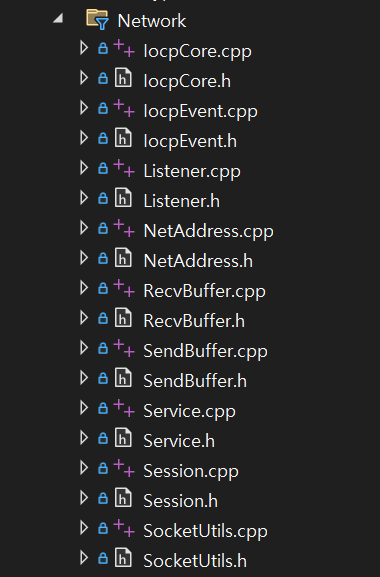
작성자(학번 이름) : 2019180052 한창우

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **주차** | 6주차 | **기간** | 2024.7. 29~ 8. 4 | **지도교수** |  |
| 이번주 한일 요약 | [인프런]  “**게임 서버 프레임워크”** 섹션 수강 내용 분석 및 정리 | | | | |

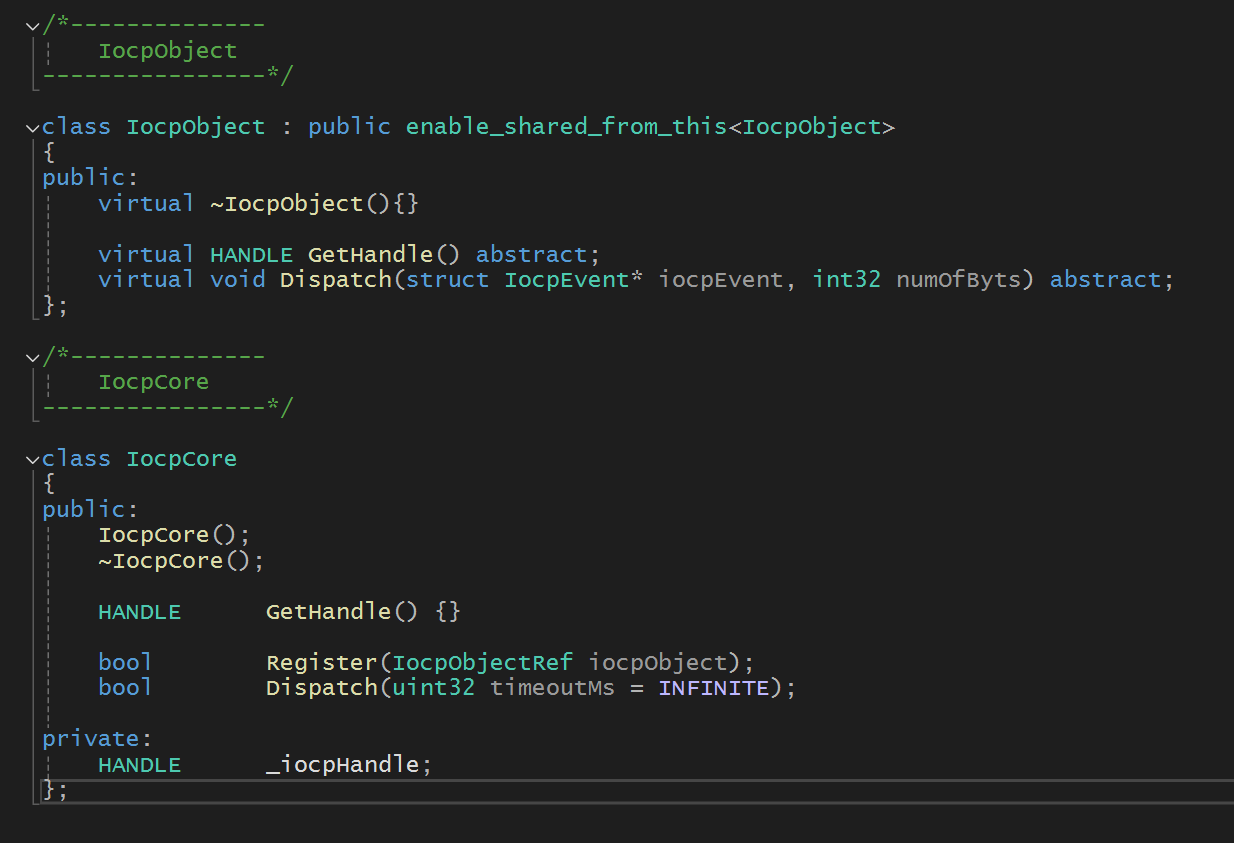
<상세 수행내용>

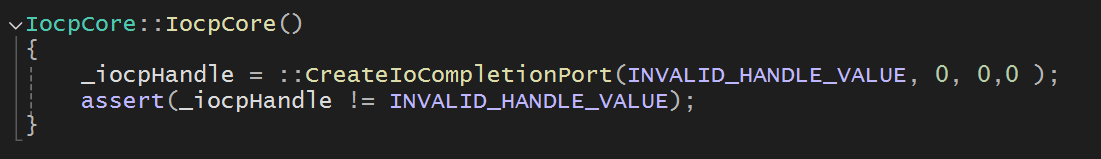
서버 프레임 워크

아래는 게임 서버 프레임워크를 구축하는 데 사용된 cpp 파일과 헤더 파일들입니다. 각 소스 파일이 어떤 역할을 하는지 정리하고자 한다.

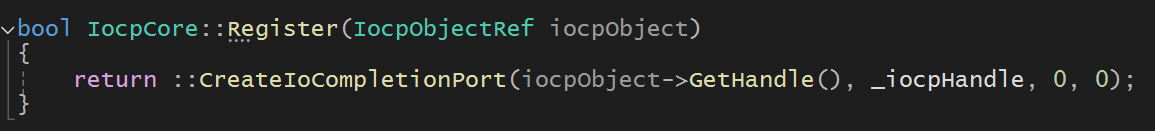


**IocpCore.cpp**

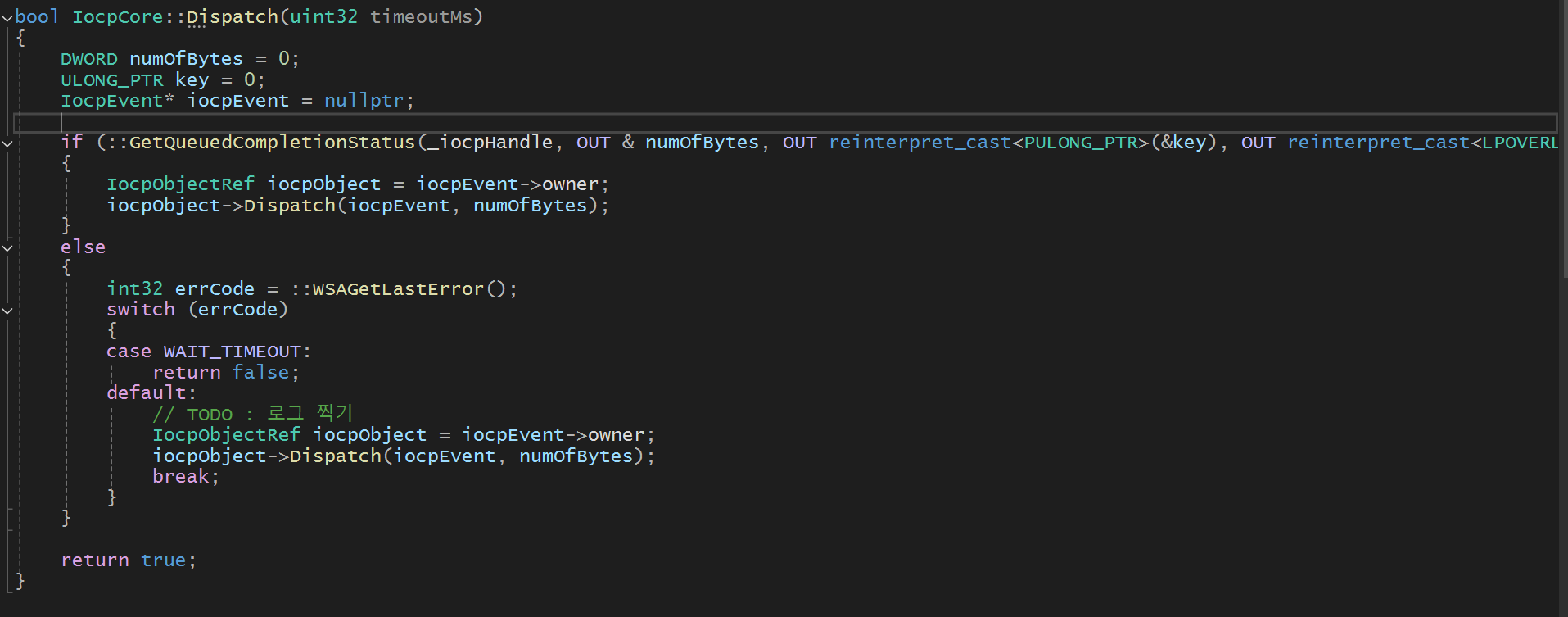




이 게임 서버 프레임워크는 IOCP 모델로 구현되어있다. 일단 서버의 핵심이 IOCP객체를 생성자에서 생성하게 된다.



