Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Программирование сетевых приложений

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

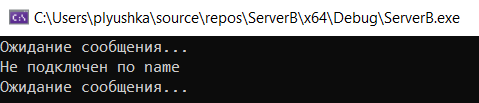
Преподаватель: Некрасова А.П.

Минск 2024

**Задание 1.** Разработайте функцию **GetRequestFromClient.** 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**, то функция **GetRequestFromClient** должна завершиться с кодом возврата **false.**

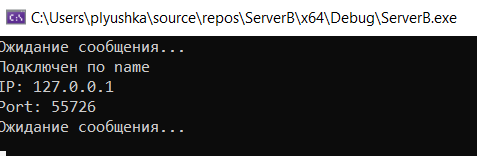
|  |
| --- |
| bool GetRequestFromClient(char\* name, short port, struct sockaddr\* from, int\* flen)  {  SOCKET sS;  sS = socket(AF\_INET, SOCK\_DGRAM, NULL);  if (sS == INVALID\_SOCKET) {  cout << "Создание сокета не удалось: " << WSAGetLastError() << endl;  }  SOCKADDR\_IN server\_addr;  server\_addr.sin\_family = AF\_INET;  server\_addr.sin\_port = htons(port);  server\_addr.sin\_addr.s\_addr = INADDR\_ANY;  if (bind(sS, (SOCKADDR\*)&server\_addr, sizeof(server\_addr)) == SOCKET\_ERROR) {  cout << "Привязка не удалась: " << WSAGetLastError() << endl;  }  int recv\_len;  char buffer[50] = "";  memset(buffer, 0, sizeof(buffer));  cout << "Ожидание сообщения..." << endl;  while (true)  {  if ((recv\_len = recvfrom(sS, buffer, sizeof(buffer), NULL, from, flen)) == SOCKET\_ERROR)  {  if (WSAGetLastError() == WSAETIMEDOUT) {  closesocket(sS);  return false;  }  else {  closesocket(sS);  throw SetErrorMsgText("Ошибка: ", WSAGetLastError());  }  }  closesocket(sS);  if (strcmp(buffer, name) == 0)  return true;  else  return false;  }  } |

Создайте новое приложение **ServerB**, вызывающее функцию **GetRequestFromClient**.Пусть позывной сервера будет ***Hello***. Запустите приложение **ServerB** иубедитесь, 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.

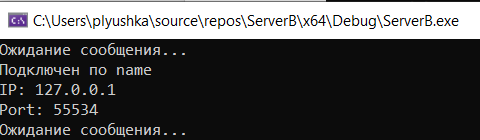
|  |
| --- |
| for (int i = 1; i <= numMessages; ++i) {  strcpy(bto, "Hello"); // Отправляемая строка  if ((send\_len = sendto(cC, bto, strlen(bto), 0, (sockaddr\*)&servAddr, sizeof(servAddr))) == SOCKET\_ERROR) {  throw SetErrorMsgText("Ошибка при отправке сообщения (sendto): ", WSAGetLastError());  }  cout << "Сообщение отправлено: " << bto << endl;  Sleep(1);  sockaddr\_in from;  int fromlen = sizeof(from);  int recv\_len;  // Получение ответа от сервера  if ((recv\_len = recvfrom(cC, bfrom, sizeof(bfrom), 0, (sockaddr\*)&from, &fromlen)) == SOCKET\_ERROR) {  throw SetErrorMsgText("Ошибка при получении сообщения (recvfrom): ", WSAGetLastError());  }  bfrom[recv\_len] = '\0';  cout << "Получено сообщение от сервера: " << bfrom << endl;  Sleep(1);  } |



**Задание 2.** Разработайте функцию **PutAnswerToClient**. 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**.

|  |
| --- |
| bool PutAnswerToClient(char\* name, const struct sockaddr\_in\* to, int\* lto)  {  SOCKET sS = socket(AF\_INET, SOCK\_DGRAM, NULL);  if (sS == INVALID\_SOCKET) {  throw SetErrorMsgText("socket: ", WSAGetLastError());  }  int sent\_len;  if ((sent\_len = sendto(sS, name, strlen(name), 0, (sockaddr\*)to, \*lto)) == SOCKET\_ERROR) {  int errorCode = WSAGetLastError();  closesocket(sS);  cout << "sendto failed with error: " << errorCode << " (" << GetErrorMsgText(errorCode) << ")" << endl;  throw SetErrorMsgText("sendto: ", errorCode);  }  if (closesocket(sS) == SOCKET\_ERROR) {  throw SetErrorMsgText("closesocket: ", WSAGetLastError());  }  return true;  } |

