**네트워크 라이브러리 및 채팅 서버 클라이언트 구현 문서**

**1.제출일자: 2022 년 01월 25일**

**2.포트폴리오: 네트워크 라이브러리 및 채팅 서버 및 클라이언트 구현**

**3.작성자: 최지원**

**<목차>**

1. **프로젝트의 소개 및 개요**
   1. 프로젝트 소개
      1. TCP와 UDP 장단점
      2. 비동기 및 동기 입출력의 차이점
      3. 입출력 모델의 종류 및 내용
2. **프로젝트의 설계 및 세부적 기능 단위로 다이어그램 기술**
   1. UML 다이어그램
      1. IOCP 완료 포트 서버 시퀀스 다이어그램 (첨부)
      2. IOCP 완료 포트 서버 클래스 다이어그램 (첨부)
      3. WSAAsyncSelect 클라이언트 시퀀스 다이어그램 (첨부)
      4. WSAAsyncSelect 클라이언트 클래스 다이어그램 (첨부)
      5. 통신에 쓰이는 프로토콜 패킷 구조 (첨부)
3. **프로젝트의 구현된 주요 기술 단위로 분석 및 설계 기술**
   1. IOCP(Input Output Completion Port) 기반 채팅 서버
      1. 주요 변수 및 함수 설계
   2. DirectX11, ImGui를 적용한 WSAASyncSelect 기반 채팅 클라이언트
      1. 주요 변수 설계 및 함수 설계
4. **최종 결과 및 추가된 내용**
   1. 결과물 화면 (첨부)
   2. 범용성/ 유연성/ 확장성/ 간결성 고려하여 추가된 내용
5. **프로젝트의 소개 및 개요**
6. **프로젝트 소개**

본 프로젝트는 IOCP(I/O Completion Port) 입출력 모델 기반의 TCP/IP 네트워크 라이브러리와WSAAsyncSelect 입출력 모델 기반의 네트워크 라이브러리를 자체적으로 구축한다.

자체적으로 구축한 라이브러리를 활용하여 여러 유저를 수용 가능한 채팅 서버와 DirectX11 ImGui(그래픽 사용자 인터페이스 라이브러리)를 적용한 채팅 클라이언트를 구현한 프로젝트이다. 실시간으로 서버와 클라이언트가 패킷을 주고받으며 채팅을 할 수 있는 프로그램이다.

1. **TCP와 UDP 장단점**

네트워크 프로그램을 만들 때는 먼저 사용할 소켓 타입을 결정해야한다. 소켓에는 두가지 정보를 입력해야 하는데, 주소 체계와 소켓의 타입이다. 주소 체계에는 다음과 같은 종류가 있다.

|  |  |
| --- | --- |
| Name(이름) | Address Family(주소체계) |
| AF\_INET | IPv4 인터넷 주소 체계 |
| AF\_INET6 | IPv6 인터넷 주소 체계 |
| AF\_LOCAL | 로컬 통신 UNIX 주소 체계 |

소켓 타입에는 TCP와 UDP로 나뉘게 된다. TCP는 Transmission Control Protocol의 약자이고, UDP는 User Datagram Protocol의 약자이자, 두 프로토콜 모두 전송계층에 속한다.

전송 계층이란, IP에 의해 전달되는 패킷의 오류를 검사하고 재전송 요구 등의 제어를 담당한다. 하지만 서로 다른 특징을 가지고 있다.

|  |  |  |
| --- | --- | --- |
| 구분 | tCP | UDP |
| 신뢰성 | 신뢰성을 위해 ACK, Checksum 등 사용 | 신뢰성이 없음 |
| 연결성 | 연결 지향성 Connection을 맺고 통신 | 비 연결성 |
| 재전송 | 재전송 요청 (오류 및 패킷 손실 검출시) | 재전송 없음 |
| 특징 | 흐름제어를 위해 Windowing 사용  속도는 느려도 신뢰성 순서성을 제공 | 신뢰성을 보장하지 않지만 고속 데이터 전송, 실시간 전송에 적합 |
| 용도 | 신뢰성이 필요한 통신 | 총 패킷 수가 적은 통신, 동영상 및 음성 등 멀티미디어 통신 |

TCP는 연결 지향성이어서 Connection을 맺고 통신을 한다. 연결이 되어 있어서 전송이 되었는지 확인이 가능하다. 즉, 신뢰성이 보장된다. UDP는 일방적으로 한 쪽에서 보내기만 한다.

그런 특성으로 UDP는 10byte 데이터를 보내고 싶으면 한 번에 10byte를 모두 전송되거나 10byte 전부 실패한다. 반면에 TCP는 10 byte가 도착될 수도 5byte씩 2번 도착할 수도 1byte씩 10번 도착될 수도 있다.

TCP는 신뢰성이 있는 전송이 중요할 때에 사용하는 프로토콜이며, 전송 순서와 신뢰성 있는 데이터를 전송하는 장점이 있지만, UDP보다 비교적 느리다 라는 단점이 있다. UDP는 TCP보다 속도가 빠르고 네트워크 부하가 적다는 장점이 있지만, 신뢰성 있는 데이터 전송을 보장하지 않는다는 단점이 있다.

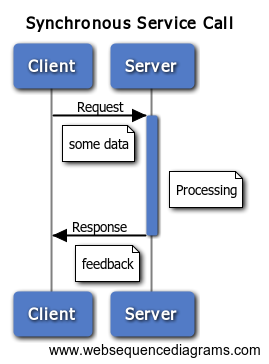
즉, 신뢰성이 요구되는 프로그램에서는 TCP를 사용하고 간단한 데이터를 고속으로 전송하고자 하면 UDP를 사용한다.

프로그래밍적으로 접근하자면 UDP는 연결이 안되어 있기 때문에, 프로그램적으로 신뢰성을 보장해줘야 한다. 또한 한 번에 받기 때문에, 나중에 받은 것이 먼저 보냈던 걸 수도 있다. 어떤 게 먼저 보낸 건지 시간 설정을 고려해야 할 것이다.