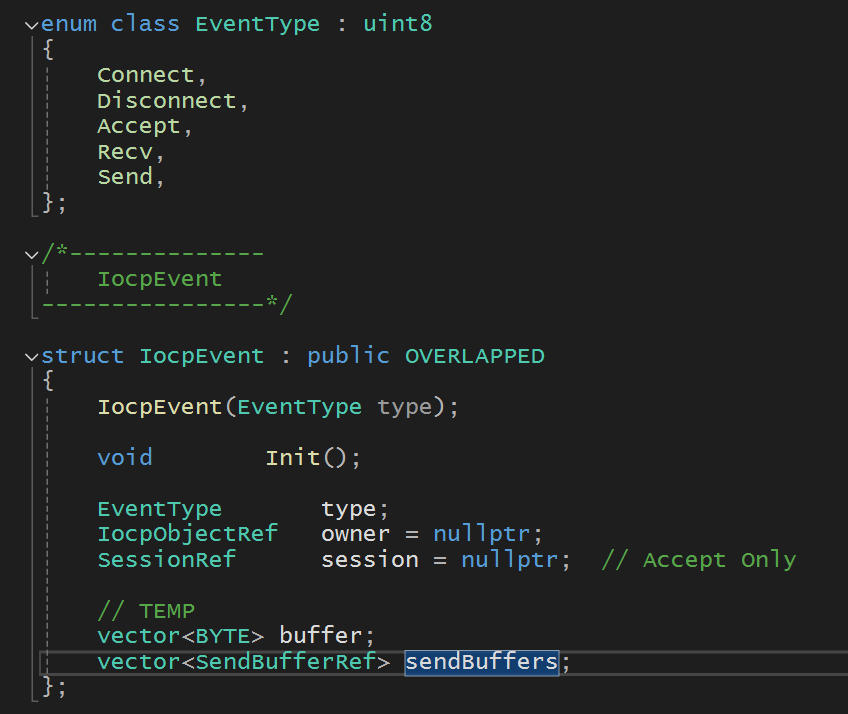
생성된 IOCP에 네트워크 통신을 위해 만든 소켓을 등록한다. IOCP 생성 함수와 소켓등록 함수가 동일하지만 전달하는 매개변수가 다르기 때문에, 그 사용 용도는 다르다.



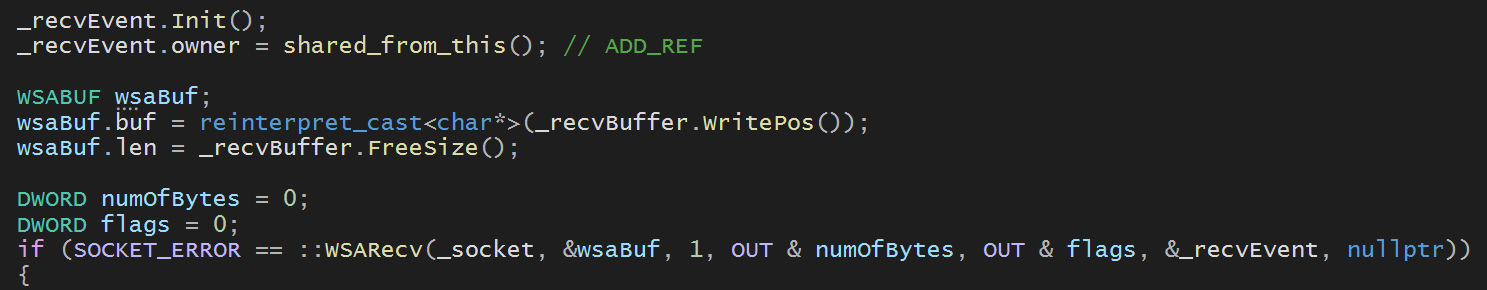
Dispatch 라는 함수에서는 “GetQueuedCompletionStatus” 라는 함수를 호출한다. 이 함수는 스레드들이 대기하여 완료통지 큐를 확인하여 처리할 일감이 있는지 확인하는 것이다.

사용자가 WSARecv, WSASend, WSAAccept, WSAconnect 같은 함수를 호출하여 비동기IO 작업을 요청하게 되면, 커널에서 요청을 저장했다가 해당 작업이 완료하면 완료통지 큐에 이를 등록하고, 유저에게 통보해주는 것이다.

**IocpEvent.cpp**



IocpEvent 객체는 “OVERLAPPED” 를 상속받는다. 상속 받는 이유는 다음과 같다.

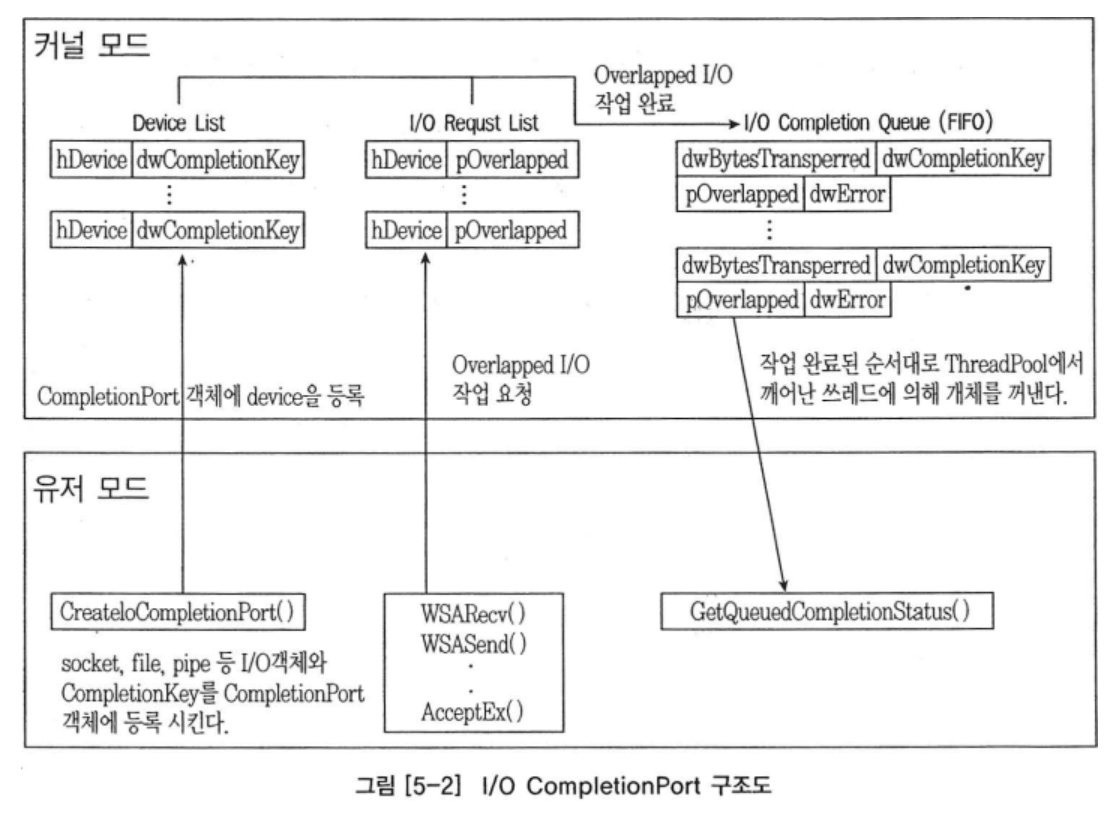


위는 WSARecv를 호출하여 클라이언트로부터 받을 데이터가 있다면 수신하겠다는 함수이다.

함수를 호출 할 때 넘겨주는 매개변수를 살펴보면, 자신의 소켓과 데이터를 넘겨 받을 버퍼 그리고 \_recvEvent 라는 객체가 있다. \_recvEvent는 IocpEvent 클래스의 객체이다.

WSARecv를 호출하면 OVERLAPPED 객체의 주소를 넘겨주기 떄문에 IocpEvent 클래스가 OVERLAPPED 구조체를 상속 받으므로써 IocpEvent의 객체를 매개변수로 넘겨줄 수 있는 것이다.

