1. 클라이언트 요약 보고서

목차

---

0. 기능 요구사항

1. 클라이언트 지원 (TCP/UDP, IPv4/6 모두 지원)
    - ☑ ~~각종 도형 및 메세지 전송 (다각형 함수 사용 X)~~
    - ☑ ~~카카오톡 1 기능 지원 (읽음 알림)~~
    - ☑ ~~파일 전송~~
2. 서버 지원 (TCP/UDP, IPv4/6 모두 지원)
    - ☑ ~~소켓 입출력 모델 사용~~
    - ☑ ~~서버에서 넌블록킹 소켓 사용~~
    - ☑ ~~클라이언트 관리 기능 GUI~~
3. 서버-클라이언트 공통 지원 (TCP/UDP, IPv4/6 모두 지원)
    - ☑ ~~채팅 ID 추가~~
    - ☑ ~~메세지 전송 방식 : 고정 + 가변 길이~~
    - ☑ ~~UDP 프로토콜 지원~~
    - ☑ ~~소켓 옵션 활용(TTL, SO_REUSEADDR, 멀티캐스트, 넌 블록킹소켓)~~

---

1. 클라이언트 기능 요약(기능서, 최종기능)

1) 서버 접속

- 클라이언트는 원하는 원하는 서버의 IP주소와 포트번호를 이용해 {TCP/UDP, IPv4/v6}중 원하는 방식으로 접속을 할 수 있다.

2) 클라이언트 식별자 = CLIENT ID (20자리) + 고유 랜덤 번호

- 클라이언트는 서버와 다른 사용자에게 본인을 알릴 수 있는 ID(String)를 가지고 접속을 할 수 있다.
- 클라이언트는 Window 실행시에 발행되는 본인만의 고유 식별번호를 가지고 서버에 접속을 할 수 있다.
- clientUniqueID = (rand() * 19 - 19) % 256;

3) 서버 접속 후 채팅

- 클라이언트는 서버에 접속을 성공한 경우 해당 서버에 접속 되어있는 다른 클라이언트들과 서로 메세지를 주고 받을 수 있다.
- 채팅 메세지 전송시에는 클라이언트의 이름, 보낼 메세지, 보낸 시간이 포함된다.

4) 서버 접속 후 그림판 사용

- 클라이언트는 서버에 접속을 성공한 경우 해당 서버에 접속 되어있는 다른 클라이언트들과 그림판을 이용해 서로에게 그림이 그려지도록 할 수 있다.
- 그림판에서 보낼 수 있는 데이터 종류는 1-펜 그림, 2-직선, 3-삼각형, 4-사각형, 5-원, 6-지우개로 총 6가지가 있다.
- 펜 색상은 검은색부터 1-빨강, 2-초록, 3-분홍, 4- 파랑 총 5가지 색을 가지고 있다.
- 그림판 메세지 전송시에는 색, 그림 시작-종료 좌표, 굵기, 반지름이 포함된다.
- 해당 클라이언트는 본인의 그림판을 비우거나, 삭제 또는 새로 만들 수 있다.

5) 송신 및 수신 채팅 메세지 확인

- 클라이언트는 수신받는 메세지만을 Window에 출력한다.
- 클라이언트가 송신한 내용은 서버로 먼저 전달되고, 서버에서는 접속하고있는 모든 클라이언트에게 메세지를 전부 전송한다.
- 클라이언트는 수신한 메세지의 식별자가 본인이 보낸 것 이라면 송신 EditController(오른쪽)에 출력, 아니라면 수신 EditController(왼쪽)에 출력한다.

6) 카카오톡 1 기능(메세지 읽음 알림)

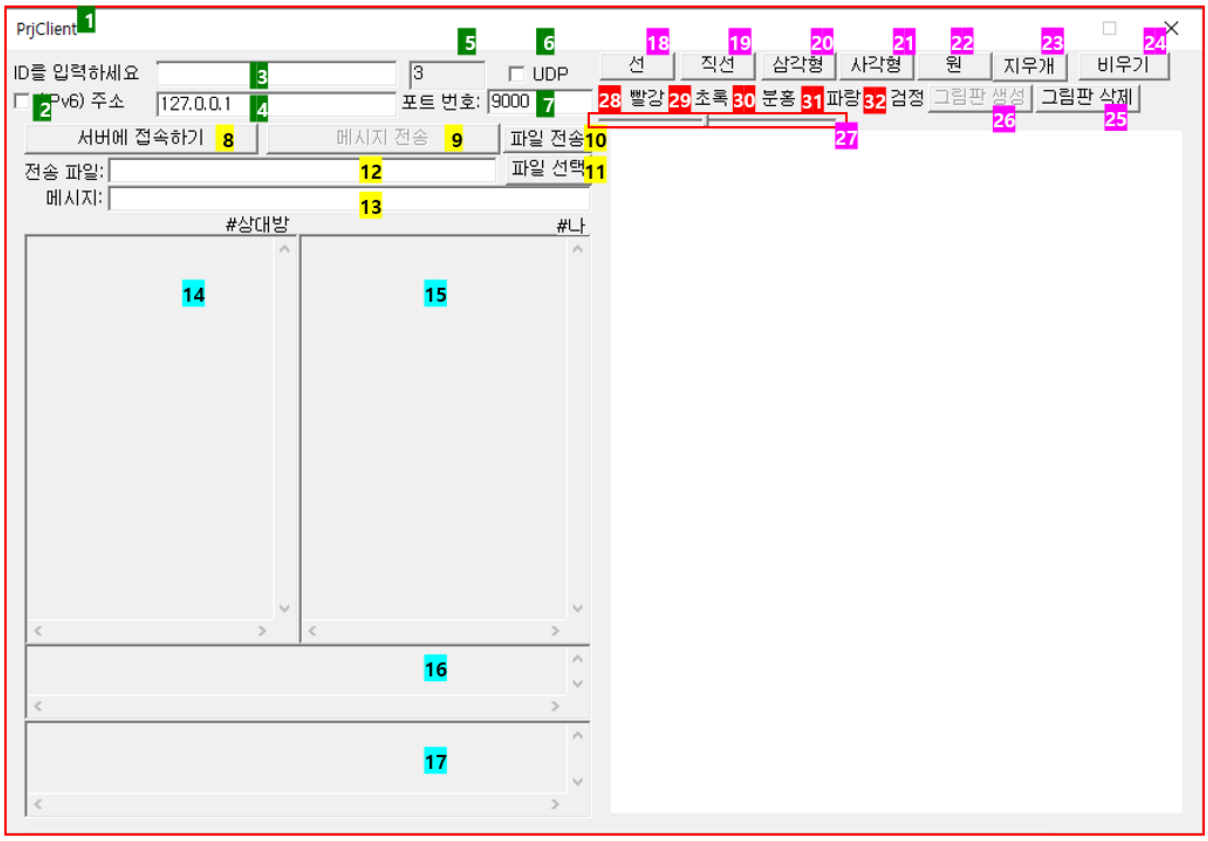
- 대화방을 띄우지 않고있다가 대화를 치려고 다시 Window창을 활성화 시키면 서버로 해당 클라이언트가 다시 대화를 시도하려고 하는 신호를 보낸다. (기준- 메세지 입력 란에 커서가 올라갈 때)
- 클라이언트는 읽음 알림이 도착한다면 본인이 보낸 메세지를 상대방이 봤을것이라고 확인이 가능하다.
- 읽음 알림 메세지는 채팅메세지 출력 EditControll 바로 밑에 있는 곳에 출력된다.

7) 파일 선택 후 송/수신 기능

- 클라이언트는 본인이 가지고 있는 파일을 선택해서 서버로 보낼 수 있다.
- 서버에 접속해있는 다른 클라이언트는 누가 어떤 파일을 보냈는지 알 수 있으며,
  본인이 송신한 파일이 아니라면 수신받은 파일 정보는 모두 “프로젝트 받은 파일” 디렉터리에 저장된다.

