Socket Programming

35기 이찬규

목차

[1. TCP 와 UDP의 장단점 3](#_Toc528860566)

[2. 비동기 입출력과 동기 입출력 4](#_Toc528860567)

[3. 소켓 입출력 모델 4](#_Toc528860568)

[3.1. Select 4](#_Toc528860569)

[3.2. WSAASyncSelect 5](#_Toc528860570)

[3.3. WSAEventSelect 5](#_Toc528860571)

[3.4. Overlapped I/O 5](#_Toc528860572)

[3.5. IOCP 6](#_Toc528860573)

[4. 채팅서버 설계 7](#_Toc528860574)

[4.1. 서버 설계 7](#_Toc528860575)

[4.2. 클라이언트 설계 7](#_Toc528860576)

[5. 채팅서버 분석 8](#_Toc528860577)

[5.1. 서버 분석 8](#_Toc528860578)

[5.1.1. 개요 8](#_Toc528860579)

[5.1.2. ChatServer 9](#_Toc528860580)

[5.1.3. Acceptor 9](#_Toc528860581)

[5.1.4. ServerModel (IOCP) 10](#_Toc528860582)

[5.1.5. StreamPacket 10](#_Toc528860583)

[5.2. 클라이언트 분석 11](#_Toc528860584)

# TCP 와 UDP의 장단점

|  |  |
| --- | --- |
| TCP | UDP |
| 연결형 프로토콜 | 비연결형 프로토콜 |
| 신뢰성 있는 데이터 전송 | 신뢰성 없는 데이터 전송 |
| 일대일 통신 | 일대일 통신,  일대다 통신 |
| 데이터 경계 구분 안 함 | 데이터 경계 구분함 |

[표 1-1] TCP와 UDP의 특징

TCP소켓은 연결지향형 소켓으로, 연결지향형 소켓은 다른 연결지향형 소켓 하나와 연결이 가능하다. 그래서 연결지향형 소켓은 일대일 통신만이 가능하다.

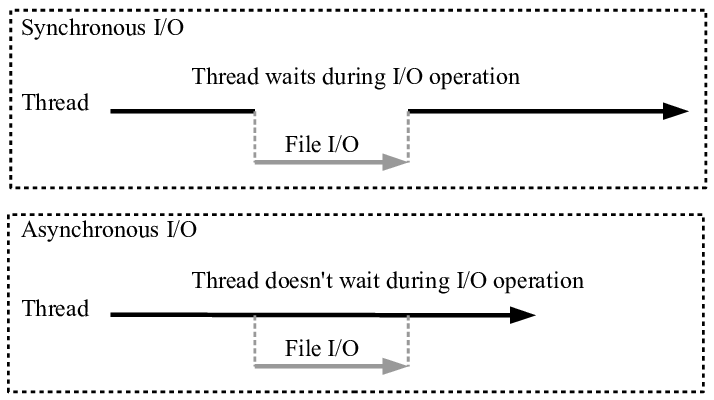
반면에 UDP 소켓은 비 연결지향형 소켓으로, TCP소켓과 달리 연결이라는 개념이 필요하지 않다. 그래서 TCP소켓으로 서버를 구성하게 되면 연결된 클라이언트마다 각각의 소켓을 만들어야 하지만, UDP 소켓은 소켓 하나로 처리를 한다. 그로 인해 TCP소켓은 일대일 통신만 가능한데 반해서 UDP소켓은 일대다 통신이 가능하게 된다.

TCP소켓은 데이터를 전송하게 되면 받는 쪽에서 데이터가 온전히 수신되었음을 송신한 소켓에 전달을 한다. 만약 송신을 했는데, 수신되었다고 답이 오지 않으면 다시 전송을 한다. 이를 통해 신뢰성 있는 데이터 전송이 가능해진다. 반면에 UDP소켓은 데이터를 송신만 할 뿐 받는 쪽에서 받았는지 아닌지 확인을 하지 않기 때문에 신뢰성이 없는 데이터 전송이라고 볼 수 있다.

