LAB Inter-process coms UDP Sockets

Table of Contents

C++ Inter-process communications using UDP sockets.

Background -The ASIO library for UDP sockets.

For writing our sockets, we will use the asio.hpp header library. We could write Windows-specific code, but let's be platform-independent and anticipate the future C++26 standard.

Some Notes on using UDP and using the asio cross-platform C++ library:
UDP is connectionless, so unlike TCP sockets for servers, there's no listen, accept, or connect to do.

The Server:

- Instantiate the asio library object
  - `asio::io_context io;`
- Define a. asio::ip::udp::socket as type UDP
  - socket socket(io, asio::ip::udp::endpoint(socket type, port)
- If we are using IPv4 this socket type is asio::ip::udp::v4()
- Define an "endpoint" – an object to store the sender's address and port:
  - `asio::ip::udp::endpoint sender_endpoint;`
- The server waits for a message to be sent to it. We use
  - socket.receive_from() takes a buffer and the endpoint to fill with the data and the sender's information, respectively.

The client sends a datagram to the server.

- Identify which port we should connect to
  - `asio::ip::udp::endpoint server_endpoint( asio::ip::make_address("127.0.0.1"), 8080);`
  - `asio::ip::udp::socket socket(io);`
- Our client socket does not need a specific port to be set since we are connecting to a known server socket the OS can choose a port number, and the server will know our port number when we connect. We just open the socket we previously defined for the server
  - `socket.open(asio::ip::udp::v4());`
- Our client sends a datagram to the server.
  - socket.send_to(asio::buffer(message), server_endpoint); // Send the message to the given server IP address/port.
- In this example, our communication is unidirectional.

UDP is a Message Datagram type of connection, so ASIO will call the OS's API and set the socket to be the datagram type. In the example in the handbook, we used TCP, so in that case OS's API would be set to the streaming type.

Running these programs

We need to incorporate the asio library files into our program.

1. Copy both the asio folder and the asio.hpp from Canvas into the folder you are running these programs from.
2. To tell the compiler where to search for the asio library, we need to include current directory in the include search path.
3. Run the code. It will fail, but it will also create the tasks.json file.
4. In VS Code you do that by adding the following line to .vscode/tasks.json in the *args* section: "-I.",

To compile by hand on Visual Studio code – In the terminal prompt:
For Windows Studio compiler use: (Windows may also need you to specify the library ws2_32.lib)
For the cluster PCs VS Code task.json file should have the following information

```
  "tasks": [
     {
        "type": "cppbuild",
        "label": "C/C++: g++.exe build active file",
        "command": "C:\\Program Files (x86)\\NAG\\EFBuilder 
7.1\\nagfor\\lib\\NewMinGW64\\bin\\g++.exe",  // This must be a C++ compiler NOT gcc.exe. This is the 
path for the compiler on the cluster PCs.
        "args": [
          "-fdiagnostics-color=always",
          "-g",
          "${file}",
          "-o",
          "${fileDirname}\\${fileBasenameNoExtension}.exe",
          "-I.",    // This tells the compiler to search in the current directory for the ASIO files
          "-lws2_32",   // This tells the linker to add the Winsock Windows library
          "-D_WIN32_WINNT=0x0601"
        ],
```

If you are having problems with using **std::cout** then you are probably using the gcc (C compiler) rather than the g++ (C++ compiler).

For Linux g++ compiler

- g++ udp_server.cpp -pthread -o udp_server
- g++ udp_client.cpp -pthread -o udp_client

To Run it:
In one terminal: udp_server.exe
In another terminal: udp_client.exe

Task 1: Communication with a unidirectional UDP socket and the ASIO library.

For this lab task, you will create a UDP socket-based inter-local-process client/server example. The client will send data to the server. UDP is bidirectional, like TCP but in this example we just send data from the client to the server.

Expected Output

Server:
```
UDP server listening on port 8080...
Received from 127.0.0.1:54733 → Hello from portable UDP client!
Received message from 127.0.0.1:xxxxx → Hello from UDP client!
```
Client:
```
Message sent to server: Hello from portable
```

1. Below is the server code. Have a go at writing the client code (it is given in the solutions). Understand what the code is doing and play with it.
2. Use the run/debug program to compile the programmes, but once they are compiled, open multiple terminals in VS Code and run the two programmes from them to help you observe what is happening. When running multiple programmes, using the terminal is easier than using the Run/Debug button.
3. Try to run two udp_server programs simultaneously. The catch statement will be invoked, and it will inform you that this is not possible.
4. Change the client program to send messages continuously, but add the following delay
    a. `std::this_thread::sleep_for(std::chrono::seconds(1));  // Wait 1 second`
5. Run two client programs simultaneously – does that work? Do they have different port numbers?
6. If you run a single client, but there is no server running. Then what happens, and why?