---

2. GUI 구성요소 요약



#	ID	Attr	부가설명
1	IDD_DIALOG1	다이얼로그 박스	메인 화면
2	IDC_ISIPV6	체크 박스	IPv6로 접속 여부를 결정한다.
3	IDC_USERID	에디트 컨트롤	클라이언트의 ID 문자열을 입력한다.
4	IDC_IPADDR	에디트 컨트롤	접속하려는 서버의 주소를 입력한다.
5	IDC_UNIQUEID	에디트 컨트롤	클라이언트의 고유 식별번호이다. (랜덤발행)
6	IDC_UDPCHECK	체크 박스	UDP 접속 여부를 결정한다.
7	IDC_PORT	에디트 컨트롤	접속하려는 서버의 포트번호를 입력한다.
8	IDC_CONNECT	버튼	유저 ID를 입력한 후 서버에 접속 요청을 한다.
9	IDC_SENDMSG	버튼	클라이언트가 작성한 메세지를 서버에 송신한다.
10	IDC_SENDFILE	버튼	선택한 파일을 전송한다.
11	IDC_SELECTFILE	버튼	팝업 창에서 파일을 선택할 수 있다.
12	IDC_FILEMSG	에디트 컨트롤	선택한 파일의 이름을 출력한다.
13	IDC_MSG	에디트 컨트롤	서버에 보낼 채팅 메세지를 입력한다.
14	IDC_STATUS	에디트 컨트롤	서버에서 수신된 메세지를 출력한다.
15	IDC_STATUS2	에디트 컨트롤	클라이언트가 보낸 메세지를 출력한다.
16	IDC_ONE	에디트 컨트롤	읽음 알림 메세지를 출력한다.
17	IDC_FILERECV	에디트 컨트롤	파일 수신 알림 메세지를 출력한다. (송신인, 저장경로 : 파일명)
18	IDC_LINE	버튼	일반 펜으로 그림을 그린다,
19	IDC_STRA	버튼	펜으로 직선을 그린다.
20	IDC_TRIA	버튼	삼각형을 그릴 수 있다.
21	IDC_RECT	버튼	사각형을 그릴 수 있다.
22	IDC_CIRC	버튼	원을 그릴 수 있다.
23	IDC_ERAS	버튼	그림판의 내용을 지울 수 있다,
24	IDC_BOARDCLEAR	버튼	그림판 프로시저를 새 하얀화면으로 초기화한다.

#	ID	Attr	부가설명
25	IDC_DELBOARD	버튼	그림판 프로시저를 없앤다.
26	IDC_NEWBOARD	버튼	새로운 그림판 프로시저를 생성한다.
27	IDC_THICK	슬라이더 컨트롤	펜 굵기를 조절할 수 있다.
28	IDC_COLORRED	라디오 버튼	펜 색을 빨간색으로 변경한다.
29	IDC_COLORGREEN	라디오 버튼	펜 색을 초록색으로 변경한다.
30	IDC_COLORBLU	라디오 버튼	펜 색을 파란색으로 변경한다.
31	IDC_COLORBLACK	라디오 버튼	펜 색을 검은색으로 변경한다.
32	IDC_COLORPINK	라디오 버튼	펜 색을 분홍색으로 변경한다.

---

3. 기능 상세

a) 사용자 정의 상수(서버-클라이언트 공용 시그널)

1. **사용자 정의 윈도우 메세지**
   WM_USER = 사용자가 사용할 수 있는 예약된 공간 포인터

```
// 사용자 정의 윈도우 메시지
#define WM_DRAWIT    (WM_USER+1)
```

2. **통신 정보 상수 (통신할 서버 IP주소, 포트번호를 가르킴)**

```
// 1) 통신정보 IP, PORT
#define MULTICAST_SEND_IPv4 "235.7.8.1"
#define MULTICASTIPv4 "235.7.8.2"

#define MULTICAST_SEND_IPv6 "FF12::1:2:3:9"
#define MULTICASTIPv6 "FF12::1:2:3:4"

#define SERVERIPV4  "127.0.0.1"
#define SERVERIPV6  "::1"
#define SERVERPORT  9000
#define REMOTEPORT  9000
```

3. **전송할 메세지 구조체 크기 및 필드 크기 (서버-클라이언트 공용)**
   구성: 타입판별 변수, 클라이언트 문자열 ID, 메세지 내용, 메세지 송신 시간, 클라이언트 고유번호 변수

```
// 2) 전송 메세지 크기
#define BUFSIZE     284                                                 // 구조체 크기 (기본 통신 고정 길이)
#define MSGSIZE     (BUFSIZE-(sizeof(int)*2)-ID_SIZE-CHECK_WHO-TIME_SIZE)// 보내는 채팅 메세지 사이즈
#define ID_SIZE     20                                                  // 클라이언트 ID (문자열 길이)
#define CHECK_WHO   1                                                   // 서버 보낸 식별 글
#define TIME_SIZE   23                                                  // 보낸 시간 정보 길이
#define FILE_SIZE   BUFSIZE - 4 - ID_SIZE                               // 송/수신 받는 파일메세지 버퍼 크기
```

4. **송/수신 메세지 식별 타입 (서버-클라이언트 공용)**

```
// 3) 전송 메세지 타입
// 3-1) 기본 메세지 전송 정보
#define CHATTING    2000                // type: 채팅
#define DRAWLINE    2001                // type: 선 그리기
#define DRAWSTRA    2002                // type: 직선 그리기
#define DRAWTRIA    2003                // type: 삼각형 그리기
#define DRAWRECT    2004                // type: 사각형 그리기
#define DRAWCIRC    2005                // type: 원 그리기
#define DRAWERAS    2006                // type: 지우개

// 3-2) 서버 Controll 메세지 전송 정보
#define ACCESS      3000                // type: 클라이언트 아이디 서버에 최초 전송
#define KICKOUT     3001                // type: 서버에서 추방 메세지

#define READCHECK   4000                // type: 읽음 알림 메세지
#define FILEINIT    4001                // type: 파일을 보내겠다고 알릴때 사용
#define FILEBYTE    4002                // type: 파일내용 전송시 사용
#define FILEEND     4003                // type: 파일 Open 후에 파일데이터를 모두 보낸 후에 사용 (feof)
```

b) 송/수신 메세지 정의 (size = 284, 모두 같은 크기의 구조체)

1. COMM_MSG : 메세지 최초 수신 구조체
2. CHAT_MSG : 사용자 송/수신 채팅 및 각종 알림 메세지 구조체
3. DRAWLINE_MSG : 사용자 송/수신 그림 메세지 구조체
4. FILE_MSG : 사용자 송/수신 파일 메세지 구조체

→ 사용자는 송신 시에 메세지 타입에 따라 구조체 포인터를 이용해 메모리 내에 있는 값을 그대로 전달한다.

→ 수신 시에는 같은 크기의 구조체를 이용하여 메세지 타입을 판별하고 포인터를 매칭 하는 방식으로 사용한다.

1. COMM_MSG 구조체

```
// S-1) 서버로부터 받는 메세지 포인터
// sizeof(COMM_MSG) == 284
struct COMM_MSG{
    int  type;
    char dummy[BUFSIZE-4];
};
```

→ 서버와 클라이언트가 식별할 수 있는 메세지 타입이 4가지이다 (전송/ 최초 접속 / 카카오톡 1기능/ 추방)

- <채팅 메세지> g_chatmsg : type = CHATTING
- <최초 접속> g_initmsg : type = ACCESS
- <카카오톡 1 기능> g_kakaoTlak1 : type = READCHECK
- 그림 타입, KICKOUT, 파일관련 타입은 아래 다른 란에서 설명

→ 기본동작 구조 g_chatmsg : 서버로부터 오는 메세지를 맨 처음에 받는 구조체이다.