TCP소켓은 소켓끼리 연결이 되어있는 통로로 데이터가 전송이 된다. 그로 인한 특징은 중간에 데이터의 소멸없이 전송이 되고, 전송한 순서대로 데이터를 받는다. 또한 전송되는 데이터의 경계가 존재하지 않는다. 데이터의 경계가 존재하지 않기 때문에 송신 측에서 여러 번의 함수 호출로 데이터를 전송해도, 수신 측에서는 이를 한번에 다 받을 수 있다.

UDP소켓은 소켓과의 연결이 없으므로, 전송된 데이터가 중간에 손실의 우려가 있고, 데이터의 경계가 존재한다. 그래서 만약 송신 측에서 데이터를 5번으로 나누어서 보냈다면, 수신 측에서도 5번의 함수 호출을 통해 데이터를 수신해야 한다. 이처럼 UDP소켓은 데이터의 손실이 일어날 수도 있고, 신뢰성을 보장받지 못하는 대신 빠른 속도의 데이터 전송을 지향한다. 이로 인해 전송된 순서에 상관없이 데이터가 수신이 된다.

# 비동기 입출력과 동기 입출력



[그림 2-1] 동기 입출력과 비동기 입출력

동기 입출력은 입출력 함수가 호출되고 해당 입출력 작업이 완료가 될 때 함수가 반환이 되는 형태이다. 이로 인해 동기 입출력은 입출력이 완료가 되기 전까지 쓰레드는 다른 작업을 할 수가 없다.

반면에 비동기 입출력은 입출력 함수를 호출하는 시점에 바로 반환을 해서, 입출력 작업이 진행되는 중에도 쓰레드는 다른 작업을 진행할 수가 있다. 때문에 비동기 입출력에서 쓰레드는 다른 작업을 진행한 후에 입출력 작업이 완료되었는지 확인을 해야 할 필요가 있다.

# 소켓 입출력 모델

## Select

Select 모델은 관찰하고자 하는 소켓들을 모아 놓고 소켓들의 이벤트 발생 여부를 판단할 수 있는데, 이로 인해 블로킹 소켓을 쓰고 하나의 쓰레드만 쓰는 프로그램에서도 블로킹 상태에 놓이지 않고 프로그램이 작동하는 모델이다. 또한 리눅스에도 같은 함수가 있어서 호환성 측면에서도 좋은 모델이라 할 수 있다. 이벤트는 3가지 종류가 있는데 수신한 데이터를 지니고 있는지를 판단하는 ReadSet과 블로킹 되지 않고 데이터 전송이 가능한 소켓을 찾는 WriteSet과 예외상황 발생여부를 판단하는 ExceptSet이 있다. 소켓을 이 Set들에 바인딩하고 Select함수를 호출하면 이벤트가 발생했을 때 Select함수가 반환이 된다. select함수의 반환 값은 변화가 발생한 소켓의 개수를 반환하는데 때문에 이벤트가 발생할 때 마다 바인딩된 소켓들을 for문을 통해 순회해야 하는 단점이 있다. 그 밖에도 for문을 한번 돌면서 이벤트들을 처리할 때 마다 fd\_set의 값을 0으로 초기화 하는 과정이 필요하고, 하나의 쓰레드에서 64명의 인원밖에 수용하지 못하는 단점이 있다. 그래서 소규모 서버를 운영하는데 유용하다고 할 수 있다.

