
Hanbit eBook
Realtime 27
Boost.Asio를 이용한
네트워크 프로그래밍
최흥배 지음
한빛미디어
Hanbit Media, Inc.

Boost.Asio를 이용한 네트워크 프로그래밍

지은이_ **최흥배**

2003년부터 현재에 이르기까지, 보드 게임부터 MMORPG 게임을 아우르는 다양한 온라인 게임의 서버 프로그램을 만들어온 개발자다. 게임 개발자로서 프로그래밍 언어 중 C++을 주 언어로, C#을 보조 언어로 사용하고 있다. 요즘은 C++11 프로그래밍과 Linux 플랫폼 프로그래밍, 심도 있는 .NET 기술, JavaScript에 대해 공부하고 있다. 기술이나 경험을 공유하는 것을 좋아하여 게임 개발자 커뮤니티나 세미나 강연을 통해 다른 프로그래머와 활발히 교류한다. 웹이 대중화되기 전부터 프로그래밍 공부를 해와서 그런지 여전히 새로운 기술을 배울 때는 책을 더 선호하여, 지금도 매달 새로운 프로그래밍 관련 책을 읽으며 연구하고 있다. 현재 티쓰리엔터테인먼트 삼국지천 팀에서 근무 중이다.

Boost.Asio를 이용한 네트워크 프로그래밍

초판발행 2013년 6월 4일

지은이 최흥배 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-611-1 15000 / **정가** 9,900원

책임편집 배용석 / **기획** 김병희 / **교정** 안선화
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
마케팅 박상용, 박주훈, 정민하

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanb.co.kr / **이메일** ask@hanb.co.kr

Published by HANBIT Media, Inc. Printed in Korea

저자 서문

필자가 처음 온라인 게임 서버를 만들 때만 해도, 이해하기 어려운 IOCP 방식보다 기존의 BSD Socket 방식이 네트워크 프로그래밍(Windows 플랫폼에서)에 더 좋다는 논의로 프로그래머 커뮤니티 사이트에서 격렬한 논쟁이 벌어지기도 했었다. 그러나 지금은 고성능 네트워크 프로그래밍에 당연하다는 듯이 IOCP를 사용하는 비동기 IO 방식을 사용한다.

Windows 플랫폼에서 고성능 네트워크 프로그램을 만들 때 필수적이기는 하지만, 사용 방법이 간단하지 않고 다른 플랫폼(가령, Linux)에서 사용할 수 없다는 단점이 있다(타 플랫폼의 비동기 프로그래밍 방식도 비슷하다). 네트워크 프로그래밍 경험이 적은 경우라면 학습하는 데 적지 않은 시간이 소요되고, IOCP로 만든 프로그램이 실제로 올바르게 동작하지 않을 수도 있다. 또 요즘은 방대한 유저를 수용하기 위해 많은 서버 컴퓨터를 운영하고 있는데, OS 라이센스 문제로 무료 플랫폼인 Linux의 사용 역시 증가하는 추세다. 하지만 Linux에서는 IOCP로 만든 프로그램을 사용할 수 없으므로 프로그램을 새로 만들어야 한다는 문제가 있다.

위에서 밝힌 문제들을 해결할 수 있는 방법의 하나가 C++ 오픈 소스 라이브러리로 유명한 Boost 라이브러리의 'Asio'를 사용하는 것이다. Asio를 사용하면 비동기 IO 프로그래밍을 손쉽게 사용할 수 있고, 코드가 간결해져서 유지보수가 좋아진다. 또한, Boost가 멀티 플랫폼을 지원하기 때문에 Windows 플랫폼에서 만든 프로그램을 거의 수정 없이 Linux로 전환할 수 있다.

C++ 프로그래머라면 대부분 Boost 라이브러리를 알고 있을 테지만, 아직 이에 관한 국내 출판물은 없는 상태다. 그래서 Boost 라이브러리의 일부만 알고 있는 경우가 많으며, 아직은 Asio에 관해서도 잘 모르는 사람이 많은 것 같다. 이 책을 통해서 Asio를 쉽게 공부하여 하루빨리 고성능 네트워크 프로그램을 만들 수 있게 되길

바란다.

이 책은 전자책으로 출판되는 만큼 배포도 쉬울 것이라 생각한다. 필자는 이러한 전자책의 장점을 잘 활용하여, 책에 싣지 못한 Asio에 대한 정보나 팁 등을 출판 이후에도 가능하면 지속적으로 업데이트해 나가려고 한다.

책을 준비하며 도움을 준 사람들에게 감사의 마음을 전한다. 먼저, 결혼 이후에도 집에서 프로그래밍 공부를 할 수 있도록 많이 배려해주고 언제나 건강을 챙겨주는 사랑하는 아내 선민 씨에게 고마움을 전하고 싶다.