TCP는 패킷이 쪼개져서 연속적으로 들어오기 때문에 10Byte를 보냈으면 10Byte가 다 전송이 되었다는 걸 인지하고 포장해서 데이터로 써야한다. 패킷 상단에 몇 바이트를 보내는 패킷인지 명시하는 것을 고려해야한다.

1. **비동기 및 동기 입출력의 차이점**

동기와 비동기는 어떠한 프로세스를 처리하는데 절차적인 측면이다. 네트워크 프로그램에서 accept, connect, send, recv 등의 함수는 스레드를 블록상태에 놓이게 한다. 예를 들어, 서버에서 데이터를 받을 준비가 되었지만, 데이터를 보내지 않는다면, 애플리케이션은 블록상태에서 대기를 한다. 이를 해결하기 위해서 논블록킹 소켓이 등장한다. 이는 소켓의 동작 방식을 나타내는 말이고, 입출력의 서비스 동작방식에서는 동기, 비동기로 나뉜다.

비동기 입출력의 필요성은 예를 들어 동영상 스트리밍을 볼 때, 영상을 모두 로드하고 영상을 시청하려면 영상을 로드 하는 동안은, 영상을 볼 수 가 없다. 비동기 I/O는 데이터 수신과 영상의 출력이 동시에 일어난다. WinAPI의 파일 입출력은 이러한 비동기 파일 입출력을 지원하고, 비동기 입출력과 비동기 통지를 결합한 형태의 대표적으로 Overlapped 모델과 IOCP 모델이 있다.

동기(Synchronous) 방식은 데이터의 요청과 결과가 한 번에 일어난다. 사용자가 서버에게 패킷을 보낸다면, 서버가 요청에 따른 응답을 보내기 전까지 다른 활동을 할 수 없으면 기다려야 한다. 동기 방식의 장점은 설계가 간단하고 직관적이어서 유지보수 디버깅이 쉽다. 단점으로는 다른 작업을 할 수가 없어서 프레임 시간을 낭비한다.

그림 동기 입출력

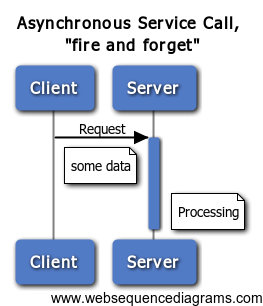
비동기(Asynchronous)방식은 요청을 보냈을 때 응답과는 무관하게 다른 활동을 수행이 가능하다. 장점으로는 요청에 따른 결과가 반환되는 시간 동안 다른 작업을 수행할 수 있다. 단점으로는 예외처리를 고려하고 동기화 객체를 사용하는 등, 동기적 방식에 비해 많은 비용이 발생한다.