→ 서버로부터 들어오는 데이터를 해당 구조체에 저장하고, 타입에 따라 구조체 포인터에 값을 넘겨준다.

```
// 사용예시
COMM_MSG comm_msg;
CHAT_MSG* chat_msg;
DRAWLINE_MSG* draw_msg;
CHAT_MSG* fileinit_recv;
FILE_MSG* fileRecv;

retval = recvn(g_sock, (char*)&comm_msg, BUFSIZE, 0);
if (comm_msg.type == CHATTING)
      chat_msg = (CHAT_MSG*)&comm_msg;
else if (comm_msg.type == READCHECK)
      chat_msg = (CHAT_MSG*)&comm_msg;
else if (comm_msg.type == DRAWERAS)
      draw_msg = (DRAWLINE_MSG*)&comm_msg;
else if (comm_msg.type == FILEINIT)
      fileinit_recv = (CHAT_MSG*)&comm_msg;
else if (comm_msg.type == FILEEND)
      fileinit_recv = (CHAT_MSG*)&comm_msg;
else if (comm_msg.type == FILEBYTE && recvFile != NULL)
```

```
        fileRecv = (FILE_MSG*)&comm_msg;
    else
        draw_msg = (DRAWLINE_MSG*)&comm_msg;
```

2. CHAT_MSG

```
// S-2) 채팅 메시지 정보 구조체
// sizeof(CHAT_MSG) == 284
struct CHAT_MSG{
    int  type;
    char client_id[ID_SIZE];
    char buf[MSGSIZE];
    char whenSent[TIME_SIZE];
    int whoSent;
};
```

→ 클라이언트가 송신한 메세지 내용을 포함하는 구조체이다.

→ 1-데이터가 채팅 메세지라는 것을 명시해줄 type 변수

→ 2-클라이언트 ID(20자리 문자열), 2-메세지 내용, 3-메세지 송신 시간, 4-클라이언트 식별 번호 로 이루어져있다.

3. DRAWLINE_MSG

```
// S-3) 그림 정보 포함 구조체
// sizeof(DRAWLINE_MSG) == 284
struct DRAWLINE_MSG{
    int  type;
    int  color;
    int  x0, y0;
    int  x1, y1;
    int  width;
    int  r;
    char dummy[BUFSIZE - 28];
    int whoSent;
};
```

→ 클라이언트가 송/수신할 그림의 데이터를 포함하는 구조체이다.

→ 1- 그림 데이터 타입(선, 직선, 삼각형, 사각형, 원, 지우개)

→ 2- 색, 3- 시작좌표, 4- 종료좌표, 5- 굵기 가 들어간다. (나머지는 사용 x)

4. FILE_MSG

```
struct FILE_MSG {
    int type;
    char client_id[ID_SIZE];
    char buf[FILE_SIZE];
};
```

→ 클라이언트가 송/수신할 파일 데이터의 송신자와 파일 데이터 조각을 담은 메세지이다.

→ 1- 파일 데이터타입, 2-파일 송신자, 3- 파일 데이터 조각

c) 전역변수 및 프로시져 지역변수

```
// 1) TCP소켓 전역변수
static SOCKET        g_sock;

// 2) UDP소켓 전역변수   (송/수신 UDP 소켓)
static SOCKET      listen_sock_UDPv4;
static SOCKET      send_sock_UDPv4;
static SOCKET      listen_sock_UDPv6;
static SOCKET      send_sock_UDPv6;

// 3) 소켓 주소 정보 (IP, Port, Family)
```

```
static char          g_ipaddr[64];        // 서버 IP 주소
static u_short       g_port;              // 서버 포트 번호
static BOOL          g_isIPv6;            // IPv4 or IPv6 주소?
static BOOL          g_isUDP;             // 체크하면 UDP, 아니면 그냥 TCP

// 4) 스레드 핸들(소켓 통신용)
static HANDLE        g_hClientThread;
static HANDLE        g_hReadEvent, g_hWriteEvent; // 이벤트 핸들
static volatile      BOOL g_bStart;               // 통신 시작 여부

// 5) 윈도우 전역 변수
static HINSTANCE     g_hInst;             // 응용 프로그램 인스턴스 핸들
static HWND          g_hDrawWnd;          // 그림판 윈도우
static HWND          g_hButtonSendMsg;    // '메시지 전송' 버튼
static HWND          g_hEditRecv;         // 받은 메시지 출력 (상대방)
static HWND          g_hEditSend;         // 보낸 메시지 출력 (내가 보냄)
static HWND          g_EditUserRead;      // (카카오톡 1) 읽음 알림메세지 출력
static BOOL          g_boardValid;        // 현재 그림판 활성화 상태

// 6) 통신 메세지 정보
static HWND          hEditUserID;         // 사용자 ID
static CHAT_MSG      g_chatmsg;           // 기본 채팅메세지 프로토콜 형태
static DRAWLINE_MSG  g_drawmsg;           // 그림 정보 메세지 프로토콜 형태
static int           g_drawcolor;         // 선 색상

static int       clientUniqueID;        // 클라이언트 식별 번호
static char      strUniqueID[5] = { 0,0,0,0,0 };  // itoa(clientUniqueID, strUniqueID, 10);

// 7) 추가 채팅 프로토콜
static CHAT_MSG   g_initmsg;            // 최초 전송 메세지(메세지 내용없이 사용자 ID를 보냄)
static CHAT_MSG   kakaoTalk1;           // 카카오톡 1 기능을 위한 메세지(g_chatmsg)에 Focus를 받으면 전송
static CHAT_MSG   fileinit_msg;         // 파일 전송 시작/종료 알림 메세지(송신 식별 정보 및 파일 이름)

// 8) 파일 관련 전역변수(WINDOW 에서 사용할 파일 변수)
OPENFILENAME OFN;                           // Window 파일 변수
const UINT nFileNameMaxLen = MSGSIZE;       // 보낼 파일 명 길이
WCHAR szFileName[nFileNameMaxLen];          // 보낼 파일 명
FILE_MSG      fileRecv;                     // 받은 파일 데이터 조각 이진 버퍼
FILE*       recvFile;                       // 받을 파일 포인터
```

d) 사용자 정의 함수 & 스레드

```
// 0) TCP/UDP, IPv4/6 프로토콜에 따른 일괄 전송함수
int SendByProtocol(char* msg);
// 1) TCP 소켓 통신 스레드
DWORD WINAPI ClientMain(LPVOID arg);
DWORD WINAPI ReadThread(LPVOID arg);
DWORD WINAPI WriteThread(LPVOID arg);
// 1-1) TCP Receive 수신 메세지 처리 함수
int recvn(SOCKET s, char* buf, int len, int flags);

// 2) UDP 소켓 통신 스레드
DWORD WINAPI WriteThread_UDP(LPVOID);
DWORD WINAPI ReadThread_UDP(LPVOID);
DWORD WINAPI ClientMainUDP(LPVOID);
DWORD WINAPI WriteThread_UDPv6(LPVOID);
DWORD WINAPI ReadThread_UDPv6(LPVOID);

// Window) WINAPI 프로시저 (CALLBACK 함수)
BOOL CALLBACK DlgProc(HWND, UINT, WPARAM, LPARAM);      // 대화상자 프로시저
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);   // 자식 윈도우 프로시저

// 3) EditControll 출력 함수
void DisplayText_Recv(char* fmt, ...);  // 수신 EditControll 출력 메세지
void DisplayText_Send(char* fmt, ...);  // 송신 EditControll 출력 메세지
void DisplayText_KAKAOTALKONE(char* fmt, ...); // 읽음 알림 EditControll 출력 메세지
void DisplayText_FILESTATUS(char* fmt, ...); // 파일 status
char* DatetoString(char* fmt, ...);  // 날짜 form 반환 함수
char* getCurrentTime();              // 날짜 반환 함수
// 4) 오류 출력 함수
void err_quit(char* msg);
void err_display(char* msg);

// 5) 파일 전송 관련 함수
char* getFileName(char* fullPath);
int SendFile(char* fileName);
```