또 '온라인서버제작자모임'을 통해서 온라인 게임 서버 개발에 관한 정보를 공유해 주고 커뮤니티를 관리하느라 수고가 많은 임영기님, 이지현님, 이욱진님, 허승욱님, 최우영군에게 고마움을 전한다. 마지막으로 게임 개발자 지망생 시절부터 같이 공부해왔고, 현재 게임 업계에서 같이 일하고 있으며, 평소 싫은 내색 하나 없이 필자의 부탁을 잘 들어주는 조현진군에게 고맙다는 말을 전하고 싶다.

집필을 마무리하며

지은이 **최흥배**

도움을 주신 분들

Technical Reviewer

김승혁

현재 블루사이드 Kingdom under fire 2 online team에서 서버 프로그래머로 근무 중이다. '우공이산'을 신조로 빠르게 진화하는 IT기술에 도태되지 않으려, 노력하는 프로그래머다.

구승모

현재 nhn next에서 게임서버 전공 교수로 테라의 전투시스템 구현 및 서버 기반 제작 참여하였다.

김승혁

NCsoft Lineage Eternal 팀 서버 프로그래머로 유연한 구조와 최소한의 코드를 선호하는 서버 개발자다.

Beta Reader

박동섭, 송형주, 이근호, 이정재

대상 독자 및 소스 코드

초급 | **초·중급** | 중급 | 중·고급 | 고급

이 책은 C++ 프로그래밍의 기본과 BSD Socket을 사용한 아주 기본적인 네트워크 프로그래밍에 대해서는 알고 있는 데 반해, 네트워크 프로그래밍 경험이 적고 고성능 네트워크 프로그래밍을 위한 비동기 IO 프로그래밍에 대해서는 잘 모르는 개발자를 대상으로 한다.

또한, 기존 방식 대신 Boost.Asio를 사용한 고성능 멀티 플랫폼 대응의 네트워크 프로그램을 만드는 방법은 물론, 비동기 프로그래밍을 일반적인 프로그래밍에서 사용하는 방법 역시 다루고 있기 때문에, IOCP 등의 비동기 IO 네트워크 프로그래밍 경험이 있는 중급 이상의 개발자에게도 도움이 될 것이다.

이론적인 설명보다는 Asio를 사용하는 실용적인 프로그래밍에 초점을 맞추고 있는 만큼, 책에서는 이에 관한 모든 부분을 세세하게 다루지 않는다. 그러나 책을 읽고 난 후 즉시 Asio를 사용하여 프로그래밍하는 것이 가능하도록 구성했다.

필자의 경험상 프로그래밍 관련 책에서 익힌 것은 반드시 직접 코딩해보고, 배운 기술을 사용하여 직접 프로그램을 만들어보아야 자기 것으로 만들 수 있다. 그러니 그냥 눈으로만 보지 말고 꼭 프로그램을 직접 만들어보기 바란다.

소스 코드는 다음 링크를 통해서 다운받을 수 있다.

- http://www.hanb.co.kr/exam/2611

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

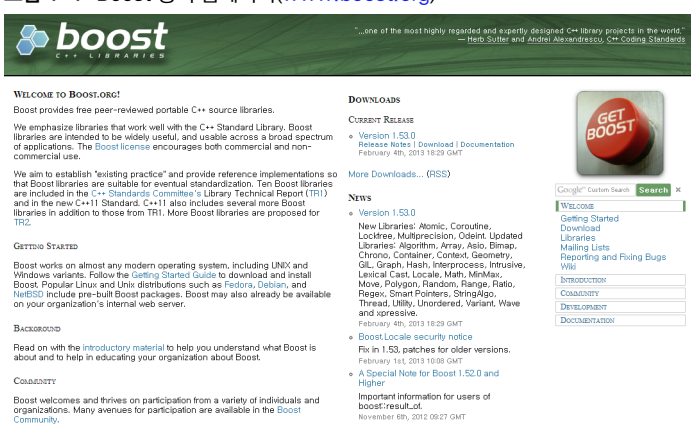
차례

1 | Boost 라이브러리

Boost 라이브러리를 알고 있는가? 아직 모르고 있다면, C++ 프로그래머로서 적지 않은 손해를 보고 있는 셈이다. Boost 라이브러리에 있는 유용한 라이브러리를 사용하면, 개발에 필요한 기능들을 직접 만들지 않아도 될뿐더러 효율성 또한 좋아서 C++ 프로그래밍의 생산성을 크게 올릴 수 있기 때문이다.

Boost 라이브러리는 C++ 프로그래머를 위한 유용한 오픈 소스 C++ 라이브러리의 모음으로, 수많은 C++ 고급 프로그래머들이(C++ 표준 위원회 멤버 등) 개발에 참여하고 있다. 그래서 Boost 라이브러리는 기능 면에서 실용적이며 안정성 또한 높다.

그림 1-1 Boost 공식 홈페이지(www.boost.org)



아직까지 Boost 라이브러리를 사용해본 적이 없다면 지금 바로 사용하기를 권한다. 아직도 일부에서는 Boost 라이브러리의 안정성을 의심하기도 하는데, 이제 무의미한 의심은 그만해도 될 것 같다. 한국의 게임 개발 회사 상당수가 이미 Boost

라이브러리를 사용하고 있으니 말이다. 무슨 설명이 더 필요한가!