UDP Server (udp_server.cpp)
```
#define ASIO_STANDALONE  // We are just using Asio, not Boost
#include <asio.hpp>
#include <iostream>
#include <array>

int main() {
    try {
        asio::io_context io; // Instantiate the asio library object
        // Create a UDP IPv4 socket bound to port 8080 on ALL IPv4 interfaces (0.0.0.0:8080),
not just localhost.
        asio::ip::udp::socket socket(io, asio::ip::udp::endpoint(asio::ip::udp::v4(), 8080));
        std::cout << "UDP server listening on port 8080..." << std::endl;
        std::array<char, 1024> buffer; // Create a 1 kiByte buffer to receive the data from the
client
        asio::ip::udp::endpoint sender_endpoint; // Used to store the information about who
send us the data.
        while (true) { // Continuously receive data from the client
            std::size_t bytes_received = socket.receive_from(asio::buffer(buffer),
sender_endpoint);  // The std::array prevents buffer overruns as it includes the buffer length
            std::string msg(buffer.data(), bytes_received);
            std::cout << "[Server rec'd message from " << sender_endpoint.address().to_string()
                << ":" << sender_endpoint.port() << "]  " << msg << std::endl;
        }
    }
    catch (std::exception& e) {
        std::cerr << "Server error: " << e.what() << std::endl;
    }
    return 0;
}
```

Task 2: Communication with a bidirectional UDP socket and the ASIO library.

Now extend the code in the solution for task 1 to create a bidirectional communication that:

1. The client starts a timer.
2. The client sends the integer number “1” to the server
3. The server receives the integer number, adds one to it and sends it to the client.
4. The client receives the integer, adds 1 to it, and sends it to the server.
5. The client stops when the number reaches 1,000,000
6. The client prints the time taken to run the program, to give us an indication of how much time is spent in the inter-local-process communication.

Then write the client program to find out how long it takes to count internally to 1,000,000 without sending data to the server. How different are the run times?

Task 3: Inter-remote-process communication with UDP.

1. Alter your task code so that it works across two different computers. To do this, you need to change the IP address in the client program.
2. How much time is now spent in the inter-remote-process communication, and how does that compare to the other setups?

Hints:
In our server code, we are already binding to all network interfaces (not just localhost).

- `asio::ip::udp::socket socket(io, asio::ip::udp::endpoint(asio::ip::udp::v4(), 8080));`

In our client code, we need to change the endpoint so that is point to the servers LAN IP address (assuming the two computers are on the same LAN. For example,

- udp::endpoint server_endpoint(asio::ip::make_address("192.168.1.10"), 8080);

To find the IP address of a computer, open a terminal window

- WindowsOS: in Powershell > ipconfig
  - Look for the **“IPv4 Address”** line under the active network adapter (e.g., “Wi-Fi” or “Ethernet”).
- LinuxOS Terminal > ip addr show
  - Look for the **inet** line under your active interface (like eth0 or wlp2s0).

Computers are designed to be protected from external network connections through built-in operating system firewalls and router firewalls.
If you are having problems connecting to a computer, you can try

1. “pinging” it, e.g.
   - Terminal on the client > ping 192.168.1.10
2. Netcat (nc)
   - Terminal on the server > nc -u -l 8080
   - Terminal on the client > echo "test" | nc -u 192.168.1.10 8080
   - Did you notice the pipe “|” in that command 😊
   - If you see the message, UDP routing is working.

In the solutions section, I’ve put an enhanced UDP server program that tells you its IP address when it starts up:

Solutions

Task 1: Communication with a unidirectional UDP socket and the ASIO library.

UDP Client (udp_client.cpp)

```
#define ASIO_STANDALONE
#include <asio.hpp>
#include <iostream>
#include <string>
#include <chrono>
#include <thread>

using udp = asio::ip::udp;  // Alias for cleaner code

int main() {
    try {
        asio::io_context io;
        udp::endpoint server_endpoint(asio::ip::make_address("127.0.0.1"), 8080);
        udp::socket socket(io, udp::v4()); // Create the socket and open it
        std::string message = "Hello from UDP client!";

        while (true) {
            socket.send_to(asio::buffer(message), server_endpoint);
            std::cout << "Message sent to server: " << message << std::endl;
            std::this_thread::sleep_for(std::chrono::seconds(1));  // Wait 1 second
        }
    }
    catch (std::exception& e) {
        std::cerr << "Client error: " << e.what() << std::endl;
    }

    return 0;
}
```

If you run your client, but there is no server running, what happens, and why?