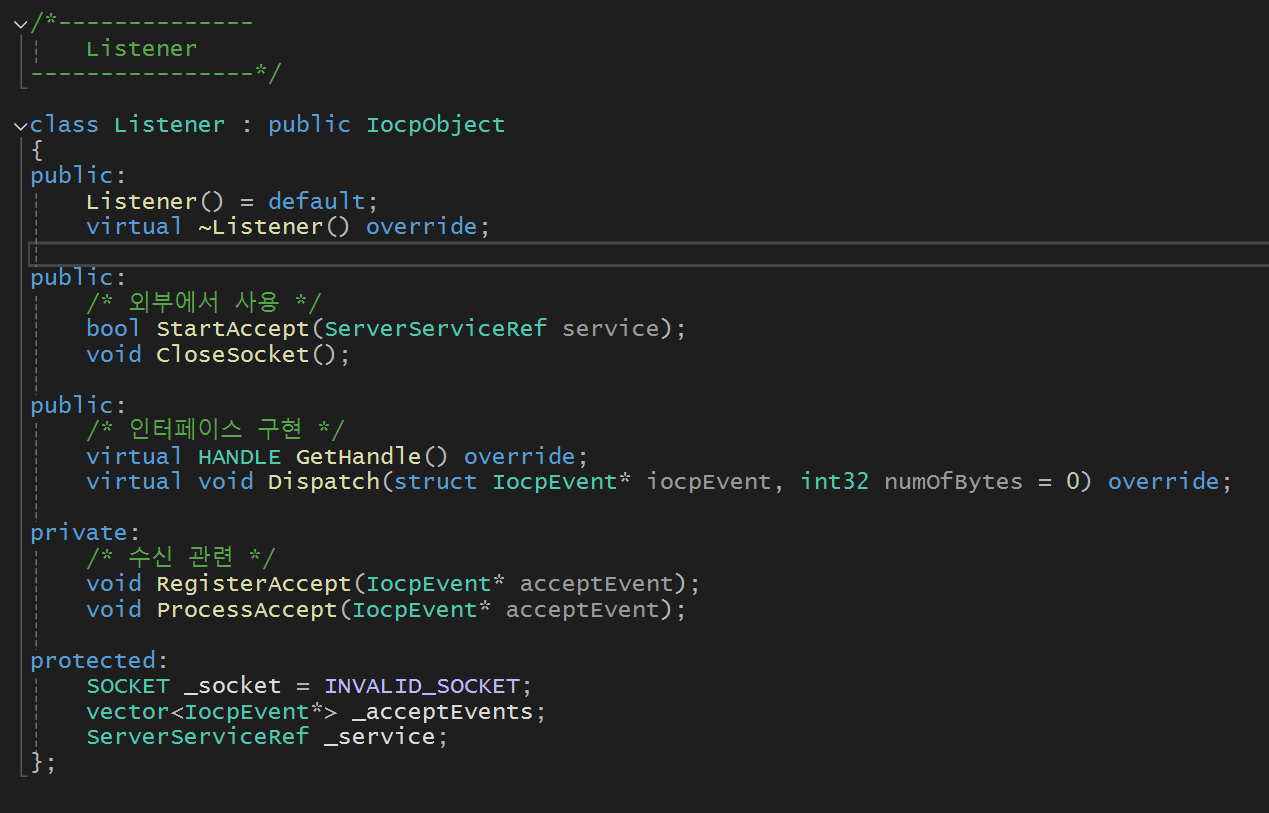
이것의 장점은 우리가 만든 IocpEvent클래스에 우리가 따로 쓰고싶은 정보들도 추가로 멤버데이터로 들게할 수 있다는 것이다.



위는 IOCP 구조도인데, 우리가 WSARecv 매개변수로 보낸 overlapped 객체를 GetQueuedCompletionStatus 함수에서 이를 다시 돌려받기 때문에 이렇게 활용하는 방식은 매우 당연한 것이다.

제가 학교에서 게임서버 강의를 수강할 때는, GetQueuedCompletionStatus 로 넘겨 받는 인자 중에 dwCompletionKey 가 있는데, 이를 활용하여 해당 클라이언트를 찾아서 작업을 처리했었다.

하지만 이 강의에서는 IocpEvent 객체에 클라이언트의 정보를 포함하여 작업을 처리하고 있다.

**Listener.cpp**

Listener class는 클라이언트가 서버에 접속하기 위해, 서버에서 소켓을 만들고 서버의 주소를 바인딩하고 Listen 함수를 호출하여 클라이언트를 받을 준비를 하기 위한 클래스이다.

흐름을 요약하자면,

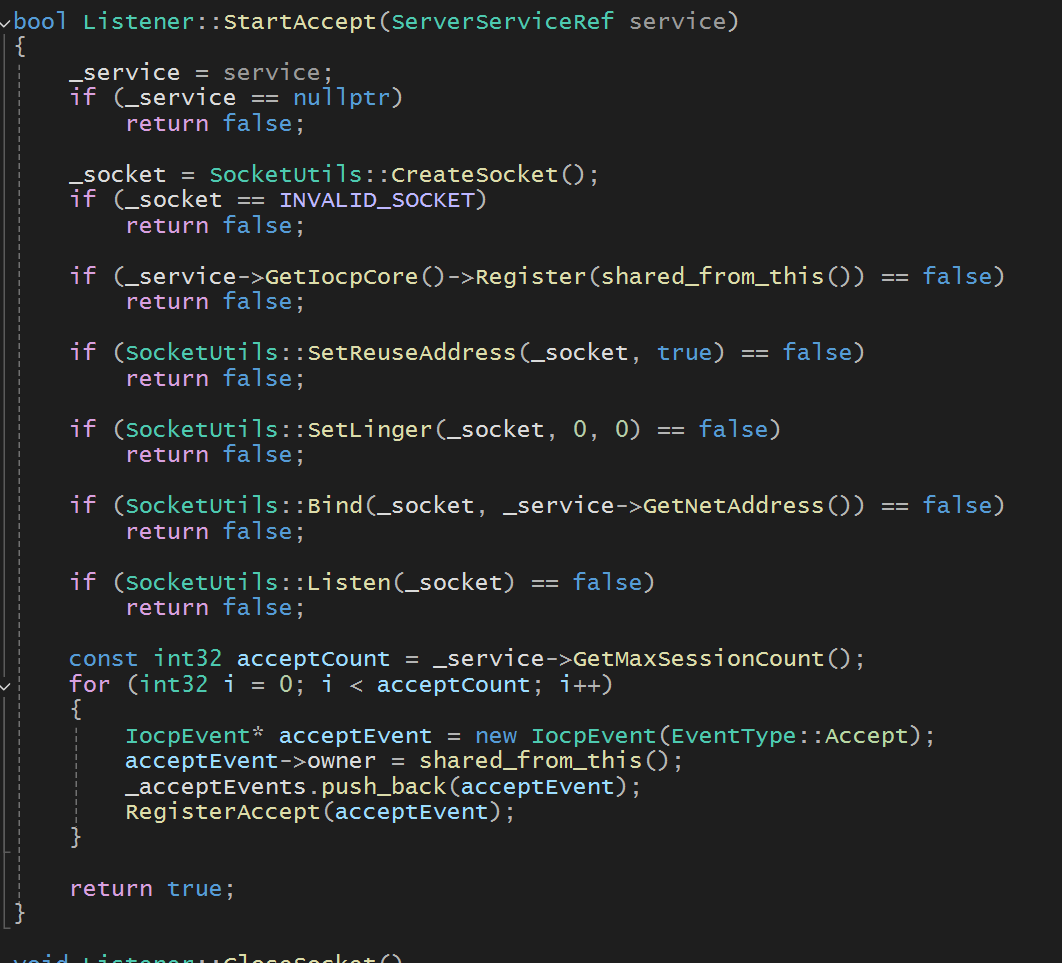
Server

* 소켓을 생성 -> 주소 바인딩 -> Listne함수 호출 -> connect 요청이 오면 accept 함수를 호출하여 수락

Client

* 소켓을 생성 -> 서버의 주소로 connect함수 호출하여 접속 요청

그러므로 Listener 클래스에는 멤버 변수로 소켓을 가지고 있는 것이다.