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

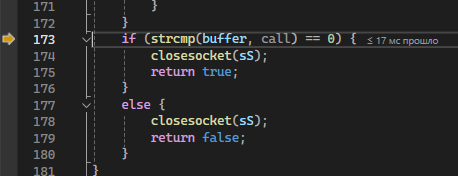


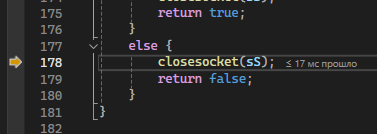
**7.5.4. Разработка клиентской части распределенного приложения**

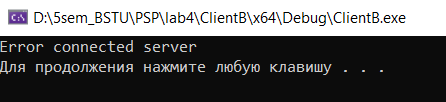
**Задание 4.** Разработайте функцию **GetServer**. Функция предназначена для отправки широковещательного запроса в локальную сеть (всем компьютерам сегмента локальной сети) с позывным сервера. Предполагается, что на одном (или на нескольких) компьютере сети есть сервер **ServerB**,которыйпросушивает 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**. Любойдругой аварийный код завершения должен приводить к исключительной ситуации (оператор **throw**), соответствующей функциям обработки ошибок разработанных в практическом занятии № 2.

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.

|  |
| --- |
| bool GetServer(char\* call, short port, struct sockaddr\* from, int\* flen) {  SOCKET sS;  sS = socket(AF\_INET, SOCK\_DGRAM, NULL);  if (sS == INVALID\_SOCKET) {  cout << "Socket creation failed: " << WSAGetLastError() << endl;  }  SOCKADDR\_IN broadcast\_addr;  broadcast\_addr.sin\_family = AF\_INET;  broadcast\_addr.sin\_port = htons(port);  broadcast\_addr.sin\_addr.s\_addr = INADDR\_BROADCAST;  int broadcast = 1;  int sent\_len;  int recv\_len;  char buffer[50];  if (setsockopt(sS, SOL\_SOCKET, SO\_BROADCAST, (char\*)&broadcast, sizeof(broadcast)) == SOCKET\_ERROR) {  throw SetErrorMsgText("Opt: ", WSAGetLastError());  closesocket(sS);  }  int timeout = 20000;  if (setsockopt(sS, SOL\_SOCKET, SO\_RCVTIMEO, (char\*)&timeout, sizeof(timeout)) == SOCKET\_ERROR) {  closesocket(sS);  throw SetErrorMsgText("Setsockopt (SO\_RCVTIMEO): ", WSAGetLastError());  }  if (sent\_len = sendto(sS, call, strlen(call), NULL, (sockaddr\*)&broadcast\_addr, sizeof(broadcast\_addr)) == SOCKET\_ERROR) {  closesocket(sS);  throw SetErrorMsgText("Sendto: ", WSAGetLastError());  }  memset(buffer, 0, sizeof(buffer));  if (recv\_len = recvfrom(sS, buffer, sizeof(buffer) - 1, NULL, from, flen) == SOCKET\_ERROR)  {  closesocket(sS);  if (WSAGetLastError() == WSAETIMEDOUT) {  return false;  }  else {  throw SetErrorMsgText("Ошибка: ", WSAGetLastError());  }  }  if (strcmp(buffer, call) == 0) {  closesocket(sS);  return true;  }  else {  closesocket(sS);  return false;  }  } |

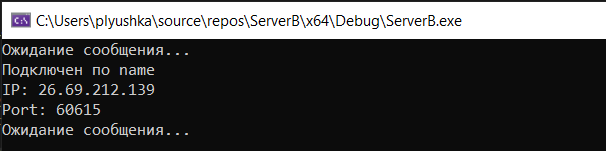
****

****

****

**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.