그림 비동기 입출력

**iii. 입출력 모델의 종류 및 내용**

|  |  |
| --- | --- |
| **모델** | **내용** |
| **Select 모델** | Select 모델은 동기 입출력을 수행하기 위해 필요한 경우 대기 중인 하나 이상의 소켓 상태를 결정한다.  소켓을 만들고, Select함수를 통해 FD\_SET으로 해당 소켓을 등록하여 FD\_ISSET으로 관찰할 수 있는 방식이다.  **fd\_set ->** Readfds – 수신할 데이터가 있을 때  Writefds – 쓰기 가능한 상태일 때, 연결 성공했을 때  Exceptfds – 연결이 실패했을 때  소켓모드와 상관없이 한 스레드에서 여러 소켓을 처리할 수 있고, 윈도우, 유닉스 등 여러 운영체제에서 사용할 수 있어서 이식성이 좋지만, 성능은 여섯 가지 모델 중 가장 좋지 않다.  64개 이상의 소켓을 처리하려면 여러 개의 스레드를 사용해야한다. |
| **WSAAsyncSelect 모델** | Async는 비동기적이라는 의미로 윈속의 소켓이벤트를 윈도우 메시지를 통해서 비동기적으로 받는다. 윈도우 메시지를 통해 성공 여부를 판단한다. 윈도우 메시지 큐에 의존해 소켓의 처리가 이루어져 많은 수의 소켓 이벤트를 처리해야 하는 경우 큐가 고갈될 수 있다. 윈도우 메시지를 사용자 정의하여 원하는 네트워크 이벤트를 발생시킨다.  **네트워크 이벤트**  **FD\_WRITE** – WSAConnect 함수 호출하고 소켓이 처음 연결되었을 때, WSAAccept 호출 뒤 연결 수락되었을 때 send, WSASend, sendto, WSASendTO 함수 호출 후에 WSAEWOULDBLOCK이 리턴이 돼서 전송 버퍼가 비워졌을 때 이벤트 발생한다. 즉, 보낼 수 있는 상태가 되면 발생한다.  **FD\_ACCEPT –** 클라이언트가 접속했을 때  **FD\_READ –** 수신 가능할 때  **FD\_CLOSE –** 접속 종료할 때  **FD\_CONNECT –** 접속 완료될 때  **FD\_OBB –** 데이터가 도착할 때  소켓 이벤트를 윈도우 메시지 형태로 처리해서 윈도우 친화적이다. 서버보다 클라이언트에 유리하다. 하지만 하나의 윈도우 프로시저에서 일반 윈도우 메시지와 소켓 메시지를 처리해야 해서 성능저하의 요인이 된다. 또한 윈도우여야만 적용이 가능하다는 단점이 있다. |
| **WSAEventSelect 모델** | 두개 이상의 스레드가 협력해서 작업을 하는 경우에 상황에 따라 동기화가 필요하다. 그때 동기화 객체가 쓰이는데, event객체는 그 중 하나이며, 커널 오브젝트이기도 하다.  소켓과 WSACreateEvent()로 이벤트 객체를 생성하여 짝지어서 네트워크 이벤트가 발생함을 애플리케이션에서 알 수 있게 되는 방식이다.  WaitForMultipleEvent() 함수로 신호 상태가 된 커널 오브젝트에 네트워크 이벤트가 발생했다는 사실을 알 수 있기 때문이다. 반환 값은 신호 상태로 변화된 이벤트 배열의 인덱스이다.  또한, WSAEnumNetworkEvents()로 구체적인 네트워크 이벤트를 알려주는 함수도 있다.  Select 모델과 WSAAsyncSelect 모델의 특성을 혼합한 형태로 비교적 뛰어난 성능을 가짐에도 WSAAsyncSelect와 달리 윈도우 메시지가 아닌 이벤트 오브젝트를 사용해 윈도우가 아니어도 구현 가능하다.  단점으로는64개 이상의 소켓을 처리하려면 여러 개의 스레드를 사용해야한다. |
| **Overlapped 모델 I**  **(Overlapped Event)** | Overlapped은 중첩 입출력으로 응용 프로그램은 입출력 함수를 호출한 후에도 입출력 작업의 완료 여부와 무관하게 다른 작업을 하는 비동기 입출력을 지원한다.  IO가 여러개 중첩되는 것, 요청하고 기다릴 필요없이 리턴하고  다른 일을 수행하는 것이 Overlapped IO이고, 통지하는 방법에 따라 종류가 나뉜다.      그림 중첩 입출력  WSAEventSelect 모델과 비슷하게 이벤트 객체를 사용한다. 소켓 입출력 작업이 완료되면, 운영체제는 응용 프로그램이 등록한 이벤트 객체를 신호 상태로 바꾼다. 이벤트 객체를 관찰함으로써 입출력 작업 완료를 감지할 수 있다.  비동기 입출력을 지원하는 소켓을 생성하고 WSACreateEvent() 함수로 대응하는 이벤트 객체도 같이 생성한다. 비동기 입출력을 지원하는 소켓 함수를 호출할 때 WSAOVERLAPPED 구조체의 hEvent변수에 이벤트 개체 핸들 값을 넣어 전달한다.  동기 입출력이 완료가 되지 않았다면 오류를 리턴 하며. 오류 코드는 WSA\_IO\_PENDING으로 된다. 입출력이 완료가 되면 운영체제는 이벤트 객체를 신호 상태로 만들어 통보한다.  WSAWaitForMultipleEvent() 함수가 리턴 되면 WSAGetOverlappedResult()함수로 비동기 입출력 결과를 확인하고 데이터를 처리한다.  비동기 입출력을 통해 성능이 뛰어나지만, 이벤트 객체로 64개 이상의 소켓을 처리하려면 여러 개의 스레드를 사용해야 하는 단점이 있다. |
| **Overlapped 모델 II**  **(Overlapped Callback)** | 소켓 입출력이 완료되면 등록해둔 함수를 자동으로 호출한다. 일반적으로 운영체제가 호출하는 응용 프로그램 함수를 콜백함수라고 하는데, Overlapped 모델에서는 완료 루틴(completion routine) 이라고 한다.  비동기 입출력을 지원하는 소켓을 생성한다. 비동기 입출력 함수(WSASend, WSARecv)를 호출할 때 완료 루틴의 시작 주소를 함수 인자로 전달한다. 비동기 입출력 작업이 완료되지 않으면 WSA\_IO\_PENDING이 된다. 운영체제에 입출력 작업요청을 한다. WaitForSingleObjectEx(), WaitForMultipleObjectEx(), SleepEx(), WSAWaitForMultipleEvent() 등 함수로 해당 스레드는 완료 루틴이 호출될 수 있는 상태인 Alertable wait 상태로 진입하게 한다.  비동기 입출력 작업이 완료되면, 운영체제는 스레드의 APC큐에 결과를 저장하고 완료 루틴을 호출한다.  **APC 큐 : asynchronous procedure call queue는 비동기 입출력 결과를 저장을 위해 운영체제가 각 스레드에 할당하는 메모리 영역이다.**  비동기 입출력 함수를 호출한 스레드가 Alertable wait 상태에 있으면 운영체제는 APC 큐에 저장된 정보를 참조하여 완료 루틴을 호출한다. 완료 루틴 호출이 끝나면, 스레드는 Alertable wait 상태에서 빠져나온다. 스레드가 비동기 입출력 결과를 계속 처리하기 위해서 다시 Alertable wait 상태에 진입하는 동작 원리를 갖는다.  장점으로는 비동기 입출력을 통해 뛰어난 성능을 갖는다. 단점으로는 모든 비동기 소켓 함수에 대해 완료 루틴을 사용할 수 있는 건 아니다. 콜백 함수의 결과를 해당 콜백 함수를 수행한 스레드 만이 알 수 있다. |
| **IOCP**  **Completion Port** | 비동기 입출력으로 입출력을 요청하고도 다른 일을 수행하는 것이 Overlapped IO이다. 이 Overlapped IO를 이벤트와 콜백을 사용해 작업완료 여부를 체크하는 방법 중 하나로 윈도우에서 제공하는 일반적인 I/O 모델 중 최고 성능을 갖는 IOCP이다.  커널 객체(운영체제의 핵심적인 기능 프로세스, 스레드 메모리 관리 기능 제공하는 객체)이다. IOCP 시스템을 관리하는 객체를 커널이 만들어서 우리에게 HANDLE를 넘겨주는 것이다.   |  |  | | --- | --- | | IOCP 자료구조 | 설명 | | Device list | hDevice, Completion Key로 구성되어 키 값으로 값을 얻을 수 있게 됨 | | IO Completion Queue | FIFO로 앞에서부터 자료를 빼서 처리, I/O가 완료 되면 정보(이벤트)를 저장함 | | Waiting Thread Queue | LIFO로 마지막에 사용된 스레드를 다시 사용함, 스레드 ID가 저장됨  스레드 풀 역할 | | Released Thread List | Waiting Thread Queue(스레드 풀)에서 꺼내 온 쓰레드 정보 | | Paused Thread List | Release쓰레드가 Suspend상태되면 저장되고, Suspend상태가 해제되면 다시 Released Thread List로 올라감 |   IOCP 특징 중 하나가, 쓰레드 풀로 쓰레드를 재사용하는 것인데,  IOCP 본인의 스레드 큐를 가지고 작업을 하고 완료된 IO가 있다면 알려준다.  재사용이 가능한 스레드를 유지하는 스레드 풀링은  WSASend/ WSARecv같은 비동기 입출력 함수를 쓰면  스레드 내부적으로 APC 큐가 생성이 된다. 여기에 완료된 결과를 저장하는데  APC큐는 자동으로 생성되고 파괴된다. 그리고 그 큐는 본인 스레드에서만  확인이 가능하는데, 반면에 관리자 역할을 하는 IOCP는 모든 걸 확인이 가능하며, CloseHandle()함수 호출하여 파괴한다. 입출력 완료 포트를 접근하는 스레드를 별도로 두는데 이를 작업 스레드(WorkerThread라고 한다.)    예를 들어 관리자(IOCP)가 일꾼(1번 입출력), 일꾼(2번 입출력) 중  1번 일꾼의 IO작업이 끝났 다하면 이를 처리하기 위해 Waiting Thread Queue에  가장 최근에 들어온 스레드를 깨운다. I/O가 완료될 때마다 기다리고 쓰레드가 알아서 I/O작업을 해 성능이 좋다. -> 스레드 풀  또한, 특정 스레드에서 작업을 하던 간에IOCP 큐자체가 기타 스레드 결과가 저장이 된다.  즉, A 스레드에서 작업해도 B스레드에서도 확인이 가능하다. |