메인) WinMain : 윈속 초기화 및 주요 변수 초기화, 다이얼로그박스 생성

1. 클라이언트 식별 번호를 rand()와 추가 공식을 이용해서 자동으로 하나 발행한다.

```
srand(time(NULL));
clientUniqueID = (rand() * 19 - 19) % 256;
itoa(clientUniqueID, strUniqueID, 10);
```

2. 송/수신 시에 사용자가 보낼 메세지 EditControll 을 스레드가 자원 동기화 하기위한 읽기/쓰기 이벤트를 만든다.

```
// 2) 스레드 Read/Write 이벤트 핸들 생성
g_hReadEvent = CreateEvent(NULL, FALSE, TRUE, NULL);
if (g_hReadEvent == NULL) return 1;
g_hWriteEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
if (g_hWriteEvent == NULL) return 1;
```

3. 메세지에서 사용할 클라이언트의 전송 구조체의 전역변수의 기본 필드를 초기화한다.

```
// 3) 전송 메세지 전역변수 최초 초기화
g_chatmsg.type = CHATTING;  // 채팅 메세지 타입 = CHATTING (2000)
g_drawmsg.type = DRAWLINE;  // 그림 메세지 타입 = DRAWLINE (2001)
g_drawmsg.color = RGB(0, 0, 0);
g_drawmsg.width = 5;

// 4) 서버 접속 성공시 최초 전송 메세지(사용자 식별정보)
g_initmsg.type = ACCESS;    // 접속 메세지 타입 = ACCESS (3000)
strncpy(g_initmsg.buf, "CLIENT_ACCESS", MSGSIZE);
kakaoTalk1.type = READCHECK;
strncpy(kakaoTalk1.buf, "가 채팅에 들어왔습니다!(메세지를 읽음)", MSGSIZE);

g_chatmsg.whoSent = g_drawmsg.whoSent = g_initmsg.whoSent = kakaoTalk1.whoSent = clientUniqueID;
```

4. 대화상자 프로시져를 시작한다.

```
// 5) 대화상자 인스턴스 생성
g_hInst = hInstance;
DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, DlgProc);
```

0) SendByProtocol(char* msg): 프로토콜 별 일괄 전송 (일반 함수)

1. 서버와 연결된 후에 사용되는 함수
2. 보낼 메세지 포인터를 파라미터로 받아서 현재 연결되어있는 상태에 따라서 스레드와 상관없이 서버에게 메세지를 일괄적으로 보낸다.

```
int SendByProtocol(char* msg) {
  int retval;
  if (g_isUDP == false) {
    retval = send(g_sock, (char*)msg, BUFSIZE, 0);
  }
  else {
    if (g_isIPv6 == false)
      retval = sendto(send_sock_UDPv4, (char*)msg, BUFSIZE, 0
        , (SOCKADDR*)&remoteaddr_v4, sizeof(remoteaddr_v4));
    else
      retval = sendto(send_sock_UDPv6, (char*)msg, BUFSIZE, 0
        , (SOCKADDR*)&remoteaddr_v6, sizeof(remoteaddr_v6));
  }

  if (retval == 0 || retval == SOCKET_ERROR)
    return FALSE;

  return TRUE;
}
```

Window-1) DlgProc : 다이얼로그 박스 프로시져 (메세지 핸들)

- 지역변수를 선언하고 WM_INITDIALOG에서 컨트롤 핸들을 가져오고 초기화한다.
- 윈도우 클래스를 선언하고 그림판을 사용하기 위한 자식프로시져를 생성한다.

```
WNDCLASS wndclass;
    wndclass.style = CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra = 0;
    wndclass.cbWndExtra = 0;
    wndclass.hInstance = g_hInst;
    wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName = NULL;
    wndclass.lpszClassName = "DrawBoardClass";
    if (!RegisterClass(&wndclass)) return 1;
    g_boardValid = FALSE;

    // C-1) 자식 윈도우 생성
    if (g_boardValid == FALSE) {
      g_hDrawWnd = CreateWindow("DrawBoardClass", "DrawBoardWindow", WS_CHILD,
        450, 60, 425, 508, hDlg, (HMENU)NULL, g_hInst, NULL);
      if (g_hDrawWnd == NULL) return 1;
      ShowWindow(g_hDrawWnd, SW_SHOW);
      UpdateWindow(g_hDrawWnd);

      g_boardValid = TRUE;
      EnableWindow(hNewBoard, FALSE);
    }
```

- 윈도우 컨트롤 핸들 처리 (기능 식별)

1. 굵기 조절 스크롤

```
case WM_HSCROLL: // 1. 굵기 조절 스크롤
    g_drawlinemsg.width = SendDlgItemMessage(hDlg, IDC_THICK, TBM_GETPOS, 0, 0);
    return 0;
```

2. WM_COMMAND:
   switch(LOWORD(wParam)){

   a. 파일 선택 창에서 파일 선택
   → Open File Dialog 를 통해서 파일을 선택(모든 파일 확장자)하고 확인을 누르면,
   → 선택한 파일명이 파일명 EditControll에 출력된다.

```
case IDC_SELECTFILE:
      memset(&OFN, 0, sizeof(OPENFILENAME));
      OFN.lStructSize = sizeof(OPENFILENAME);
      OFN.hwndOwner = hDlg;
      OFN.lpstrFilter = TEXT("All Files(*.*)\0*.*\0");

      OFN.lpstrFile = (LPSTR)szFileName;
      OFN.nMaxFile = nFileNameMaxLen;
      if (0 != GetOpenFileName(&OFN))
      {
        SetWindowText(hFileName, OFN.lpstrFile);
      }
      return TRUE;
```

   b. 파일 전송
   → 파일 선택 후에 출력된 파일명을 가지고 SendFile 함수에 파라미터로 넘겨준뒤 파일을 전송한다.
   → 넘겨받은 파라미터 파일명으로 파일포인터를 fopen한 뒤에 서버에게 파일 데이터를 보낼 것이라는
   1- FILEINIT 타입 메세지(전송 시작 알림)를 먼저보내고,
   2- 파일을 바이너리 형태로 읽은뒤 일괄 전송함수(SendByProtocol)를 통해 보낸다. (type =FILEBYTE)
   3- 파일의 끝까지 내용을 모두 보냈으면 FILEEND타입의 메세지를 보냄으로써 파일전송을 마친다.

```
case IDC_SENDFILE:
      char fileName[nFileNameMaxLen];
```

```
      ZeroMemory(fileName, nFileNameMaxLen);
      if (GetDlgItemText(hDlg, IDC_FILEMSG, (LPSTR)fileName, nFileNameMaxLen) != NULL)
        return SendFile(fileName);

      return TRUE;
```

```
int SendFile(char *fileName) {
  // 1- 파일 보낼 것 이라고 먼저 알리기
  fileinit_msg.type = FILEINIT;
  char fullPath[MSGSIZE];
  strncpy(fullPath, fileName, MSGSIZE);
  strncpy(fileinit_msg.buf, getFileName(fileName), MSGSIZE);

  SendByProtocol((char*)&fileinit_msg);

  // 2- 파일 읽고 보내기
  FILE_MSG sendfile = { FILEBYTE, }; // 파일 버퍼
  strncpy(sendfile.client_id, fileinit_msg.client_id, ID_SIZE);

  DisplayText_Send("%s파일 전송 시작\r\n", fileinit_msg.buf);
  DisplayText_Recv("\r\n");
  FILE* send_fp = fopen(fullPath, "rb");
  if (send_fp == NULL) {
    DisplayText_Send("%s 파일 전송 실패 입니다\r\n", fileName);
    DisplayText_Recv("\r\n");
    return FALSE;
  }
  while (!feof(send_fp)) {
    fread(sendfile.buf, FILE_SIZE, 1, send_fp);
    SendByProtocol((char*)&sendfile);
  }

  // 3- 파일 내용 모두 전송완료
  fileinit_msg.type = FILEEND;
  SendByProtocol((char*)&fileinit_msg);
  DisplayText_Send("%s 파일 전송 완료\r\n", fileinit_msg.buf);
  DisplayText_Recv("\r\n");

  fclose(send_fp);
  return TRUE;
}
```

