|  |  |
| --- | --- |
| #include <winsock2.h>  #include <WS2tcpip.h>  #include <bits/stdc++.h>  // need link with Ws2\_32.lib  #pragma comment(lib, "Ws2\_32.lib")  using namespace std;  int main()  {      WORD wVersion = MAKEWORD(2, 2); // Specify the version of Winsock you want to use                                      // 2.2 -> x=2, y=2      WSADATA wsaData;      if (WSAStartup(wVersion, &wsaData) != 0) // Phiên bản Winsock cần tải, Con trỏ trỏ đến cấu trúc LPWSADATA      {          cout << "WSAStartup failed with error" << GetLastError() << endl;          return 1;      }      cout << "WSAStartup completed." << endl;      char ip[] = "127.0.0.";      char stCodeString[10];      unsigned int stCode = 106200228;      unsigned int num = (stCode % 255) + 1;      sprintf(stCodeString, "%d", num);      strcat(ip, stCodeString);      cout << "IP address: " << ip << endl;      // inet\_addr      unsigned long ip\_addr1 = inet\_addr(ip);      if (ip\_addr1 == INADDR\_NONE)      {          printf("Can't convert IP by inet\_addr\n");      }      else      {          cout << "Convert inet\_addr completed:" << ip\_addr1 << endl;      }      // inet\_pton      unsigned long ip\_addr2;      if (inet\_pton(AF\_INET, ip, &ip\_addr2) == 1)      {          cout << "Convert inet\_pton completed:" << ip\_addr2 << endl;      }      else      {          cout << "Can't convert IP by inet\_pton\n";      }      // Call WSACleanup when application finishes      if (WSACleanup() != 0)          cout << "Clean failed with error." << GetLastError() << endl;      return 0;  } | Bài 1: |

Bài 2: TCP: xử lý chuỗi

|  |  |
| --- | --- |
| Server | Client |
| #include <winsock2.h>  #include <ws2tcpip.h>  #include <bits/stdc++.h>  using namespace std;  #define SERVER\_PORT 5500  #define SERVER\_ADDR "127.0.0.1"  #define BUFF\_MAXSIZE 1024  #pragma comment(lib, "Ws2\_32.lib")  string process(string input)  {      string output = "";      for (char c : input)          if (isalpha(c))              output += tolower(c);      return (output == "exit" || output == "quit") ? "good bye" : output;  }  int main()  {      WORD wVersion = MAKEWORD(2, 2);      WSADATA wsaData;      if (WSAStartup(wVersion, &wsaData) != 0)      {          cout << "WSAStartup failed with error " << GetLastError() << endl;          return 1;      }      cout << "WSAStartup completed." << endl;      SOCKET listenSock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);      if (listenSock == INVALID\_SOCKET)      {          cout << "Creating socket failed with code " << WSAGetLastError() << endl;          WSACleanup(); // Clean up before returning          return 1;      }      cout << "Creating socket completed successfully.\n";      sockaddr\_in tcpServerAddr;      tcpServerAddr.sin\_family = AF\_INET;      tcpServerAddr.sin\_port = htons(SERVER\_PORT);      tcpServerAddr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR);      if (bind(listenSock, (sockaddr \*)&tcpServerAddr, sizeof(tcpServerAddr)) == SOCKET\_ERROR)      {          cout << "Bind API failed with code " << WSAGetLastError() << endl;          closesocket(listenSock); // Clean up before returning          WSACleanup();          return 1;      }      cout << "Bind API completed successfully.\n";      if (listen(listenSock, 5) == SOCKET\_ERROR)      {          cout << "Listen failed with code " << WSAGetLastError() << endl;          closesocket(listenSock); // Clean up before returning          WSACleanup();          return 1;      }      cout << "Server is listening for requests..." << endl;      sockaddr\_in clientAddr;      char buff[BUFF\_MAXSIZE], clientIP[INET\_ADDRSTRLEN];      int ret, clientAddrLen = sizeof(clientAddr), clientPort;      SOCKET NewConnection = accept(listenSock, (sockaddr \*)&clientAddr, &clientAddrLen);      if (NewConnection == INVALID\_SOCKET)      {          cout << "Connection failed with code: " << WSAGetLastError() << endl;          closesocket(listenSock); // Clean up before returning          WSACleanup();          return 1;      }      inet\_ntop(AF\_INET, &clientAddr.sin\_addr, clientIP, sizeof(clientIP));      clientPort = ntohs(clientAddr.sin\_port);      cout << "Connection is established: IP = " << clientIP << " at port = " << clientPort << endl;      while (1)      {          ret = recv(NewConnection, buff, BUFF\_MAXSIZE, 0);          if (ret == SOCKET\_ERROR)          {              cout << "Error with code " << WSAGetLastError() << endl;              break;          }          else if (ret == 0)          {              cout << "Client disconnected." << endl;              break;          }          else          {              buff[ret] = '\0'; // Null-terminate the received data              cout << "Received message from client " << clientIP << ":" << clientPort << ": " << buff << endl;              string buffStr(buff);              buffStr = process(buffStr);              const char \*message = buffStr.c\_str();              ret = send(NewConnection, message, strlen(message), 0);              if (ret == SOCKET\_ERROR)              {                  cout << "Error with code " << WSAGetLastError() << endl;                  break;              }          }      }      shutdown(NewConnection, SD\_BOTH);      closesocket(NewConnection);      closesocket(listenSock);      WSACleanup();      return 0;  } | #include <winsock2.h>  #include <ws2tcpip.h>  #include <iostream>  using namespace std;  #define SERVER\_PORT 5500  #define SERVER\_ADDR "127.0.0.1"  int main()  {      WORD wVersionRequested = MAKEWORD(2, 2);      WSADATA wSadata;      if (WSAStartup(wVersionRequested, &wSadata) != 0)      {          cout << "WSAStartup failed: \n";          return 0;      }      cout << "WSAStartup completed." << endl;      sockaddr\_in serverAddr;      serverAddr.sin\_family = AF\_INET;      serverAddr.sin\_port = htons(SERVER\_PORT);      serverAddr.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_ADDR);      SOCKET client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);      if (client == INVALID\_SOCKET)      {          cout << "Creating socket failed with code " << WSAGetLastError() << endl;          WSACleanup(); // Clean up before returning          return 1;      }      cout << "Creating socket completed successfully.\n";      if (connect(client, (sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)      {          cout << "Connection failed with code " << WSAGetLastError() << endl;          return 0;      }      cout << "Connection completed successfully." << endl;      // STEP 3: Send data (example)      const char \*message = "eXit2a@";      int bytesSent = send(client, message, strlen(message), 0);      if (bytesSent == SOCKET\_ERROR)      {          cout << "Send failed with error: " << WSAGetLastError() << endl;      }      // STEP 4: Receive data (example)      char buffer[1024];      int bytesReceived = recv(client, buffer, sizeof(buffer), 0);      if (bytesReceived == SOCKET\_ERROR)      {          cout << "Receive failed with error: " << WSAGetLastError() << endl;      }      else      {          buffer[bytesReceived] = '\0';          cout << "Received: " << buffer << endl;      }      // STEP 5: Shutdown the connection (if needed)      shutdown(client, SD\_BOTH); // SD\_BOTH closes both send and receive      // STEP 6: Close sockets and clean up      closesocket(client);      WSACleanup();      return 0;  } |

Bài 3: UDP – chuyển DNS thành IP

|  |  |
| --- | --- |
| Server | Client |
| // cd UDP  // g++ UDPServer.cpp -o UDPServer.exe -lws2\_32  // .\UDPServer.exe 8080  #include <bits/stdc++.h>  #include <winsock2.h> // Include the Windows Sockets API header  #include <stdio.h>  #include <ws2tcpip.h>  using namespace std;  string process(char \*hostname)  {      string st = "";      struct addrinfo hints, \*result, \*rp;      ZeroMemory(&hints, sizeof(hints));      // hints.ai\_family = AF\_UNSPEC;     // Allow IPv4 or IPv6      hints.ai\_family = AF\_INET;       // Allow IPv4      hints.ai\_socktype = SOCK\_STREAM; // Use TCP      int status = getaddrinfo(hostname, NULL, &hints, &result);      if (status != 0)      {          fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(status));          WSACleanup();          return "Not found information";      }      for (rp = result; rp != NULL; rp = rp->ai\_next)      {          void \*addr;          char ipstr[INET6\_ADDRSTRLEN];          if (rp->ai\_family == AF\_INET)          {              struct sockaddr\_in \*ipv4 = (struct sockaddr\_in \*)rp->ai\_addr;              addr = &(ipv4->sin\_addr);          }          else          {              struct sockaddr\_in6 \*ipv6 = (struct sockaddr\_in6 \*)rp->ai\_addr;              addr = &(ipv6->sin6\_addr);          }          inet\_ntop(rp->ai\_family, addr, ipstr, sizeof(ipstr));          string s(ipstr);          st += "\n" + s;      }      freeaddrinfo(result);      return st;  }  int main(int argc, char \*argv[])  {      if (argc != 2)      {          cerr << "Usage: serverUDP.exe <port\_number>" << endl;          return 1;      }      int port = atoi(argv[1]);      WSADATA wsaData;      if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)      {          cerr << "WSAStartup failed!" << endl;          return 1;      }      cout << "WSAStartup successfully!" << endl;      SOCKET serverSocket = socket(AF\_INET, SOCK\_DGRAM, 0);      if (serverSocket == INVALID\_SOCKET)      {          cerr << "Socket creation failed!" << endl;          WSACleanup();          return 1;      }      cout << "Socket creates successfully!" << endl;      sockaddr\_in serverAddr;      serverAddr.sin\_family = AF\_INET;      serverAddr.sin\_addr.s\_addr = INADDR\_ANY;      serverAddr.sin\_port = htons(port);      if (bind(serverSocket, (sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)      {          cerr << "Binding failed!" << endl;          closesocket(serverSocket);          WSACleanup();          return 1;      }      cout << "Binding successfully!" << endl;      char buffer[1024];      sockaddr\_in clientAddr;      int clientAddrLen = sizeof(clientAddr);      while (true)      {          int bytesReceived = recvfrom(serverSocket, buffer, sizeof(buffer), 0, (sockaddr \*)&clientAddr, &clientAddrLen);          if (bytesReceived == SOCKET\_ERROR)          {              cerr << "Error in recvfrom()" << endl;              break;          }          buffer[bytesReceived] = '\0';          string st = process(buffer);          const char \*message = st.c\_str();          sendto(serverSocket, message, strlen(message), 0, (sockaddr \*)&clientAddr, clientAddrLen);      }      closesocket(serverSocket);      WSACleanup();      return 0;  } | // g++ UDPclient.cpp -o UDPclient.exe -lws2\_32  // .\UDPclient.exe 127.0.0.1 8080 quora.com  #include <iostream>  #include <winsock2.h>  using namespace std;  int main(int argc, char \*argv[])  {      if (argc != 4)      {          cerr << "Usage: clientUDP.exe <server\_ip> <port\_number> <domain\_name>" << endl;          return 1;      }      const char \*serverIP = argv[1];      int port = atoi(argv[2]);      const char \*domainName = argv[3];      WSADATA wsaData;      if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)      {          cerr << "WSAStartup failed!" << endl;          return 1;      }      cout << "WSAStartup successfully!" << endl;      SOCKET clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);      if (clientSocket == INVALID\_SOCKET)      {          cerr << "Socket creation failed!" << endl;          WSACleanup();          return 1;      }      cout << "Socket creates successfully!" << endl;      sockaddr\_in serverAddr;      serverAddr.sin\_family = AF\_INET;      serverAddr.sin\_port = htons(port);      serverAddr.sin\_addr.s\_addr = inet\_addr(serverIP);      if (sendto(clientSocket, domainName, strlen(domainName), 0, (sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)      {          cerr << "Error in sendto()" << endl;          closesocket(clientSocket);          WSACleanup();          return 1;      }      char buffer[1024];      sockaddr\_in serverResponse;      int serverResponseLen = sizeof(serverResponse);      int bytesReceived = recvfrom(clientSocket, buffer, sizeof(buffer), 0, (sockaddr \*)&serverResponse, &serverResponseLen);      if (bytesReceived == SOCKET\_ERROR)      {          cerr << "Error in recvfrom()" << endl;      }      else      {          buffer[bytesReceived] = '\0';          cout << "Resolved IP Address: " << buffer << endl;      }      closesocket(clientSocket);      WSACleanup();      return 0;  } |

Bài 4: Login TCP

|  |  |
| --- | --- |
| Server | Client |