**소켓 입출력 모델 (Select, WSAAsyncSelect, WSAEventSelect는 동기 입출력과 비동기 통지를 결합한 형태**

**비동기 소켓 입출력 모델은 (Overlapped, Completion Port)는 비동기 입출력과 비동기 통지를 결합한 형태**

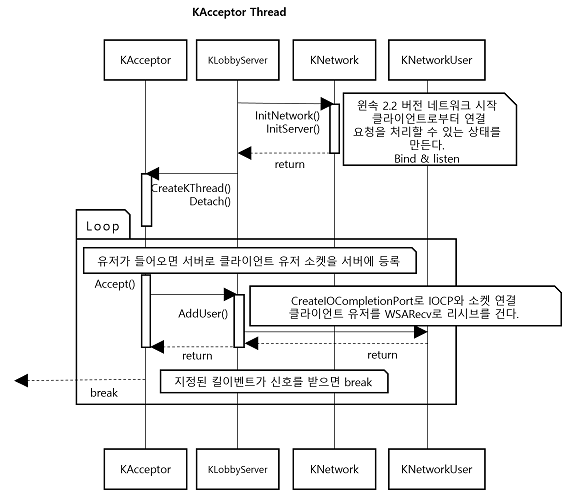
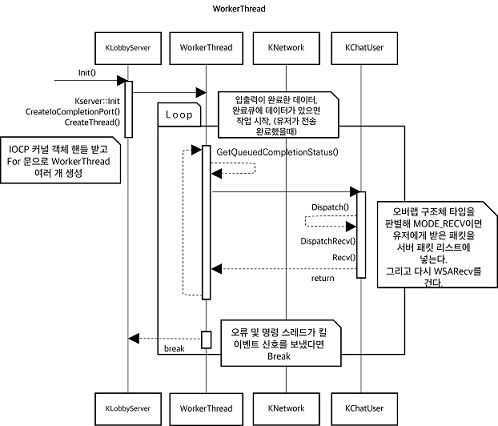
1. **프로젝트의 설계 및 세부적 기능 단위로 다이어그램 기술**
   1. UML 다이어그램
      1.  **IOCP 완료 포트 서버 시퀀스 다이어그램 (첨부)**

그림 1 클래스 다이어그램

그림 KAcceptor Thread 시퀀스 다이어그램

|  |  |
| --- | --- |
| KAcceptor :: Thread : 유저를 받아드리고, 완료 포트와 소켓 연결 담당 | |
| 1 | 서버 네트워크 초기화 |
| 2 | KAcceptor 스레스 생성 및 시작, 스레드 핸들 운영체제로 반환 |
| 3 | 유저가 들어오면 유저 소켓을 서버에 등록 |
| 4 | CreateIOCompletionPort함수로 **IOCP와 소켓 연결**, 클라이언트 유저를 수신 가능 상태로 만든다. |
| 5 | 서버의 스레드 킬 이벤트가 발생할 시에 반복문 탈출 |



**3**

그림 WorkerThread 시퀀스 다이어그램

|  |  |
| --- | --- |
| **Worker Thread : IOCP의 완료 큐에 완료된 입출력 작업하는 스레드** | |
| 1 | KLobbyServer에서 커널 객체 핸들 받고 여러 개의 WorkerThread 스레드 생성 및 시작 |
| 2 | 반복문을 돌며 GetQueuedCompletionStatus()로 완료된 입출력 확인 |
| 3 | 완료된 입출력이 있다면, 오버랩 구조체의 모드를 판별하고 패킷을 받고 패킷 풀에 저장 다시 WSARecv 호출 |
| 4 | 서버의 스레드 킬 이벤트가 발생할 시에 반복 문 탈출 |

