**Модуль по отображению электронной картографической информации от обеспечивающих систем корабля на «Систему отображения », при применении изделий по различным объектам.**

**Текст программы**

2017

**АННОТАЦИЯ**

В данном документе приведен текст программы «Картографическое ядро» и «Модуль графического отображения карт».

Исходным языком, используемым для данной разработки, является С. Результатом компиляции и сборки исходных кодов являются бинарные файлы, готовые к исполнению.

Оглавление

[1.Текст программы «Картографическое ядро» 4](#__RefHeading___Toc8256_1066461007)

[2.Текст программы «Модуль графического отображения карт» 17](#__RefHeading___Toc8258_1066461007)

# Текст программы «Картографическое ядро»

Листинг 1: Файл Commander.cpp

|  |
| --- |
| #include "Commander.h"  Client::Client() : toStop(false), id(-1)  {  //clTh = new std::thread();  }  Client::~Client()  {  delete clTh;  }  void Client::setId(int id)  {  this->id = id;  }  int Client::getId()  {  return id;  }  void Client::setThread(std::thread\* th)  {  clTh = th;  }  std::thread\* Client::getThread()  {  return clTh;  }  Commander::Commander() : toStopServer(false), toStopListenSocket(false),  liSocIsStoped(false),clients(), freeId(0)  {  map\_lock = new std::recursive\_mutex();  }  Commander::~Commander()  {  for (std::map<int, Client\*>::iterator it = clients.begin();  it != clients.end(); ++it)  {  delete it->second;  }  delete map\_lock;  }  void Commander::setLsStruct(LsStruct\* lsSt)  {  lsStruct = lsSt;  }  void Commander::stopServer()  {  auto iResult = closesocket(lsStruct->lsSocket);  //в цикле происходит удаление элемента map со всеми последствиями  for (std::map<int, Client\*>::iterator it = clients.begin();  it != clients.end();)  {  auto client = it->second;  it++;  stopClient(client->getId());  }  lsStruct->lsTh->join();  toStopListenSocket = true;  toStopServer = true;  }  bool Commander::stopClient(int id)  {  Client\* client;  try{  client = clients.at(id);  }  catch (const std::out\_of\_range& oor)  {  return false;  }  client->toStop = true;  client->getThread()->join();  clientStoped(client->getId());  return true;  }  void Commander::stopListenSocket()  {  toStopListenSocket = true;  }  //Возвращает ссылку на команды для клиента  Client\* Commander::addClient()  {  int id = getNextId();  auto newClient = new Client();  newClient->setId(id);  map\_lock->lock();  clients.insert(std::pair<int, Client\*>(id, newClient));  map\_lock->unlock();  return newClient;  }  void Commander::clientStoped(int id)  {  map\_lock->lock();  auto client = clients.at(id);  delete client;  clients.erase(id);  map\_lock->unlock();  }  bool Commander::serverIsStoped()  {  if (clients.size() == 0 && liSocIsStoped == true)  return true;  return false;  }  //Возвращает свободный id для клиента  //toDo простоя увелечение ид - это фигня  int Commander::getNextId()  {  return freeId++;  } |

Листинг 2: Файл Commander.h

|  |
| --- |
| #pragma once  #include <map>  #include <thread>  #include <winsock2.h>  #include <mutex>  class Client  {  public:  Client();  ~Client();  void setId(int id);  int getId();  void setThread(std::thread\* th);  std::thread\* getThread();  bool toStop;  private:  int id;  std::thread\* clTh;  };  struct LsStruct  {  SOCKET lsSocket;  std::thread\* lsTh;  };  class Commander  {  public:  Commander();  ~Commander();  Client\* addClient();  void setLsStruct(LsStruct\* lsSt);  //command  void stopServer();  bool stopClient(int id);  void stopListenSocket();  //answer  void clientStoped(int id);  void LiSocStoped();  //??  bool serverIsStoped();  bool toStopServer;  bool toStopListenSocket;  bool liSocIsStoped;  private:  int getNextId();  int freeId;  LsStruct\* lsStruct;  std::map<int, Client\*> clients;  std::recursive\_mutex\* map\_lock;  }; |