c. 카카오톡 1 기능 유사 구현
→ 메세지 입력창에 커서가 올라가면 (FOCUS가 주어지면) 소켓으로 서버에 읽음 메세지를 보낸다.

```
case IDC_MSG: // 2-1) 사용자 메세지 EditControll
      if (HIWORD(wParam) == EN_SETFOCUS) { // EditControll 포커스 얻을 시에(칠 준비하면, 톡방 들어갈시?)
        strncpy(kakaoTalk1.whenSent, getCurrentTime(), 23);
        if (g_isUDP == false) {
          send(g_sock, (char*)&kakaoTalk1, BUFSIZE, 0);
        }
        else {
          if (g_isIPv6 == false) {
            sendto(send_sock_UDPv4, (char*)&kakaoTalk1, BUFSIZE, 0
              , (SOCKADDR*)&remoteaddr_v4, sizeof(remoteaddr_v4));
          }
          else {
            sendto(send_sock_UDPv6, (char*)&kakaoTalk1, BUFSIZE, 0
              , (SOCKADDR*)&remoteaddr_v6, sizeof(remoteaddr_v6));
          }
        }
      }
      return TRUE;
```

d. 그림판 지우기, 생성, 삭제
→ 자식 윈도우 프로시저를 삭제하고 다시 생성하는 방식이다.
→ 삭제와 비우기는 반드시 그림판이 생성되어있을때 가능하며, 그림판 생성은 그림판이 삭제되었을때 활성화 된다.

```
case IDC_BOARDCLEAR: // 2-2) 그림판 비우기
      if (g_boardValid == TRUE) {
        DestroyWindow(g_hDrawWnd);
        g_hDrawWnd = CreateWindow("DrawBoardClass", "DrawBoardWindow", WS_CHILD,
          450, 60, 425, 415, hDlg, (HMENU)NULL, g_hInst, NULL);
        if (g_hDrawWnd == NULL) return 1;
        ShowWindow(g_hDrawWnd, SW_SHOW);
        UpdateWindow(g_hDrawWnd);
```

```
        g_boardValid = TRUE;
        EnableWindow(hNewBoard, FALSE);
      }

      return TRUE;

    case IDC_NEWBOARD: // 2-2) 그림판 새로생성
      if (g_boardValid == FALSE) {
        g_hDrawWnd = CreateWindow("DrawBoardClass", "DrawBoardWindow", WS_CHILD,
          450, 60, 425, 415, hDlg, (HMENU)NULL, g_hInst, NULL);
        if (g_hDrawWnd == NULL) return 1;
        ShowWindow(g_hDrawWnd, SW_SHOW);
        UpdateWindow(g_hDrawWnd);

        g_boardValid = TRUE;
        EnableWindow(hNewBoard, FALSE);
        EnableWindow(hBoardClear, TRUE);
      }
      return TRUE;

    case IDC_DELBOARD: // 2-2) 그림판 삭제
      if (g_boardValid == TRUE) {
        DestroyWindow(g_hDrawWnd);

        g_boardValid = FALSE;
        EnableWindow(hNewBoard, TRUE);
        EnableWindow(hBoardClear, FALSE);
      }
      return TRUE;
```

e. UDP, IPv6 접속 여부 결정 (여부에 따라 IP, Port 에디트 컨트롤 미리 정해진 값으로 변경시킴)
   → case IDC_ISIPV6 | case_UDPCHECK

f. 소켓 서버 접속 시작 (서버에 접속하기 버튼)
   → 사용자가 설정한 TCP/UDP, IPv4/6 중 선택하여 접속을 할 수있다.
   → 선택한 프로토콜 종류에 따라 실행되는 스레드는 반드시 한개이다 (TCP, UDPv4, UDPv6)
   → 접속 후에는 접속과 관련된 모든 버튼이나 EditControll 을 비활성화 상태로 만든다. (접속버튼 등)

```
case IDC_CONNECT:
      if (GetDlgItemText(hDlg, IDC_USERID, (LPSTR)g_chatmsg.client_id, ID_SIZE) != NULL) {
        // 메세지에 사용자 이름 추가
        GetDlgItemText(hDlg, IDC_USERID, (LPSTR)g_initmsg.client_id, ID_SIZE);
        GetDlgItemText(hDlg, IDC_USERID, (LPSTR)kakaoTalk1.client_id, ID_SIZE);
        GetDlgItemText(hDlg, IDC_USERID, (LPSTR)g_chatmsg.client_id, ID_SIZE);
        // 접속 방식 확인
        g_isIPv6 = SendMessage(hButtonIsIPv6, BM_GETCHECK, 0, 0);
        g_isUDP = SendMessage(hUDPCheck, BM_GETCHECK, 0, 0);
        g_port = GetDlgItemInt(hDlg, IDC_PORT, NULL, FALSE);

        if (g_isUDP == false) { // TCP 연결
          GetDlgItemText(hDlg, IDC_IPADDR, g_ipaddr, sizeof(g_ipaddr));
          // 소켓 TCP 통신 스레드 시작
          g_hClientThread = CreateThread(NULL, 0, ClientMain, NULL, 0, NULL);

        }
        else { // UDP 연결
          // 소켓 UDP통신 스레드 시작
          g_hClientThread = CreateThread(NULL, 0, ClientMainUDP, NULL, 0, NULL);
        }

        if (g_hClientThread == NULL) { // 스레드 시작 확인
          MessageBox(hDlg, "UDP 클라이언트를 시작할 수 없습니다."
            "\r\n프로그램을 종료합니다.", "실패!", MB_ICONERROR);
          EndDialog(hDlg, 0);
        }
        else {
          EnableWindow(hButtonConnect, FALSE);
          while (g_bStart == FALSE); // 서버 접속 성공 기다림
          EnableWindow(hButtonIsIPv6, FALSE);
          EnableWindow(hEditIPaddr, FALSE);
          EnableWindow(hEditPort, FALSE);
          EnableWindow(g_hButtonSendMsg, TRUE);
          EnableWindow(hUDPCheck, FALSE);
          SetFocus(hEditMsg);
        }
        return TRUE;
      }
      return TRUE;
```

g. 소켓 접속 후 메세지 작성 후 전송
→ 스레드 간 쓰기/읽기 를 완료 시에 전송이 가능하도록 함
→ 전송시에는 전송 시간을 가지고 그 내용을 문자열로 포함하여서 전송
→ 쓰기가 완료될때까지 다시 기다림(스레드 이벤트를 이용하여)

```
case IDC_SENDMSG:
      // 읽기 완료를 기다림
      WaitForSingleObject(g_hReadEvent, INFINITE);
      GetDlgItemText(hDlg, IDC_MSG, g_chatmsg.buf, MSGSIZE);
      strncpy(g_chatmsg.whenSent, getCurrentTime(), 23);
      // 쓰기 완료를 알림
      SetEvent(g_hWriteEvent);
      // 입력된 텍스트 전체를 선택 표시
      SendMessage(hEditMsg, EM_SETSEL, 0, -1);
      return TRUE;
```

h. 그외 그림판 기능들 (도형 , 색 변경)

case IDC_COLORRED, case IDC_COLORGREEN, case IDC_COLORBLUE,
case IDC_COLORBLACK, case IDC_COLORPINK
case IDC_LINE, case IDC_STRA, case IDC_TRIA, case IDC_RECT, case IDC_CIRC, case IDC_ERAS:

1, 2) TCP / UDP 소켓 통신 스레드

<소켓에서 수신한 메세지의 타입에 따른 클라이언트 동작> (3.기능상세-**b 송/수신 메세지 정의** 참고)

