**\*\*\* -lboost\_system** when compile!

### **Creating an endpoint in the client to designate the server**

1. Obtain the server application's IP address and port number. The IP address should be specified as a string in the dot-decimal (IPv4) or hexadecimal (IPv6) notation.

2. Represent the raw IP address as an object of the asio::ip::address class.

3. Instantiate the object of the asio::ip::tcp::endpoint class from the address object created in step 2 and a port number.

4. The endpoint is ready to be used to designate the server application in Boost.Asio communication related methods.

#include <boost/asio.hpp>

#include <iostream>

using namespace boost;

int main()

{

// Step 1. Assume that the client application has already

// obtained the IP-address and the protocol port number.

std::string raw\_ip\_address = "127.0.0.1";

unsigned short port\_num = 3333;

// Used to store information about error that happens

// while parsing the raw IP-address.

boost::system::error\_code ec;

// Step 2. Using IP protocol version independent address

// representation.

asio::ip::address\_v4 ip\_address =

asio::ip::address::from\_string(raw\_ip\_address, ec);

if (ec.value() != 0) {

// Provided IP address is invalid. Breaking execution.

std::cout

<< "Failed to parse the IP address. Error code = "

<< ec.value() << ". Message: " << ec.message();

return ec.value();

}

// Step 3.

asio::ip::tcp::endpoint ep(ip\_address, port\_num);

// Step 4. The endpoint is ready and can be used to specify a

// particular server in the network the client wants to

// communicate with.

return 0;

}

### **Creating a server endpoint**

1. Obtain the protocol port number on which the server will listen for incoming requests.

2. Create a special instance of the asio::ip::address object representing all IP addresses available on the host running the server.

3. Instantiate an object of the asio::ip::tcp::endpoint class from the address object created in step 2 and a port number.

4. The endpoint is ready to be used to specify to the operating system that the server wants to listen for incoming messages on all IP addresses and a particular protocol port number.

#include <boost/asio.hpp>

#include <iostream>

using namespace boost;

int main()

{

// Step 1. Here we assume that the server application has

//already obtained the protocol port number.

unsigned short port\_num = 3333;

// Step 2. Create special object of asio::ip::address class

// that specifies all IP-addresses available on the host. Note

// that here we assume that server works over IPv6 protocol.

asio::ip::address ip\_address = asio::ip::address\_v6::any();

// Step 3.

asio::ip::tcp::endpoint ep(ip\_address, port\_num);

// Step 4. The endpoint is created and can be used to

// specify the IP addresses and a port number on which

// the server application wants to listen for incoming

// connections.

return 0;

}

**Creating an active socket**

Client:

1. Create an instance of the asio::io\_service class or use the one that has been created earlier.

2. Create an object of the class that represents the transport layer protocol (TCP or UDP) and the version of the underlying IP protocol (IPv4 or IPv6) over which the socket is intended to communicate.

3. Create an object representing a socket corresponding to the required protocol type. Pass the object of asio::io\_service class to the socket's constructor.

4. Call the socket's open() method, passing the object representing the protocol created in step 2 as an argument.

#include <boost/asio.hpp>

#include <iostream>

using namespace boost;

int main()

{

// Step 1. An instance of 'io\_service' class is required by

// socket constructor.

asio::io\_service ios;

// Step 2. Creating an object of 'tcp' class representing

// a TCP protocol with IPv4 as underlying protocol.

asio::ip::tcp protocol = asio::ip::tcp::v4();

// Step 3. Instantiating an active TCP socket object.

asio::ip::tcp::socket sock(ios);

// Used to store information about error that happens

// while opening the socket.

boost::system::error\_code ec;

// Step 4. Opening the socket.

sock.open(protocol, ec);

if (ec.value() != 0) {

// Failed to open the socket.

std::cout

<< "Failed to open the socket! Error code = "

<< ec.value() << ". Message: " << ec.message();

return ec.value();

}

return 0;

}

**Domain Name System (DNS)**

+ Doesn't name a particular physical device but an IP address that can be assigned to a device.

+ When given a DNS name, before a client can communicate with a corresponding server application, it must first resolve the name to obtain IP addresses associated with that name.

1. Obtain the DNS name and the protocol port number designating the server application and represent them as strings.

2. Create an instance of the asio::io\_service class or use the one that has been created earlier.

3. Create an object of the resolver::query class representing a DNS name resolution query.

4. Create an instance of DNS name resolver class suitable for the necessary protocol.

5. Call the resolver's resolve() method, passing a query object created in step 3 to it as an argument.

#include <boost/asio.hpp>

#include <iostream>

using namespace boost;

int main()

{

// Step 1. Assume that the client application has already

// obtained the DNS name and protocol port number and

// represented them as strings.

std::string host = "samplehost.com";

std::string port\_num = "3333";

// Step 2.

asio::io\_service ios;

// Step 3. Creating a query.

asio::ip::tcp::resolver::query resolver\_query(host,

port\_num, asio::ip::tcp::resolver::query::numeric\_service);

// Step 4. Creating a resolver.

asio::ip::tcp::resolver resolver(ios);

// Used to store information about error that happens

// during the resolution process.

boost::system::error\_code ec;

// Step 5.

asio::ip::tcp::resolver::iterator it =

resolver.resolve(resolver\_query, ec);

// Handling errors if any.

if (ec != 0) {

// Failed to resolve the DNS name. Breaking execution.

std::cout << "Failed to resolve a DNS name."

<< "Error code = " << ec.value()

<< ". Message = " << ec.message();

return ec.value();

}

return 0;

}

**Bind a socket to an endpoint**

+ Before an active socket can communicate with a remote application or a passive socket can accept incoming connection requests, they must be binded to a local IP address

1. Obtain the protocol port number on which the server should listen for incoming connection requests.

2. Create an endpoint that represents all IP addresses available on the host and the protocol port number obtained in the step 1.

3. Create and open an acceptor socket.

4. Call the acceptor socket's bind() method, passing the endpoint object as an argument to it.

#include <boost/asio.hpp>

#include <iostream>

using namespace boost;

int main()

{

// Step 1. Here we assume that the server application has

// already obtained the protocol port number.

unsigned short port\_num = 3333;

// Step 2. Creating an endpoint.

asio::ip::tcp::endpoint ep(asio::ip::address\_v4::any(),

port\_num);

// Used by 'acceptor' class constructor.

asio::io\_service ios;

// Step 3. Creating and opening an acceptor socket.

asio::ip::tcp::acceptor acceptor(ios, ep.protocol());

boost::system::error\_code ec;

// Step 4. Binding the acceptor socket.

acceptor.bind(ep, ec);

// Handling errors if any.

if (ec != 0) {

// Failed to bind the acceptor socket. Breaking

// execution.

std::cout << "Failed to bind the acceptor socket."

<< "Error code = " << ec.value() << ". Message: "

<< ec.message();

return ec.value();

}

return 0;

}