



# ASIO

MM4220 게임서버 프로그래밍  
정내훈

# 내용

---

- BOOST/ASIO 소개
- Boost/ASIO API
- Boost/ASIO 실습

# BOOST/ASIO

- Boost

- 1999년 9월 1일에 첫 공개
- C++ 프로그래밍에 사용되는 여러가지 유용한 라이브러리들의 집합
  - 선형대수, 랜덤, 멀티쓰레딩, 영상처리, 정규식, 유닛테스팅, 리플렉션... 등 160가지
- 현재 1.82가 최신버전 (2023년)
- C++11, C++17, C++20에 많이 들어갔고, C++23에도 많이 들어갈 예정

# BOOST

- Boost
  - 많은 라이브러리들이 header file로 구현됨.
    - Template Programming
    - 장점 : 호환성, 실행 속도
    - 단점 : 가독성, 컴파일 속도
    - 일부는 라이브러리 필요 (예: ASIO)
  - 라이선스는 Boost 라이선스 : 자유롭게 이용 가능
  - 한번 맛을 들이면 헤어날 수 없음
    - 예) C++ 표준위원회

# ASIO

---

- Asio C++ Library ([think-async.com](http://think-async.com))
  - 이곳에서 다운 받으면, boost 제외하고 asio만 설치 가능
- 2003년 부터 개발 시작
- 2005년에 Boost에 채용

# ASIO

---

- Cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent **asynchronous** model using a modern C++ approach
  - [https://www.boost.org/doc/libs/1\\_79\\_0/doc/html/boost\\_asio.html](https://www.boost.org/doc/libs/1_79_0/doc/html/boost_asio.html)

# ASIO

- 특징
  - IOCP와는 달리 C++ API이다.
  - 기존의 Socket API를 전부 C++ API로 재작성 했다.
    - 기존 socket 사용 프로그램은 전부 재작성 해야 한다.
    - C++ API이므로 매우 직관적이고, 사용하기 편하다.
      - 과거의 잔재가 없다.
    - Lambda 함수에 매우 의존적이다.
    - Socket 객체의 관리에 주의 해야 한다.
      - destructor가 호출되어야 커널 자료구조가 반환된다
      - 포인터가 강제 된다. 멀티쓰레드 사용 시 shared\_ptr가 강제 된다.
  - 동기식 동작도 가능하다.
    - 사용법이 매우 간단하다.
    - 기존 Socket API를 C++ Layer로 감싼 것

# ASIO

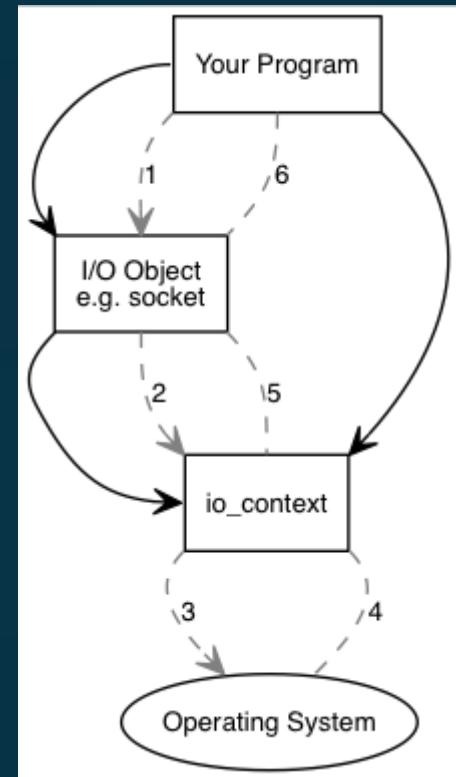
- 동작 : 동기식
  - 준비 : io\_context와 socket이 필요

```
boost::asio::io_context io_context;  
boost::asio::ip::tcp::socket socket(io_context);
```

## 1. socket API 호출

- **socket.connect**(server\_endpoint);

2. socket이 요청을 io\_context에 전달
3. io\_context가 운영체제 호출
4. 운영체제가 결과를 io\_context에 리턴
5. io\_context가 에러 처리 후 socket에 결과 전달
6. socket은 결과를 프로그램에 전달.