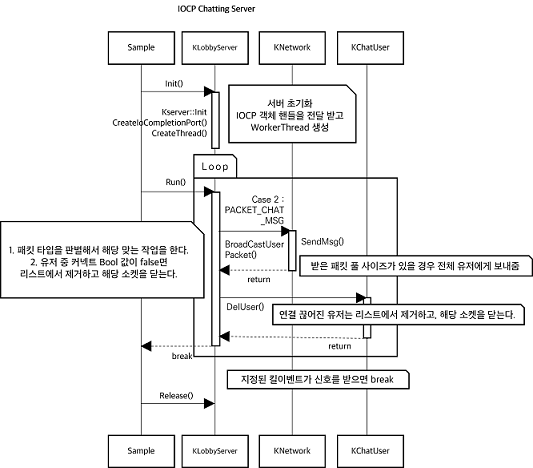


그림 Sample 메인스레드

|  |  |
| --- | --- |
| **Main Thread : WorkerThread에서 얻는 패킷풀로 중계하는 BroadCast 서버 역할, 유저 관리** | |
| 1 | CreateIoCompletionPort() IOCP 포트 객체 생성 및 핸들 반환 WorkerThread 생성 및 스레드 시작 |
| 2 | WorkerThread가 받아온 패킷 풀 사이즈가 있을 경우 패킷 프로토콜 판별해 메시지 타입이면 전체 유저에게 전송 BroadCastUserPacket(); |
| 3 | 연결이 끊어진 유저가 있을 경우 유저 리스트에 제거하고, 해당 유저 소켓을 닫는다. |
| 4 | 서버의 스레드 킬 이벤트가 발생할 시에 반복 문 탈출 |

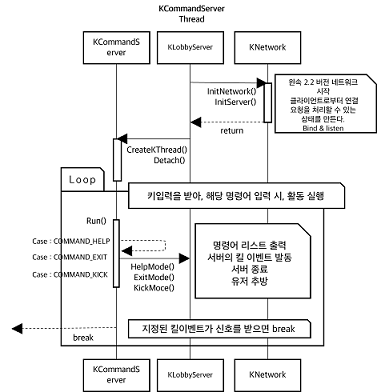


그림 KCommandServer Thread 시퀀스 다이어그램

|  |  |
| --- | --- |
| **KCommandServer : 서버 유저 인터페이스 서버 종료 등 명령어 처리 스레드** | |
| 1 | 서버 초기화 후에 스레드 생성 및 시작, 스레드 핸들 운영체제에 반환 |
| 2 | 반복문에서 블록 함수인 std::cin 입력 함수로 사용자 입력을 받음 |
| 3 | 스위치 문으로 명령어 저장된 자료구조 map에 포함되는 명령어를 입력 시 활동 시작. |
| 4 | /exit 입력 시 서버 종료 및 모든 스레드의 킬 이벤트 발동함 |
| 5 | 서버의 스레드 킬 이벤트가 발생할 시에 반복 문 탈출 |

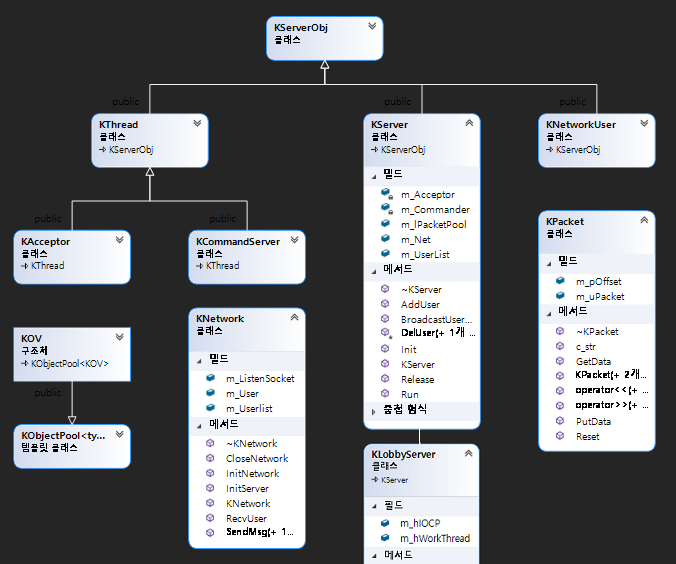
* + 1. **IOCP 완료 포트 서버 클래스 다이어그램 (첨부)**

그림 IOCP 서버 라이브러리 클래스 다이어그램

그림 1 클래스 다이어그램

|  |  |
| --- | --- |
| 구현된 기능의 주요 클래스 설명 | |
| KServerObj클래스 | 트리 서버 종료 이벤트, 임계구역 CriticalSection으로 스레드의 동기화를 담당 |
| KThread 클래스 | 스레드의 부모 클래스, 스레드 핸들 관리 및 스레드 시작을 담당 |
| KServer 클래스 | 서버의 부모 클래스로서 로비서버, 채팅서버 등 파생이 가능함, 서버 자체 패킷 풀을 가지고 있음 |
| KLobbyServer 클래스 | 서버의 자식 클래스로서 CreateIOCompletionPort 작업함 |
| KNetwork 클래스 | 네트워크 소켓에 주소 할당과 연결 요청하는 네트워크 작업 담당 |
| KNetworkUser 클래스 | 유저의 부모 클래스, Sample에서는 채팅 유저를 파생시켜 사용함 |
| KAcceptor 클래스 | 서버에 접속한 유저를 받아 서버 클래스에 전달한다. |
| KCommandServer클래스 | 서버의 사용자 인터페이스 제공, 명령어 입력 시 해당 활동 수행 |
| KObjectPool 클래스 | 오브젝트풀의 부모 클래스, 원형 큐로 구조로 재사용을 위해 템플릿을 사용 |
| KPacket 클래스 | 헤더에 패킷 헤더가 있는 패킷 관리 클래스, 패킷 생성 및 파싱 담당 |