종류	type	동작
클라이언트간 채팅 메세지	CHATTING	클라이언트간 송/수신 채팅 메세지를 나타내고 화면에 출력한다.
클라이언트간 그림 지우기	DRAWERAS	클라이언트가 그림판에서 지우개 기능을 사용하여 그림판의 그림이 지워진다.
클라이언트 읽음 알림	READCHECK	클라이언트가 채팅을 할 준비가 됨 (읽음). 사용자가 읽었음을 메세지로 출력한다.
서버로부터 추방 메세지	KICKOUT	서버에서 클라이언트 본인을 추방함. 소켓 스레드 통신이 종료된다
다른 클라이언트에서 파일 전송 시작을 알림	FILEINIT	메세지에 포함되어 있는 파일명을 이용해 정해진 디렉터리에 Write 파일 포인터를 생성한다. 이때 파일을 송신한 클라이언트의 ID와 파일명이 메세지로 출력된다.
다른 클라이언트에서 파일 데이터 조각을 전송	FILEBYTE	FILEINIT 타입 메세지를 받고 만들어놓은 파일포인터에다가 파일 수신받은 파일 데이터 조각을 Write한다.
다른 클라이언트에서 파일 전송 종료를 알림	FILEEND	상대 클라이언트의 파일전송이 끝났다는 의미이므로 Write파일 포인터를 fclose()하고, 파일 저장이 완료되었다는 메세지를 출력한다.
클라이언트간 그림 그리기	위 type을 제외한 모두	클라이언트가 그림판에서 그림 기능을 사용. 자식 프로시저인 그림판 위도우에 메세지가 전달된다. 전달된 값에 따라 그림판에 그림이 그려진다.

1-1) DWORD WINAPI ClientMain(LPVOID) → TCP 소켓 통신 스레드

1. 스레드 내에서 접속을 원하는 서버의 주소 구조체(SOCKADDR_IN | IN6)를 초기화하고, 클라이언트 소켓을 해당 주소에 connect요청을 하여 연결을 시도한다.
2. 접속(연결) 성공 후에 서버에게 최초로 메세지를 하나 보낸다(g_initmsg)
3. TCP소켓 통신을 위한 읽기/쓰기 스레드를 실행하고 두 스레드가 끝날때 까지 기다린다.
4. ReadThread 에서 서버로 부터 메세지 타입이 KICKOUT(3001)로 된 메세지가 오고, 그 메세지의 타겟이 본인의 아이디와 식별번호와 같다면 통신을 종료한다.

```
// 주요 코드(connect() 생략)
etval = send(g_sock, (char*)&g_initmsg, BUFSIZE, 0);
  MessageBox(NULL, "서버에 TCP로 접속했습니다.", "성공!", MB_ICONINFORMATION);
```

```
  // 읽기 & 쓰기 스레드 생성
  HANDLE hThread[2];
  hThread[0] = CreateThread(NULL, 0, ReadThread, NULL, 0, NULL);
  hThread[1] = CreateThread(NULL, 0, WriteThread, NULL, 0, NULL);
  if (hThread[0] == NULL || hThread[1] == NULL) {
    MessageBox(NULL, "스레드를 시작할 수 없습니다."
      "\r\n프로그램을 종료합니다.",
      "실패!", MB_ICONERROR);
    exit(1);
  }

  g_bStart = TRUE;
  // 스레드 종료 대기
  retval = WaitForMultipleObjects(2, hThread, FALSE, INFINITE);
```

1-2) DWORD WINAPI ReadThread(LPVOID) → TCP 수신 스레드

1. 스레드 시작후 서버로부터 사용자 정의 수신 함수를 통해서 메세지 수신을 받는다

```
int recvn(SOCKET s, char* buf, int len, int flags)
{
  int received;
  char* ptr = buf;
  int left = len;

  while (left > 0) {
    received = recv(s, ptr, left, flags);
    if (received == SOCKET_ERROR)
      return SOCKET_ERROR;
    else if (received == 0)
      break;
    // left = 256
    left -= received;
    ptr += received;
  }

  return (len - left);
}
```

2. <소켓에서 수신한 메세지의 타입에 따라 동작>에서 명시해둔 메세지 타입에 따른 동작을 수행한다.

```
if (retval == 0 || retval == SOCKET_ERROR || comm_msg.type == KICKOUT ) {
      chat_msg == (CHAT_MSG*)&comm_msg;
      if (!strcmp(chat_msg->client_id, g_chatmsg.client_id) && chat_msg->whoSent == g_chatmsg.whoSent)
        break;
      else
        continue;
    }
    if (comm_msg.type == CHATTING) {
      chat_msg = (CHAT_MSG*)&comm_msg;
      DisplayText_Recv("\r\n");DisplayText_Send("\r\n");
      if (!strcmp(chat_msg->client_id, g_chatmsg.client_id) && chat_msg->whoSent == g_chatmsg.whoSent) {
        DisplayText_Send("[%s] %s\r\n", chat_msg->client_id, chat_msg->buf);
        DisplayText_Recv("%s\r\n", chat_msg->whenSent);
      }
      else
      {
        DisplayText_Recv("[%s(%d)] %s\r\n", chat_msg->client_id, chat_msg->whoSent, chat_msg->buf);
        DisplayText_Send("%s\r\n", chat_msg->whenSent);
      }
    }
    else if (comm_msg.type == READCHECK) {
      chat_msg = (CHAT_MSG*)&comm_msg;
      DisplayText_KAKAOTALKONE("[%s]-[%s(%d)]%s\r\n", chat_msg->whenSent, chat_msg->client_id, chat_msg->whoSent, chat_msg->bu
    }
    else if (comm_msg.type == DRAWERAS) {
      draw_msg = (DRAWLINE_MSG*)&comm_msg;
      g_drawcolor = RGB(255, 255, 255);
      g_drawlinemsg.type = draw_msg->type;
      g_drawlinemsg.x0 = draw_msg->x0;
      g_drawlinemsg.y0 = draw_msg->y0;
      g_drawlinemsg.x1 = draw_msg->x1;
      g_drawlinemsg.y1 = draw_msg->y1;

      g_drawlinemsg.width = draw_msg->width;
      SendMessage(g_hDrawWnd, WM_DRAWIT,
        MAKEWPARAM(draw_msg->x0, draw_msg->y0),
        MAKELPARAM(draw_msg->x1, draw_msg->y1));
    }
```

```
    else if (comm_msg.type == FILEINIT) {
      fileinit_recv = (CHAT_MSG*)&comm_msg;
      ZeroMemory(fileN, 280);
      if (strcmp(fileinit_recv->client_id, fileinit_msg.client_id) == 0)continue;

      DisplayText_FILESTATUS("\"%s\" 파일 수신 from %s(%d)\r\n", fileinit_recv->buf, fileinit_recv->client_id, fileinit_recv->wh
      strncpy(fileN, "프로젝트 받은 파일\\", 280);
      strncat(fileN, fileinit_recv->buf, strlen(fileinit_recv->buf));
      recvFile = fopen(fileN, "wb");
    }
    else if (comm_msg.type == FILEEND) {
      fileinit_recv = (CHAT_MSG*)&comm_msg;
      if (strcmp(fileinit_recv->client_id, fileinit_msg.client_id) == 0)continue;

      DisplayText_FILESTATUS("%s가 보낸 \"%s 파일 저장\"\r\n", fileinit_recv->client_id, fileN);
      fclose(recvFile);
    }
    else if (comm_msg.type == FILEBYTE && recvFile != NULL) {
      fileRecv = (FILE_MSG*)&comm_msg;
      if (strcmp(fileRecv->client_id, fileinit_msg.client_id) == 0)continue;
      fwrite(fileRecv->buf, MSGSIZE, 1, recvFile);
    }
    else {
      draw_msg = (DRAWLINE_MSG*)&comm_msg;
      g_drawcolor = draw_msg->color;
      g_drawlinemsg.type = draw_msg->type;
      g_drawlinemsg.x0 = draw_msg->x0;
      g_drawlinemsg.y0 = draw_msg->y0;
      g_drawlinemsg.x1 = draw_msg->x1;
      g_drawlinemsg.y1 = draw_msg->y1;

      g_drawlinemsg.width = draw_msg->width;
      SendMessage(g_hDrawWnd, WM_DRAWIT,
        MAKEWPARAM(draw_msg->x0, draw_msg->y0),
        MAKELPARAM(draw_msg->x1, draw_msg->y1));
    }
   }
   return 0;
```