2011년 8월에 표준 작업이 끝난 C++11 버전에는 많은 Boost 라이브러리가 새로운 표준 라이브러리로 추가되었다. 예를 들어 Visual Studio(이하 VS) 2008 SP를 설치하면 사용할 수 있는 tr1 라이브러리 대부분이 Boost 라이브러리다. 그래서 C++11 버전을 지원하지 않는 VC를 사용하더라도, Boost 라이브러리를 사용하면 C++의 새로운 기능을 적지 않게 사용할 수 있다.

Boost.Asio의 사용 방법과 장점을 알아보기 전에, 우선 Boost 라이브러리 설치 방법과 특징을 간략하게 살펴보자.

1.1 Boost 라이브러리 설치

Boost 라이브러리는 공식 웹 사이트와 Boost 라이브러리 컨설팅 BoostPro에서 다운받을 수 있다.

- Boost 공식 웹 사이트 : http://www.boost.org/users/history/version_1_52_0.html
- BoostPro 웹 사이트 : http://www.boostpro.com/download/

Boost 라이브러리 중 OS(Operating System, 운영체제)의 커널 기능을 사용하는 라이브러리(thread, filesystem 등)는 직접 빌드해야 한다. VS에서 BoostPro를 사용한다면, 이미 빌드된 lib이나 dll 파일을 얻을 수 있다(VS 2003버전 ~ VS 2010 버전을 사용한다면, 시스템 종류(32비트나 64비트 운영체제에 상관없이) BoostPro를 실행할 수 있다).

다만 BoostPro를 이용하면 Boost 라이브러리의 최신 버전을 공식 사이트에서 받는 것보다 상당히 늦게 받을 수밖에 없다는 단점이 있다. 또한 Windows 이외

의 플랫폼을 지원하지도 않으며, 최신 버전의 VS를 지원하지 않을 수도 있다[01](VS 2012를 지원하지 않는다).

Boost 라이브러리를 사용하기 위해 본인이 직접 라이브러리를 빌드해야 하는 것이 꽤 까다롭게 여겨질 수도 있다. 하지만 Boost 라이브러리의 대부분은 헤더 파일을 포함하는 것만으로 사용할 수 있으며, Boost 라이브러리에서 제공하는 툴을 사용하면 빌드하는 것도 아주 간단하다.

빌드하는 방법에 대해 자세히 알고 싶다면 다음의 웹 사이트를 참고하기 바란다(이외에도 네이버나 구글에서 검색하면 다양한 Boost 라이브러리 빌드 방법을 찾을 수 있다).

- Boost 라이브러리 빌드하기: http://jacking.tistory.com/986

1.2 멀티 플랫폼 지원

필자는 게임 개발자로서 경력을 쌓고 있는데, 몇 년 전까지만 해도 게임은 대부분 Windows 플랫폼용으로 개발했다. 때문에 멀티 플랫폼이라는 단어는 별 의미가 없었다. 그러나 지금은 모바일 플랫폼이 무서운 기세로 성장하는 추세라서 iOS나 Android 등 비(非) Windows 플랫폼을 무시할 수 없게 되었다.

Boost 라이브러리의 장점 중 하나가 멀티 플랫폼 지원이다. 이처럼 Boost 라이브러리가 Windows 이외에도 Linux OS, iOS, Android 등을 지원하기 때문에 모바일 플랫폼에 적절히 대응할 수 있다. 또한 거의 모든 플랫폼에 유연하게 대응할 수 있기 때문에, Boost 라이브러리를 사용하여 얻은 지식과 경험의 가치는 쉽게 바래지 않으며 오랫동안 이용 가능하다.

---

01 이 책을 집필하던 2013년 2월 초, BoostPro의 설립자가 다른 회사로 이직하면서 회사 문을 닫았다. 때문에 이후 1.5.2 버전부터는 빌드된 라이브러리를 제공하지 않을 것 같다.

iOS나 Android에서 Boost 라이브러리를 사용하는 방법을 알고 싶다면, 구글에서 'boost ios'와 'boost android'를 각각 검색해보기 바란다. 다양한 자료를 찾아볼 수 있을 것이다(Boost 라이브러리나 iOS, android는 다양한 버전이 있으니 필요할 때 직접 검색하기 바란다).

이상으로 Boost 라이브러리에 대해 짧게 설명했다. 핵심 부분은 이해했을 것이니, 이제 본격적인 주제로 들어가보자.

2 | Boost.Asio

Boost.Asio는 Boost 라이브러리 중 하나로, 네트워크 및 비동기 프로그래밍과 관련된 라이브러리다. Boost.Asio를 사용하면 어렵고 복잡한 고성능 멀티 플랫폼 네트워크 프로그래밍과 비동기 프로그래밍을 아주 쉽게 할 수 있다. Boost.Asio의 특징은 다음과 같다.