Листинг 3: Файл TSPServer.h

|  |
| --- |
| #pragma once  #define \_CRT\_SECURE\_NO\_WARNINGS  #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS //ошибка в inet\_addr()  #include <winsock2.h>  #include <Ws2tcpip.h>  #include <stdio.h>  #include <thread>  #include "Commander.h"  #include "LicenseChecker.h"  #include <string>  #include <sys/stat.h>  #include <sstream>  #pragma comment(lib,"Ws2\_32.lib")  //#define DEFAULT\_BUFLEN 1234  #define PKT\_LEN 1024  class TSPServer  {  public:  TSPServer();  ~TSPServer();  int startServer();  private:  Commander commander;  int readn(SOCKET fd, char \*bp, size\_t len, int flag);  void listenNewClients(SOCKET ListenSocket);  void serveNewClient(SOCKET AcceptSocket, Client\*);  void listenComands();  void clCommandCLOSE(Client\* client);  int sendn(SOCKET fd, char \*bp, size\_t len, int flag);  }; |

Листинг 4: Файл TSPServer.cpp

|  |
| --- |
| #include "TSPServer.h"  TSPServer::TSPServer()  {  }  TSPServer::~TSPServer()  {  }  int TSPServer::startServer()  {  SOCKET ListenSocket = INVALID\_SOCKET;  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData); //старт  if (iResult != NO\_ERROR) {  wprintf(L"WSAStartup failed with error: %ld\n", iResult);  return 1;  }  //создание сокета прослушивания  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);//AF\_INET - IPv4;  //AF\_inet - IPV6;  //AF\_UNSPEC - не важена версия IP  // The socket address to be passed to bind  if (ListenSocket == INVALID\_SOCKET) {  wprintf(L"socket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return 1;  }  sockaddr\_in service;  service.sin\_family = AF\_INET;  service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");//  service.sin\_port = htons(27015);  //Привязывание сокета  iResult = bind(ListenSocket, (SOCKADDR\*)&service, sizeof(service));  if (iResult == SOCKET\_ERROR) {  wprintf(L"bind failed with error %u\n", WSAGetLastError());  closesocket(ListenSocket);  WSACleanup();  return 1;  }  else  wprintf(L"bind returned success\n");  // Listen for incoming connection requests  // on the created socket  if (listen(ListenSocket, SOMAXCONN) == SOCKET\_ERROR)  wprintf(L"listen function failed with error: %d\n", WSAGetLastError());    auto listen = new std::thread(&TSPServer::listenNewClients,this,std::ref(ListenSocket));  auto lsStruct = new LsStruct();  lsStruct->lsSocket = ListenSocket;  lsStruct->lsTh = listen;  commander.setLsStruct(lsStruct);  //listen.detach();    //поток прослушивания команд в cmd  std::thread lsCom(&TSPServer::listenComands,this);  lsCom.join();  delete listen;  delete lsStruct;  WSACleanup();  wprintf(L"Server stoped\n");  system("pause");//ожидание нажатия  return 0;  }  int TSPServer::readn(SOCKET fd, char \*bp, size\_t len, int flag)  {  int cnt;  int iResult;  cnt = len;  while (cnt > 0)  {  iResult = recv(fd, bp, cnt, flag);  if (iResult < 0) /\* Ошибка чтения? \*/  {  if (WSAGetLastError() == WSAEINTR) /\* Вызов прерван? \*/  continue; /\* Повторить чтение. \*/  return -1; /\* Вернуть код ошибки. \*/  }  if (iResult == 0) /\* Конец файла? \*/  return len - cnt; /\* Вернуть неполный счетчик. \*/  bp += iResult;  cnt -= iResult;  }  return len;  }  //Поток общения клиента с сервером  void TSPServer::serveNewClient(SOCKET AcceptSocket,Client\* clCom)  {  char recvbuf[PKT\_LEN];  int recvbuflen = PKT\_LEN;  std::string hardwareParam;  std::string licenseKey;    int connectDown, iResult;  iResult = readn(AcceptSocket, recvbuf, recvbuflen, 0);  if (iResult > 0)  {  std::stringstream ss;  ss.str(recvbuf);  ss >> hardwareParam >> licenseKey;  LicenseChecker checker = LicenseChecker(hardwareParam, licenseKey);  std::string answer = checker.checkLicense();  int iResult = send(AcceptSocket, answer.c\_str(), PKT\_LEN, 0);  if (iResult == SOCKET\_ERROR) {  wprintf(L"send failed with error: %d\n", WSAGetLastError());  return;  }  }  else  if (iResult < 0)  {  printf("Client %d:recv failed: %d\n",clCom->getId(),WSAGetLastError());  }  printf("Connection closed\n");  iResult = shutdown(AcceptSocket, SD\_RECEIVE);  if (iResult == SOCKET\_ERROR) {  wprintf(L"Client %d: shutdown failed with error: %d\n", clCom->getId(), WSAGetLastError());  //return;  }  iResult = closesocket(AcceptSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"Client %d: close failed with error: %d\n", clCom->getId(), WSAGetLastError());  //return;  }  //commander.clientStoped(clCom->getId());  }  //Поток ожидания новых клиентов  void TSPServer::listenNewClients(SOCKET ListenSocket)  {  SOCKET AcceptSocket;  wprintf(L"Waiting for client to connect...\n");  // Accept the connection.  do{  AcceptSocket = accept(ListenSocket, NULL, NULL);  if (commander.toStopListenSocket == true) break; //to do  if (AcceptSocket == INVALID\_SOCKET) {  auto err = WSAGetLastError();  wprintf(L"ListenSocket: accept failed with error: %ld\n",err);  if (err == 10004)  return;  continue;  //closesocket(ListenSocket);  //WSACleanup();  //return 1;  }  else  {  auto clCom = commander.addClient();  wprintf(L"Client:%d connected.\n",clCom->getId());  std::thread\* clTh;  clTh = new std::thread(&TSPServer::serveNewClient,this, std::ref(AcceptSocket), clCom);  clCom->setThread(clTh);  }  } while (commander.toStopListenSocket==false);  auto iResult = closesocket(ListenSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"ListenSocket: close failed with error: %d\n",  WSAGetLastError());  return;  }  else  {  commander.liSocIsStoped = true;  return;  }  }  void TSPServer::listenComands()  {  char command[255];  do  {  gets\_s(command);  if (strcmp(command, "exit") == 0)  {  commander.stopServer();  //while (commander.serverIsStoped()==false){}  break;  }  else  if (atoi(command)>=0)  commander.stopClient(atoi(command));  } while (1);  }  //toDo Как вызвать отсюда commander::stopClient, чтобы клиент удалился?  void TSPServer::clCommandCLOSE(Client\* client)  {  //std::thread closeThread(&Commander::stopClient, commander, client->getId());  //closeThread.detach();  client->toStop = true;  //commander.stopClient(client->getId());  }  //Как readn, только для отправки.  int TSPServer::sendn(SOCKET fd, char \*bp, size\_t len, int flag)  {  int iResult = send(fd, bp, len, flag);  if (iResult == SOCKET\_ERROR) {  wprintf(L"send failed with error: %d\n", WSAGetLastError());  //closesocket(AcceptSocket);  //return;  }  return iResult;  } |