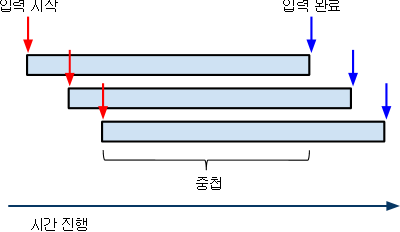
## WSAASyncSelect

WSAASyncSelect는 소켓과 관련된 이벤트를 윈도우 프로시저에 전달이 되기 때문에 하나의 쓰레드에서도 여러 소켓의 이벤트를 처리할 수 있다. 윈도우 프로시저에서 소켓의 이벤트를 전달받기 위해서는 사용자 정의 윈도우 메시지를 만들고 WSAAsyncSelect함수를 통해 소켓과 받고 싶은 네트워크 이벤트를 등록해야 한다. WSAASyncSelect를 사용할 때 유의점은 이벤트가 발생해서 메시지를 받을 때 그에 대응하는 소켓 함수를 호출하지 않으면, 해당 소켓은 동일한 이벤트의 메시지를 다시 수신할 수 없다. 따라서 메시지가 발생하면 그에 대응하는 함수를 호출해야 하며, 호출하지 못한 경우엔 PostMessage를 이용해서 메시지 큐에 직접 넣어서 해당 이벤트에 대한 메시지를 다시 호출하게 해야 한다.

## WSAEventSelect

WSAEventSelect는 WSAEvent를 통해 소켓의 네트워크 이벤트 발생 여부를 파악하는 모델이다. 기본적으로 WSAEvent배열을 생성하고 소켓을 생성할 때 마다 배열의 Event 하나씩 WSAEventSelect함수를 통해서 해당 소켓과 이벤트를 바인딩하고 처리할 네트워크 이벤트를 등록한다. 이벤트 발생 여부는 WSAWaitforMultipleEvents로 받을 수 있는데, 반환 값에서 WSA\_WAIT\_EVENT\_0을 빼면 이벤트 배열에서의 인덱스 값을 얻을 수 있다. 이벤트는 동시에 발생할 수 있기 때문에 해당 이벤트만 처리를 하면 안되고, 반환 받은 인덱스로부터 마지막까지 순회하면서 다른 소켓에서는 이벤트가 일어나지 않았나 확인하는 절차가 필요하다. WSAWaitForMultipleEvents는 이벤트 발생 여부만 알 수 있고 어떤 이벤트가 발생했는지는 알 수 없는데 이는 WSAEnumNetworkEvents의 인자 값에 들어가는 LPWSANETWORKEVENT 구조체로 알 수 있다.

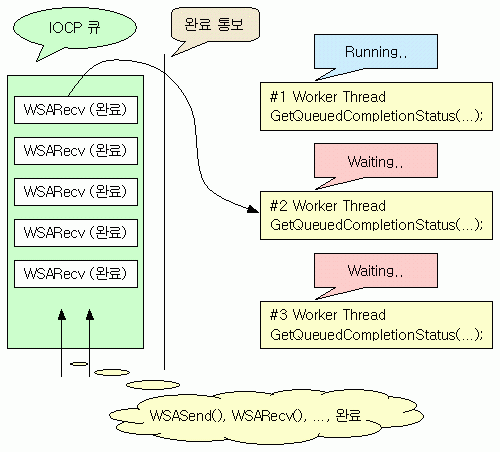
## Overlapped I/O



[그림 3-1] Overlapped I/O

기존의 소켓 입출력 함수는 동기 입출력 방식으로 해당 함수를 호출하면 이벤트가 발생하기전까지는 블로킹 되는 형태로 진행이 된다. 반면에 Overlapped I/O는 비동기 입출력 함수를 통해서 소켓의 입출력을 하게 되는데 이를 통해서 여러 개의 소켓의 데이터를 동시에 다룰 수 있게 된다. Overlapped I/O를 사용하기 위해선 기존의 send 와 recv 함수가 아닌 WSASend, WSARecv 함수를 사용하는데, 입출력의 완료를 확인하는 방법이 2가지가 있다. 하나는 이벤트 객체를 통해서 받는 방법이 있고, 또 하나는 콜백함수인 Completion Routine를 이용하는 방법이 있다. 입출력 함수를 호출할 때 Overlapped구조체를 인자로 전달해줘야 하는데 이벤트를 하나 만들어서 Overlapped 구조체에 멤버로 설정해주면, 입출력 완료 시점에 해당 이벤트가 신호상태로 바뀌게 된다. 결과적으로 쓰레드는 대기함수를 통해서 입출력 완료를 받게 된다. Completion Routine을 설정하게 되면 입출력이 완료되면 이 함수가 호출이 된다. 그런데 해당 쓰레드에서 중요한 작업을 진행중일 땐 Completion Routine함수가 실행이 되면 안되므로 쓰레드가 Alertable Wait 상태로 되면 완료함수가 호출이 된다. SleepEx, WaitForSingleObjectEx, WaitForMultipleObjectEx 등을 호출하면 쓰레드는 Alertable Wait 상태가 되고 이 시간에 완료함수가 호출이 된다.