| #include <winsock2.h>  #include <ws2tcpip.h>  #include <bits/stdc++.h>  using namespace std;  #define SERVER\_PORT 5500  #define SERVER\_ADDR "127.0.0.1"  #define BUFF\_MAXSIZE 1024  #pragma comment(lib, "Ws2\_32.lib")  struct UserData  {  string id;  string password;  bool active;  bool login;  };  map<int, string> loginWith;  vector<UserData> userData;  vector<UserData> readUserCSV(string filename)  {  vector<UserData> userDataRead;  ifstream file(filename);  if (!file.is\_open())  cerr << "Error: Unable to open file " << filename << endl;  string line;  getline(file, line);  while (getline(file, line))  {  stringstream lineStream(line);  string cell;  UserData user;  // Read id, password, active, and login  if (getline(lineStream, user.id, ',') &&  getline(lineStream, user.password, ',') &&  getline(lineStream, cell, ',') &&  stringstream(cell) >> user.active &&  getline(lineStream, cell, ',') &&  stringstream(cell) >> user.login)  {  userDataRead.push\_back(user);  }  else  {  cerr << "Error: Invalid data format in line" << endl;  }  }  file.close();  return userDataRead;  }  void updateUserCSV(string filename)  {  ofstream file(filename, ios::out | ios::trunc);  if (!file.is\_open())  {  cerr << "Error: Unable to create/open file " << filename << endl;  return;  }  file << "ID,Password,Active,Login\n";  for (const UserData &user : userData)  {  file << user.id << "," << user.password << "," << user.active << "," << user.login << "\n";  }  file.close();  }  string process(const string &input, int clientPort)  {  userData = readUserCSV("database.csv"); // Update data from database  istringstream iss(input);  string request, id, password, temp;  iss >> request >> id >> password >> temp;  if (request == "login" && id != "" && password != "" && temp == "")  {  if (loginWith[clientPort] != "")  return "You logged in";  for (UserData &user : userData)  {  if (user.id == id)  {  if (user.password == password)  {  // Bỏ cmt nếu muốn 1 tài khoản chỉ vào được 1 client duy nhất (Code hiện tại: 1 tài khoản vào được nhiều client như Fb)  // if (user.login == 0)  // {  loginWith[clientPort] = id;  user.login = 1;  updateUserCSV("database.csv");  return "Login successfully";  // }  // else  // return "You logged at another client";  }  else  return "Wrong password";  }  }  return "Id does not exist";  }  else if (request == "logout" && id == "") // Check if redundant params  {  if (loginWith[clientPort] == "")  return "Please login";  for (auto &user : userData)  if (user.id == loginWith[clientPort])  user.login = 0;  loginWith[clientPort] = "";  updateUserCSV("database.csv");  return "Logout successfully";  }  else  return "Wrong message format";  }  int main()  {  WORD wVersion = MAKEWORD(2, 2);  WSADATA wsaData;  if (WSAStartup(wVersion, &wsaData) != 0)  {  cout << "WSAStartup failed with error " << GetLastError() << endl;  return 1;  }  cout << "WSAStartup completed." << endl;  SOCKET listenSock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (listenSock == INVALID\_SOCKET)  {  cout << "Creating socket failed with code " << WSAGetLastError() << endl;  WSACleanup(); // Clean up before returning  return 1;  }  cout << "Creating socket completed successfully.\n";  sockaddr\_in tcpServerAddr;  tcpServerAddr.sin\_family = AF\_INET;  tcpServerAddr.sin\_port = htons(SERVER\_PORT);  tcpServerAddr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR);  if (bind(listenSock, (sockaddr \*)&tcpServerAddr, sizeof(tcpServerAddr)) == SOCKET\_ERROR)  {  cout << "Bind API failed with code " << WSAGetLastError() << endl;  closesocket(listenSock); // Clean up before returning  WSACleanup();  return 1;  }  cout << "Bind API completed successfully.\n";  if (listen(listenSock, 5) == SOCKET\_ERROR)  {  cout << "Listen failed with code " << WSAGetLastError() << endl;  closesocket(listenSock); // Clean up before returning  WSACleanup();  return 1;  }  cout << "Server is listening for requests..." << endl;  sockaddr\_in clientAddr;  char buff[BUFF\_MAXSIZE], clientIP[INET\_ADDRSTRLEN];  int ret, clientAddrLen = sizeof(clientAddr), clientPort;  SOCKET NewConnection = accept(listenSock, (sockaddr \*)&clientAddr, &clientAddrLen);  if (NewConnection == INVALID\_SOCKET)  {  cout << "Connection failed with code: " << WSAGetLastError() << endl;  closesocket(listenSock); // Clean up before returning  WSACleanup();  return 1;  }  inet\_ntop(AF\_INET, &clientAddr.sin\_addr, clientIP, sizeof(clientIP));  clientPort = ntohs(clientAddr.sin\_port);  cout << "Connection is established: IP = " << clientIP << " at port = " << clientPort << endl;  while (1)  {  ret = recv(NewConnection, buff, BUFF\_MAXSIZE, 