Boost.Asio

- Boost.Asio: Asynchronous I/O in Boost Library.

References:

- Boost.Asio documentation:
  http://www.boost.org/libs/asio/
- Boost.Asio tutorial:
  http://www.boost.org/doc/libs/1_62_0/doc/html/boost_asio/tutorial.html
- Boost.Asio examples:
  http://www.boost.org/doc/libs/1_62_0/doc/html/boost_asio/examples.html
- Online book:
  http://en.highscore.de/cpp/boost/asio.html
  http://www.highscore.de/cpp/boost/asio.html (German)
  http://zh.highscore.de/cpp/boost/asio.html (Chinese)
- JTC1/SC22/WG21 paper N3389:
  http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2012/n3389.pdf

Boost.Asio

- Boost.Asio
  - is a cross-platform C++ library for network and low-level I/O programming
  - provides developers with a consistent asynchronous model in C++.
- Support **`kqueue()`**, **`epoll()`**, **`/dev/poll`**, and the traditional **`select()`**

Equivalents: Socket API to Boost.Asio

BSD Socket API Elements	Equivalents in Boost.Asio
socket descriptor - int(POSIX) or SOCKET(Windows)	For TCP: ip::tcp::socket, ip::tcp::acceptor
in_addr, in6_addr	ip::address, ip::address_v4, ip::address_v6
sockaddr_in, sockaddr_in6	For TCP: ip::tcp::endpoint
socket()	For TCP: ip::tcp::acceptor::open(), ip::tcp::socket::open()
poll(), select(),pselect()	io_service::run(), io_service::run_one(), io_service::poll(), io_service::poll_one() Note: in conjunction with asynchronous operations.

BSD Socket API Elements	Equivalents in Boost.Asio
ioctl()	For TCP: ip::tcp::socket::io_control()
listen()	For TCP: ip::tcp::acceptor::listen() basic_socket_acceptor::listen()
accept()	For TCP: ip::tcp::acceptor::accept() basic_socket_acceptor::accept()
bind()	For TCP: ip::tcp::acceptor::bind(), ip::tcp::socket::bind()
close()	For TCP: ip::tcp::acceptor::close(), ip::tcp::socket::close()
connect()	For TCP: ip::tcp::socket::connect()

BSD Socket API Elements	Equivalents in Boost.Asio
readv(), recv(),read()	For TCP: ip::tcp::socket::read_some(), ip::tcp::socket::async_read_some(), ip::tcp::socket::receive(), ip::tcp::socket::async_receive()
recvfrom()	For UDP: ip::udp::socket::receive_from(), ip::udp::socket::async_receive_from() basic_datagram_socket::receive_from(), basic_datagram_socket::async_receive_from()
send(), write(),writev()	For TCP: ip::tcp::socket::write_some(), ip::tcp::socket::async_write_some(), ip::tcp::socket::send(), ip::tcp::socket::async_send()
sendto()	For UDP: ip::udp::socket::send_to(), ip::udp::socket::async_send_to() basic_datagram_socket::send_to(), basic_datagram_socket::async_send_to()

BSD Socket API Elements	Equivalents in Boost.Asio
getaddrinfo(), gethostbyaddr(), gethostbyname(), getnameinfo(), getservbyname(), getservbyport()	For TCP: ip::tcp::resolver::resolve(), ip::tcp::resolver::async_resolve()
gethostname()	ip::host_name()
getpeername()	For TCP: ip::tcp::socket::remote_endpoint()
getsockname()	For TCP: ip::tcp::acceptor::local_endpoint(), ip::tcp::socket::local_endpoint()
getsockopt()	For TCP: ip::tcp::acceptor::get_option(), ip::tcp::socket::get_option()

Example: Client

- From https://theboostcpplibraries.com/boost.asio-network-programming

```
#include <boost/asio/io_service.hpp>
#include <boost/asio/write.hpp>
#include <boost/asio/buffer.hpp>
#include <boost/asio/ip/tcp.hpp>
#include <array>
#include <string>
#include <iostream>

using namespace boost::asio;
using namespace boost::asio::ip;

io_service ioservice;
tcp::resolver resolv{ioservice};
tcp::socket tcp_socket{ioservice};
std::array<char, 4096> bytes;
```

```
void read_handler(const boost::system::error_code &ec,
  std::size_t bytes_transferred)
{
  if (!ec)
  {
    std::cout.write(bytes.data(), bytes_transferred);
    tcp_socket.async_read_some(buffer(bytes), read_handler);
  }
}

void connect_handler(const boost::system::error_code &ec)
{
  if (!ec)
  {
    std::string r =
      "GET / HTTP/1.1\r\nHost: theboostcpplibraries.com\r\n\r\n";
    write(tcp_socket, buffer(r));
    tcp_socket.async_read_some(buffer(bytes), read_handler);
  }
}
```

```
void resolve_handler(const boost::system::error_code &ec,
  tcp::resolver::iterator it)
{
  if (!ec)
    tcp_socket.async_connect(*it, connect_handler);
}

int main()
{
  tcp::resolver::query q{"theboostcpplibraries.com", "80"};
  resolv.async_resolve(q, resolve_handler);
  ioservice.run()
}
```

Example: Server

```
#include <boost/asio/io_service.hpp>
#include <boost/asio/write.hpp>
#include <boost/asio/buffer.hpp>
#include <boost/asio/ip/tcp.hpp>
#include <string>
#include <ctime>

using namespace boost::asio;
using namespace boost::asio::ip;

io_service ioservice;
tcp::endpoint tcp_endpoint{tcp::v4(), 2014};
tcp::acceptor tcp_acceptor{ioservice, tcp_endpoint};
tcp::socket tcp_socket{ioservice};
std::string data;
```

```
void write_handler(const boost::system::error_code &ec,
  std::size_t bytes_transferred)
{
  if (!ec)
    tcp_socket.shutdown(tcp::socket::shutdown_send);
}

void accept_handler(const boost::system::error_code &ec)
{
  if (!ec)
  {
    std::time_t now = std::time(nullptr);
    data = std::ctime(&now);
    async_write(tcp_socket, buffer(data), write_handler);
  }
}

int main()
{
  tcp_acceptor.listen();
  tcp_acceptor.async_accept(tcp_socket, accept_handler);
  ioservice.run();
}
```

Boost.Asio and Threads

- Cross Thread

```
int Thread1Run() {   ...  async_read(ioservice, read_handler);   ... }
int Thread2Run() {   ...  ioservice.run();   ... }
int read_handler(...) {   ...  /* handle read event */ ... }
```

- Thread Safety
  - In general, it is safe to make concurrent use of distinct objects, but unsafe to make concurrent use of a single object.
  - However, types such as io_service provide a stronger guarantee that it is safe to use a single object concurrently.
    - Safe, with the specific exceptions of the reset() and notify_fork() functions. (See http://www.boost.org/doc/libs/1_62_0/doc/html/boost_asio/reference/io_service.html)
- Thread Pools
  - Multiple threads may call io_service::run() to set up a pool of threads from which completion handlers may be invoked.

```
int Thread1Run() {   ...  ioservice.run();   ... }
int Thread2Run() {   ...  ioservice.run();   ... }
```

- Strands: Use threads without explicit locking.
  - A strand is defined as a strictly sequential invocation of event handlers (i.e. no concurrent invocation).