Листинг 5: Файл DataAccessObject.cpp

|  |
| --- |
| #include "DataAccessObject.h"  DataAccessObjecct::DataAccessObjecct()  {  }  DataAccessObjecct::~DataAccessObjecct()  {  }  //key - значение ключа  //Возвращает значение аппаратного обеспечения, если оно уже зарегистрированно  //Возвращает NULL если не зарегистрированно  //Возвращает пустую строку, если лицензионный ключ не существует.  std::string DataAccessObjecct::getHardwareByLicenseKey(std::string key)  {  sqlite3 \*db;  char \*zErrMsg = 0;  int rc;  std::string sql;  std::string\* hardware = new std::string("");    /\* Open database \*/  rc = sqlite3\_open("licenses.db", &db);  if (rc){  fprintf(stderr, "Can't open database: %s\n", sqlite3\_errmsg(db));  return(0);  }  else{  fprintf(stderr, "Opened database successfully\n");  }  /\* Create SQL statement \*/  sql = "SELECT hardware FROM licenses WHERE licenseKey = \'" + key + "\';";  /\* Execute SQL statement \*/  rc = sqlite3\_exec(db, sql.c\_str(), getHardwareByLicenseKeyCallback,  (void\*)hardware, &zErrMsg);  if (rc != SQLITE\_OK){  fprintf(stderr, "SQL error: %s\n", zErrMsg);  sqlite3\_free(zErrMsg);  }  else{  fprintf(stdout, "Operation done successfully\n");  }  sqlite3\_close(db);  std::string hw = std::string(\*hardware);  delete hardware;  return hw;  }  int DataAccessObjecct::getHardwareByLicenseKeyCallback(void \*ptrData, int argc, char \*\*argv, char \*\*azColName){  std::string\* hardware = (std::string\*)ptrData;  for (int i = 0; i<argc; i++){  \*hardware = argv[i] ? argv[i] : "NULL";  }  return 0;  }  int DataAccessObjecct::setHardwareToLicenseKey(std::string licenseKey, std::string hardware)  {  sqlite3 \*db;  char \*zErrMsg = 0;  int rc;  std::string sql;  const char\* str = "";  /\* Open database \*/  rc = sqlite3\_open("licenses.db", &db);  if (rc){  fprintf(stderr, "Can't open database: %s\n", sqlite3\_errmsg(db));  return(0);  }  else{  fprintf(stderr, "Opened database successfully\n");  }  /\* Create SQL statement \*/  sql = "UPDATE licenses SET hardware = \'" + hardware +  "\' WHERE licenseKey = \'" + licenseKey + "\';";  /\* Execute SQL statement \*/  rc = sqlite3\_exec(db, sql.c\_str(), setHardwareToLicenseKeyCallback,  (void\*)str, &zErrMsg);  if (rc != SQLITE\_OK){  fprintf(stderr, "SQL error: %s\n", zErrMsg);  sqlite3\_free(zErrMsg);  }  else{  fprintf(stdout, "Operation done successfully\n");  }  sqlite3\_close(db);  return 0;  }  int DataAccessObjecct::setHardwareToLicenseKeyCallback(void \*data, int argc, char \*\*argv, char \*\*azColName)  {  int i = 0;  return 0;  } |