0);  if (ret == SOCKET\_ERROR)  {  cout << "Error with code " << WSAGetLastError() << endl;  break;  }  else if (ret == 0)  {  cout << "Client disconnected." << endl;  break;  }  else  {  buff[ret] = '\0'; // Null-terminate the received data  cout << "Received message from client " << clientIP << ":" << clientPort << ": " << buff << endl;  string buffStr(buff);  buffStr = process(buffStr, clientPort);  const char \*message = buffStr.c\_str();  ret = send(NewConnection, message, strlen(message), 0);  if (ret == SOCKET\_ERROR)  {  cout << "Error with code " << WSAGetLastError() << endl;  break;  }  }  }  shutdown(NewConnection, SD\_BOTH);  closesocket(NewConnection);  closesocket(listenSock);  WSACleanup();  return 0;  } | #include <winsock2.h>  #include <ws2tcpip.h>  #include <iostream>  using namespace std;  #define SERVER\_PORT 5500  #define SERVER\_ADDR "127.0.0.1"  int main()  {      WORD wVersionRequested = MAKEWORD(2, 2);      WSADATA wSadata;      if (WSAStartup(wVersionRequested, &wSadata) != 0)      {          cout << "WSAStartup failed: \n";          return 0;      }      cout << "WSAStartup completed." << endl;      sockaddr\_in serverAddr;      serverAddr.sin\_family = AF\_INET;      serverAddr.sin\_port = htons(SERVER\_PORT);      serverAddr.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER\_ADDR);      SOCKET client = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);      if (client == INVALID\_SOCKET)      {          cout << "Creating socket failed with code " << WSAGetLastError() << endl;          WSACleanup(); // Clean up before returning          return 1;      }      cout << "Creating socket completed successfully.\n";      if (connect(client, (sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)      {          cout << "Connection failed with code " << WSAGetLastError() << endl;          return 0;      }      cout << "Connection completed successfully." << endl;      // STEP 3: Send data (example)      string messageSend;      cout << "To Login: login <id> <password>\n To Logout: logout\n";      while (1)      {          cout << "(Client) -->";          getline(cin, messageSend);          const char \*message = messageSend.c\_str();          int bytesSent = send(client, message, strlen(message), 0);          if (bytesSent == SOCKET\_ERROR)          {              cout << "Send failed with error: " << WSAGetLastError() << endl;          }          // STEP 4: Receive data (example)          char buffer[1024];          int bytesReceived = recv(client, buffer, sizeof(buffer), 0);          if (bytesReceived == SOCKET\_ERROR)          {              cout << "Receive failed with error: " << WSAGetLastError() << endl;          }          else          {              buffer[bytesReceived] = '\0';              cout << "(Server) " << buffer << endl;          }      }      // STEP 5: Shutdown the connection (if needed)      shutdown(client, SD\_BOTH); // SD\_BOTH closes both send and receive      // STEP 6: Close sockets and clean up      closesocket(client);      WSACleanup();      return 0;  } |

Bài 5: Login UDP

|  |  |
| --- | --- |
| Server | Client |
| #include <winsock2.h>  #include <ws2tcpip.h>  #include <bits/stdc++.h>  using namespace std;  #define SERVER\_PORT 5500  #define SERVER\_ADDR "127.0.0.1"  #define BUFF\_MAXSIZE 1024  #pragma comment(lib, "Ws2\_32.lib")  struct UserData  {      string id;      string password;      bool active;      bool login;  };  map<int, string> loginWith;  vector<UserData> userData;  vector<UserData> readUserCSV(string filename)  {      vector<UserData> userDataRead;      ifstream file(filename);      if (!file.is\_open())          cerr << "Error: Unable to open file " << filename << endl;      string line;      getline(file, line);      while (getline(file, line))      {          stringstream lineStream(line);          string cell;          UserData user;          // Read id, password, active, and login          if (getline(lineStream, user.id, ',') &&              getline(lineStream, user.password, ',') &&              getline(lineStream, cell, ',') &&              stringstream(cell) >> user.active &&              getline(lineStream, cell, ',') &&              stringstream(cell) >> user.login)          {              userDataRead.push\_back(user);          }          else          {              cerr << "Error: Invalid data format in line" << endl;          }      }      file.close();      return userDataRead;  }  void updateUserCSV(string filename)  {      ofstream file(filename, ios::out | ios::trunc);      if (!file.is\_open())      {          cerr << "Error: Unable to create/open file " << filename << endl;          return;      }      file << "ID,Password,Active,Login\n";      for (const UserData &user : userData)      {          file << user.id << "," << user.password << "," << user.active << "," << user.login << "\n";      }      file.close();  }  string process(const string &input, int clientPort)  {      userData = readUserCSV("database.csv"); // Update data from database      istringstream iss(input);      string request, id, password, temp;      iss >> request >> id >> password >> temp;      if (request == "login" && id != "" && password != "" && temp == "")      {          if (loginWith[clientPort] != "")              return "You logged in";          for (UserData &user : userData) {              if (user.id == id)              {                  if (user.password == password)                  {                      // Bỏ cmt nếu muốn 1 tài khoản chỉ vào được 1 client duy nhất (Code hiện tại: 1 tài khoản vào được nhiều client như Fb)                      // if (user.login == 0)                      // {                      loginWith[clientPort] = id;                      user.login = 1;                      updateUserCSV("database.csv");                      return "Login successfully";                      // }                      // else                      //     return "You logged at another client";                  }                  else                      return "Wrong password";              }          }          return "Id does not exist";      }      else if (request == "logout" && id == "") // Check if redundant params      {          if (loginWith[clientPort] == "")              return "Please login";          for (auto &user : userData)              if (user.id == loginWith[clientPort])                  user.login = 0;          loginWith[clientPort] = "";          updateUserCSV("database.csv");          return "Logout successfully";      }      else          return "Wrong message format";  }  int main() {      WORD wVersion = MAKEWORD(2, 2);      WSADATA wsaData;      if (WSAStartup(wVersion, &wsaData) != 0)      {          cout << "WSAStartup failed with error " << GetLastError() << endl;          return 1;      }      cout << "WSAStartup completed." << endl;      SOCKET udpServerSock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);      if (udpServerSock == INVALID\_SOCKET) {          cout << "Creating socket failed with code " << WSAGetLastError() << endl;          WSACleanup();          return 1;      }      cout << "Creating socket completed successfully.\n";      sockaddr\_in udpServerAddr;      udpServerAddr.sin\_family = AF\_INET;      udpServerAddr.sin\_port = htons(SERVER\_PORT);      udpServerAddr.sin\_addr.s\_addr = inet\_addr(SERVER\_ADDR);      if (bind(udpServerSock, (sockaddr \*)&udpServerAddr, sizeof(udpServerAddr)) == SOCKET\_ERROR) {          cout << "Bind API failed with code " << WSAGetLastError() << endl;          closesocket(udpServerSock);          WSACleanup();          return 1;      }      cout << "Bind API completed successfully.\n";      sockaddr\_in clientAddr;      char buff[BUFF\_MAXSIZE], clientIP[INET\_ADDRSTRLEN];      int ret, clientAddrLen = sizeof(clientAddr), clientPort;      while (1) {          ret = recvfrom(udpServerSock, buff, BUFF\_MAXSIZE, 0, (sockaddr \*)&clientAddr, &clientAddrLen);          if (ret == SOCKET\_ERROR) {              cout << "Error with code " << WSAGetLastError() << endl;              break;          }          Else {              inet\_ntop(AF\_INET, &clientAddr.sin\_addr, clientIP, sizeof(clientIP));              clientPort = ntohs(clientAddr.sin\_port);              buff[ret] = '\0';              cout << "Received message from client " << clientIP << ":" << clientPort << ": " << buff << endl;              string buffStr(buff);              buffStr = process(buffStr, clientPort);              const char \*message = buffStr.c\_str();              ret = sendto(udpServerSock, message, strlen(message), 0, (sockaddr \*)&clientAddr, clientAddrLen);              if (ret == SOCKET\_ERROR)              {                  cout << "Error with code " << WSAGetLastError() << endl;                  break;              }          }      }      closesocket(udpServerSock);      WSACleanup();      return 0;  } | // g++ UDPclient.cpp -o UDPclient.exe -lws2\_32  // .