## IOCP

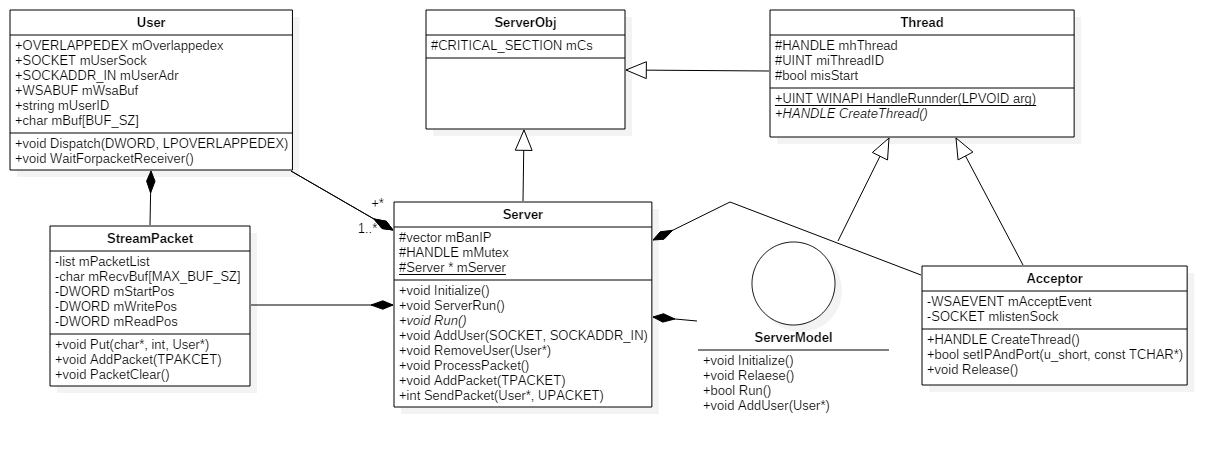


[그림 3-2] IOCP

Overlapped I/O 방식은 입출력의 완료를 반환 받는 작업을 함수를 호출한 쓰레드에서만 처리할 수 있다. 반면 IOCP는 다른 쓰레드에서도 처리할 수 있게 된다. 우선 Completion Port를 만들고 난 뒤 클라이언트의 접속이 생기면 소켓과 반환 받을 키 값을 Completion Port에 바인딩한다. 바인딩 된 소켓에 입출력 이벤트가 발생하면 그림 3-2와 같이 IOCP큐에 쌓이게 되고, 큐에 쌓인 이벤트들은 GetQueuedCompletionStatus함수로 반환 받을 수 있다. 이를 처리하는 쓰레드를 WorkerThread라 하는데, WorerThread는 입출력 이벤트가 없을 시에는 계속 블로킹 상태로 대기를 하다가 큐에 쌓이면 쓰레드가 깨어나서 작업을 하게 되는 방식이다. 일반적으로 WorkerThread는 CPU코어의 개수만큼 생성해서 사용한다.

# 채팅서버 설계

## 서버 설계

[그림 4-1] 채팅서버 다이어그램

서버에서 사용하는 클래스들은 모두 SeverObj를 상속하여 사용한다. ServerObj는 동기화 객체인 CRITICAL\_SECTION을 멤버로 가지며 필요에 따라 이 멤버변수로 임계영역을 설정한다.