Листинг 6: Файл DataAccesssObject.h

|  |
| --- |
| #pragma once  #include <string>  #include "sqlite3.h"  class DataAccessObjecct  {  public:  DataAccessObjecct();  ~DataAccessObjecct();  std::string getHardwareByLicenseKey(std::string);  int setHardwareToLicenseKey(std::string, std::string);  std::string addLicenseKey(std::string licenseKey);  private:  static int getHardwareByLicenseKeyCallback(void \*data, int argc, char \*\*argv, char \*\*azColName);  static int setHardwareToLicenseKeyCallback(void \*data, int argc, char \*\*argv, char \*\*azColName);    }; |

Листинг 7: Файл LicenseChecker.cpp

|  |
| --- |
| #include "LicenseChecker.h"  LicenseChecker::LicenseChecker(std::string ahardwareParam, std::string alicenseKey)  {  hardwareParam = ahardwareParam;  licenseKey = alicenseKey;  }  LicenseChecker::~LicenseChecker()  {  }  std::string LicenseChecker::checkLicense()  {  //Pair pair = checkKeyValidation();  DataAccessObjecct dao = DataAccessObjecct();  std::string hw = dao.getHardwareByLicenseKey(licenseKey);  if (hw == "")  {  return "\_1 Invalid Key";  }  else if (hw == "NULL")  {  dao.setHardwareToLicenseKey(licenseKey, hardwareParam);  }  else  {  return "\_2 Invalid hardware parameters";  }  return getHash();  }  //ToDO sqlite3  Pair LicenseChecker::checkKeyValidation()  {  Pair pair = Pair();  DataAccessObjecct dao = DataAccessObjecct();  std::string hw = dao.getHardwareByLicenseKey(licenseKey);  if (hw == "")  pair.key = "";  else if (hw == "NULL")  {  }  if (licenseKey != "LIC")  {  pair.key = "";  pair.hardwareParam = "";  }  else  {  pair.key = "LIC";  pair.hardwareParam = "VA50\_HC\_CR";  }  return pair;  }  std::string LicenseChecker::getHash()  {  std::string output = sha256(hardwareParam);  return output;  } |

Листинг 8: Файл LicenseChecker.h

|  |
| --- |
| #pragma once  #include "string"  #include "sha256.h"  #include "DataAccessObject.h"  //using namespace std;  struct Pair  {  std::string key;  std::string hardwareParam;  };  class LicenseChecker  {  public:  LicenseChecker(std::string hardwareParam, std::string licenseKey);  ~LicenseChecker();  std::string checkLicense();  private:  std::string hardwareParam, licenseKey;  std::string getHash();  Pair checkKeyValidation();  }; |

Листинг 9: Файл main.cpp

|  |
| --- |
| #include "TSPServer.h"  #include "DataAccessObject.h"  int main(void)  {  TSPServer\* server = new TSPServer();  server->startServer();  return 0;  } |