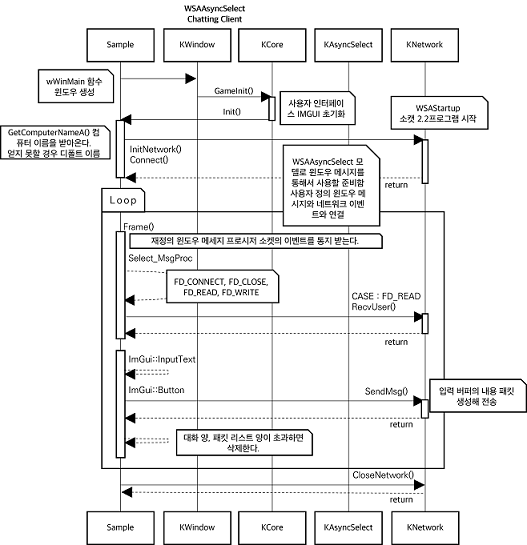
* + 1. **WSAAsyncSelect 클라이언트 시퀀스 다이어그램 (첨부)**

그림 WSAAsyncSelect 클라이언트 시퀀스 다이어그램

|  |  |
| --- | --- |
| **WSAAsyncSelect 클라이언트 시퀀스 다이어그램** | |
| 1 | wWinMain 함수 윈도우 생성 후 GUI 라이브러리 ImGui 초기화 및 객체 생성 |
| 2 | 소켓 생성 WSAAsyncSelect로 사용자 정의 윈도우 메시지로 네트워크 결과 받아올 준비 |
| 3 | 연결 실 패시, 아이피 입력 창 띄움 재시도. |
| 4 | 재정의 윈도우 메시지 프로시저로 소켓의 이벤트를 통지받는다. |
| 5 | FD\_READ 수신 가능하면 패킷 읽어 옴, 보낼 수 있으면 FD\_Write  IMGUI의 Input text와 button 활용해서 전송버튼 누르면 패킷 전송한다. |
| 6 | 대화 양, 패킷 리스트 양이 초과하면 삭제한다. |
| 7 | 네트워크 종료 |

* + 1. **WSAAsyncSelect 클라이언트 클래스 다이어그램 (첨부)**

****

그림 WSAAsyncSelect 클라이언트 클래스 다이어그램

|  |  |
| --- | --- |
| 구현된 기능의 주요 클래스 설명 | |
| KNetWork 클래스 | WSAStartup 및 메시지 전송 등 네트워크 관리 클래스 |
| KAsyncSelect클래스 | 소켓을 만들고 연결, 윈도우 메시지를 통해 비동기적으로 소켓 활용한다  재정의 윈도우 메시지 프로시저로 입출력 통지 반환 |
| KNetworkUser | 유저 관리 클래스 |
| KPacket 클래스 | 헤더에 패킷 헤더가 있는 패킷 관리 클래스, 패킷 생성 및 파싱 담당 |

* + 1. **통신에 쓰이는 프로토콜 패킷 구조 (첨부)**

패킷은 Pack 과 Bucket의 합성어로, 운송에서 적당한 덩어리로 나눠 행선지를 표시하는 꼬리표를 붙이는데, 이를 통신에 접목한 것이다. TCP프로토콜은 큰 데이터를 보낼 때 패킷이 끊어져서 보내진다. 프로그램에서 보내는 데이터도 패킷을 만들어서, 패킷 헤더에는 해당 데이터의 종류나, 사이즈를 먼저 보낸다. 그 패킷 헤더만큼 읽었으면 모든 데이터 사이즈만큼 읽을 때까지 계속 읽어오면 되는 로직이다. 즉, 컴퓨터 간 데이터를 주고받을 때 전송되는 데이터 조각이다.

데이터 조각인 패킷은 데이터 교환에서 서로 다른 시스템 간에 교환하기 위해서는 반드시 규정, 약속이 필요한데 이를 ‘프로토콜’이라고 한다.

게임에서는 게임 패킷 프로토콜을 정해서 데이터나, 구조체를 한 번에 보내는 것이 보다, 패킷을 만들어서 보내면 최적의 사이즈로 전송할 수 있어서 메모리 절약이 가능하다는 이점이 있다. 또한 암호화가 가능하다.

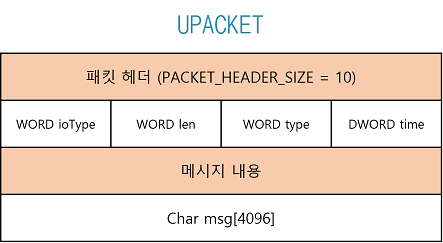


그림 사용된 패킷 구조

위 그림은 본 프로젝트에 사용된 패킷 구조이다.

ioType은 입출력 타입을 결정하고, len은 메시지의 길이를 결정한다. type은 열거형 Enum으로 구성된 채팅메시지 PACKET\_CHAT\_MSG, 혹은 PACKET\_LOBBY\_USER\_INFO등으로 패킷의 메시지형을 결정한다. time은 보낸 시간을 결정한다. 패킷 헤더가 보내지면, 패킷 헤더의 의미대로 메시지 내용이 해석되며, 최적의 사이즈로 보낼 수가 있다.

1. **프로젝트의 구현된 주요 기술 단위로 분석 및 설계 기술**
   1. IOCP(Input Output Completion Port) 기반 채팅 서버
      1. 주요 변수 설계