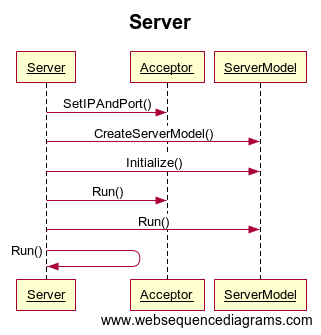
서버의 주요 구성 요소로 Acceptor와 ServerModel을 만든다. Acceptor는 클라이언트의 접속을 처리하는 클래스고, ServerModel은 클라이언트와 서버 간의 입출력을 처리하는 클래스이다. ServerModel은 인터페이스로 정의하고, 이를 통해 입출력 모델을 바꾸고 싶은 경우가 생기면 언제든지 ServerModel을 상속하는 새로운 클래스를 만들어서 작동이 되게 구성하였다. 가장 먼저 구현하고자 하는 ServerModel은 IOCP이다.

## 클라이언트 설계

클라이언트는 쓰레드를 한 개만 추가하고, 메인 쓰레드는 출력 및 서버로부터 받은 패킷을 처리하는 작업을 하고, 추가한 쓰레드에서는 입력을 담당하게 된다. 입력 작업은 하나의 쓰레드에서 전담하는 방식이기 때문에 블로킹형태인 send를 사용하여 입력을 하게하고, 출력의 경우는 Overlapped I/O 방식을 사용하고 반환결과를 받는 방식은 CompletionRoutine을 사용한다.

# 채팅서버 분석

## 서버 분석



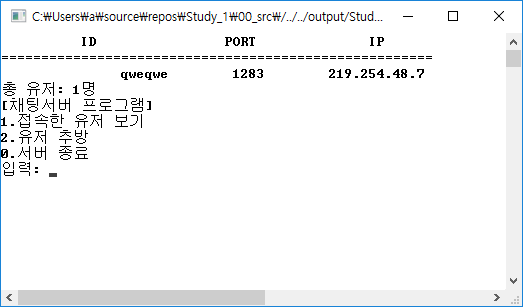
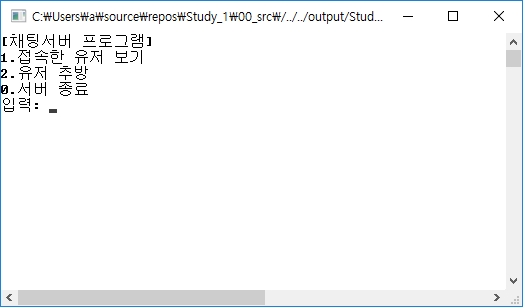
[그림 5-1] 서버의 시퀀스 다이어그램

### 개요

서버를 구동 시키기 위해선 서버의 Initialize함수를 통해서 서버의 멤버변수들을 초기화 시켜야 한다. 이 함수에선 기본적인 윈도우 소켓을 사용하기 위해 WSAStartup함수를 호출한다. 그리고Acceptor로 포트번호와 IP주소를 보내서 소켓을 만들고 보내준 포트번호와 IP주소를 통해 소켓에 bind한다. 마지막으로 listen함수까지 실행하면, 소켓이 클라이언트의 접속을 받을 준비가 끝나게 된다.

서버모델은 멤버를 포인터로 가지고 있어서 이 단계에서 할당을 받고 초기화를 진행하게 하였다. 서버모델은 스태틱 함수로 포인터를 반환하는 함수를 가지고 있는데 이 함수의 인자를 통해서 서버의 모델을 다른 형태로도 사용할 수 있게 구현하였다. 현재 구현 돼있는 모델은 IOCP이다. 초기화 과정이 모두 끝나면 ServerRun함수를 통해 서버가 돌아가게 된다. 앞서 초기화했던 멤버들을 실행시키고 서버의 Run을 호출하는데 이는 서버를 상속받는 클래스에서 정의하여서 메인 쓰레드에서 진행할 일을 구현할 수 있게끔 하였다.