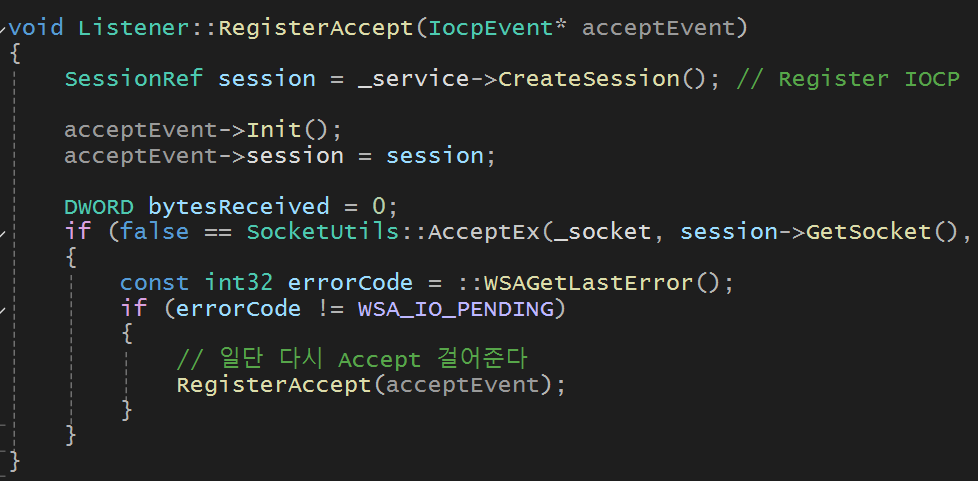


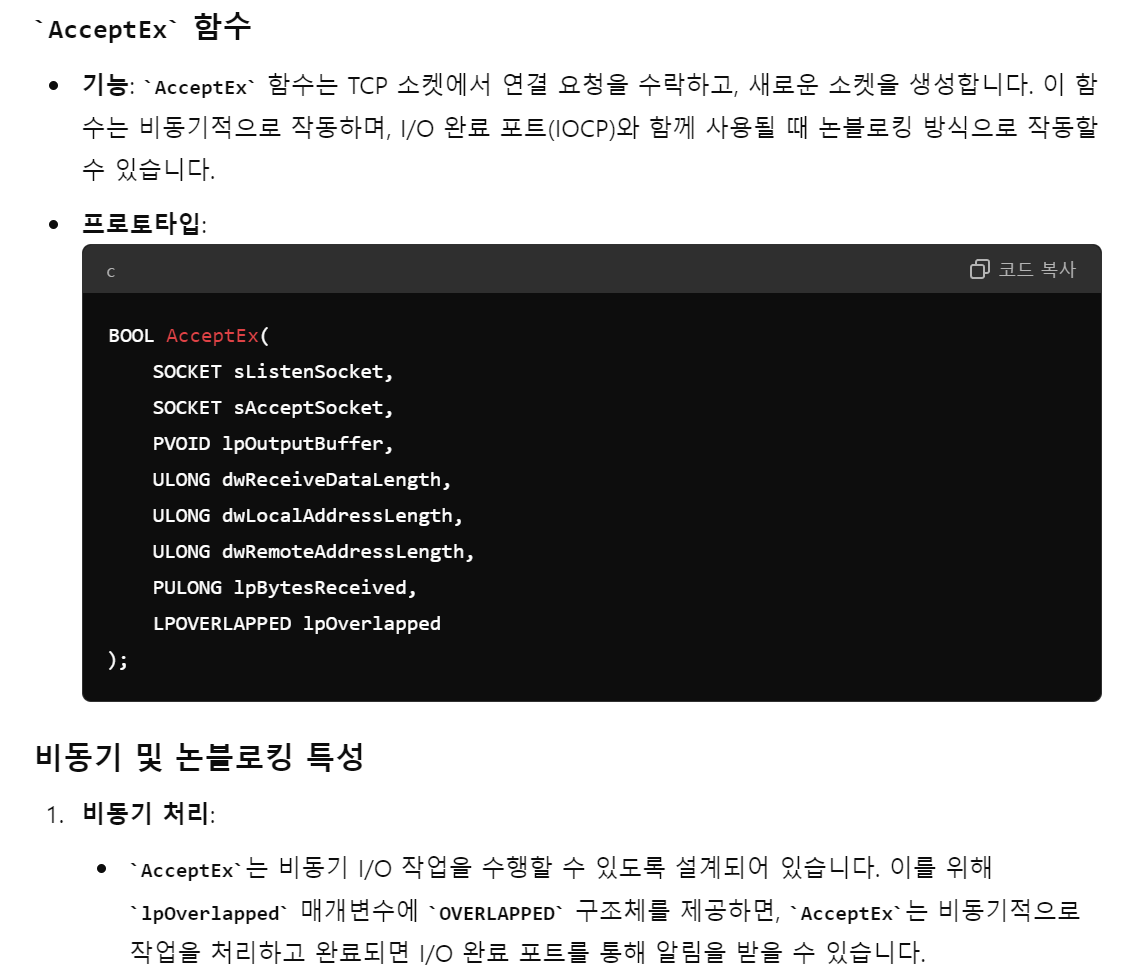
소켓을 생성하고 IOCP에 Listener 소켓을 등록한다. 그리고 소켓 옵션을 설정하는 함수들을 거침

중요하게 볼 것은 Bind와 Listen 함수를 호출하여 클라이언트를 받을 준비를 하는 것이다.

그리고 Listener 클래스는 IocpObject**(유저영역에서 만든 클래스)** 클래스를 상속받고 있다.

이전에 언급했던 WSARecv, WSASend, Accept 등 이런 함수를 호출 할 때, 매개변수로 등록하는 것 중에 소켓도 등록한다고 설명했다. 일단 소켓을 가지고 있는 객체는 IocpObejct를 다 상속 받는다고 보면 된다. (가상 함수 이용을 위해)



Bind와 Listen을 하였다면 Accept함수를 호출하여 클라이언트의 접속 요청이 있으면 받아주게 되는 것이다. 위 AcceptEx는 비동기 함수이다.

위 코드를 보면,

Session 객체를 만들어서 새로운 소켓을 만든다. 이 소켓은 Listener 소켓과 다른 클라이언트를 받아주게 되면, 해당 클라이언트와 통신하기 위한 소켓이다.

(Session class는 뒤에 설명예정, Session 클래스도 IocpObject를 상속받고있음)

Accept를 호출하고 실제로 접속 요청이 와서 수락을 하게되면, 커널에서 완료통지 큐에 일감을 넣고 유저에게 통보한다.

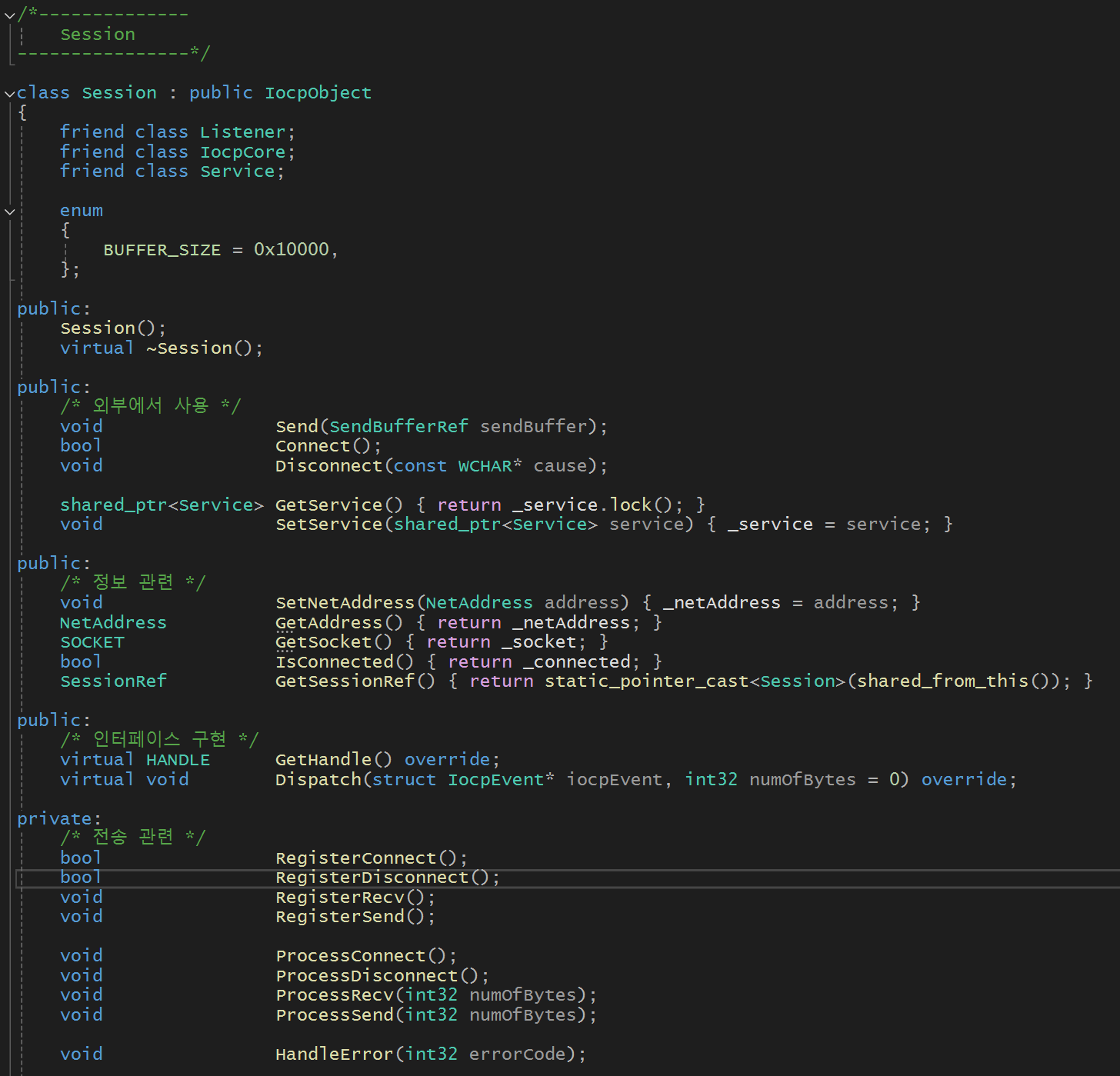


완료통지를 통보 받고 이를 처리하는 스레드는 클라이언트의 소켓의 정보를 따로 저장한다.

