TCP ECHO SERVER

INTRODUCTION

1. Computer Networks

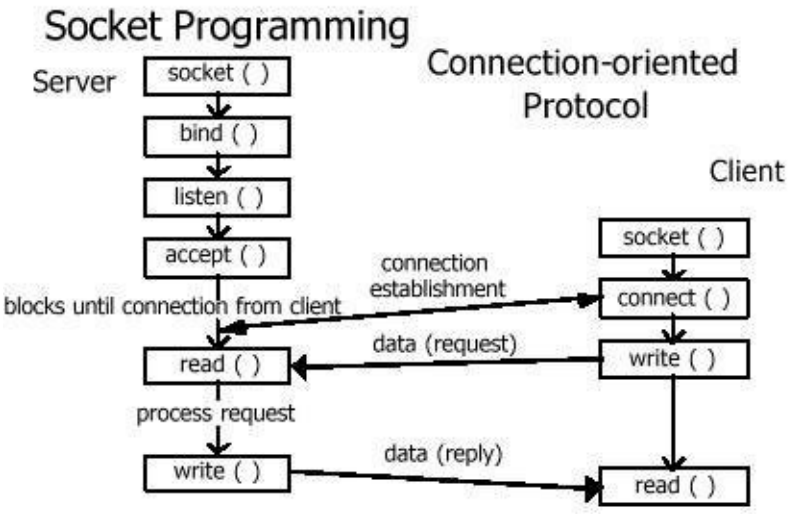
In computer networking, protocols are used often to define a set of rules and procedure that enable devices and systems to communicate. TCP refers to the Transmission Control Protocol, a core protocol is used to send data in the form of message units between computers and the Internet. TCP divides the message sent into packets(unit of data), and keeps track of this packets.

Server are computers that handle and process requests over the Internet. In networking, servers are accessed through a client-server programming model. Whereby a server waits for a request from the client, which could be a given application on a computer. When the client requests information from a server , the server receives the request and processes it by sending back some information to the client. For example, web browsers are used as clients to access web servers which hold web pages as html documents, when the request is received by the server, the documents are sent to the client, the web browser.

Echo servers are used to echo back the input to the client. When the client connects to the server, it starts sending strings, once the server receives the string , it sends it back to the client.

2. Socket Programming

In the client-server model , Sockets are used as the end points of communication and used by operating systems to provide programs access to a network.



TCP ECHO SERVER PROGRAMMING

The TCP Echo Server is programmed using the Boost.Asio libraries to create both a client and a server source file.

The client aspect is done with the following procedures:

1. Use a resolver to find the server socket on the expected host and port.
2. Create a client socket to connect to the server.
3. The client is asked to enter a message.
4. A terminator is used to signal when the server is done.

The server aspect is done with the following procedures:

1. Create an acceptor to find the socket on specific port.
2. Use a shared socket pointer to access buffer
3. Accept socket and connect
4. Read and write data back to echo

CODE

```
/**********echo_client.cpp***********/

#include <cstdlib>
#include <cstring>
#include <iostream>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;

enum { max_length = 1024 };

int main(int argc, char* argv[])
{
  try
  {
    if (argc != 3)
    {
      std::cerr << "Usage: blocking_tcp_echo_client <host> <port>\n";
      return 1;
    }

    boost::asio::io_service io_service;          //io_service represents your program's link to
                                                 //the operating system's I/O services.

    tcp::resolver resolver(io_service);
    tcp::resolver::query query(tcp::v4(), argv[1], argv[2]);
    tcp::resolver::iterator iterator = resolver.resolve(query);
```

```
    tcp::socket s(io_service);
    boost::asio::connect(s, iterator);

    using namespace std; // For strlen.
    std::cout << "Client Message: ";
    char request[max_length];
    std::cin.getline(request, max_length);
    size_t request_length = strlen(request);
    boost::asio::write(s, boost::asio::buffer(request, request_length));

    char reply[max_length];
    size_t reply_length = boost::asio::read(s,
        boost::asio::buffer(reply, request_length));
    std::cout << "Echo Message: ";
    std::cout.write(reply, reply_length);
    std::cout << "\n";
  }
  catch (std::exception& e)
  {
    std::cerr << "Exception: " << e.what() << "\n";
  }

  return 0;
}

/**********echo_server.cpp***********/

#include <cstdlib>
#include <iostream>
#include <boost/bind.hpp>
#include <boost/smart_ptr.hpp>
#include <boost/asio.hpp>
#include <boost/thread/thread.hpp>

using boost::asio::ip::tcp;

const int max_length = 1024;

typedef boost::shared_ptr<tcp::socket> socket_ptr;

void session(socket_ptr sock)
{
  try
  {
    for (;;)
    {
      char data[max_length];
```

```
      boost::system::error_code error;
      size_t length = sock->read_some(boost::asio::buffer(data), error);
      if (error == boost::asio::error::eof)
        break;                                  // Connection closed cleanly by peer.
      else if (error)
        throw boost::system::system_error(error); // Some other error.

      boost::asio::write(*sock, boost::asio::buffer(data, length));
    }
  }
  catch (std::exception& e)
  {
    std::cerr << "Exception in thread: " << e.what() << "\n";
  }
}

void server(boost::asio::io_service& io_service, unsigned short port)
{
  tcp::acceptor a(io_service, tcp::endpoint(tcp::v4(), port));
  for (;;)
  {
    socket_ptr sock(new tcp::socket(io_service));
    a.accept(*sock);
    boost::thread t(boost::bind(session, sock));
  }
}

/********** main ***************/
int main(int argc, char* argv[])
{
  try
  {
    if (argc != 2)
    {
      std::cerr << "Usage: blocking_tcp_echo_server <port>\n";
      return 1;
    }

    boost::asio::io_service io_service;

    using namespace std;                        // For atoi.
    server(io_service, atoi(argv[1]));
  }
  catch (std::exception& e)
  {
    std::cerr << "Exception: " << e.what() << "\n";
  }

  return 0;
```

}