1-3) DWORD WINAPI WriteThread(LPVOID) → TCP 송신 스레드

1. 채팅 메세지 EditControll 이나 그림판에서 발생하는 모든 데이터를 정해진 구조체에 알맞게 값을 할당하여 서버에게 보낸다. (파일전송은 SendFile 함수에서만)
2. 클라이언트는 보낸 메세지를 송신 채팅 EditControll에 바로 출력하는 것이 아니고 서버에 먼저 보내지고 본인이 보낸 메세지가 맞으면 그 다음에 송신 채팅 EditControll에 출력이 되는 것이다. 다른 클라이언트가 보낸 메세지는 수신 EditControll에 출력된다.

```
DWORD WINAPI WriteThread(LPVOID arg)
{
  int retval;

  // 서버와 데이터 통신
  while (1) {
    // 쓰기 완료 기다리기
    WaitForSingleObject(g_hWriteEvent, INFINITE);

    // 문자열 길이가 0이면 보내지 않음
    if (strlen(g_chatmsg.buf) == 0) {
      // '메시지 전송' 버튼 활성화
      EnableWindow(g_hButtonSendMsg, TRUE);
      // 읽기 완료 알리기
      SetEvent(g_hReadEvent);
      continue;
    }

    // 데이터 보내기
    retval = send(g_sock, (char*)&g_chatmsg, BUFSIZE, 0);
    if (retval == SOCKET_ERROR) {
      break;
    }

    // '메시지 전송' 버튼 활성화
    EnableWindow(g_hButtonSendMsg, TRUE);
    // 읽기 완료 알리기
    SetEvent(g_hReadEvent);
  }

  return 0;
}
```

2-1) DWORD WINAPI ClientMainUDP(LPVOID) → UDP 소켓 통신 스레드

1. 스레드 내에서 서버와 통신할 소켓 listen, send UDP 소켓을 생성하여 할당한다.
2. 송/수신 스레드를 실행하고 두 스레드가 끌날때 까지 기다린다.
3. ReadThread 에서 서버로 부터 메세지 타입이 KICKOUT(3001)로 된 메세지가 오고, 그 메세지의 타겟이 본인의 아이디와 식별번호와 같다면 통신을 종료한다.

```
// 주요 코드
HANDLE hThread[2];

  if (g_isIPv6 == false) {
    // socket()
    listen_sock_UDPv4 = socket(AF_INET, SOCK_DGRAM, 0);
    if (listen_sock_UDPv4 == INVALID_SOCKET) err_quit("socket()");

    send_sock_UDPv4 = socket(AF_INET, SOCK_DGRAM, 0);
    if (send_sock_UDPv4 == INVALID_SOCKET) err_quit("socket()");

    hThread[0] = CreateThread(NULL, 0, ReadThread_UDP, NULL, 0, NULL);
    hThread[1] = CreateThread(NULL, 0, WriteThread_UDP, NULL, 0, NULL);
    MessageBox(NULL, "서버에 UDPv4로 접속했습니다.", "성공!", MB_ICONINFORMATION);
  }
```

2-2) DWORD WINAPI ReadThread_UDP(LPVOID) → UDPv4 수신 스레드

1. 스레드 내에서 접속을 원하는 서버의 IP주소 SOCKADDR_IN 구조체에 bind 시키고 멀티 캐스트 그룹에 가입한다.
2. TCP와 다르게 UDP 에서는 recvfrom()함수를 이용하여 메세지들을 수신한다.
3. 수신 후에 동작은 TCP에서와 동일하다.
4. 통신이 종료되면 (recvfrom()의 리턴 값이 -1 or SOCKETERROR or 추방메세지) 가입한 멀티캐스트 그룹을 탈퇴한다.

2-3) DWORD WINAPI WriteThread_UDP(LPVOID); → UDPv4 송신 스레드

1. 스레드 시작시에 서버에 서버에게 최초로 메세지를 하나 보낸다(g_initmsg)
2. 채팅 메세지 EditControll 이나 그림판에서 발생하는 모든 데이터를 정해진 구조체에 알맞게 값을 할당하여 서버에게 보낸다.
3. 클라이언트는 보낸 메세지를 송신 채팅 EditControll에 바로 출력하는 것이 아니고 서버에 먼저 보내지고 본인이 보낸 메세지가 맞으면 그 다음에 송신 채팅 EditControll에 출력이 되는 것이다. 다른 클라이언트가 보낸 메세지는 수신 EditControll에 출력된다.
4. TCP에서 사용하는 주소나, 소켓만 다를 뿐 동작은 동일하다.

2-4) DWORD WINAPI ReadThread_UDPv6(LPVOID)

1. SOCKADDR_IN6 구조체에 원하는 서버의 IPv6주소를 bind 시키는 것외에는 위 2-2) ReadThread_UDP에서의 동작과 모두 같다.
   (멀티캐스트도 v6용으로 사용)

2-5) DWORD WINAPI WriteThread_UDPv6(LPVOID)

1. 위 2-3) WriteThread_UDP와 동작이 모두 같다.

Window-2) LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM)

(책 참고)

1. 프로시저 생성 - WM_CREATE
   → 프로시저를 생성하고 흰 화면으로 시작한다.

```
case WM_CREATE:
    hDC = GetDC(hWnd);
    // 화면을 저장할 비트맵 생성
```

```
    cx = GetDeviceCaps(hDC, HORZRES);
    cy = GetDeviceCaps(hDC, VERTRES);
    hBitmap = CreateCompatibleBitmap(hDC, cx, cy);

    // 메모리 DC 생성
    hDCMem = CreateCompatibleDC(hDC);

    // 비트맵 선택 후 메모리 DC 화면을 흰색으로 칠함
    SelectObject(hDCMem, hBitmap);
    SelectObject(hDCMem, GetStockObject(WHITE_BRUSH));
    SelectObject(hDCMem, GetStockObject(WHITE_PEN));
    Rectangle(hDCMem, 0, 0, cx, cy);

    ReleaseDC(hWnd, hDC);
    return 0;
```

2. 왼쪽 버튼 클릭 다운 - WM_LBUTTONDOWN
   → 버튼을 클릭시에 시작 좌표를 지정한다.
3. 마우스 움직임 - WM_MOUSEMOBE
   → 마우스가 눌린채로 움직일 때 g_drawmsg의 좌표에 값을 채우고 소켓에 이동한 좌표 정보를 전송한다.
4. 왼쪽 버튼 클릭 업 - WM_LBUTTONUP
   → 단순 선이나 지우개 기능을 사용하지않고 도형기능을 사용할때 시작과 끝 좌표 정보를 전송한다.
5. 사용자 정의 윈도우 메세지 WM_DRAWIT
   → comm_msg.type 이 그림그리는 타입일 경우 WM_DRAWIT으로 자식 프로시저에 메세지를 전달
   → 수신 받는 메세지 타입에 따른 동작하는 방식