# ASIO

- 동작 : 비동기식 (시작 부분)

1. socket API 호출

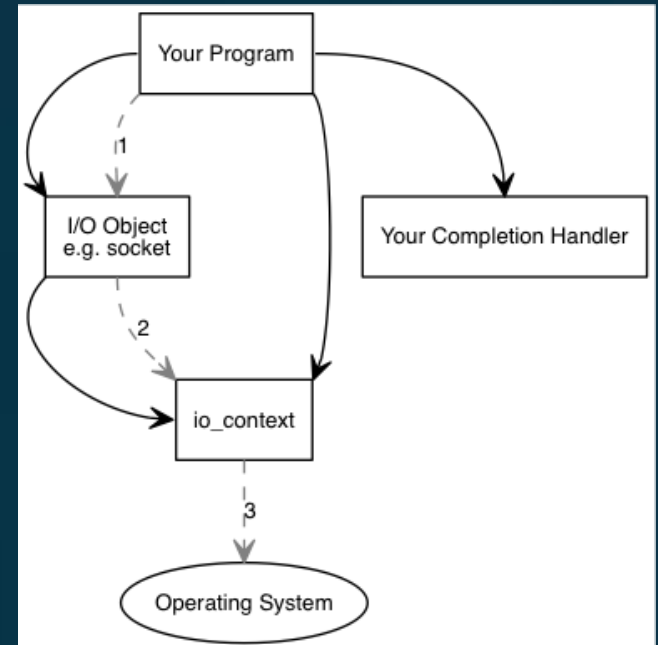
```
socket.async_connect(server_p, my_handler);
```

- my\_handler는 완료했을 때 호출되는 함수

```
void my_handler(const boost::system::error_code& ec);
```