- 네트워크 프로그래밍에서 주로 사용한다.
- Asynchronous I/O(비동기 입출력)를 지원하여, 상대적으로 많은 시간이 걸리는 I/O 작업을 빠르게 처리할 수 있다.
- 파일 입출력, 시리얼 포트 입출력, 범용적인 비동기 프로그래밍에 사용할 수 있다.
- 멀티 플랫폼을 지원한다.

2.1 멀티 플랫폼 지원

멀티 플랫폼을 지원하기 위해 해당 플랫폼의 특수한 기능을 지원하지 않는 라이브러리도 있지만, Boost.Asio는 각 플랫폼에서 지원하는 특정 기능을 대부분 지원한다. 그래서 Boost.Asio는 멀티 플랫폼을 지원하면서도 각 플랫폼의 전용 API보다 성능이 떨어지지 않는다.

Boost.Asio의 OS 플랫폼별 지원 기능 중에서 특히나 중요한 것이 비동기 입출력(Asynchronous I/O) 지원 여부다. Boost.Asio를 사용하여 고성능 네트워크 프로그래밍을 하려면 비동기 입출력 지원이 필수이기 때문이다. 플랫폼별 Boost.Asio의 비동기 입출력 기능 구현 방법은 표 2-1에서 확인할 수 있다.

표 2-1 Boost.Asio 구현을 위해 플랫폼에서 사용한 API

플랫폼	Boost.Asio 구현
Linux Kernel 2.4	비동기를 지원하지 않으므로 select만 사용. FD_SIZE 크기에 연결 수 제한
Linux Kernel 2.6 이상	epoll 사용
FreeBSD	Kqueue 사용
Mac OS X	Kqueue 사용
Solaris	/dev/poll 사용
Windows 2000 이상	Ovelapped I/O와 IOCP 사용

여기서 잠깐_ 비동기 입출력(Asynchronous I/O)

- 네트워크 프로그래밍에서의 blocking, non-blocking, syncronous, asyncronous에 관한 설명
  http://devsw.tistory.com/142
- 비동기 I/O를 이용한 Boost 애플리케이션 실행에 관한 설명
  https://www.ibm.com/developerworks/linux/library/l-async/

2.2 신뢰성

오픈 소스 라이브러리를 사용할 때 가장 신경 쓰이는 부분이 신뢰성이다. 신뢰성 때문에 오픈 소스를 사용하지 않고 직접 만들어서 사용하는 경우도 있는데, 다행히 Boost.Asio는 신뢰성에 관한 한 걱정하지 않아도 된다.

현재 Boost.Asio는 많은 곳에서 사용되고 있다. 국내 한 유명 IT 회사는 회사 내의 C++ 네트워크 라이브러리로 Boost.Asio를 사용하고 있으며, 이미 출시된 유명 온라인 게임에서도 사용하고 있다. 필자가 일했던 곳을 포함한 몇몇 회사는 온라인 P2P용 캐주얼 게임과 MMORPG에서도 Boost.Asio를 사용 중이다. Boost.Asio의 신뢰성이 낮다면 기업들이 사용하겠는가. 답은 이미 나와 있다.

이처럼 Boost.Asio는 오픈 소스답게 많은 곳에서 사용 중이며, 버그가 있었다면 이미 발견하여 수정했을 확률이 아주 높다. 그리고 C++11 이후의 새로운 C++ 표준 STL에 네트워크 라이브러리로 포함될 확률도 꽤 높다. 반면 우리가 직접 만든 것은 우리만 사용하기 때문에, 사용 시간이나 사용처가 Boost.Asio에 비해 확연히 적다. 어떤 의미에서는 우리가 직접 만든 것에 버그가 있을 확률이 더 높다.[01]

2.3 성능

필자는 PC 온라인 게임의 서버 프로그래머로 일하고 있어서, 멀티 플랫폼 대응보다는 성능적인 측면을 더 중요하게 생각한다. 서버 프로그램은 대부분 플랫폼 하나를 정해서 개발하고, 클라이언트를 최대한 많이 연결할 수 있어야 하며, 빠르게 데이터를 주고받을 수 있어야 하기 때문에 필자 같은 서버 프로그래머는 필연적으로 성능에 민감하다. Boost.Asio와 '직접 만든 네트워크 라이브러리'의 성능 차이를 그림 2-1의 자동차 사진을 통해 비교해보겠다.

