("Startup:", WSAGetLastError());  if (WSACleanup() == SOCKET\_ERROR)  throw SetErrorMsgText("Cleanup:", WSAGetLastError());  }  catch (string errorMsgText)  {  cout << endl << errorMsgText;  }  return 0;  } |



**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**, указывающийтип соединения. Воспользуйтесь примером из раздела 3.7 пособия. Убедитесь в работоспособности приложения.

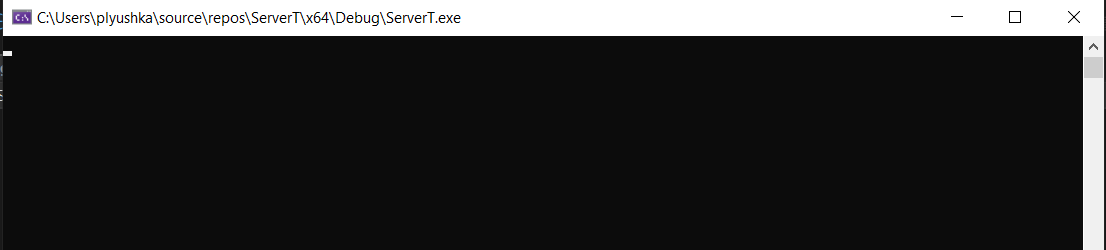
|  |
| --- |
| if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0) // Версия 2.0 сокетов  throw SetErrorMsgText("Startup:", WSAGetLastError());  if ((sS = socket(AF\_INET, SOCK\_STREAM, NULL)) == INVALID\_SOCKET) // IPv4 (AF\_INET) и TCP (SOCK\_STREAM используется для потока)  throw SetErrorMsgText("Socket:", WSAGetLastError());  ...  if (closesocket(cS) == SOCKET\_ERROR) {  throw SetErrorMsgText("Ошибка закрытия сокета клиента:", WSAGetLastError());  }  cout << "Сокет клиента закрылся удачно." << endl; |

**Задание 5.** Добавьте в приложение **ServerT** вызовфункций **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.

|  |
| --- |
| SOCKADDR\_IN serv{};  serv.sin\_family = AF\_INET; // Протокол IPv4  serv.sin\_port = htons(2000); // Порт 2000  serv.sin\_addr.s\_addr = INADDR\_ANY; // Сервер будет слушать все доступные сетевые интерфейсы  if (bind(sS, (LPSOCKADDR)&serv, sizeof(serv)) == SOCKET\_ERROR) // Связывает сокет с указанным IP-адресом и портом  throw SetErrorMsgText("Bind:", WSAGetLastError());  if (listen(sS, SOMAXCONN) == SOCKET\_ERROR) // Сервер начинает ожидать входящих соединений  throw SetErrorMsgText("Listen:", WSAGetLastError()); |

**Задание 6.** Добавьте в приложение **ServerT** вызовфункции **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.

|  |
| --- |
| SOCKET cS;  SOCKADDR\_IN clnt;  memset(&clnt, 0, sizeof(clnt));  int lclnt = sizeof(clnt);  if ((cS = accept(sS, (sockaddr\*)&clnt, &lclnt)) == INVALID\_SOCKET)  throw SetErrorMsgText("Accept:", WSAGetLastError()); |

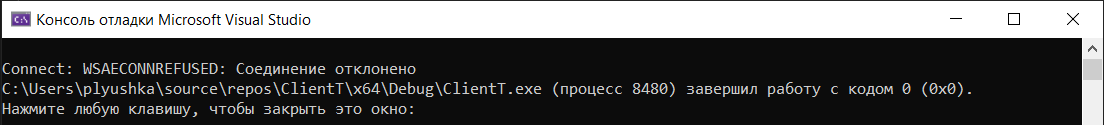


**7.3.4. Разработка клиентской части распределенного приложения**

**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.

|  |
| --- |
| SOCKADDR\_IN serv;  serv.sin\_family = AF\_INET;  serv.sin\_port = htons(2000);  if (inet\_pton(AF\_INET, "127.0.0.1", &serv.sin\_addr) <= 0)  throw SetErrorMsgText("Inet\_pton: ", WSAGetLastError());  if ((connect(sS, (sockaddr\*)&serv, sizeof(serv))) == SOCKET\_ERROR)  throw SetErrorMsgText("Connect: ", WSAGetLastError()); |



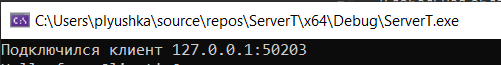
**7.3.5. Обмен данными между сервером и клиентом**

**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** завершилсябез ошибок.



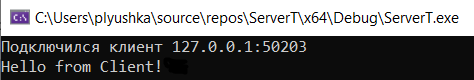
**Задание 10.** Доработайте программу сервера **ServerT** таким образом, чтобы после подсоединения клиента на экран консоли **ServerT** выводился IP-адрес и порт, подсоединившегося клиента. Необходимые значения находятся в структуре **SOCKADDR\_IN**, которая заполняется функцией **accept**. Используйте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

|  |
| --- |
| char clientIP[INET\_ADDRSTRLEN];  if (inet\_ntop(AF\_INET, &clnt.sin\_addr, clientIP, INET\_ADDRSTRLEN) == NULL)  // Преобразует IP-адрес клиента из числового формата в строку  {  cerr << "Ошибка преобразования IP-адреса клиента." << endl;  }  else  {  cout << "Подключился клиент " << clientIP << ":" << ntohs(clnt.sin\_port) << endl;  } |