2. socket이 io\_context에 전송

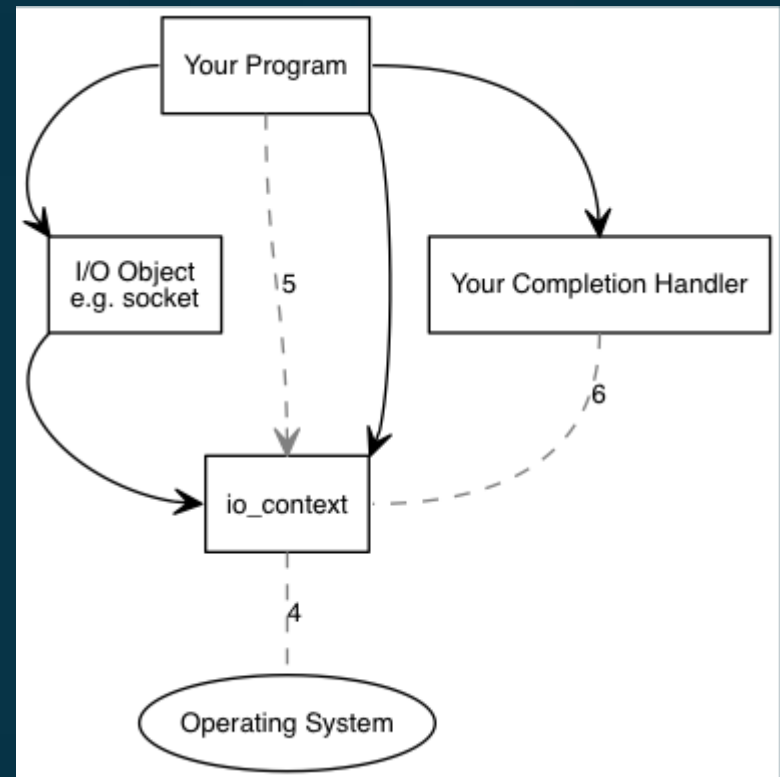
3. io\_context가 운영체제 호출



# ASIO

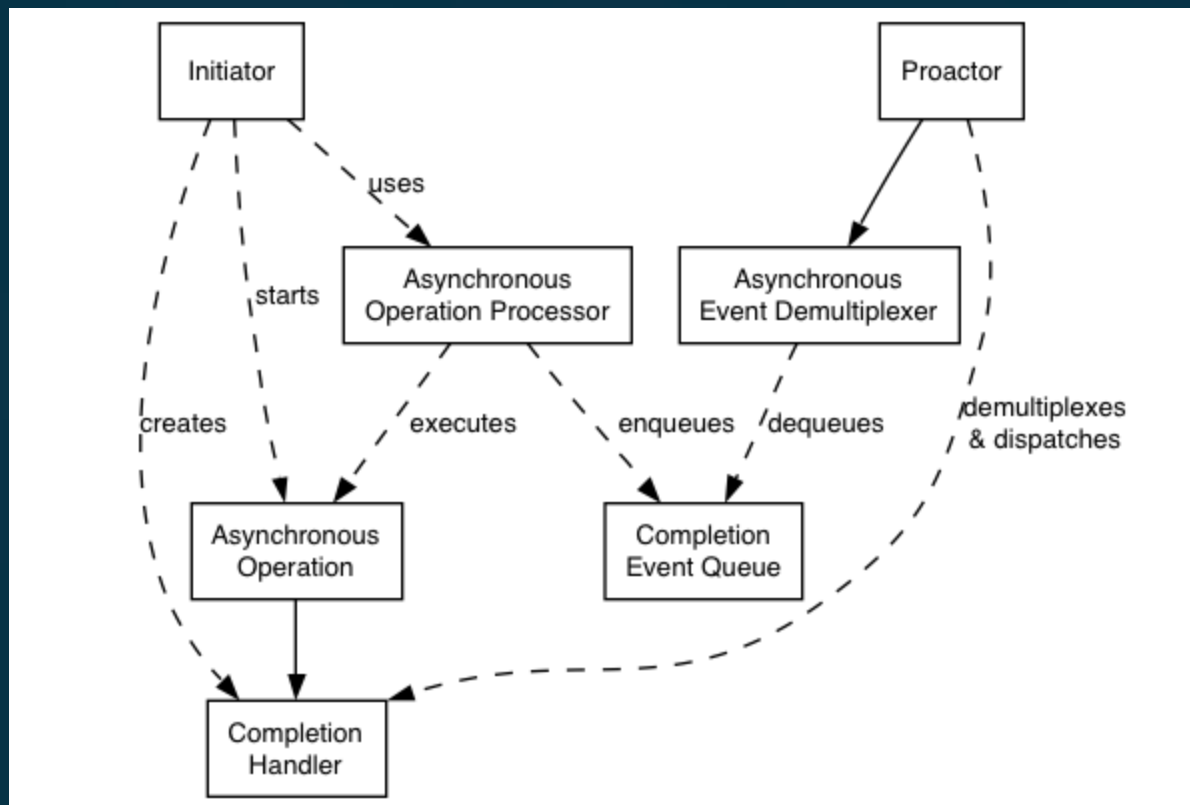
- 동작 : 비동기식 (완료 부분)

4. 운영체제가 결과를 queue에 넣어놓고 io\_context에 완료를 알려줌.
5. 프로그램에서 `io_context::run()`을 호출  
`run()`은 완료를 발견할 때 까지 block
6. `run()`이 완료를 발견하면 결과를 queue에서 빼낸 후 `my_handler`에 전달.



# ASIO

- 비동기 구조 : Proactor 패턴
  - 내부에 event queue를 갖고 epoll같은 reactor 패턴도 Proactor 패턴으로 변환



# ASIO

- IOCP와의 차이

```
worker_thread()  
{  
    while(true) {  
        GQCS(g_iocp,...);  
        if (is_recv)  
            ProcessBuffer();  
    }  
}
```

```
WSARecv(sock, WSABUF)
```

```
worker_thread()  
{  
    io_context->run();  
}
```

```
sock.async_read(  
    buffers,  
    ProcessBuffer();
```

# ASIO

- 멀티쓰레드 연동
  - `io_context` 객체에 대한 모든 접근은 thread safty를 보장한다.
  - 여러 개의 쓰레드에서 `io_context::run()`을 호출함으로서 병렬성을 얻을 수 있다.
  - 내부적으로 비동기처리를 위해 별도의 쓰레드를 사용할 수 있는데, 이것의 동작은 최소화 되어있고, 사용자에게 숨겨져 있다.

# ASIO

- Strands

- handler가 여러 쓰레드에서 동시에 처리될 때 순서가 뒤바뀌는 것을 막기위한 구조
- handler를 같은 strand에 등록하면 handler의 호출이 직렬화 된다.
  - 원래는 mutex를 사용해서 직렬화 해야 하나. strand를 사용하면 mutex없이 효율적으로 커널레벨에서 직렬화 된다고 한다. => 거짓말. Linux의 경우 내부적으로 mutex를 사용해서 직렬화 한다. => 성능 하락의 주범. 쓰지 말 것.