그림 2-1 네트워크 라이브러리 성능 차 비교

01 Boost 라이브러리 1.4 버전 이후의 Asio에서는 안정성을 넘어서 성능 튜닝이 많이 되었다.

네트워크 라이브러리를 직접 만들 경우, 특별한 OS와 프로그램에 특화되게 만들 수 있어서, 이론적으로 보자면 사진 (A)의 F1 머신 같이 엄청난 성능을 낼 수 있다. Boost.Asio는 다양한 OS와 범용적인 곳에서 사용하기 때문에 (A)보다는 좀 떨어지는 (C)의 스포츠카 정도의 성능을 낼 수 있다. 그런데 실제로 우리가 직접 만들어 보면 아주 잘 만들어야 (A)와 같은 성능을 낼 수 있다. 누구나 F1 머신 수준의 성능을 낼 수 있을까? 이 책을 읽고 있는 독자라면 답을 알고 있을 것이다. 지식이나 경험이 부족하면 사실 사진 (B)의 일반 자동차 정도의 성능도 내기 어렵다.

아주 특별한 경우가 아니라면, Boost.Asio는 고성능을 요구하는 네트워크 프로그래밍에도 사용할 수 있다(고성능 네트워크 프로그램을 만드는 곳이 온라인 게임인데, 여기서도 문제없이 잘 사용하고 있다).

2.4 편의성 및 범용성

아무리 좋은 것이라도 사용하기 어렵다면 효율성이 떨어진다(업무에 필요 없다면 사용을 꺼릴 것이다).

그림 2-2 개발 편의도 비교

필자는 예전에 건강을 위해 양파즙을 구입하려고 한 적이 있는데, 사진 (a)와 같이 즙으로 만들어 판매하는 것을 사면 간단할뿐더러 신뢰할 만한 것을 구할 수 있었다. 그러나 (a)를 구매해서는 필자에게 어울리는, 좀 다른 양파즙을 얻기 어렵다는 문제가 있었다. 즉, 필자가 (a)에 맞추어야 했다. 필자에게 꼭 맞는 양파즙을 구하려면 사진 (b)와 같이 직접 생양파를 사서 처음부터 만들어야 했는데, 그러면 필자가 직접 해야 할 일이 너무 많고 질 좋은 양파를 구하는 것도 문제였다. 이에 반해 사진 (c)처럼 어느 정도 손질되어 있는 양파를 구매하면 (a)와 (b)의 장점을 취할 수 있었다. 그래서 필자는 (c)를 구매해서, 스스로에게 꼭 맞는 양파즙을 만들어 먹고 기운을 차렸다.

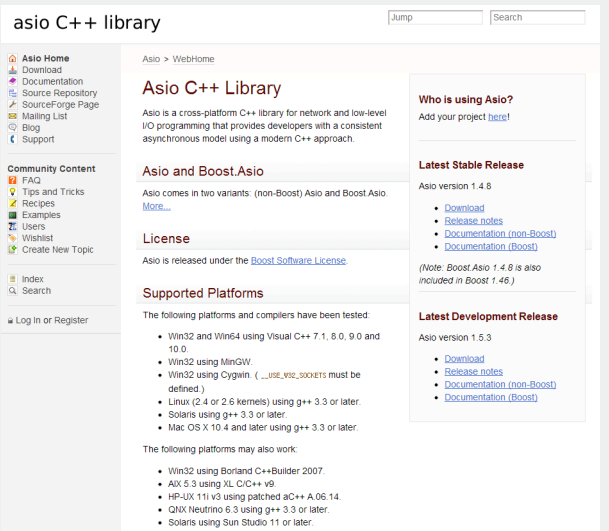
이 예를 네트워크 프로그램을 만드는 것에 대입하면 (a)는 상용 라이브러리를 사용하는 경우고, (b)는 자신이 직접 네트워크 라이브러리를 만드는 경우며, (c)는 Boost.Asio를 사용한 경우다.

Boost.Asio를 사용하여 네트워크 프로그래밍하면 네트워크 라이브러리를 직접 만드는 것에 비해 개발 속도와 성능, 신뢰도가 높아지는 것은 물론, 자신이 원하는 기능을 추가하는 데 좀 더 많은 시간을 쏟을 수 있다.

네트워크 프로그래밍이나 비동기 프로그래밍은 일반적으로 쉽게 접할 수 있는 것이 아니라서, Boost 라이브러리는 알아도 그에 포함되어 있는 Asio에 대해서는 잘 모르는 분들이 많으리라 생각한다. 필자가 지금까지 주절주절 설명한 것도 그 때문이다. 앞서 밝혔듯이 이 책은 실전에 도움되는 것을 목표로 하고 있으니 Boost.Asio에 대한 이론적인 설명은 이것으로 끝내겠다. 다음 장부터는 간단한 소켓 프로그래밍 예제를 통해서 '고성능 비동기 네트워크 프로그래밍'과 '범용적 비동기 프로그래밍'에 대해 알아보겠다.

NOTE_ Boost.Asio는 Boost 라이브러리와 별도로 만들어졌다가 이후에 Boost 라이브러리에 편입되었다. 그래서 지금도 Boost 라이브러리와 별도로 사이트가 운영되고 있는데, 여기서도 Asio 라이브러리를 다운받을 수 있다.