# Текст программы **«****Модуль графического отображения карт»**

Листинг 10: Файл LocalRepository.h

|  |
| --- |
| #pragma once  #include <fstream>  #include <iostream>  #include "string"  #include "rapidjson/prettywriter.h"  #include "rapidjson/filereadstream.h"  #include "rapidjson/document.h"  using namespace rapidjson;  using namespace std;  class LocalRepository// : public MemberRepository  {  public:  LocalRepository();  ~LocalRepository();  bool getFirstRuning();  bool setFirstRuning(bool);  bool setLicenseKey(string);  string getLicenseKey();  bool setActivateKey(string);  string getActivateKey();  bool setUsbKeyVid(string);  string getUsbKeyVid();  bool setUsbKeyPid(string);  string getUsbKeyPid();  private:  bool isFirstRuning;  string licenseKey;  string activeKey;  string usbKeyVid;  string usbKeyPid;  bool serializeData();  bool deserializeData();  };  class Data  {  public:  Data(){}  ~Data(){}  bool isFirstRuning;  string licenseKey;  string activeKey;  string usbKeyVid;  string usbKeyPid;  template <typename Writer>  void Serialize(Writer& writer) const {  writer.StartObject();  writer.String("isFirstRuning");  writer.Bool(isFirstRuning);  writer.String("licenseKey");  #if RAPIDJSON\_HAS\_STDSTRING  writer.String(licenseKey);  #else  writer.String(licenseKey.c\_str(), static\_cast<SizeType>(licenseKey.length()));  #endif  writer.String("activeKey");  #if RAPIDJSON\_HAS\_STDSTRING  writer.String(activeKey);  #else  writer.String(activeKey.c\_str(), static\_cast<SizeType>(activeKey.length()));  #endif  writer.String("usbKeyVid");  #if RAPIDJSON\_HAS\_STDSTRING  writer.String(usbKeyVid);  #else  writer.String(usbKeyVid.c\_str(), static\_cast<SizeType>(usbKeyVid.length()));  #endif  writer.String("usbKeyPid");  #if RAPIDJSON\_HAS\_STDSTRING  writer.String(usbKeyPid);  #else  writer.String(usbKeyPid.c\_str(), static\_cast<SizeType>(usbKeyPid.length()));  #endif  writer.EndObject();  }  }; |

Листинг 11: Файл main.cpp

|  |
| --- |
| #include "MyProgram.h"  #include "Protect.h"  #include "iostream"  #include "LocalRepository.h"  using namespace std;  int main(void)  {  Protect protect = Protect();  int res;  do  {  res = protect.checkProtection();  switch (res)  {  case 0:  {  cout << "License successful\n" << endl;  break;  }  case -1:  {  cout << "Please, enter license key" << endl;  string licenseKey;  cin >> licenseKey;  protect.setLicenseKey(licenseKey);  break;  }  case -11:  {  cout << "Please, enter license key" << endl;  string licenseKey;  cin >> licenseKey;  protect.setLicenseKey(licenseKey);  break;  }  case -12:  {  cout << "Server not available, repeat later\n" << endl;  system("pause");  return -1;  break;  }  case -13:  {  cout << "Wrong license key, please repeat\n" << endl;  string licenseKey;  cin >> licenseKey;  protect.setLicenseKey(licenseKey);  break;  }  case -14:  {  cout << "Wrong hardware for this license key\n" << endl;  system("pause");  return -2;  break;  }  default:  cout << "Unknown error\n" << endl;  system("pause");  return -2;  break;    }  } while (res != 0);  if (protect.checkUSBKeyProtection())  {  cout << "USB Key not found" << endl;  system("pause");  return -1;  }  else  {  cout << "USB Key is correct" << endl;  }  MyProgram prog = MyProgram();  prog.run();  return 0;  } |

Листинг 12: Файл MemberRepository.h

|  |
| --- |
| #pragma once  #include "string"  using namespace std;  class MemberRepository  {  public:  virtual bool getFirstRuning() = 0;  virtual bool setFirstRuning(bool) = 0;  virtual bool setLicenseKey(string) = 0;  virtual string getLicenseKey() = 0;  virtual bool setActivateKey(string) = 0;  virtual string getActivateKey() = 0;  }; |