(만약 해당 클라이언트가 악성 유저라면 IP를 차단하기 위함)

그리고 RegisterAccept 함수로 진입하여 다시 Accept 함수를 호출하여 다음 클라이언트를 받는 작업을 하게된다. 반복하여 얘기하자면, Listener 소켓과 새로운 Session을 만들어 소켓을 만들고 Acceptex함수를 호출하는 것.

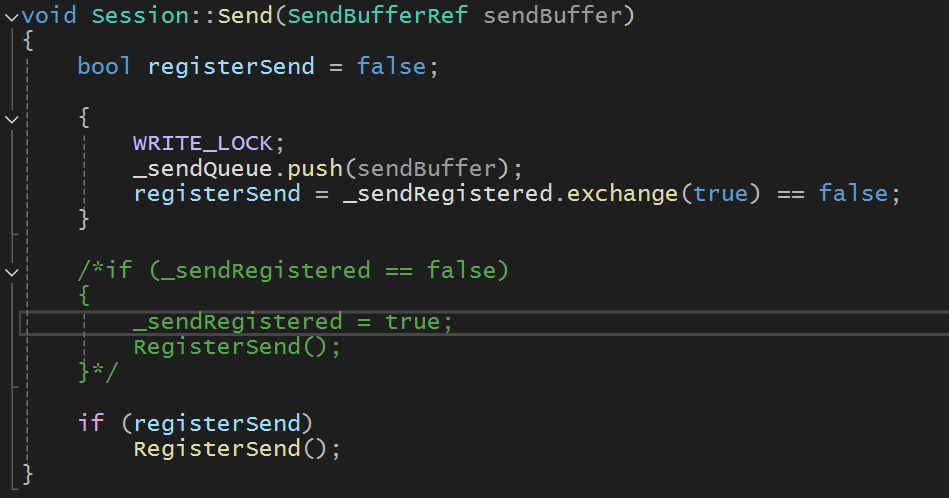
**Session.cpp**





Session은 서버에 접속하는 클라이언트들을 관리하기 위한 클래스이다. 클라이언트 한명이 하나의Session인 것이다. 그러므로 접속이 완료되면, 서버는 클라이언트와 Session클래스에 등록되어있는 소켓으로 서로 통신하는 것이다. (Listener에서 통신 과정을 설명함)

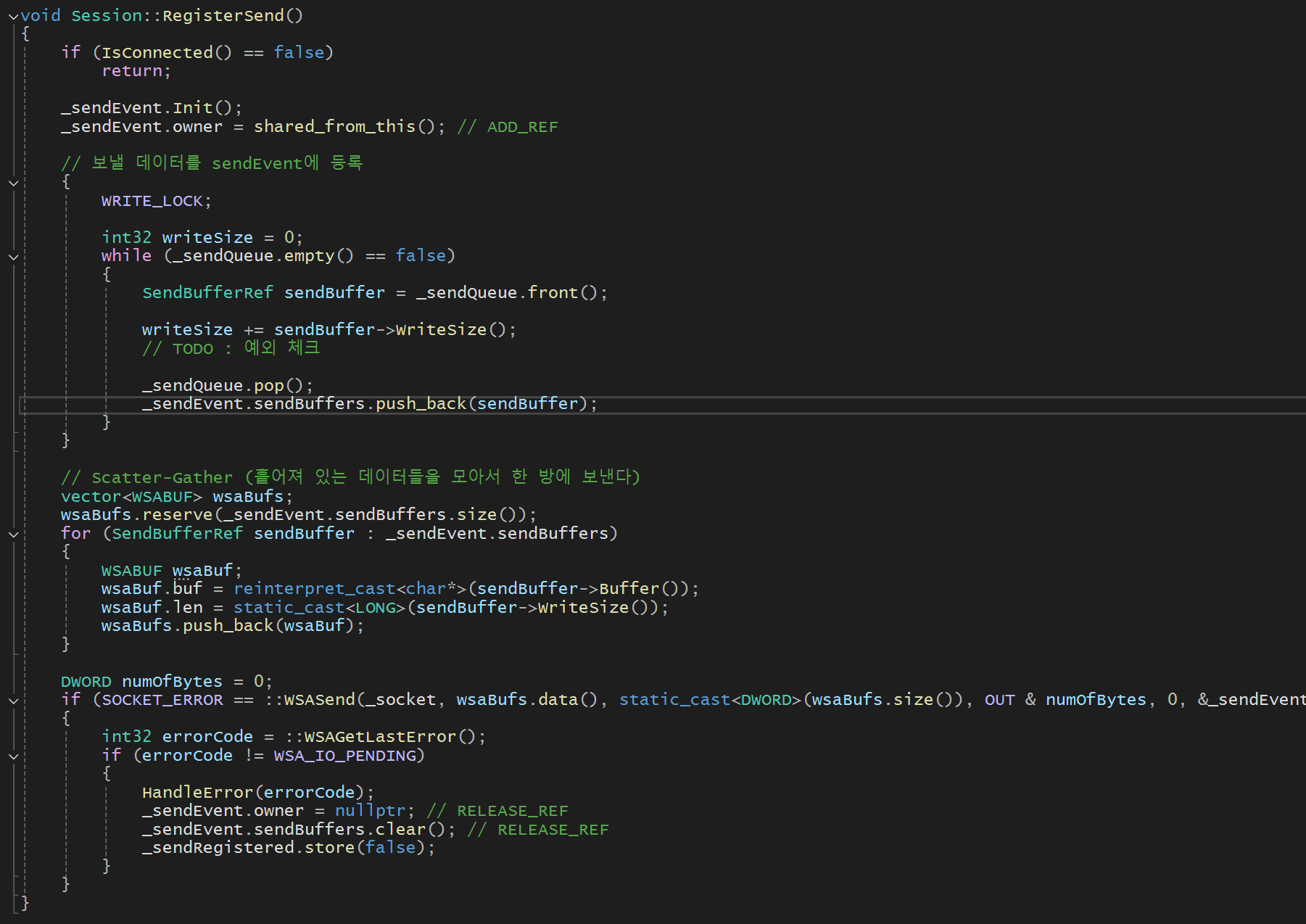
앞으로 클라이언트에서 무엇인가 요청이 들어오면 Session 클래스에서 이를 확인하고 처리하게 되는 것이다. 서버와 클라이언트 모두 Session 클래스를 사용하여 서로 주고 받는 데이터를 처리하는 것이다.



Session에서 데이터를 Send하기 위한 코드 분석.

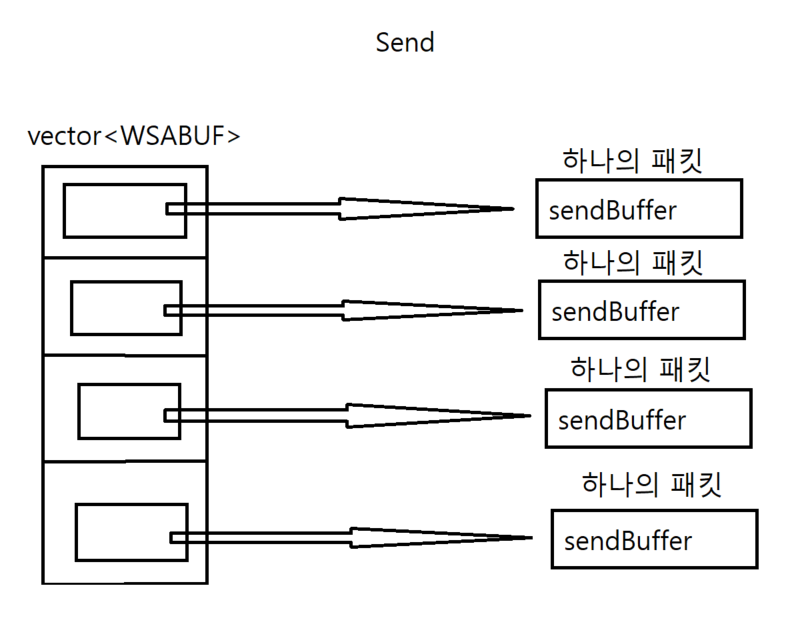
1. 락을 건다.
2. 보내고자하는 데이터를 큐에 넣는다.
3. registerSend를 true를 반환받는 스레드는 RegisterSend 함수로 진입할 수 있다.