# 내용 (2023-화목)

---

- BOOST/ASIO 소개
- Boost/ASIO API
- Boost/ASIO 실습

# ASIO 준비

- TCP API를 전부 새로 작성
  - 따라서, 기존의 socket api 재사용은 전무하다.
  - 주소는 `ip::tcp::endpoint` 클래스로 관리한다.
    - 기존의 `struct sockaddr`를 대체
  - Name resolving은 `ip::tcp::resolver` 클래스를 사용한다.

```
ip::tcp::resolver resolver(my_io_context);
ip::tcp::resolver::query query("www.boost.org", "http");
ip::tcp::resolver::iterator iter = resolver.resolve(query);
ip::tcp::resolver::iterator end; // End marker.
while (iter != end)
{
    ip::tcp::endpoint endpoint = *iter++;
    std::cout << endpoint << std::endl;
}
```



# ASIO 준비

- 클라이언트 API

```
ip::tcp::socket socket(my_io_context);  
boost::asio::connect(socket, resolver.resolve(query));  
또는 socket.connect(resolver.resolve(query));
```

# ASIO 준비

- 서버 API

```
ip::tcp::acceptor acceptor(my_io_context, my_endpoint);  
...  
ip::tcp::socket socket(my_io_context);  
acceptor.accept(socket);
```

# ASIO 준비

- 송수신 API

```
size_t boost::asio::ip::tcp::socket::write_some(  
    const ConstBufferSequence& buffers,  
    boost::system::error_code& ec  
)  
size_t boost::asio::ip::tcp::socket::read_some(  
    const ConstBufferSequence& buffers,  
    boost::system::error_code& ec  
)
```

# ASIO 준비

- 비동기 송수신 API

```
boost::asio::async_write(AsyncWriteStream& s,  
    const ConstBufferSequence& buffers,  
    BOOST_ASIO_MOVE_ARG(WriteHandler) handler);  
boost::asio::ip::tcp::socket::async_read_some(  
    const MutableBufferSequence& buffers,  
    BOOST_ASIO_MOVE_ARG(ReadHandler) handler);
```

```
void handler(  
    const boost::system::error_code& error,  
    std::size_t bytes_transferred;  
);
```

# ASIO 준비

BSD Socket API Elements	Equivalents in Boost.Asio
socket descriptor - int (POSIX) or SOCKET (Windows)	For TCP: <a href="#">ip::tcp::socket</a> , <a href="#">ip::tcp::acceptor</a> For UDP: <a href="#">ip::udp::socket</a> <a href="#">basic_socket</a> , <a href="#">basic_stream_socket</a> , <a href="#">basic_datagram_socket</a> , <a href="#">basic_raw_socket</a>
in_addr, in6_addr	<a href="#">ip::address</a> , <a href="#">ip::address_v4</a> , <a href="#">ip::address_v6</a>
sockaddr_in, sockaddr_in6	For TCP: <a href="#">ip::tcp::endpoint</a> For UDP: <a href="#">ip::udp::endpoint</a> <a href="#">ip::basic_endpoint</a>
accept()	For TCP: <a href="#">ip::tcp::acceptor::accept()</a> <a href="#">basic_socket_acceptor::accept()</a>
bind()	For TCP: <a href="#">ip::tcp::acceptor::bind()</a> , <a href="#">ip::tcp::socket::bind()</a> For UDP: <a href="#">ip::udp::socket::bind()</a> <a href="#">basic_socket::bind()</a>
close()	For TCP: <a href="#">ip::tcp::acceptor::close()</a> , <a href="#">ip::tcp::socket::close()</a> For UDP: <a href="#">ip::udp::socket::close()</a> <a href="#">basic_socket::close()</a>
connect()	For TCP: <a href="#">ip::tcp::socket::connect()</a> For UDP: <a href="#">ip::udp::socket::connect()</a> <a href="#">basic_socket::connect()</a>
getaddrinfo(), gethostbyaddr(), gethostbyname(), getnameinfo(), getservbyname(), getservbyport()	For TCP: <a href="#">ip::tcp::resolver::resolve()</a> , <a href="#">ip::tcp::resolver::async_resolve()</a> For UDP: <a href="#">ip::udp::resolver::resolve()</a> , <a href="#">ip::udp::resolver::async_resolve()</a> <a href="#">ip::basic_resolver::resolve()</a> , <a href="#">ip::basic_resolver::async_resolve()</a>
gethostname()	<a href="#">ip::host_name()</a>
getpeername()	For TCP: <a href="#">ip::tcp::socket::remote_endpoint()</a> For UDP: <a href="#">ip::udp::socket::remote_endpoint()</a> <a href="#">basic_socket::remote_endpoint()</a>
getsockname()	For TCP: <a href="#">ip::tcp::acceptor::local_endpoint()</a> , <a href="#">ip::tcp::socket::local_endpoint()</a> For UDP: <a href="#">ip::udp::socket::local_endpoint()</a> <a href="#">basic_socket::local_endpoint()</a>
getsockopt()	For TCP: <a href="#">ip::tcp::acceptor::get_option()</a> , <a href="#">ip::tcp::socket::get_option()</a> For UDP: <a href="#">ip::udp::socket::get_option()</a> <a href="#">basic_socket::get_option()</a>
inet_addr(), inet_aton(), inet_pton()	<a href="#">ip::address::from_string()</a> , <a href="#">ip::address_v4::from_string()</a> , <a href="#">ip::address_v6::from_string()</a>
inet_ntoa(), inet_ntop()	<a href="#">ip::address::to_string()</a> , <a href="#">ip::address_v4::to_string()</a> , <a href="#">ip::address_v6::to_string()</a>