Листинг 13: Файл LocalRepository.cpp

|  |
| --- |
| #include "LocalRepository.h"  LocalRepository::LocalRepository()  {  deserializeData();  }  LocalRepository::~LocalRepository()  {  serializeData();  }  bool LocalRepository::getFirstRuning()  {  if (!deserializeData())  return NULL;  return isFirstRuning;  }  bool LocalRepository::setFirstRuning(bool k)  {  isFirstRuning = k;  return serializeData();  }  bool LocalRepository::setLicenseKey(string key)  {  licenseKey = key;  return serializeData();  }  string LocalRepository::getLicenseKey()  {  if (!deserializeData())  return NULL;  return licenseKey;  }  bool LocalRepository::setActivateKey(string key)  {  activeKey = key;  return serializeData();  }  string LocalRepository::getActivateKey()  {  if (!deserializeData())  return NULL;  return activeKey;  }  bool LocalRepository::setUsbKeyVid(string key)  {  usbKeyVid = key;  return serializeData();  }  string LocalRepository::getUsbKeyVid()  {  if (!deserializeData())  return NULL;  return usbKeyVid;  }  bool LocalRepository::setUsbKeyPid(string key)  {  usbKeyPid = key;  return serializeData();  }  string LocalRepository::getUsbKeyPid()  {  if (!deserializeData())  return NULL;  return usbKeyPid;  }  bool LocalRepository::serializeData()  {  Data data = Data();  data.activeKey = activeKey;  data.isFirstRuning = isFirstRuning;  data.licenseKey = licenseKey;  data.usbKeyVid = usbKeyVid;  data.usbKeyPid = usbKeyPid;  StringBuffer sb;  PrettyWriter<StringBuffer> writer(sb);  data.Serialize(writer);  ofstream fout("data.txt");  if (!fout.is\_open())  cout << "file not opened" << endl;  else  {  fout << sb.GetString() << endl;  fout.close();  }  return true;  }  bool LocalRepository::deserializeData()  {  FILE\* fp;  errno\_t err;  err = fopen\_s(&fp, "data.txt", "rb"); // non-Windows use "r"  if (err != 0)  {  cout << "file not opened" << endl;  return false;  }  char readBuffer[65536];  FileReadStream is(fp, readBuffer, sizeof(readBuffer));  Document document;  document.ParseStream(is);  if (!document.IsObject());  assert(document.HasMember("isFirstRuning"));  assert(document["isFirstRuning"].IsBool());  isFirstRuning = document["isFirstRuning"].GetBool();  assert(document.HasMember("licenseKey"));  assert(document["licenseKey"].IsString());  licenseKey = document["licenseKey"].GetString();  assert(document.HasMember("activeKey"));  assert(document["activeKey"].IsString());  activeKey = document["activeKey"].GetString();  assert(document.HasMember("usbKeyVid"));  assert(document["usbKeyVid"].IsString());  usbKeyVid = document["usbKeyVid"].GetString();  assert(document.HasMember("usbKeyPid"));  assert(document["usbKeyPid"].IsString());  usbKeyPid = document["usbKeyPid"].GetString();  fclose(fp);  return true;  } |

Листинг 14: Файл MyProgram.cpp

|  |
| --- |
| #include "MyProgram.h"  MyProgram::MyProgram()  {  }  MyProgram::~MyProgram()  {  }  int MyProgram::run()  {  std::cout << "Hello world" << std::endl;  system("pause");  return 0;  } |

Листинг 15: Файл MyProgram.h

|  |
| --- |
| #pragma once  #include "iostream"  class MyProgram  {  public:  MyProgram();  ~MyProgram();  int run();  private:  }; |

Листинг 16: Файл NetClient.h

|  |
| --- |
| #include "NetClient.h"  NetClient::NetClient(string aparam, string alicenseKey)  {  hardwareParam = aparam;  licenseKey = alicenseKey;  }  NetClient::~NetClient()  {  }  string NetClient::getActivateKey()  {  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData); //старт  if (iResult != NO\_ERROR) {  wprintf(L"WSAStartup failed with error: %ld\n", iResult);  return "";  }  //создание сокета для подключения  SOCKET ConnectSocket = INVALID\_SOCKET;  ConnectSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);//AF\_INET - IPv4;  if (ConnectSocket == INVALID\_SOCKET) {  wprintf(L"socket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return "";  }  //СОздание соединения с сервером  sockaddr\_in clientService;  clientService.sin\_family = AF\_INET;  clientService.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  clientService.sin\_port = htons(27015);  //----------------------  // Connect to server.  iResult = connect(ConnectSocket, (SOCKADDR \*)& clientService, sizeof(clientService));  if (iResult == SOCKET\_ERROR) {  wprintf(L"connect function failed with error: %ld\n", WSAGetLastError());  iResult = closesocket(ConnectSocket);  if (iResult == SOCKET\_ERROR)  wprintf(L"closesocket function failed with error: %ld\n", WSAGetLastError());  WSACleanup();  return "";  }  wprintf(L"Connected to server.\n");  string answer = commun(ConnectSocket);  //Завершение соединения  iResult = shutdown(ConnectSocket, SD\_RECEIVE);  if (iResult == SOCKET\_ERROR) {  wprintf(L"shutdown failed with error: %d\n", WSAGetLastError());  WSACleanup();  return "";  }  //закрытие сокета  iResult = closesocket(ConnectSocket);  if (iResult == SOCKET\_ERROR) {  wprintf(L"close failed with error: %d\n", WSAGetLastError());  WSACleanup();  return "";  }  WSACleanup();  return answer;  }  string NetClient::commun(SOCKET socket)  {  string data = hardwareParam + " " + licenseKey;  int sendBufLen = DEFAULT\_BUFLEN;  int iResult = send(socket, data.c\_str(), DEFAULT\_BUFLEN, 0);  if (iResult == SOCKET\_ERROR) {  wprintf(L"send failed with error: %d\n", WSAGetLastError());  closesocket(socket);  WSACleanup();  return "";  }  char recvbuf[DEFAULT\_BUFLEN + 1];  int recvbuflen = DEFAULT\_BUFLEN;  iResult = readn(socket, recvbuf, recvbuflen, 0);  if (iResult > 0)  {  return string(recvbuf);  }  else  {  if (iResult == 0)  printf("Connection closed\n");  else  if (iResult < 0)  printf("recv failed: %d\n", WSAGetLastError());  return "";  }    }  int NetClient::readn(SOCKET fd, char \*bp, size\_t len, int flag)  {  int cnt;  int iResult;  cnt = len;  while (cnt > 0)  {  iResult = recv(fd, bp, cnt, flag);  if (iResult < 0) /\* Ошибка чтения? \*/  {  if (WSAGetLastError() == WSAEINTR) /\* Вызов прерван? \*/  continue; /\* Повторить чтение. \*/  return -1; /\* Вернуть код ошибки. \*/  }  if (iResult == 0) /\* Конец файла? \*/  return len - cnt; /\* Вернуть неполный счетчик. \*/  bp += iResult;  cnt -= iResult;  }  return len;  } |

