Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Программирование сетевых приложений

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Некрасова А.П.

Минск 2024

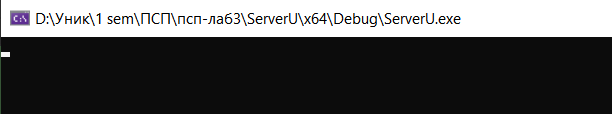
**Лабораторная работа № 3. Обмен данными без установки соединения**

**Цель:** 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).

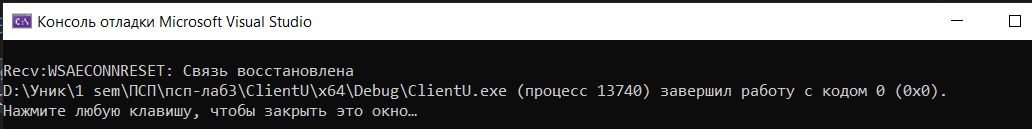
**Задание 1**. Ознакомьтесь со схемой взаимодействия процессов без установки соединения в распределенном приложении, приведенной разделе 3.4 пособия (рисунок 3.4.1). Определите основные отличия этой схемы от схемы взаимодействия процессов с установкой соединения. Разработайте программу **ServerU**, реализующую 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.

|  |
| --- |
| int main()  {  setlocale(LC\_ALL, "RUS");  SOCKET sS;  WSADATA wsaData;  try  {  if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0)  throw SetErrorMsgText("Startup:", WSAGetLastError());  if ((sS = socket(AF\_INET, SOCK\_DGRAM, NULL)) == INVALID\_SOCKET)  throw SetErrorMsgText("Socket:", WSAGetLastError());  int bufferSize = 512;  if (setsockopt(sS, SOL\_SOCKET, SO\_RCVBUF, (char\*)&bufferSize, sizeof(bufferSize)) == SOCKET\_ERROR) {  throw SetErrorMsgText("Byffer error:", WSAGetLastError());  }  SOCKADDR\_IN serv;  serv.sin\_family = AF\_INET;  serv.sin\_port = htons(2000);  serv.sin\_addr.s\_addr = INADDR\_ANY;  if (bind(sS, (LPSOCKADDR)&serv, sizeof(serv)) == SOCKET\_ERROR)  throw SetErrorMsgText("Bind:", WSAGetLastError());  SOCKADDR\_IN clnt;  while (true) {  int lc = sizeof(clnt);  char ibuf[50];  int lb = 0, lobuf = 0;  memset(ibuf, 0, sizeof(ibuf));  if (lb = recvfrom(sS, ibuf, sizeof(ibuf), NULL, (sockaddr\*)&clnt, &lc) == SOCKET\_ERROR)  throw SetErrorMsgText("Recv:", WSAGetLastError());  cout << ibuf << endl;  if ((lobuf = sendto(sS, "Сообщение принято", strlen(ibuf) + 1, NULL, (sockaddr\*)&clnt, sizeof(clnt))) == SOCKET\_ERROR)  throw SetErrorMsgText("Send:", WSAGetLastError());  Sleep(200);  }  if (closesocket(sS) == SOCKET\_ERROR)  throw SetErrorMsgText("Closesocket:", WSAGetLastError());  if (WSACleanup() == SOCKET\_ERROR)  throw SetErrorMsgText("Cleanup:", WSAGetLastError());  }  catch (string errorMsgText)  {  cout << endl << errorMsgText;  }  return 0;  } |

**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.



**Задание 3**. Создайте новое С++ -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.

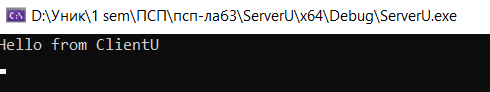


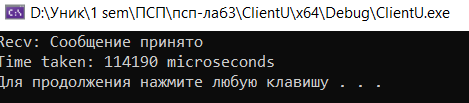
Ошибка возникает из-за попытки отправить данные на недоступный адрес.