- Asio C++ library : http://think-async.com/Asio



3 | 간단한 Echo 서버, 클라이언트 프로그램 만들기

이제 Boost.Asio를 사용하여 네트워크 프로그래밍을 어떻게 하는지 본격적으로 알아보자. 이전에 OS의 API(Windows 혹은 Linux의 Socket API)를 사용하여 네트워크 프로그래밍을 해본 적이 있다면, Boost.Asio를 이용해서 네트워크 프로그래밍을 하면 얼마나 간단해지는지 알고 있을 것이다.

여기서는 Boost.Asio 사용 방법을 설명하는 것이 핵심이므로, 사용 방법에 집중할 수 있도록 예제를 최대한 간단하게 만들었다. 실용적인 예제는 아니지만 코드가 짧고 간단해서 Boost.Asio 사용 방법을 배우는 데 좋을 것이다.

3.1 Boost.Asio를 사용하기 위한 준비

Boost.Asio를 사용하려면, 먼저 Boost 라이브러리를 빌드하여("1.1 Boost 라이브러리 설치" 참고) Boost.Asio와 관련된 lib 파일을 만들어야 한다. lib 파일이 준비되었다면, 다음과 같은 헤더 파일 하나만 추가하여 Boost.Asio를 사용할 수 있다.

```
#include <boost/asio.hpp>
```

이것으로 Boost.Asio를 사용할 준비는 끝났다. 이제 사용만 하면 된다. 정말 간단하지 않은가? 참고로, lib 파일 없이 헤더 파일만 추가하는 간단한 작업으로 Boost.Asio를 사용할 수도 있다.

예제 프로그램은 Boost.Asio를 사용한 간단한 네트워크 프로그래밍으로, 동기 I/O 버전의 TCP 프로토콜을 사용하는 Echo 서버와 클라이언트다.

3.2 동기 I/O 방식의 TCP/IP Echo 서버

예제 3-1은 TCP/IP 프로토콜을 서버 프로그램으로 사용하여 하나의 클라이언트 접속을 받은 후 클라이언트가 보내는 메시지를 받으면, 그 메시지를 다시 클라이언트에 보내주는 일종의 Echo 서버 프로그램[01]이다.

예제 3-1 SynchronousTCPServer

```
#include <SDKDDKVer.h>
// 'SDKDDKVer.h'는 VS 2012에서 프로젝트 만들면 자동으로 넣어주는
// 헤더 파일이다. 없어도 되는 헤드 파일이다.

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

const char SERVER_IP[] = "127.0.0.1";
const unsigned short PORT_NUMBER = 31400;

int main()
{
   boost::asio::io_service io_service;
   boost::asio::ip::tcp::endpoint endpoint( boost::asio::ip::tcp::v4(),
      PORT_NUMBER );
   boost::asio::ip::tcp::acceptor acceptor( io_service, endpoint );

   boost::asio::ip::tcp::socket socket(io_service);
   acceptor.accept(socket);
   std::cout << "클라이언트 접속" << std::endl;

   for (;;)
```

01 예제는 에코(Echo) 서버/클라이언트다. 상대방이 보낸 것을 그대로 돌려주는 에코의 뜻 그대로, 서버는 클라이언트가 보낸 데이터를 다시 그대로 클라리언트에 보낸다.

```
    {
        std::array<char, 128> buf;
        buf.assign(0);
        boost::system::error_code error;
        size_t len = socket.read_some(boost::asio::buffer(buf), error);

        if( error )
        {
            if( error == boost::asio::error::eof )
            {
                std::cout << "클라이언트와 연결이 끊어졌습니다" << std::endl;
            }
            else
            {
                std::cout << "error No: " << error.value() << " error Message: "
                    << error.message() << std::endl;
            }
            break;
        }

        std::cout << "클라이언트에서 받은 메시지: " << &buf[0] << std::endl;
        char szMessage[128] = {0,};
        sprintf_s( szMessage, 128-1, "Re:%s", &buf[0] );
        int nMsgLen = strnlen_s( szMessage, 128-1 );

        boost::system::error_code ignored_error;
        socket.write_some(boost::asio::buffer(szMessage, nMsgLen), ignored_error);
        std::cout << "클라이언트에 보낸 메시지: " << szMessage << std::endl;
    }
    getchar();
    return 0;
}
```

예제 3-1은 코드가 짧아서 전체가 한눈에 들어올 것이다. 한눈에 들어오지 않는다고 해도, 필자가 하나하나씩 설명할 테니 편안한 마음으로 잘 따라오기 바란다. 이 예제를 제대로 이해하면 Boost.Asio로 간단한 네트워크 프로그램을 만들 수 있으며, 앞으로 살펴볼 기능이 추가된 예제 역시 쉽게 이해할 수 있다(네트워크 프로그래밍에 관한 경험이 부족하다고 생각하는 개발자라면 2장과 3장의 예제 코드를 모두 외워서 사용하기 바란다. 코드가 짧아서 생각보다 외우기 쉬우며, 코드를 외워두면 자신감도 생기고 이해가 더 잘 될 것이다).