다른 스레드는 보내고자하는 데이터를 큐에 넣기만 한다.

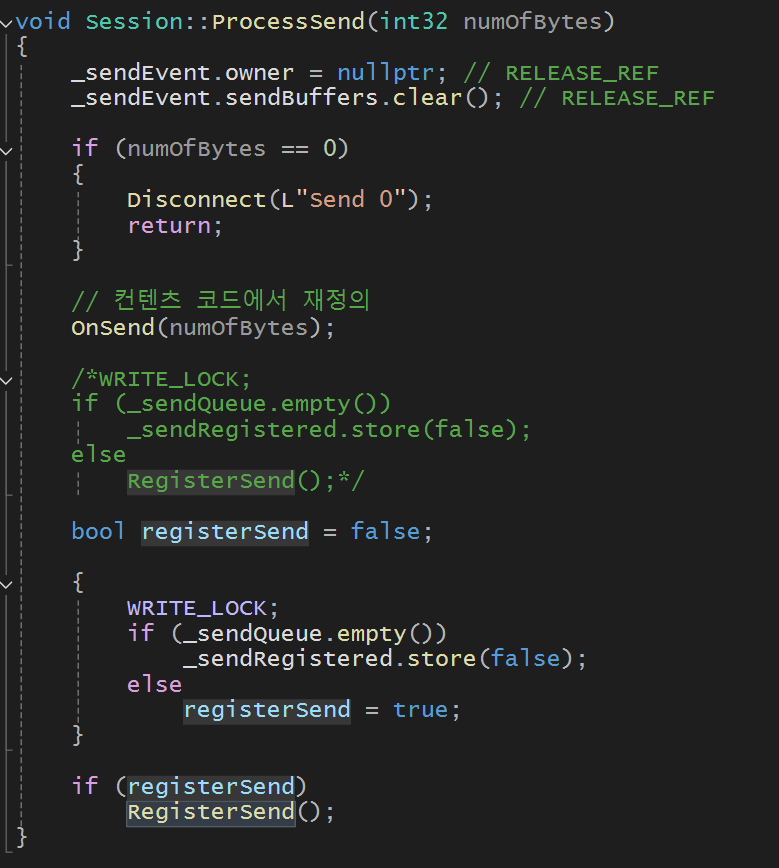


1. SendQueue에 들어있는 모든 데이터들을 \_sendEvent(iocpEvent) 객체의 멤버 데이터인 sendBuffers 벡터에 다 넣는다. 데이터들의 총 사이즈도 저장.
2. Scatter-Gather 기법을 이용하여 WSABUF를 받는 벡터를 wsabufs 를 만든다.
3. Wsabufs에 지금까지 sendBuffers 벡터에 넣었던 모든 데이터들을 넣는다.
4. WSASend를 해서 지금까지 SendQueue에 있던 모든 데이터들을 클라이언트에게 전송 한다.

**(중요! SendQueue는 공유 데이터이므로 락을 걸고 접근하는 것이다.)**

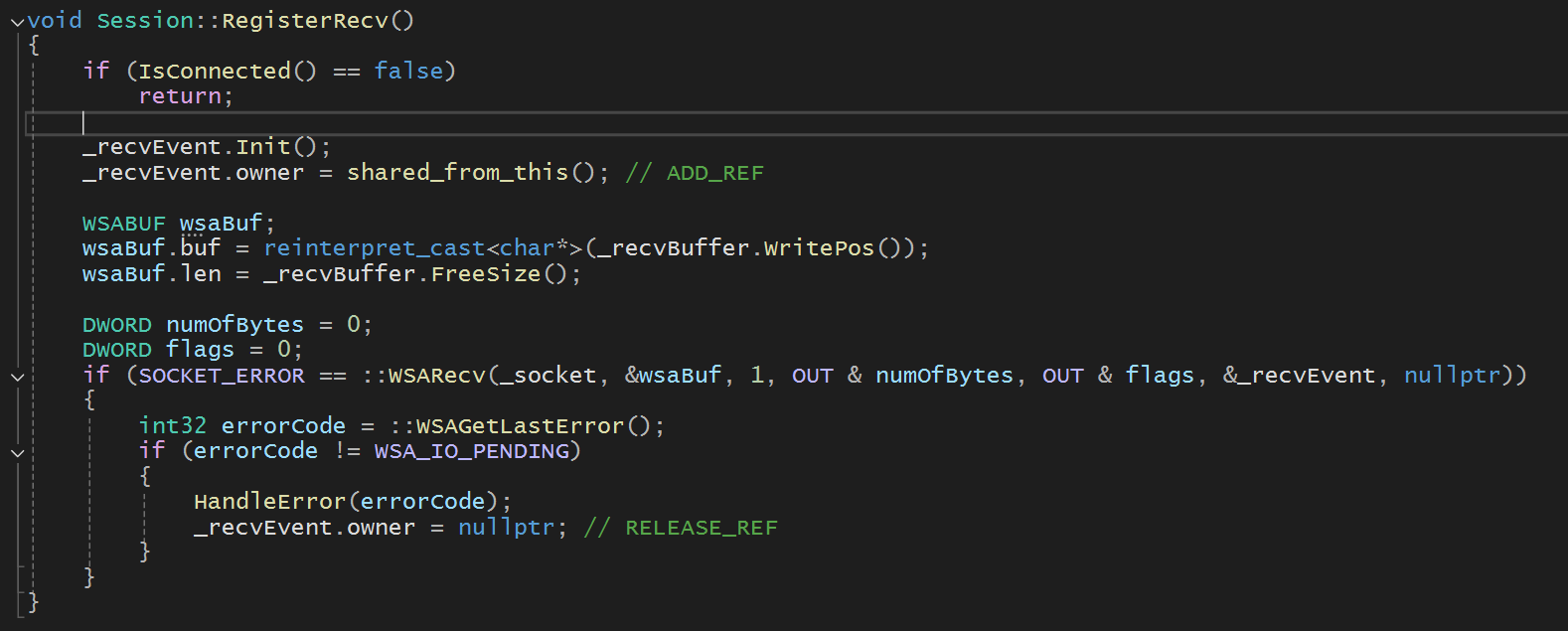


위 Send하는 과정에서 데이터를 Scatter-Gather 기법으로 보내는 것을 그림으로 그려봤다.



Send를 마치면 ProcessSend함수가 실행된다. 여기서 주목할 부분은 \_sendQeueu를 확인하여

보내야할 데이터가 또 추가로 들어왔다면 또 WSASend를 하므로 데이터 전송이 delay 되는 것을 막는 것이다.



외부에서 들어온 데이터를 받을 버퍼를 만들고 WSARecv를 호출하는 코드이다.

그리고 데이터를 전송 받게되어 커널에서 통지가 오면



완료통지를 받고 깬 스레드가 ProcessRecv를 실행한다.

그리고 OnRecv를 호출하여 패킷의 내용에 따라 적절한 처리를 수행하는 것이다.

그리고 다시 RegisterRecv로 진입하여 WSARecv를 호출하는 것이다.

다음에 또 올 데이터를 받기 위함!

**Send 와 Recv 는 차이가 있다.**

위의 코드들을 분석해보면 Recv의 경우에는 한번만 하게되고, Send는 여러 번 할 수 있어야한다. 서버의 입장에서 여러 클라이언트의 요청이 들어올 것이다. 그렇다면 요청에 맞는 데이터들을 패킷으로 만들어서 send 해야하므로 한 번만 send를 해서는 안되고 요청이 들어올 때 마다 send를 해야한다.

위 코드에서는 \_sendEvent (IocpEvent를 상속받은) 라는 객체를 재사용하기 위해서 락을 걸고 맨처음 WSASend를 수행하는 스레드가 보내는 것이고, 그 찰나에 다른 스레드가 send를 하면 중복된 WSASend 호출과 \_sendEvent의 이용을 피하기 위해 sendQueue에 자신이 보낼 데이터만 넣는 것이다.

학교에서 게임서버를 수강했을 때는, send 할 때마다 Overlapped 구조체를 동적할당하여 그냥 보내버렸는데 강의와는 차이가 있었다.