Листинг 17: Файл NetClient.h

|  |
| --- |
| #pragma once  #include "string"  #define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS //ошибка в inet\_addr()  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <winsock2.h>  #include <Ws2tcpip.h>  #include <stdio.h>  #include <cstring>  #include <iostream>  #include <string>  #include <sys/stat.h>  #include <memory>  #define DEFAULT\_BUFLEN 1024  #pragma comment(lib,"Ws2\_32.lib")  using namespace std;  class NetClient  {  public:  NetClient(string hardwareParam, string licenseKey);  ~NetClient();  string getActivateKey();  string commun(SOCKET socket);  private:  string hardwareParam;  string licenseKey;  int readn(SOCKET fd, char \*bp, size\_t len, int flag);  }; |

Листинг 18: Файл Protect.cpp

|  |
| --- |
| #include "Protect.h"  Protect::Protect()  {  rep = LocalRepository();  licenseKey = "";  }  Protect::~Protect()  {  }  int Protect::checkProtection()  {  if (rep.getFirstRuning())  {  //отсутствует лисензионный ключ  if (licenseKey == "")  return -1;  //  int res = registerProduct();  if (res != 0)  return res;  rep.setLicenseKey(licenseKey);  rep.setFirstRuning(false);  }  else  {  licenseKey = rep.getLicenseKey();  }  if (checkLicense() != 0)  {  rep.setLicenseKey("");  rep.setFirstRuning(true);  return -3;  }    return 0;  }  bool Protect::setLicenseKey(string key)  {  //ToDo размер ключа  licenseKey = key;  return true;  }  //ToDo нужно бОльше параметров  string Protect::getHardwareParameters()  {  LPDWORD lType[100];  //char pData[255]; //PVOID pData[100];  TCHAR pData[255] = { 0 };  DWORD lSize = 100;  LPCSTR BaseKey = "SOFTWARE\\Microsoft\\.NETFramework";  int i = RegGetValue(HKEY\_LOCAL\_MACHINE,  TEXT("HARDWARE\\DESCRIPTION\\System\\BIOS"),  TEXT("BaseBoardProduct"),  RRF\_RT\_REG\_SZ,  NULL, &pData, &lSize);  return string(pData);  }  int Protect::registerProduct()  {  if (licenseKey == "")  return -11;  string hardwareParam;  string activateKey;  hardwareParam = getHardwareParameters();  NetClient client(hardwareParam, licenseKey);  activateKey = client.getActivateKey();  if (activateKey == "")  return -12;  else  {  if (activateKey[0] == '\_')  {  if (activateKey[1] == '1')  {  return -13;  }  else  {  return -14;  }  }  }  rep.setActivateKey(activateKey);  return 0;  }  int Protect::checkLicense()  {  if (licenseKey == "")  return -2;  string hash;  hash = getHash();  if (hash != rep.getActivateKey())  return -1;  return 0;  }  string Protect::getHash()  {  string hardwareParam = getHardwareParameters();  string output = sha256(hardwareParam);  return output;  }  int Protect::checkUSBKeyProtection()  {  vector <USBPair> vec = getUSBPairs();  USBPair usbKey(rep.getUsbKeyVid(), rep.getUsbKeyPid());  bool haveUSBKey = false;  for (USBPair pair : vec)  {  pair.vid = sha256(pair.vid);  pair.pid = sha256(pair.pid);  if (pair == usbKey)  {  haveUSBKey = true;  break;  }  }  if (haveUSBKey != true)  return -1;  return 0;  }  vector<USBPair> Protect::getUSBPairs() {  DWORD deviceIndex = 0;  SP\_DEVINFO\_DATA deviceInfoData;  SP\_DEVICE\_INTERFACE\_DATA deviceInterfaceData;  deviceInfoData.cbSize = sizeof(deviceInfoData);  vector <USBPair> vec;  //buried somewhere deep in the ddk  static GUID GUID\_DEVINTERFACE\_USB\_HUB = { 0xf18a0e88, 0xc30c, 0x11d0, { 0x88, 0x15, 0x00, 0xa0, 0xc9, 0x06, 0xbe, 0xd8 } };  static GUID GUID\_DEVINTERFACE\_USB\_DEVICE = { 0xA5DCBF10L, 0x6530, 0x11D2, { 0x90, 0x1F, 0x00, 0xC0, 0x4F, 0xB9, 0x51, 0xED } };  static GUID GUID\_DEVINTERFACE\_USB\_HOST\_CONTROLLER = { 0x3abf6f2d, 0x71c4, 0x462a, { 0x8a, 0x92, 0x1e, 0x68, 0x61, 0xe6, 0xaf, 0x27 } };  HDEVINFO deviceInterfaceSet = SetupDiGetClassDevs(&GUID\_DEVINTERFACE\_USB\_DEVICE, NULL, NULL, DIGCF\_DEVICEINTERFACE | DIGCF\_PRESENT);  //HDEVINFO deviceInterfaceSet = SetupDiGetClassDevs(NULL, NULL, NULL, DIGCF\_PRESENT | DIGCF\_ALLCLASSES | DIGCF\_DEVICEINTERFACE);  //HDEVINFO deviceInterfaceSet = SetupDiGetClassDevs(NULL, NULL, NULL, DIGCF\_ALLCLASSES | DIGCF\_PRESENT);  if (deviceInterfaceSet == INVALID\_HANDLE\_VALUE)  {  // Insert error handling here.  return vec;  }  while (SetupDiEnumDeviceInfo(deviceInterfaceSet, deviceIndex, &deviceInfoData))  {  deviceInfoData.cbSize = sizeof(deviceInfoData);  ULONG IDSize;  CM\_Get\_Device\_ID\_Size(&IDSize, deviceInfoData.DevInst, 0);  TCHAR\* deviceID = new TCHAR[IDSize];  CM\_Get\_Device\_ID(deviceInfoData.DevInst, deviceID, MAX\_PATH, 0);  char vid[5];  char pid[5];  strncpy\_s(vid, deviceID + 8, 4);  strncpy\_s(pid, deviceID + 17, 4);  vec.push\_back(USBPair(vid, pid));  deviceIndex++;  }  SetupDiDestroyDeviceInfoList(deviceInterfaceSet);  return vec;  } |

Листинг 18: Файл Protect.h

|  |
| --- |
| #pragma once  #pragma comment (lib, "Setupapi.lib")  #include "cstring"  #include "LocalRepository.h"  #include "NetClient.h"  #include "sha256.h"  #include <SetupAPI.h>  #include <cfgmgr32.h>  #include <tchar.h>  #include <vector>  #include <stdio.h>  #pragma comment(lib, "user32.lib")  using namespace std;  //Наверное не нужен  struct USBPair  {  string vid;  string pid;  public:  USBPair(string v, string p)  {  vid = v;  pid = p;  }  bool equals(USBPair pair)  {  if (pair.vid == vid && pair.pid == pid)  return true;  return false;  }  friend bool operator==(const USBPair& left, const USBPair& right)  {  if (left.vid == right.vid && left.pid == right.pid)  return true;  return false;  }  };  class Protect  {  public:  Protect();  ~Protect();  int checkProtection();  bool setLicenseKey(string key);  int checkUSBKeyProtection();  private:  string getHardwareParameters();  int registerProduct();  int checkLicense();  string getHash();  vector<USBPair> getUSBPairs();    LocalRepository rep;  string licenseKey;  }; |