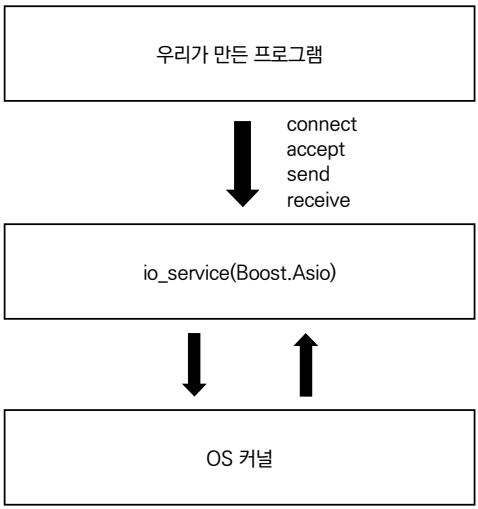
3.2.1 클라이언트 접속 준비

```
boost::asio::io_service io_service;
```

boost::asio::io_service는 Boost.Asio의 핵심 중의 핵심이다. io_service는 커널에서 발생한 I/O 이벤트를 '디스패치[02]' 해주는 클래스로, io_service를 통해서 커널에서 발생한 네트워크상의 접속 받기, 접속하기, 데이터 받기, 데이터 보내기 이벤트를 알 수 있다. 그래서 Boost.Asio에서 socket과 같은 객체를 생성할 때 io_service를 인자로 넘겨줘야 한다.

02 다중 태스킹 환경에서 우선순위가 높은 작업이 수행될 수 있도록 시스템 자원을 할당하는 것을 말한다.

그림 3-1 동기 버전에서의 io_service




```
boost::asio::ip::tcp::endpoint endpoint( boost::asio::ip::tcp::v4(),
   PORT_NUMBER );
boost::asio::ip::tcp::acceptor acceptor( io_service, endpoint )
```

endpoint는 네트워크 주소를 설정한다(이 주소로 클라이언트가 접속한다). 서버와 클라이언트는 endpoint 설정 방식이 다른데, 서버는 IP 주소(IPv4 또는 IPv6)와 포트 번호를 사용한다. 예제에서는 IPv4 방식의 주소 체계를 사용했다.

acceptor 클래스는 클라이언트의 접속을 받아들이는 역할을 하는데, io_service와 endpoint를 인자로 사용했다.

3.2.2 클라이언트의 접속 받기

```
boost::asio::ip::tcp::socket socket(io_service);
acceptor.accept(socket);
```

예제 3-1은 TCP/IP 프로토콜을 사용하므로 접속한 클라이언트에 할당할 'tcp::socket'을 만든다. 그리고 만든 socket을 통해서 클라이언트가 보낸 메시지를 주고받아야 하므로 io_service를 할당한다.

위에서 만든 acceptor의 accept 멤버 함수에 클라이언트용 socket 객체를 넘겨주면 서버가 클라이언트를 접속받을 준비는 끝난다. 이 예제는 동기형 방식[03]을 사용하고 있어서, accept 멤버 함수를 호출한 후 클라이언트의 접속이 완료될 때까지 대기 상태가 된다.

3.2.3 클라이언트가 보낸 메시지 받기

```
std::array<char, 128> buf;
buf.assign(0);
```

클라이언트가 보낸 메시지를 담을 버퍼를 만든다(std::array<char, 128> buf;). 예제 3-1에서는 char형 배열과 호환되는 STL의 array 컨테이너를 사용했다. 물론 array가 아닌 char형 배열을 사용해도 괜찮다.

```
boost::system::error_code error;
```

error_code 클래스는 시스템에서 발생하는 에러 코드를 '랩핑wrapping'[04]한 클래스로, 에러가 발생하면 에러 코드와 에러 메시지를 얻을 수 있다.

```
size_t len = socket.read_some(boost::asio::buffer(buf), error);
```

socket 클래스의 read_some 멤버 함수를 사용하여 클라이언트가 보낸 데이터를 받는다. 인자로 데이터를 담을 버퍼와 에러 코드를 받을 error_code를 사용하며 데이터를 받으면 받은 데이터 크기를 반환한다. 예제 3-1은 동기형 방식이므로 데

03 요청 후 답변을 받을 때까지 더 이상 진행하지 않고 기다리는 방식이다. 답변을 받으면 다음 단계를 진행한다.
04 기본형 자료를 객체형으로 바꿀 때 사용하는 클래스다.

이터를 다 받을 때까지 대기 상태에 들어간다.