|  |  |  |  |
| --- | --- | --- | --- |
| **클래스**  **/구조체** | **타입** | **인스턴스**  **/변수명** | **내용** |
| KServerObj | CRITICAL\_SECTION | m\_cs | 유저 레벨의 스레드 동기화 객체인 크리티컬 섹션을 이용해 주로 각 스레드에서 안전하게 유저 리스트를 변경할 때 사용 |
| HANDLE | m\_hKillEvent | 이벤트 핸들로 에러나 종료 명령으로 서버를 종료시켜야 할 때 사용 |
| KThread | uintptr\_t | m\_hThread | 스레드 핸들 변수 |
| unsigned int | m\_iId | 스레드 아이디 번호 |
| bool | m\_bisStarted | 스레드 시작 유무 불값 |
| LPVOID | m\_pObject | 스레드 생성 시, 객체나 변수 인자를 전달해주고 싶으면 LPVOID형으로 받아 나중에 형변환해서 사용하는 변수 |
| KServer | KAcceptor | m\_Acceptor | 유저 받는 스레드 객체 |
| KCommandServer | m\_Commander | 사용자 명령 입력 스레드 객체 |
| std::list<KNetworkUser\*> | m\_UserList | 서버의 유저 리스트 |
| KNetwork | m\_Net | 네트워크 객체 |
| std::list<KPacket> | m\_lPacketPool | 서버의 패킷 풀 |
| typedef std::list<KNetworkUser\*>  ::iterator m\_UserIter; | m\_UserIter | 유저 리스트 iterator |
| KLobbyServer | HANDLE | m\_hWorkThread  [MAX\_WORKER\_THREAD] | GetQueuedCompletionStatus()로 완료된 입출력 확인하는 WorkerThread 객체 핸들 |
| HANDLE | m\_hIOCP | 완료포트 관리 객체 핸들 |
| KNetwork | SOCK | m\_ListenSocket | 서버의 리슨 소켓 |
| KNetworkUser\* | m\_User | 유저 포인터 변수 |
| KNetworkUser | KServer\* | m\_pServer | 유저가 갖고 있는 서버 객체 |
| SOCKET | m\_Sock | 유저의 소켓 |
| SOCKADDR\_IN | m\_Addr | 유저의 소켓의 주소 |
| std::string | m\_csName | Inet\_ntop로 변환한 유저의 아이피 string 값 |
| short | m\_iPort | Ntohs로 변환한 포트값 |
| bool | m\_bConnect | 유저의 연결 유무 |
| WSABUF | m\_WsaRecvBuffer | WSARecv로 받아놓는 패킷의 버퍼 |
| WSABUF | m\_WsaSendBuffer | WSASend로 보낼 패킷을 담은 버퍼 |
| char | m\_szRecv[2048] | 실제 데이터를 받는 버퍼 |
| char | m\_szRecvBuffer[2048] | 패킷 풀에 저장하는 버퍼 |
| int | m\_iPacketPos | 패킷의 시작 주소 |
| Int | m\_iWritePos | 현재의 저장 위치 |
| Int | m\_iReadPos | 패킷 시작 위치로부터 받은 바이트 |
| std::list<KPacket> | m\_lPacketPool | 유저 패킷 풀 |
| KObjectPool | void static void\* volatile | m\_ObjectPool  [POOL\_MAX\_SIZE] | Volatile 최적화 방지, 변수의 cache 사용 방지한다. 템블릿으로 여러 객체를 담아 놓을 수 있는 선형큐의 구조인 오브젝트 풀 배열이다.  스레드가 돌아가게 되면 통신에서는 Overlapped 구조체가 유저마다 있는데 메모리 변수 값 갱신하고 그 값을 다시 확인할 때 캐시로 확인하게 된다. 이를 방지하기 위해 volatile를 사용한다. |
| static long long volatile | g\_iHead | 생성 및 할당 인덱스 |
| static long long volatile | g\_iTail; | 소멸 및 해제 인덱스 |
| KCommanderServer | std::string | m\_Input | 명령어 입력 받는 스트링 변수 |
| int | m\_CommandMode | Switch 문에 사용되는 모드 확인 변수 |
| std::map<std::string,int>  ::iterator | m\_iter\_map | 명령어 모음 Map iterator |
| std::map<std::string, int> | m\_CommandMap | 명령어(string), 모드(int)로 짝을 이루는 Map 자료구조 명령어 리스트 변수 |
| KPacket | UPACKET | m\_uPacket | 데이터 전송에 쓰이는 패킷 객체 |
| char\* | m\_pOffset | 헤더와 별개로 메시지 길이를 저장하는 변수 |