**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.

|  |
| --- |
| int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  SOCKET sS;  WSADATA wsaData;  try  {  if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0)  throw SetErrorMsgText("Startup:", WSAGetLastError());  if ((sS = socket(AF\_INET, SOCK\_DGRAM, NULL)) == INVALID\_SOCKET)  throw SetErrorMsgText("Socket:", WSAGetLastError());  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());  string msg;  int lc = sizeof(serv);  char ibuf[50];  int lobuf = 0, libuf = 0;  auto start = chrono::high\_resolution\_clock::now();  memset(ibuf, 0, sizeof(ibuf));  msg = "Hello from ClientU";  if ((lobuf = sendto(sS, msg.c\_str(), strlen(msg.c\_str()) + 1, NULL, (sockaddr\*)&serv, sizeof(serv))) == SOCKET\_ERROR)  throw SetErrorMsgText("Send:", WSAGetLastError());  Sleep(100);  if ((libuf = recvfrom(sS, ibuf, sizeof(ibuf), NULL, (sockaddr\*)&serv, &lc)) == SOCKET\_ERROR) {  int errCode = WSAGetLastError();  if (errCode == WSAEWOULDBLOCK) {  cout << "Сервер не отвечает, попробуйте позже." << endl;  }  else {  throw SetErrorMsgText("Recv:", errCode);  }  }  cout << "Recv: " << ibuf << endl;  auto end = chrono::high\_resolution\_clock::now();  auto duration = chrono::duration\_cast<chrono::microseconds>(end - start);  cout << "Time taken: " << duration.count() << " microseconds" << endl;  if (closesocket(sS) == SOCKET\_ERROR)  throw SetErrorMsgText("Closesocket:", WSAGetLastError());  if (WSACleanup() == SOCKET\_ERROR)  throw SetErrorMsgText("Cleanup:", WSAGetLastError());  system("pause");  }  catch (string errorMsgText)  {  cout << endl << errorMsgText;  }  return 0;  } |

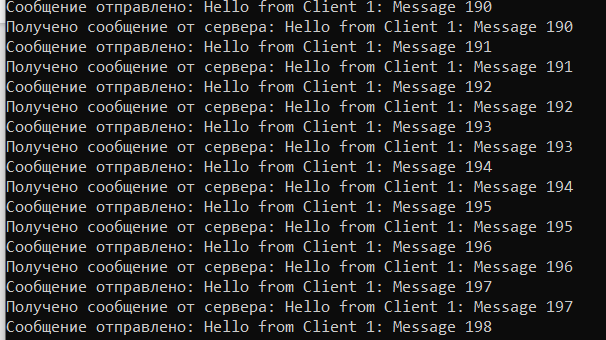
**Задание 5**. Реализуйте блоки 4 в обеих программах. Перешлите полученное сервером сообщение обратно в адрес клиента и убедитесь, что сообщение получено.



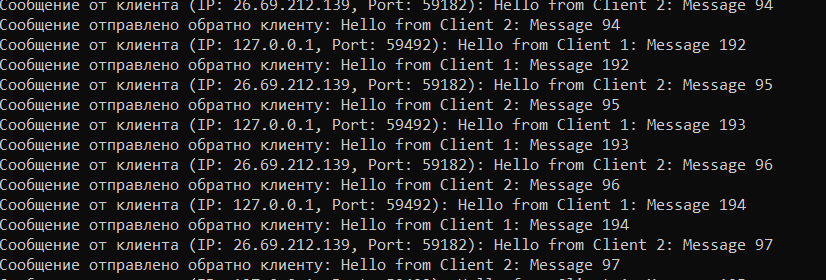


**Задание 6**. Внесите необходимые изменения в программу **ClientU** для того, чтобы программы можно было бы расположить на разных компьютерах локальной сети. Убедитесь в работоспособности приложения.

**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 **ClientТ**(разработанный в практической работе № 2). Объясните полученный результат.

