Info on Inter-process coms using TCP sockets

Table of Contents

Purpose – Implementing TCP practice

We discussed implementing TCP sockets in the handbook. This document goes over that information again and provides some code for you to test out on a computer.

How to implement ASIO TCP Server

For writing our sockets, we will use the asio.hpp header library.
In ASIO, there is an asio::io_context data type that is always needed and provides asynchronous operations via an event loop. But still needed for blocking I/O.

TCP server structure: Create io_context → create acceptor → accept → read/write.

- Create data structures needed to run the sockets:
  - tcp::endpoint (asio::ip::tcp::v4()) // create the information on the IP type, and the port number. You can also specify the required IP address (likely localhost), but if you don't, it will listen on all IP addresses on the PC.
  - acceptor(...endpoint) Wait for a connection request from a client
- Listen for a connection (wait/block) and accept it when it comes
  - acceptor.accept(socket); // Wait for a connection
  - The information about the client is stored in socket.remote_endpoint()
- To read data from the client, we need a buffer, then we read:
  - std::array<char, 1024> data;
  - std::size_t length = socket.read_some(asio::buffer(data));

Tcp_server.cpp

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

using asio::ip::tcp;

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

        // Create a TCP acceptor listening on port 65432
        tcp::acceptor acceptor(io, tcp::endpoint(tcp::v4(), 65432));
        std::cout << "TCP Server listening on port 65432\n";

        tcp::socket socket(io);
        acceptor.accept(socket);  // Wait for a connection
        std::cout << "Client connected from " << socket.remote_endpoint() << "\n";
        std::array<char, 1024> data;
        std::size_t length = socket.read_some(asio::buffer(data));
        std::cout << "Received: " << std::string(data.data(), length) << "\n";

        // Send reply
        std::string message = "Hello from server!";
        asio::write(socket, asio::buffer(message));
    }
    catch (std::exception &e) {
        std::cerr << "Error: " << e.what() << "\n";
    }
    std::cout << "Closing down\n";
}
```

How to implement ASIO TCP Client

TCP client structure: Create io_context → create socket → connect → read/write.

- Create information about what type of server service you want to connect to asio::ip::tcp::endpoint. This includes
    - IP address, converted from a string to the binary network representation using asio::ip::make_address("127.0.0.1")
- Connect our client socket the the requested server endpoint
    - socket.connect(endpoint); Request to connect to the server
- Write data to the server, we put our string into a buffer and write it:
    - asio::write(socket, asio::buffer(message));
- To read data from the server, we read into a buffer:
    - std::size_t length = socket.read_some(asio::buffer(data));

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

using asio::ip::tcp;

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

        tcp::socket socket(io);
        tcp::endpoint endpoint(asio::ip::make_address("127.0.0.1"), 65432);
        socket.connect(endpoint);
        std::cout << "Connected to TCP server\n";

        // Send message
        std::string message = "Hello from client!";
        asio::write(socket, asio::buffer(message));

        // Receive reply
        std::array<char, 1024> data;
        std::size_t length = socket.read_some(asio::buffer(data));
        std::cout << "Received: " << std::string(data.data(), length) << "\n";
    }
    catch (std::exception &e) {
        std::cerr << "Error: " << e.what() << "\n";
    }
    std::cout << "Closing down\n";
}
```