1. 한/영 Text 버퍼 결합

한글 버퍼와 영어 버퍼를 따로 두었던 것을 하나의 버퍼로 합쳐

문자를 입력하는 모습을 볼 수 있도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명- 실행 결과 화면

텍스트이(가) 표시된 사진

자동 생성된 설명

이전 위의 방법으로 한글과 영문 상태를 dwConversion 변수로 확인했는데 왜인지 영문상태 일 때 한글과 영문 모두 입력된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

이를 해결하기 위해 ChatMGR에 한/영 상태를 저장하여 구분하기로 했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이전에 영어를 입력하면 m\_textindex로 버퍼 위치를 찾고 영어 문자를 넣는 방식으로 진행했는데

한글 입력도 m\_textindex를 이용해 위치에 맞게 값을 넣어주면 잘 실행될 것이라 생각했다.

텍스트, 모니터, 화면, 블랙이(가) 표시된 사진

자동 생성된 설명

처음 if문은 조합중인 문자를 넣는 부분, 두번째 if문은 완성된 문자를 넣는 부분이다.

한글 문자를 입력 받고 출력하는 m\_textbuf에 같은 값을 넣어주는 방식으로 진행했다.

이 부분에서 WCHAR는 char과 달리 2바이트로 구성돼 있어 한/영 모두 하나의 자료형에 넣을 수 있다는 것을 알았다.

그리고 한글을 조합 중이었다가 영어 입력으로 바꿨을 때 index를 변형할 필요가 있을 것 같아 조합중인 단어가 있는 지 확인하기 위해 ChatMGR::m\_cobtext로 저장했다.

텍스트, 오렌지, 스크린샷, 가까이이(가) 표시된 사진

자동 생성된 설명

추가로 세세하게 해결해야 할 부분으론 현재 텍스트를 지울 때 단순히 index를 앞으로 보내고 문자의 끝에 NULL을 넣는 방식인데 만약 한글을 조합 중이고 중성이나 종성을 지우려 하는 작업(모음과 받침을 지우는 작업)을 하지 못한다는 단점이 있다.

1. 개인 콘솔 프로젝트 Client 네트워크 싱글 스레드화

텍스트이(가) 표시된 사진

자동 생성된 설명

이전까지 나는 클라이언트에서 네트워크를 통신할 때 스레드를 하나 더 만들어 Recv만 수행하여 데이터를 받기만 하는 서브 스레드가 하나 있는 것이 나름 괜찮은 방식이라 생각했다.

하지만 게임 서버 프로그래밍을 수강하면서 멀티 스레드를 만들었을 때 이런 방식으로 불필요한 컨텍스트 스위치를 일으키는 행위는 매우 비용이 큰 행위이며 싱글 스레드로 해결할 수 있는 방식은 싱글 스레드로 해결하는 것이 마땅하다는 것을 깨달았다.

따라서 클라이언트는 기존의 방식을 버리고, Overlapped I/O CallBack을 이용하는 방식으로 변경하는 것을 생각했다.

* 1. Socket의 확장 OVERLAPPED 구조체 변경

기존에 의미도 모르고 사용하고 있었던 OVERLAPPEDEX 구조체를 좀 더 효율적으로 사용하기 위해 게임 서버 프로그래밍 강의에서 사용하던 구조체를 수정하여 사용하기로 했다.

이전 확장 OVERLAPPED 구조체 형태

텍스트이(가) 표시된 사진

자동 생성된 설명

현재 확장 OVERLAPPED 구조체 형태

텍스트이(가) 표시된 사진

자동 생성된 설명

기본적인 기능은 같지만 패킷의 맨 앞부터 패킷의 길이와 본인의 ID를 추가 저장하여 보낸다.

또한 생성자를 만들어 데이터 버퍼를 넣었을 때 위의 작업을 알아서 해주게 끔 만들었다.

WSAOVERLAPPED를 클래스의 맨 앞에 두면 LPOVERLAPPED를 캐스팅을 이용하여 EXP\_OVER\* 으로도 부를 수 있기 때문에 전보다 더 편하게 사용할 수 있을 것이다.

* 1. Client 본격 싱글 스레드화

텍스트, 실내, 블랙, 스크린샷이(가) 표시된 사진

자동 생성된 설명

클라이언트는 do\_recv()와 do\_send()를 이용하여 Overlapped i/o를 걸어 통신하고 이 송수신이 완료되면 다음 형태의 콜백함수를 부른다.

텍스트이(가) 표시된 사진

자동 생성된 설명

recv\_callback()이 불릴 땐 데이터가 Overlapped 구조체가 포함한 버퍼에 채워졌다는 의미이니 이 정보를 처리하고 다시 do\_recv()를 불러 다음 송신을 준비한다.

send\_cllaback()이 불릴 땐 send가 완료된 상태이기 때문에 보낼 때 필요한 데이터(버퍼 등)를 해제해야 한다.

위 작업들이 잘 작성된 상태라면 클라이언트는 do\_recv()를 프로그램 시작할 때 한 번 호출하고, do\_send()는 보낼 데이터가 있을 때마다 해준다면 문제없이 작동할 것이다.

 - 프로그램이 시작할 때 한 번 호출하는 함수

 - do\_recv()를 추가하여 수신 대기를 한다.

 - 매 프레임마다 호출하는 함수

 -> 

블로킹 소켓으로 통신하던 부분을 do\_send()로 바꾼다