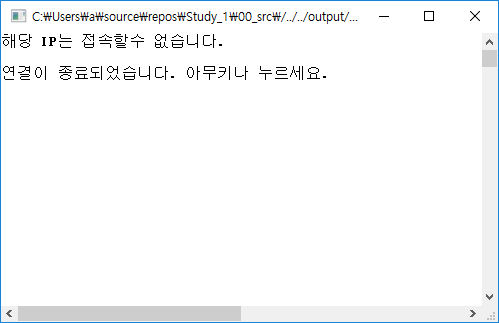
### ChatServer



[그림 5-2, 5-3] ChatServer 실행 모습

ChatServer 클래스는 Server를 상속받는 클래스로 메인쓰레드에서 진행이된다. ChatServer를 통해서 현재 서버에 접속중인 유저 목록을 확인할 수 있다. 또한 서버는 유저를 밴 시킬 수도 있는데 유저를 밴 하게 되면 서버의 밴 목록에 해당하는 유저의 IP가 등록이 되고 해당 유저는 재접속을 할 수 없게 하였다. 밴 목록은 메모리 상에 저장이 되기 때문에 프로그램 실행 중에만 유효하며 프로그램을 재시작 하게 되면 목록은 다시 초기화된다.

### Acceptor



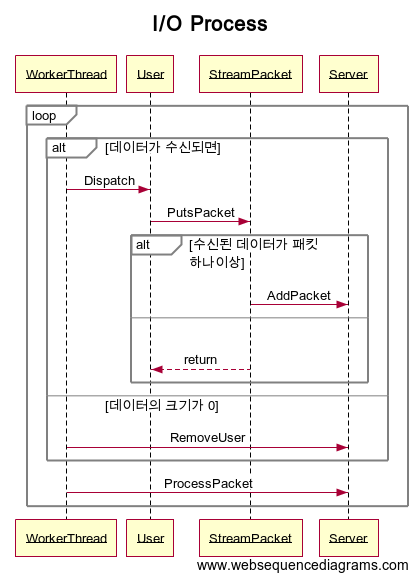
[그림 5-4] 밴을 당하게 되면 서버에 접속을 할 수 없다.

Acceptor는 클라이언트의 접속을 관리하는 객체이다. 별도의 쓰레드에서 진행이 되며, 클라이언트가 접속을 하게 되면 Acceptor는 서버의 밴 목록에서 클라이언트의 IP가 존재하는지 확인하는 작업을 거친다. 만약 밴 목록에 IP가 있다면 접속을 끊어서 서버에 들어올 수 없도록 한다. 밴 목록에 IP가 없다면 서버에 해당 클라이언트를 유저로 등록하는 과정을 거치게된다.

### ServerModel (IOCP)

현재 구현 되어있는 서버모델은 IOCP이다. IOCP는 초기화 과정 때 Comport 핸들을 만들고 GetSystemInfo 함수를 통해서 현재 컴퓨터의 CPU코어 개수를 얻게 된다. 그렇게 얻은 CPU의 코어 개수만큼 WorkerThread를 만들고 쓰레드들을 실행한다. 서버는 클라이언트를 등록하고 난 뒤 서버모델에도 해당 유저의 정보를 전달해준다. 이를 통해 Comport 핸들에 해당 클라이언트의 소켓을 바인딩 한다. WorkterThread는 실제 입출력을 처리하는 쓰레드로 CPU의 코어개수 만큼의 쓰레드가 생성이 돼서 작업을 한다. GetQueuedCompletionStatus 함수를 통해서 현재 바인딩된 소켓에 데이터 입출력이 완료 여부를 확인한다. 입출력이 완료가 되면 해당 유저객체에 받은 데이터를 그대로 보내서 그 객체내에서 데이터를 패킷별로 분류하게 하고, 연결이 끊어지면 유저리스트에서 해당 유저를 삭제하는 작업을 한다. 그 후에 서버에 저장된 패킷을 처리하는 함수를 호출하고 다시 GetQueuedCompletionStatus 함수를 호출하는 방식으로 진행된다.