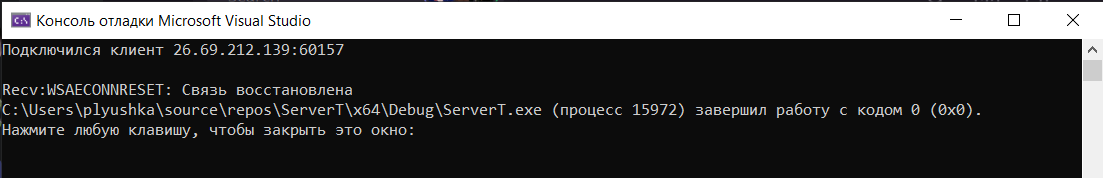


**Задание 11.** Добавьте в программу сервера **ServerT** вызов функции **recv,** а в программу клиента вызов функции **send.** Описание этих функций приводится в разделе 3.11 пособия. Перешлите текст ***Hello from Client*** от клиента серверу. Выведите полученный сервером текст на экран консоли. Обратите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

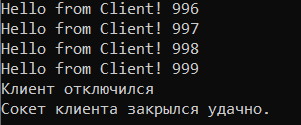
|  |
| --- |
| while (true) {  if ((libuf = recv(cS, ibuf, sizeof(ibuf), NULL)) == SOCKET\_ERROR)  throw SetErrorMsgText("Recv:", WSAGetLastError());  if (libuf == 0) {  cout << "Клиент отключился" << endl;  break;  }  cout << ibuf << endl;  if ((lobuf = send(cS, ibuf, libuf, NULL)) == SOCKET\_ERROR)  throw SetErrorMsgText("send:", WSAGetLastError());  } |



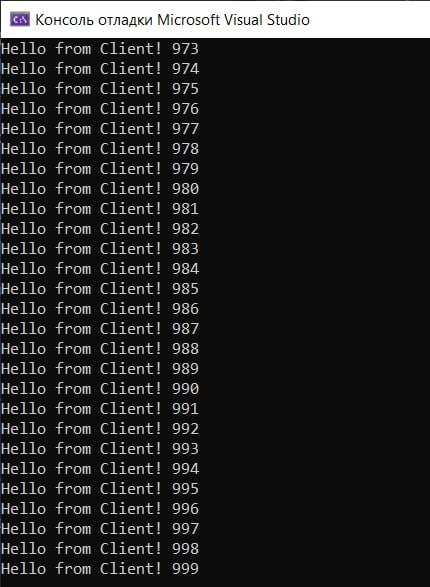
**Задание 12.** Установитепрограмму **ServerT** на другой компьютер локальной сети, а в программу **ClientT** внесите необходимые изменения, позволяющие ей установить связь с сервером. Убедитесь в работоспособности распределенного в локальной сети приложения **ClientT-ServеrT**.

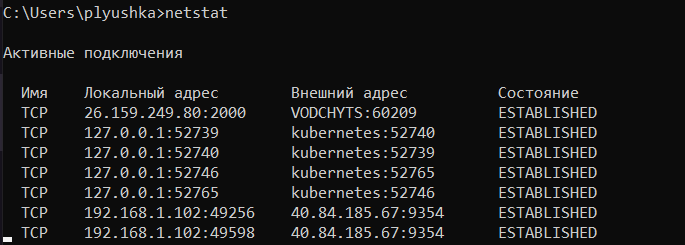


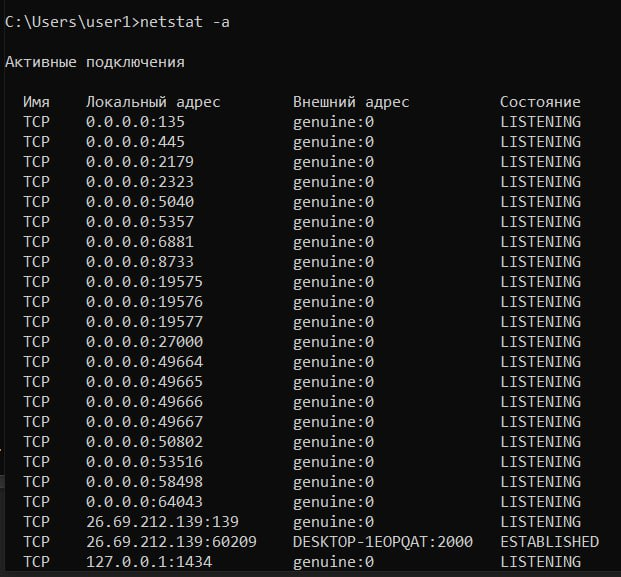
**Задание 13.** Внеситеизменения в программы  **ClientT** и **ServerT**, позволяющие 1000 раз передать сообщение типа ***Hello from Client xxx*** (***xxx*** *–* номер сообщения) от клиента серверу. Убедитесь в работоспособности в сети.

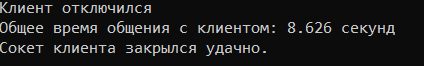


**Задание 14.**  Доработайте программы  **ClientT** и **ServerT** таким образом, чтобы программа **ServerT**  принимала последовательность сообщений вида ***Hello from Client xxx*** от программы **ClientT** и отправляла их без изменения обратно программе **ClientT**. Программа **ClientT,** получив вернувшееся 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.