* 주요 함수 설계

|  |  |  |
| --- | --- | --- |
| **클래스**  **/구조체** | **함수** | **내용** |
| KServerObj | KServerObj() | 모든 서버 스레드는 자식으로 두는 부모 클래스로 생성자 호출 시 CreateEvent로 서버를 종료하는 이벤트를 생성하고, 유저모드의 동기화 객체인 크리티컬 섹션을 초기화한다. |
| ~KServerObj() | 해제자 호출 시, 크리티컬 섹션을 삭제하고, 이벤트 핸들을 운영체제에게 넘긴다. |
| Thread | CreateKThread(LPVOID pValue) | \_beginthreadex()함수로 스레드를 시작하는 함수이다. LPVOID 매개변수로 원하는 객체나 변수를 스레드에게 겨줄 수 있다. 서버객체 포인터를 넘겨서, 스레드도 서버에 관한 작업을 할 수 있다. |
| Join() | 메인 스레드를 기다리는 함수로, 스레드가 끝날 때까지 기다리라고 명시하는 함수이다. WaitForSingleObject()를 사용한다. |
| Detach() | 스레드의 핸들을 운영체제에게 넘겨주는 함수이다. 스레드는 사용자가 임의적으로 종료할 수 없고, 커널 체로 운영체제가 관리하기 때문에, 명시적으로 운영체제에 돌려주는 함수이다. |
| KServer | Init(int port) | 서버 소켓 초기화하고, 유저 받아드리는 별도의 스레드 와 명령 처리하는 스레드를 시작한다. |
| BroadcastUserPacket  (KNetworkUser\* user) | 받은 패킷 풀 사이즈가 있을 경우 전체 유저에게 보내줌. 브로드 캐스트 기능을 하는 함수이다. |
| KLobbyServer | Init(int port) | CreateIOCompletionPort로 완료포트 관리 객체 핸들을 받아온다. 완료통지가 되면 작업을 할 스레드 WorkerThread를 생성해서 시작한다. |
| AddUser(SOCKET clientSock, SOCKADDR\_IN clientAddr) | KAcceptor 스레드에서 유저가 들어온 것을 감지하면 이 함수에서 유저 리스트에 유저를 추가한다. 그리고 소켓을 IOCP와 연결해서 입출력 완료 통지를 받게 해준다.  비동기 작업을 해야 하니 유저가 접속이 되면 리시브를 걸어 놔야 한다. WSARecv() |
| DWORD WINAPI WorkerThread(LPVOID param) | GetQueuedCompletionStatus() 완료 큐에 완료된 입출력이 있으면 작업을 한다. 받은 패킷을 조각 모음해서 패킷 풀에 저장한다. |
| Run() | 패킷 타입을 판별해서 해당 맞는 작업을 한다.  Switch문으로 사전에 정의한 프로토콜인 패킷 헤더의 타입이 PACKET\_CHAT\_MSG이라면 채팅메시지 타입으로 확인되어 모든 유저에게 해당 메시지를 전달한다. |
| KNetworkUser | Set(SOCKET sock, SOCKADDR\_IN addr, KServer\* pServer) | 유저의 소켓, 주소, 서버를 받아서 유저 객체를 초기화하고 생성한다. |
| Dispatch(DWORD dwTransfer, KOV\* ov) | Overlapped 구조체의 타입이 Send인지 Recv인지 switch문으로 판별해서 전송 받은 패킷의 데이터를 Recv인 경우에 DispatchRecv 함수를 호출해 데이터를 조각 모음해서 하나의 패킷으로 만든다. 다시 유저를 수신가능한 상태로 바꾼다. |
| Recv() | 수신 가능한 상태로 바꾸는 함수로 WSARecv를 사용한다. 비동기 처리이기 때문에 에러처리를 해줘야 한다. WSA\_IO\_PEDING이면 중첩 작업이 성공적으로 진행되었으며, 완료에 대한 측정은 나중에 이루어진다. |
| KAcceptor | Run() | 별도의 스레드로 accept() 논블록킹 함수여서 보통은 SOCKET\_ERROR에서 WSAGetLastError()를 확인해보면 WSAEWOULDBLOCK이다. 그러다가 유저가 접속하면 서버에서 유저를 추가하고 어떤 유저가 들어왔는지 출력한다.  서버에서 킬 이벤트를 발동하면 스레드의 반복문이 종료된다. |
| KCommandServer | Run() | 서버 콘솔창에 명령어를 입력하면, Map으로 저장된 명령어의 키 값과 일치하면, 해당 명령어에 맞는 작업을 실행한다. |
| ExitMode() | 서버의 킬 이벤트를 발동해서 모든 스레드 반복문을 빠져나오게 한다. |
| KObjectPool | AllFree() | InterlockedExchangePointer() 인터락 함수로 값을 안전하게 바꾼다. 메모리를 정렬해서 정해진 16의 승수 단위로 해제한다. |
| static void\* operator new(size\_t size) | new 연산자 오버로딩, 동적 메모리 할당을 정렬된 정해진 16의 승수 단위로 말록하는 \_aligned\_malloc(MEMORY\_ALLOCATION\_ALIGNMENT) 사용한다. 준비된 오브젝트가 있으면 준비된 오브젝트를 반환하고 없으면 신규로 생성해서 반환해준다. |

* 1. DirectX11, ImGui를 적용한 WSAASyncSelect 기반 채팅 클라이언트
     1. 주요 변수 설계

|  |  |  |  |
| --- | --- | --- | --- |
| **클래스**  **/구조체** | **타입** | **인스턴스**  **/변수명** | **내용** |
| KAsyncSelect | bool | m\_bConnect | 재정의 윈도우 메시지 프로시저에서 사용자 윈도우 메시지인 NETWORK\_MSG가 FD\_CLOSE를 반환 시 연결 해제한다. |
| PACKET\_HEADER | struct | PACKET\_HEADER | 패킷의 형태와 메시지의 크기를 정한다. |
| UPACKET | struct | UPACKET | 패킷으로 PACKET\_HEADER와 실질적인 메시지인 msg[4096]이 있다. |

* 주요 함수 설계

|  |  |  |
| --- | --- | --- |
| **클래스**  **/구조체** | **함수** | **내용** |
| KAsyncSelect | Connect(HWND hWnd, int type, int iport, const char\* ip)  { | WSAAsyncSelect()함수로 윈도우 메시지를 통해서 비동기적으로 소켓 활용 가능하게 한다. 소켓 구조체 채워서 입력한 아이피로 Connect 시도한다.. |
| Select\_MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) | 재정의 윈도우 메시지 프로시저이다. 사용자 윈도우 메시지인 NETWORK\_MSG를 WSAGETSELECTEVENT로 해석해 입출력 결과를 얻는다. |
| KPacket | KPacket(WORD iType) | 패킷 헤더의 type을 추가적으로 받아오는 생성자 |

1. **최종 결과 및 추가된 내용**
2. 결과물 화면 (첨부)

텍스트, 실내, 모니터, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 클라이언트 아이피 입력 및 채팅

텍스트, 모니터, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 서버 명령어 입력 및 종료

텍스트, 모니터, 스크린샷, 실내이(가) 표시된 사진

자동 생성된 설명

그림 IMGUI 유저 그래픽스 인터페이스 적용한 채팅창 모습

1. 범용성/ 유연성/ 확장성/ 간결성 고려하여 추가된 내용

- IOCP 서버에 별도의 스레드로 사용자 입력을 받아 명령을 입력하는 스레드 클래스 구현

- WSAAsyncSelect 클라이언트에 IMGUI 그래픽스 인터페이스 위젯 라이브러리를 활용해, 채팅창을 만들어서 버튼이나 입력창을 쉽게 구현하고 여러가지 동작이 가능하게 함

-라이브러리를 구축해서 다른 속성의 서버로 쉽게 이식할 수 있게 제작함

참고 문서 : TeamCenter 게이트 웨이 <https://docs.plm.automation.siemens.com/content/>