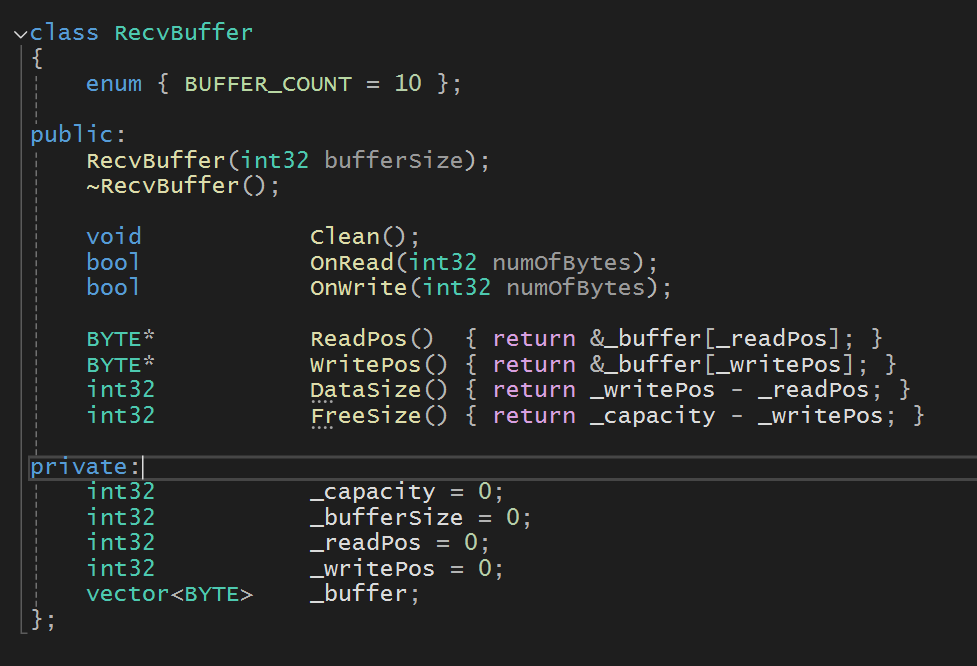
**Recv Buffer.cpp & Send Buffer.cpp**

일단 Recv Buffer 부터 정리 분석.

외부로부터 데이터를 수신하면 커널은 이를 유저가 지정한 버퍼에 복사해주고

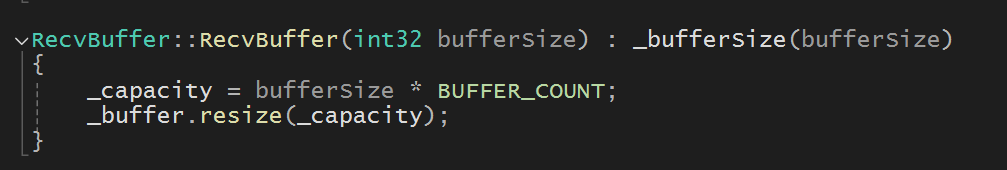
유저에게 통보를 해준다. 그에 따른 데이터 처리는 위에 요약한 것이다.

여기서는 사용자가 데이터를 전달받을 버퍼를 좀 더 효율적으로 관리하기 위해서 Recv Buffer 라는 클래스를 만든 것이다.

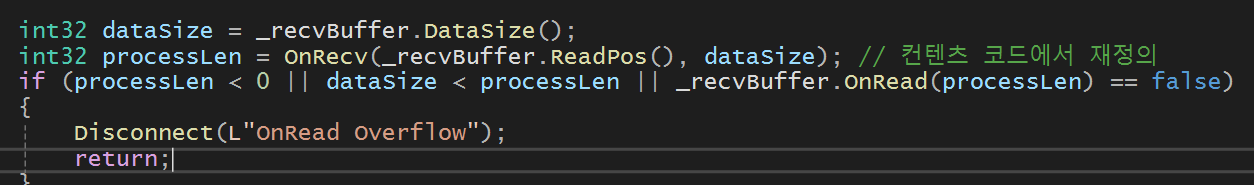


Vector<BYTE>로 데이터를 담을 버퍼를 멤버 변수로 가지고있다.

Session에서 RecvBuffer 객체를 하나 가지고 있는 것을 확인했었다.

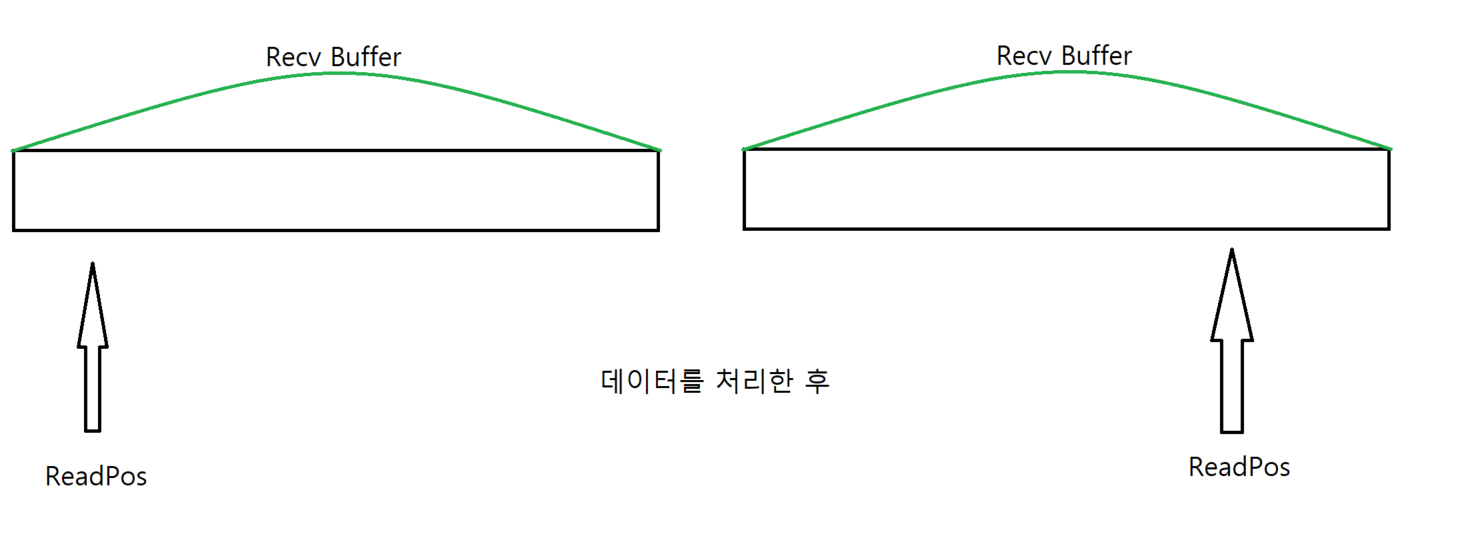


Session 클래스에서 RecvBuffer를 객체로 가지고있으니, Session을 생성하면 자동으로 RecvBuffer 객체는 생성된다. RecvBuffer 생성자에서 유저가 설정한 크기만큼의 버퍼가 동적할당된다.



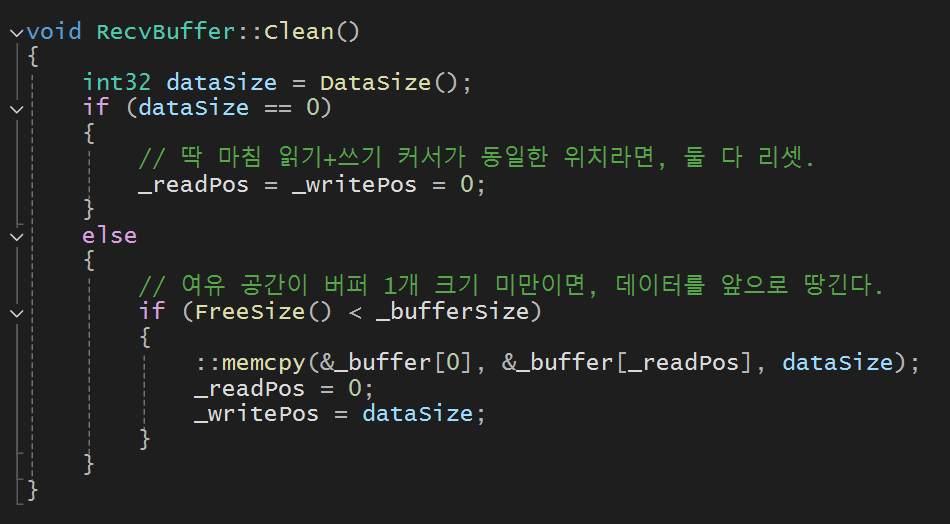
이는 Session 클래스에서 ProcessRecv 함수에서 얽어온 데이터를 처리하는 함수이다.