# ASIO 준비

BSD Socket API Elements	Equivalents in Boost.Asio
ioctl()	For TCP: <a href="#">ip::tcp::socket::io control()</a> For UDP: <a href="#">ip::udp::socket::io control()</a> <a href="#">basic socket::io control()</a>
listen()	For TCP: <a href="#">ip::tcp::acceptor::listen()</a> <a href="#">basic socket acceptor::listen()</a>
poll(), select(), pselect()	<a href="#">io context::run()</a> , <a href="#">io context::run one()</a> , <a href="#">io context::poll()</a> , <a href="#">io context::poll one()</a> Note: in conjunction with asynchronous operations.
readv(), recv(), read()	For TCP: <a href="#">ip::tcp::socket::read some()</a> , <a href="#">ip::tcp::socket::async read some()</a> , <a href="#">ip::tcp::socket::receive()</a> , <a href="#">ip::tcp::socket::async receive()</a> For UDP: <a href="#">ip::udp::socket::receive()</a> , <a href="#">ip::udp::socket::async receive()</a> <a href="#">basic stream socket::read some()</a> , <a href="#">basic stream socket::async read some()</a> , <a href="#">basic stream socket::receive()</a> , <a href="#">basic stream socket::async receive()</a> , <a href="#">basic datagram socket::receive()</a> , <a href="#">basic datagram socket::async receive()</a>
recvfrom()	For UDP: <a href="#">ip::udp::socket::receive from()</a> , <a href="#">ip::udp::socket::async receive from()</a> <a href="#">basic datagram socket::receive from()</a> , <a href="#">basic datagram socket::async receive from()</a>
send(), write(), writev()	For TCP: <a href="#">ip::tcp::socket::write some()</a> , <a href="#">ip::tcp::socket::async write some()</a> , <a href="#">ip::tcp::socket::send()</a> , <a href="#">ip::tcp::socket::async send()</a> For UDP: <a href="#">ip::udp::socket::send()</a> , <a href="#">ip::udp::socket::async send()</a> <a href="#">basic stream socket::write some()</a> , <a href="#">basic stream socket::async write some()</a> , <a href="#">basic stream socket::send()</a> , <a href="#">basic stream socket::async send()</a> , <a href="#">basic datagram socket::send()</a> , <a href="#">basic datagram socket::async send()</a>
sendto()	For UDP: <a href="#">ip::udp::socket::send to()</a> , <a href="#">ip::udp::socket::async send to()</a> <a href="#">basic datagram socket::send to()</a> , <a href="#">basic datagram socket::async send to()</a>
setsockopt()	For TCP: <a href="#">ip::tcp::acceptor::set option()</a> , <a href="#">ip::tcp::socket::set option()</a> For UDP: <a href="#">ip::udp::socket::set option()</a> <a href="#">basic socket::set option()</a>
shutdown()	For TCP: <a href="#">ip::tcp::socket::shutdown()</a> For UDP: <a href="#">ip::udp::socket::shutdown()</a> <a href="#">basic socket::shutdown()</a>
socketatmark()	For TCP: <a href="#">ip::tcp::socket::at mark()</a> <a href="#">basic socket::at mark()</a>
socket()	For TCP: <a href="#">ip::tcp::acceptor::open()</a> , <a href="#">ip::tcp::socket::open()</a> For UDP: <a href="#">ip::udp::socket::open()</a> <a href="#">basic socket::open()</a>
socketpair()	<a href="#">local::connect pair()</a> Note: POSIX operating systems only.

