Приложение Ж Листинг исходного кода

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.0)
project(one-way-sync VERSION 0.1.0)

set(CMAKE_CXX_STANDARD 20)

set (source_dir "${PROJECT_SOURCE_DIR}/src/")

SET(CMAKE_INCLUDE_PATH ${CMAKE_INCLUDE_PATH} "C:/CLI_STUFF/mingw64/boost_1_78_0/")
SET(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} "C:/CLI_STUFF/mingw64/boost_1_78_0/lib/")

set(BOOST_ROOT "C:/CLI_STUFF/mingw64/boost_1_78_0/")
set(Boost_USE_STATIC_LIBS ON)
find_package(Boost COMPONENTS system)
find_package( Threads )

include_directories(${Boost_INCLUDE_DIR})

file (GLOB source_files
   "${source_dir}/*.cpp"
   "${source_dir}/helpers/*.cpp"
   "${source_dir}/models/*.cpp"
   "${source_dir}/networking/*.cpp"
)

include_directories(
   "${source_dir}/"
)

add_compile_options(-Wall -Wextra -pedantic -Werror -pthread)

add_executable(one-way-sync ${source_files})

target_link_libraries(one-way-sync ws2_32)
target_link_libraries(one-way-sync Threads::Threads)
```

main.cpp

```
#include <boost/asio.hpp>
#include <boost/array.hpp>
#include <boost/bind/bind.hpp>
#include <thread>
#include <iostream>

#define IPADDRESS "127.0.0.1" // "192.168.1.64"
#define UDP_PORT 13253

using boost::asio::ip::udp;
using boost::asio::ip::address;

void Sender(std::string in) {
   boost::asio::io_service io_service;
   udp::socket socket(io_service);
   udp::endpoint remote_endpoint =
udp::endpoint(address::from_string(IPADDRESS), UDP_PORT);
   socket.open(udp::v4());

   boost::system::error_code err;
   auto sent = socket.send_to(boost::asio::buffer(in), remote_endpoint, 0, err);
   socket.close();
   std::cout << "Sent Payload --- " << sent << "\n";
}

struct Client {
   boost::asio::io_service io_service;
   udp::socket socket{io_service};
   boost::array<char, 1024> recv_buffer;
   udp::endpoint remote_endpoint;

   int count = 3;

   void handle_receive(const boost::system::error_code& error, size_t
bytes_transferred) {
      if (error) {
         std::cout << "Receive failed: " << error.message() << "\n";
```

```
            return;
        }
        std::cout << "Received: '" << std::string(recv_buffer.begin(),
recv_buffer.begin()+bytes_transferred) << "' (" << error.message() << ")\n";

        if (--count > 0) {
            std::cout << "Count: " << count << "\n";
            wait();
        }
    }

    void wait() {
        socket.async_receive_from(boost::asio::buffer(recv_buffer),
            remote_endpoint,
            boost::bind(&Client::handle_receive,                    this,
boost::asio::placeholders::error, boost::asio::placeholders::bytes_transferred));
    }

    void Receiver()
    {
        socket.open(udp::v4());
        socket.bind(udp::endpoint(address::from_string(IPADDRESS),
UDP_PORT));

        wait();

        std::cout << "Receiving\n";
        io_service.run();
        std::cout << "Receiver exit\n";
    }
};

int main(int argc, char *argv[])
{
    Client client;
    std::thread r([&] { client.Receiver(); });

    std::string input = argc>1? argv[1] : "hello world";
```

```
        std::cout << "Input is '" << input.c_str() << "'\nSending it to Sender Function...\n";

        for (int i = 0; i < 3; ++i) {
          std::this_thread::sleep_for(std::chrono::milliseconds(200));
          Sender(input);
        }

        r.join();
      }
```