클라이언트에서 보낸 데이터를 받으면 에러가 발생했는지 확인하기 위해 error_code를 조사해야 한다. 클라이언트의 접속이 끊어지는 경우도 read_some을 통해서 알 수 있는데, 이때는 error_code에 에러 값(boost::asio::error::eof)이 설정된다. 때문에 에러가 발생할 경우, 그것이 진짜 에러인지 아니면 접속이 끊어져 발생한 문제인지 꼭 조사해보아야 한다.

```
if( error )
{
  if( error == boost::asio::error::eof )
  {
    std::cout << "클라이언트와 연결이 끊어졌습니다" << std::endl;
  }
  else
  {
    std::cout << "error No: " << error.value() << " error Message: "
              << error.message() << std::endl;
  }
  break;
}
```

3.2.4 클라이언트에 메시지 보내기

```
char szMessage[128] = {0,};
sprintf_s( szMessage, 128-1, "Re:%s", &buf[0] );
int nMsgLen = strnlen_s( szMessage, 128-1 );

boost::system::error_code ignored_error;
socket.write_some(boost::asio::buffer(szMessage, nMsgLen),
  ignored_error);
```

3.2.3에서는 데이터를 받을 때 버퍼로 std::array를 사용했는데, 이번에는 일반적으로 자주 사용하는 char형 배열을 버퍼로 사용하였다. Boost.Asio는 C++ STL과 호환이 잘되어서 std::vector와 std::string도 버퍼로 사용할 수 있다.

메시지를 보낼 때는 socket 클래스의 write_some 멤버 함수를 사용한다. 인자는 받을 때와 비슷하게 buffer 클래스와 error_code 클래스를 사용한다. 단, 받을 때는 buffer 클래스에서 buffer 생성자에게 보낼 데이터의 양을 지정하여, szMessage 배열에 있는 데이터를 모두 보내지 않고 지정한 양만큼 보낸다.

이것으로 Echo 서버가 완성되었다. Windows나 Linux의 OS API를 사용한 네트워크 프로그램에 비해 소스 코드가 아주 짧다는 것이 확연하다. 한편, 예제 3-1은 Windows뿐만 아니라 Linux나 Mac OS 등 다양한 플랫폼에서 코드 하나 바꾸지 않고 그대로 사용할 수 있다.

> **NOTE_** Winsock API를 사용하여 예제 3-1과 비슷한 기능을 가진 서버를 만든 예
> http://blog.naver.com/liccorob/10155384326

3.3 동기 I/O 방식의 TCP/IP Echo 클라이언트

클라이언트와 서버 간에 서로 공통 부분(Asio를 초기화하는 부분과 데이터 받기/

보내기는 서버와 클라이언트가 같은 방식을 사용한다)이 있어서 예제 3-1을 이해했다면, 예제 3-2의 클라이언트 코드 역시 쉽게 이해할 수 있을 것이다.

예제 3-2 SynchronousTCPClient

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

const char SERVER_IP[] = "127.0.0.1";
const unsigned short PORT_NUMBER = 31400;

int main()
{
  boost::asio::io_service io_service;

  boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip::address::
    from_string(SERVER_IP), PORT_NUMBER);

  boost::system::error_code connect_error;
  boost::asio::ip::tcp::socket socket(io_service);
  socket.connect(endpoint, connect_error);

  if (connect_error)
  {
    std::cout << "연결 실패. error No: " << connect_error.value()
              << ", Message: " << connect_error.message() << std::endl;
    getchar();
    return 0;
  }
  else
  {
```

```
    std::cout << "서버에 연결 성공" << std::endl;
}

for (int i = 0; i < 7; ++i )
{
    char szMessage[128] = {0,};
    sprintf_s( szMessage, 128-1, "%d - Send Message", i );
    int nMsgLen = strnlen_s( szMessage, 128-1 );

    boost::system::error_code ignored_error;
    socket.write_some( boost::asio::buffer(szMessage, nMsgLen), ignored_error);

    std::cout << "서버에 보낸 메시지: " << szMessage << std::endl;

    std::array<char, 128> buf;
    buf.assign(0);
    boost::system::error_code error;
    size_t len = socket.read_some(boost::asio::buffer(buf), error);

    if( error )
    {
        if( error == boost::asio::error::eof )
        {
            std::cout << "서버와 연결이 끊어졌습니다" << std::endl;
        }
        else
        {
            std::cout << "error No: " << error.value() << " error Message: "
                << error.message().c_str() << std::endl;
        }
        break;
    }
```

```
        std::cout << "서버로부터 받은 메시지: " << &buf[0] << std::endl;
    }

    if( socket.is_open() )
    {
        socket.close();
    }

    getchar();
    return 0;
}
```

3.3.1 서버에 접속하기

```
boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip
::address::from_string(SERVER_IP), PORT_NUMBER);
```

서버(예제 3-1)의 endpoint 설정과 조금 다르다. 클라이언트는 접속할 서버의 IP 주소를 지정하는데, boost::asio::ip::address::from_string 클래스를 사용하여 문자열로 된 IP 주소를 Boost.Asio에서 사용하는 IP 주소로 변환한다.

```
boost::system::error_code connect_error;
boost::asio::ip::tcp::socket socket(io_service);

socket.connect(endpoint, connect_error);

if (connect_error)
{
    std::cout << "연결 실패. error No: " << connect_error.value()
        << ", Message: " << connect_error.message() << std::endl;
    getchar();
```