# ASIO 구현

- 운영체제 별로 최신 API를 사용해 구현
  - Linux Kernel 2.6부터 : epoll
  - Mac OS X : kqueue
  - Windows : IOCP
- 원래는 소스를 다운받아 컴파일해야 하지만, Visual Studio의 Nuget사용
  - boost-v143를 설치할 것. (Visual Studio 2022)
  - asio만 받으면 좋은데 따로 떨어져 있지 않아서 전체를 받아야함. 한 시간정도 걸림

# 내용

---

- BOOST/ASIO 소개
- Boost/ASIO API
- Boost/ASIO 실습



# 실습

---

- eClass 자료실에서 다운

# ASIO

- 실습
  - 설치
    - [www.boost.org](http://www.boost.org)에서 다운받아서 설치
    - 또는, Nuget사용



# ASIO 예제 : 에코 클라이언트

```
#include <iostream>
#include <SDKDDKVER.h>
#include <boost/asio.hpp>

using namespace std;

int main()
{
    try {
        boost::asio::io_context io_context;
        boost::asio::ip::tcp::endpoint server_addr(boost::asio::ip::address::from_string("127.0.0.1"), 3500);
        boost::asio::ip::tcp::socket socket(io_context);
        boost::asio::connect(socket, &server_addr);
        for (;;) {
            . . . // 입력 -> 전송 -> 수신 -> 출력
        }
    }
    catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }
}
```

# ASIO 예제 : 에코 클라이언트

```
for (;;) {
    std::string buf;
    boost::system::error_code error;

    std::cout << "Enter Message: ";
    std::getline(std::cin, buf);
    if (0 == buf.size()) break;

    socket.write_some(boost::asio::buffer(buf), error);
    if (error == boost::asio::error::eof) break;
    else if (error) throw boost::system::system_error(error);

    char reply[1024 + 1];
    size_t len = socket.read_some(boost::asio::buffer(reply, 1024), error);
    if (error == boost::asio::error::eof) break;
    else if (error) throw boost::system::system_error(error);

    reply[len] = 0;
    std::cout << len << " bytes received: " << reply << endl;
}
```

# ASIO 예제 : 에코 클라이언트

---

- 실습 : 컴파일 및 실행
  - Visual Studio 2022

# ASIO 예제 : 에코 서버

```
#include <iostream>
#include <sdkddkver.h>
#include <unordered_map>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;
using namespace std;

constexpr int PORT = 3500;

class session;
unordered_map <int, session> g_clients;
int g_client_id = 0;

void accept_callback(boost::system::error_code ec, tcp::socket& socket, tcp::acceptor& my_acceptor);

int main(int argc, char* argv[])
{
    try {
        boost::asio::io_context io_context;
        tcp::acceptor my_acceptor{ io_context, tcp::endpoint(tcp::v4(), PORT) };
        my_acceptor.async_accept([&my_acceptor](boost::system::error_code ec, tcp::socket socket) {
            accept_callback(ec, socket, my_acceptor); });
        io_context.run();
    } catch (std::exception& e) {
        std::cerr << "Exception: " << e.what() << "\n";
    }
}
```

# ASIO 예제 : 에코 서버

```
class session
{
    int my_id;
    tcp::socket socket_;
    enum { max_length = 1024 };
    char data_[max_length];

public:
    session() : socket_(nullptr) {
        cout << "Session Creation Error.\n";
    }
    session(tcp::socket socket, int id) : socket_(std::move(socket)), my_id(id) {
        do_read();
    }
    void do_read() {
        socket_.async_read_some(boost::asio::buffer(data_, max_length),
            [this](boost::system::error_code ec, std::size_t length) {
                if (ec) g_clients.erase(my_id);
                else g_clients[my_id].do_write(length); });
    }
    void do_write(std::size_t length) {
        boost::asio::async_write(socket_, boost::asio::buffer(data_, length),
            [this](boost::system::error_code ec, std::size_t /*length*/) {
                if (!ec) g_clients[my_id].do_read();
                else g_clients.erase(my_id); });
    }
};
```

# ASIO 예제 : 에코 서버

```
void accept_callback(boost::system::error_code ec, tcp::socket& socket, tcp::acceptor& my_acceptor)
{
    g_clients.try_emplace(g_client_id, move(socket), g_client_id);
    g_client_id++;

    my_acceptor.async_accept([&my_acceptor](boost::system::error_code ec, tcp::socket socket) {
        accept_callback(ec, socket, my_acceptor);
    });
}
```