ReadPos 함수는 현재 RecvBuffer의 멤버 데이터인 버퍼의 현재 위치를 알려주는 함수이다.

그러면 OnRecv(\_recvBuffer.ReadPos(), dataSize)를 넘겨줌으로써 RecvBuffer에 있는 데이터들을 적절히 처리할 것이고 처리한 데이터 크기 만큼 버퍼의 현재 위치는 달라질 것이다.

그 모습을 그려봤다.

데이터를 처리하였다면 다음으로는 OnRead 함수를 호출하여 이상이 없는지 체크한다.



그리고 Clean 이라는 함수도 RecvBuffer에 있는 데이터를 처리하고 호출되는 함수 중 하나이다.

저렇게 처리를 하여 유연하게 RecvBuffer를 관리할 수 있는 것이다.

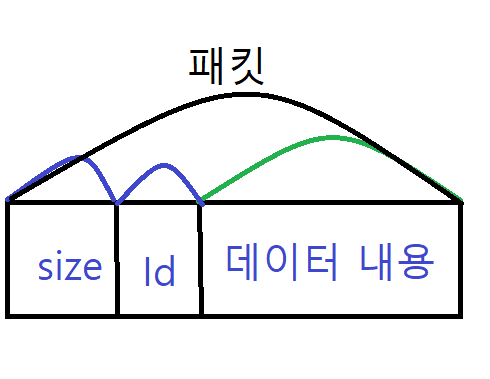
데이터를 읽어들인 크기와 처리한 크기가 같다면 커서의 위치를 0으로 되돌리면 되는 것이고,

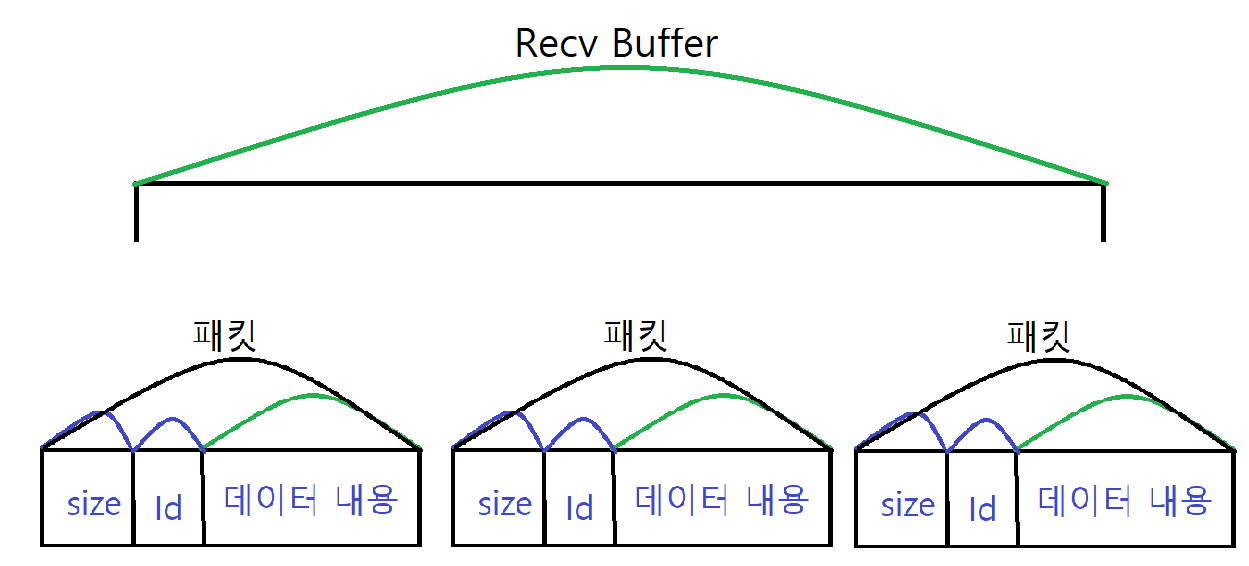
만약 RecvBuffer의 데이터를 받아들이 공간이 부족하다면 기존의 데이터들을 Index 0으로 복사하고 커서의 위치도 0으로 초기화하는 것이다.

**서버와 클라이언트간에 데이터 처리를 위한 프로토콜**

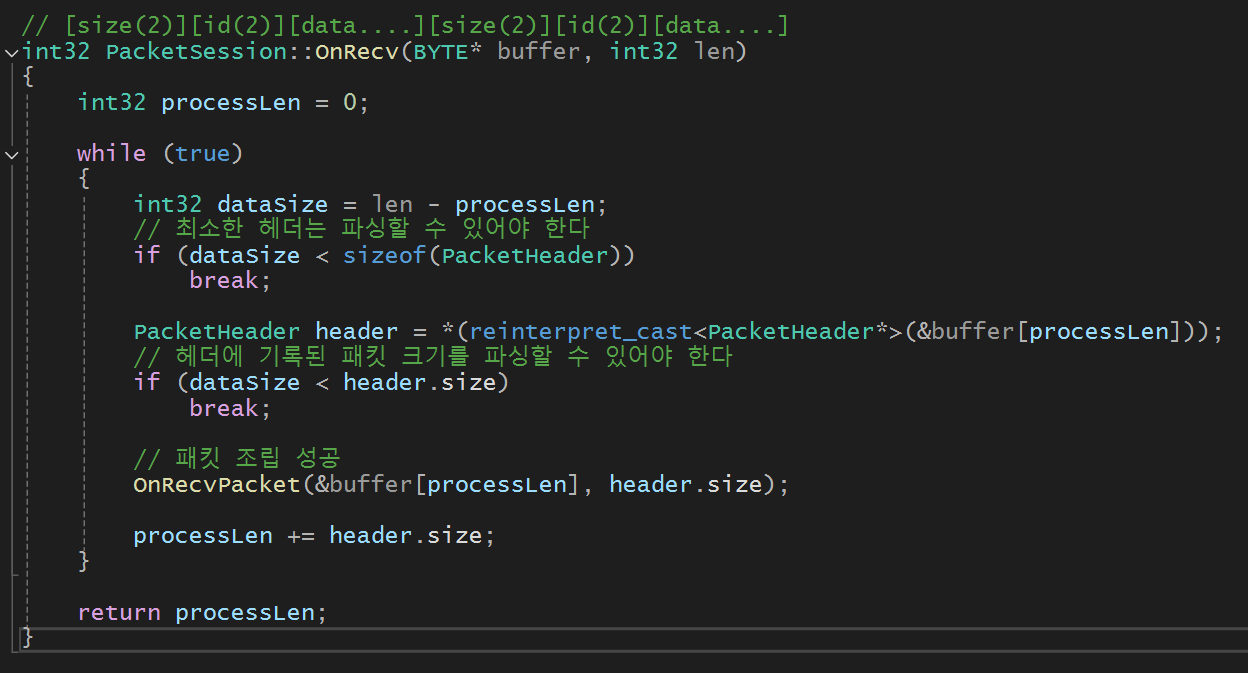
서버와 클라이언트간에 패킷을 주고 받을 것이다. 그것을 유저영역에 있는 RecvBuffer에서 패킷을 받아서 이를 적절히 처리하여 클라이언트와 서버는 통신을 할 것이다.

아래 size + id + 데이터 내용이 하나의 패킷을 이루는 것이다.





서버는 여러 클라이언트로부터 패킷을 받을 것이고 이를 각 패킷에 맞는 요청을 적절히 처리하면 되는 것이다. 클라이언트도 마찬가지다. 서버로부터 받은 패킷을 그에 게임 로직에 맞게 처리하면 되는 것이다.



위 코드를 통해 RecvBuffer에 있는 데이터들을 처리하게 된다.

**Google Protobuf**

강의에서 패킷을 만들 때, 외부 라이브러리인 Google Protobuf를 사용하여 만들었다.

이는 사용자가 클라와 서버가 주고받을 구조체 패킷 내용을 작성하면, protobuf가 해당 패킷을 알아서 변환해주고 직렬화와 역직렬화 기능을 제공하여 사용자가 편리하게 패킷을 작성하여 통신할 수 있도록 해준다. 그리고 서로 다른 언어 환경에서도 패킷을 주고받게 할 수 있다고 한다.

--------------------------------------------------------------------------------------------------------------

진행 상황 유튜브 링크:

|  |  |  |  |
| --- | --- | --- | --- |
| **문제점 정리** |  | | |
| **해결방안** |  | | |
| **다음주차** | 7주차 | **다음기간** | 8. 5 ~ 8. 12 |
| **다음주 할일** | 7주차에 계획되어있는 내용들 | | |
| **지도 교수**  **Comment** |  | | |