\UDPclient.exe 127.0.0.1 8080 dut.udn.vn  #include <iostream>  #include <winsock2.h>  using namespace std;  #define SERVER\_PORT 5500  #define SERVER\_ADDR "127.0.0.1"  int main()  {      const char \*serverIP = SERVER\_ADDR;      int port = SERVER\_PORT;      WSADATA wsaData;      if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)      {          cerr << "WSAStartup failed!" << endl;          return 1;      }      cout << "WSAStartup completed." << endl;      // Create socket      SOCKET clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);      if (clientSocket == INVALID\_SOCKET)      {          cerr << "Socket creation failed!" << endl;          WSACleanup();          return 1;      }      cout << "Creating socket completed successfully.\n";      sockaddr\_in serverAddr;      serverAddr.sin\_family = AF\_INET;      serverAddr.sin\_port = htons(port);      serverAddr.sin\_addr.s\_addr = inet\_addr(serverIP);      string messageSend;      while (1)      {          cout << "(Client) -->";          getline(cin, messageSend);          const char \*message = messageSend.c\_str();          if (sendto(clientSocket, message, strlen(message), 0, (sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)          {              cerr << "Error in sendto()" << endl;              closesocket(clientSocket);              WSACleanup();              return 1;          }          char buffer[1024];          sockaddr\_in serverResponse;          int serverResponseLen = sizeof(serverResponse);          int bytesReceived = recvfrom(clientSocket, buffer, sizeof(buffer), 0, (sockaddr \*)&serverResponse, &serverResponseLen);          if (bytesReceived == SOCKET\_ERROR)          {              cerr << "Error in recvfrom()" << endl;          }          else          {              buffer[bytesReceived] = '\0';              cout << "(Server): " << buffer << endl;          }      }      closesocket(clientSocket);      WSACleanup();      return 0;  }  **SERVER CÓ ĐK, ĐỔI PASS (UDP)**  string process(const string &input, int clientPort)  {  userData = readUserCSV("database.csv"); // Update data from database  istringstream iss(input);  string request, id, password, temp;  iss >> request >> id >> password >> temp;  if (request == "register" && id != "" && password != "" && temp == "")  {  if (loginWith[clientPort] != "")  return "You logged in";  for (UserData &user : userData)  if (user.id == id)  return "Id existing";  userData.push\_back({id, password, 1, 0});  updateUserCSV("database.csv");  return "Register successfully";  }  else if (request == "login" && id != "" && password != "" && temp == "")  {  if (loginWith[clientPort] != "")  return "You logged in";  for (UserData &user : userData)  {  if (user.id == id)  {  if (user.password == password)  {  // Bỏ cmt nếu muốn 1 tài khoản chỉ vào được 1 client duy nhất (Code hiện tại: 1 tài khoản vào được nhiều client như Fb)  // if (user.login == 0)  // {  loginWith[clientPort] = id;  user.login = 1;  updateUserCSV("database.csv");  return "Login successfully";  // }  // else  // return "You logged at another client";  }  else  return "Wrong password";  }  }  return "Id does not exist";  }  else if (request == "password" && id != "" && password != "" && temp == "")  {  if (loginWith[clientPort] == "")  return "Please logged in";  string newPassword = id;  string reEnterPw = password;  if (newPassword == reEnterPw)  {  for (UserData &user : userData)  if (user.id == loginWith[clientPort])  user.password = newPassword;  updateUserCSV("database.csv");  return "Change password successfully";  }  return "The 2 passwords are not the same";  }  else if (request == "logout" && id == "") // Check if redundant params  {  if (loginWith[clientPort] == "")  return "Please login";  for (auto &user : userData)  if (user.id == loginWith[clientPort])  user.login = 0;  loginWith[clientPort] = "";  updateUserCSV("database.csv");  return "Logout successfully";  }  else  return "Wrong message format";  } |