### StreamPacket



[그림 5-5] 입출력의 과정

StreamPacket은 받은 데이터를 패킷별로 분류를 하는 객체이다. 기본적으로 서버의 소켓은 TCP소켓이므로 데이터의 경계구분이 되지 않는다. 따라서 한번에 recv로 하나의 패킷이 온전히 수신되지 않을 수도 있고, 하나의 패킷 이상 수신될 수도 있다. 패킷은 해당 패킷의 타입과 패킷의 크기를 담는 WORD 두개로 이루어진 패킷헤더 구조체와 실제 전달한 데이터를 담는 버퍼로 이루어져 있다. StreamPacket 객체에서는 이를 통해 받은 데이터가 패킷 헤더이상이 왔는지 확인하고 수신됐다면 이를 통해 해당 패킷의 크기만큼 받은 데이터를 잘라낸다. 그렇게 온전한 패킷 하나를 완성하면 이를 서버의 패킷풀에 저장한다. 서버는 저장된 패킷을 처리할 때 해당 패킷의 타입을 확인하고 그에 맞는 방법으로 진행을 한다. 예를 들면 채팅 메시지의 경우는 보낸 유저의 ID와 시간을 패킷에 덧붙이는 작업을 거친 뒤에 유저리스트에 유저들에게 다시 전송을한다.

## 클라이언트 분석

클라이언트는 초기화 과정에서 접속하고자 하는 서버의 IP와 포트번호를 입력 받고, 해당 서버에 접속을 시도한다. 이때 연결을 담당하는 쓰레드를 하나 생성하는데, 서버와의 연결을 5초간 기다리고 그 시간동안 응답이 없으면 프로그램이 종료되게 하였다. 연결이 성공적으로 이루어지면 ID를 입력하고 채팅서버에 들어가게 되는데, 이때 쓰레드를 두개 생성한다. 하나의 쓰레드는 유저로부터 키보드를 입력 받고 입력 완료된 메시지를 서버에 전달하는 입력 전담 쓰레드이고, 또 하나는 패킷이 들어왔을 때 처리를 하는 패킷 처리 쓰레드이다. 클라이언트는 입력과 출력의 모델을 다르게 쓰는데 입력 같은 경우는 기본 방식인 send를 통해 처리하고, 출력은 Overlapped 모델을 써서 처리한다. 출력이 완료되면 CompRoutine이 호출되어야 하는데 이를 위해 메인 쓰레드는 SleepEx를 통해서 매 프레임 0.1초가 대기상태에 들어가고 만약 출력이 완료된 상태라면 이 때 CompRoutine이 호출되는 방식이다. CompRoutine은 서버와 마찬가지로 데이터를 Diapatch 과정을 진행하고, 패킷풀에 패킷이 하나이상 쌓이면 SetEvent를 호출한다. 이 이벤트로 서버에 접속할 때 만들었던 두번째 쓰레드가 작동하게 되는데, 이 쓰레드는 패킷풀에 저장된 패킷을 처리하는데, 기존에 출력된 화면을 지우고 재출력을 하는 과정이 필요하다. 이를 위해 gotoxy 함수를 통해 커서를 이동시켜 한 줄 지우고 다시 출력하는 방식을 거친다. 이를 통해 유저는 입력중인 채팅 내용은 화면에 유지하고 갱신된 채팅 내용을 확인할 수 있다. 현재 프로그램은 채팅로그를 22개까지 저장하고 화면에 출력한다. 만약 22개가 넘어가면 먼저 들어온 데이터를 빼고 새로 들어온 데이터를 넣는 방식이다.