- With UDP, this code will appear to run correctly even if there is no server!
- There is no handshake on opening the socket or when sending the data.
- If there is no server, then the data will disappear into the ether!
- Unlike TCP, with UDP, you can start the client first, and the server after.

Task 2: Communication with a bidirectional UDP socket and the ASIO library.

Server_code
```
#define ASIO_STANDALONE
#include <asio.hpp>
#include <iostream>
#include <array>

using udp = asio::ip::udp;

int main() {
    try {
        asio::io_context io;
        udp::socket socket(io, udp::endpoint(udp::v4(), 8080));
        std::cout << "UDP server listening on port 8080..." << std::endl;
        udp::endpoint client_endpoint;
        std::array<int, 1> buffer;

        while (true) {
            std::size_t bytes = socket.receive_from(asio::buffer(buffer), client_endpoint);
            if (bytes != sizeof(int)) continue;

            int value = buffer[0];
            value += 1;

            socket.send_to(asio::buffer(&value, sizeof(value)), client_endpoint);
        }
    }
    catch (std::exception& e) {
        std::cerr << "Server error: " << e.what() << std::endl;
    }
    return 0;
}
```

Client code
```
#define ASIO_STANDALONE
#include <asio.hpp>
#include <iostream>
#include <chrono>
#include <array>

using udp = asio::ip::udp;
using namespace std::chrono;

int main() {
    try {
        asio::io_context io;
        udp::socket socket(io);
        socket.open(udp::v4());
        udp::endpoint server_endpoint(asio::ip::make_address("127.0.0.1"), 8080);

        std::array<int, 1> buffer;
        buffer[0] = 1;

        auto start = high_resolution_clock::now();
        while (true) {
            // Send the integer
            socket.send_to(asio::buffer(buffer), server_endpoint);

            // Receive the updated value back
            udp::endpoint from;
            std::size_t bytes = socket.receive_from(asio::buffer(buffer), from);
            if (bytes != sizeof(int)) continue;
            // Increment and check stop condition
            buffer[0] += 1;
            if (buffer[0] >= 1'000'000) break;
        }
        auto end = high_resolution_clock::now();
        double elapsed_ms = duration_cast<milliseconds>(end - start).count();

        std::cout << "Final value: " << buffer[0] << std::endl;
        std::cout << "Elapsed time: " << elapsed_ms << " ms" << std::endl;
        std::cout << "Average time per round-trip: "
                  << (elapsed_ms * 1000.0 / 1000000) << " microseconds" << std::endl;
    }
    catch (std::exception& e) {
        std::cerr << "Client error: " << e.what() << std::endl;
    }
    return 0;
}
```

Task 3 Inter-remote-process communication with UDP.

Here's an enhanced UDP server program that tells you its IP address when it starts up:

UDP server

```
#define ASIO_STANDALONE
#include <asio.hpp>
#include <iostream>
#include <array>

using udp = asio::ip::udp;
using asio::ip::address_v4;

int main() {
    try {
        asio::io_context io;

        // Bind to all available IPv4 interfaces on port 8080
        udp::endpoint local_endpoint(udp::v4(), 8080);
        udp::socket socket(io, local_endpoint);

        // Print all local IPv4 addresses (for clarity)
        std::cout << "UDP server listening on port " << local_endpoint.port() << " ..." <<
std::endl;

        asio::ip::tcp::resolver resolver(io);
        asio::ip::tcp::resolver::query query(asio::ip::host_name(), "");
        auto results = resolver.resolve(query);

        std::cout << "Local IPv4 addresses detected:" << std::endl;
        for (auto& e : results) {
            auto addr = e.endpoint().address();
            if (addr.is_v4())
                std::cout << "  " << addr.to_string() << std::endl;
        }
        std::cout << "\nUse one of these addresses in your client code." << std::endl;
        std::cout << "--------------------------------------------\n" << std::endl;

        udp::endpoint client_endpoint;
        std::array<int, 1> buffer;

        while (true) {
            std::size_t bytes = socket.receive_from(asio::buffer(buffer), client_endpoint);
            if (bytes != sizeof(int)) continue;

            int value = buffer[0];
            value += 1;

            socket.send_to(asio::buffer(&value, sizeof(value)), client_endpoint);
        }
    }
    catch (std::exception& e) {
        std::cerr << "Server error: " << e.what() << std::endl;
    }

    return 0;
}
```