종류	타입	동작
선	DRAWLINE	펜으로 자유롭게 그린 선에 대한 정보를 그림판에 그린다.
직선	DRAWSTRA	펜으로 마우스를 클릭한 좌표(시작)부터 마우스를 클릭업 한 좌표(종료)까지에 대한 정보를 이용하여 그림판에 직선을 그린다.
삼각형	DRAWTRIA	펜으로 마우스를 클릭한 좌표(시작)부터 마우스를 클릭업 한 좌표(종료)까지에 대한 정보를 이용하여 세 꼭지점을 정하고 그림판에 삼각형을 그린다.
사각형	DRAWRECT	펜으로 마우스를 클릭한 좌표(시작)부터 마우스를 클릭업 한 좌표(종료)까지에 대한 정보를 이용하여 네 꼭지점을 정하고 그림판에 직선을 그린다.
원	DRAWCIRC	펜으로 마우스를 클릭한 좌표(시작)부터 마우스를 클릭업 한 좌표(종료)까지에 대한 정보를 이용하여 사각형에 내접한 원을 그린다.
지우개	DRAWERAS	펜으로 마우스를 자유롭게 이동한 경로에 하얀색으로 그린다.

```
// 주요코드
case WM_DRAWIT:
    hDC = GetDC(hWnd);
    hPen = CreatePen(PS_SOLID, g_drawmsg.width, g_drawcolor);

    //직선,선
    if (g_drawmsg.type == DRAWLINE || g_drawmsg.type == DRAWSTRA) {

      hOldPen = (HPEN)SelectObject(hDC, hPen);
      MoveToEx(hDC, LOWORD(wParam), HIWORD(wParam), NULL);
      LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
      SelectObject(hDC, hOldPen);

      hOldPen = (HPEN)SelectObject(hDCMem, hPen);
      MoveToEx(hDCMem, LOWORD(wParam), HIWORD(wParam), NULL);
      LineTo(hDCMem, LOWORD(lParam), HIWORD(lParam));
      SelectObject(hDC, hOldPen);

      DeleteObject(hPen);
      ReleaseDC(hWnd, hDC);
    }

    //삼각형
    else if (g_drawmsg.type == DRAWTRIA) {
      hOldPen = (HPEN)SelectObject(hDC, hPen);
      SelectObject(hDC, GetStockObject(NULL_BRUSH));
      MoveToEx(hDC, g_drawmsg.x0, g_drawmsg.y0, NULL);
      LineTo(hDC, (g_drawmsg.x1 - g_drawmsg.x0) / 2 + g_drawmsg.x0, g_drawmsg.y1);
      LineTo(hDC, g_drawmsg.x1, g_drawmsg.y0);
      LineTo(hDC, g_drawmsg.x0, g_drawmsg.y0);
      SelectObject(hDC, hOldPen);
      DeleteObject(hPen);
      ReleaseDC(hWnd, hDC);
```

```
    }

    //사각형
    else if (g_drawmsg.type == DRAWRECT) {
      hOldPen = (HPEN)SelectObject(hDC, hPen);
      SelectObject(hDC, GetStockObject(NULL_BRUSH));
      MoveToEx(hDC, g_drawmsg.x0, g_drawmsg.y0, NULL);
      LineTo(hDC, g_drawmsg.x0, g_drawmsg.y1);
      LineTo(hDC, g_drawmsg.x1, g_drawmsg.y1);
      LineTo(hDC, g_drawmsg.x1, g_drawmsg.y0);
      LineTo(hDC, g_drawmsg.x0, g_drawmsg.y0);
      SelectObject(hDC, hOldPen);
      DeleteObject(hPen);
      ReleaseDC(hWnd, hDC);
    }

    //원
    else if (g_drawmsg.type == DRAWCIRC) {
      hOldPen = (HPEN)SelectObject(hDC, hPen);
      SelectObject(hDC, GetStockObject(NULL_BRUSH));
      Ellipse(hDC, g_drawmsg.x0, g_drawmsg.y0, LOWORD(lParam), HIWORD(lParam));
      SelectObject(hDC, hOldPen);
      DeleteObject(hPen);
      ReleaseDC(hWnd, hDC);
    }
    else if (g_drawmsg.type == DRAWERAS) {
      hOldPen = (HPEN)SelectObject(hDC, hPen);
      MoveToEx(hDC, LOWORD(wParam), HIWORD(wParam), NULL);
      LineTo(hDC, LOWORD(lParam), HIWORD(lParam));
      SelectObject(hDC, hOldPen);
      hOldPen = (HPEN)SelectObject(hDCMem, hPen);
      MoveToEx(hDCMem, LOWORD(wParam), HIWORD(wParam), NULL);
      LineTo(hDCMem, LOWORD(lParam), HIWORD(lParam));
      SelectObject(hDC, hOldPen);
      DeleteObject(hPen);
      ReleaseDC(hWnd, hDC);
    }
    return 0;
```

3) EditControll 출력함수

3-1) void DisplayText_Recv(char* fmt, ...)
3-2) void DisplayText_Send(char* fmt, ...)
3-3) void DisplayText_KAKAOTALKONE(char* fmt, ...)

→ 클라이언트 송/수신 메시지, 읽음 알림 메세지를 출력해주는 함수이다.

→ g_hEditRecv : 수신 메세지 EditControll, g_hEditSend : 송신 메세지 EditControll,

→ g_EditUserRead : 읽음알림 메세지

```
void DisplayText_Recv(char* fmt, ...)
{
  va_list arg;
  va_start(arg, fmt);

  char cbuf[1024];
  vsprintf(cbuf, fmt, arg);

  int nLength = GetWindowTextLength(g_hEditRecv);
  SendMessage(g_hEditRecv, EM_SETSEL, nLength, nLength);
  SendMessage(g_hEditRecv, EM_REPLACESEL, FALSE, (LPARAM)cbuf);

  va_end(arg);
}

void DisplayText_Send(char* fmt, ...)
{
  va_list arg;
  va_start(arg, fmt);

  char cbuf[1024];
  vsprintf(cbuf, fmt, arg);

  int nLength = GetWindowTextLength(g_hEditSend);
  SendMessage(g_hEditSend, EM_SETSEL, nLength, nLength);
  SendMessage(g_hEditSend, EM_REPLACESEL, FALSE, (LPARAM)cbuf);

  va_end(arg);
```

```
}

void DisplayText_KAKAOTALKONE(char* fmt, ...)
{
  va_list arg;
  va_start(arg, fmt);

  char cbuf[1024];
  vsprintf(cbuf, fmt, arg);

  int nLength = GetWindowTextLength(g_EditUserRead);
  SendMessage(g_EditUserRead, EM_SETSEL, nLength, nLength);
  SendMessage(g_EditUserRead, EM_REPLACESEL, FALSE, (LPARAM)cbuf);

  va_end(arg);
}
```

3-4) char* DatetoString(char* fmt, ...)
3-5) char* getCurrentTime()

→ 현재시간을 문자열로 리턴해준다.

```
char* DatetoString(char* fmt, ...) {
  va_list arg;
  va_start(arg, fmt);

  char cbuf[23];
  vsprintf(cbuf, fmt, arg);

  return (char*)cbuf;
}

char* getCurrentTime() {
  time_t curTime = time(NULL);
  struct tm* pLocal = NULL;
#if defined(_WIN32) || defined(_WIN64)
  pLocal = localtime(&curTime);
#else
  localtime_r(&curTime, pLocal);
#endif
  if (pLocal == NULL) return NULL;

  return DatetoString("%04d-%02d-%02d T%02d:%02d:%02d",
    pLocal->tm_year + 1900, pLocal->tm_mon + 1, pLocal->tm_mday,
    pLocal->tm_hour, pLocal->tm_min, pLocal->tm_sec);
}
```

4) 오류 출력 함수

```
// 소켓 함수 오류 출력 후 종료
void err_quit(char* msg)
{
  LPVOID lpMsgBuf;
  FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
    NULL, WSAGetLastError(),
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    (LPTSTR)&lpMsgBuf, 0, NULL);
  MessageBox(NULL, (LPCTSTR)lpMsgBuf, msg, MB_ICONERROR);
  LocalFree(lpMsgBuf);
  exit(1);
}

// 소켓 함수 오류 출력
void err_display(char* msg)
{
  LPVOID lpMsgBuf;
  FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
    NULL, WSAGetLastError(),
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    (LPTSTR)&lpMsgBuf, 0, NULL);
  printf("[%s] %s", msg, (char*)lpMsgBuf);
  LocalFree(lpMsgBuf);
}
```

5) 파일 전송 관련 함수

char* getFileName(char* fullPath)
int SendFile(char* fileName)

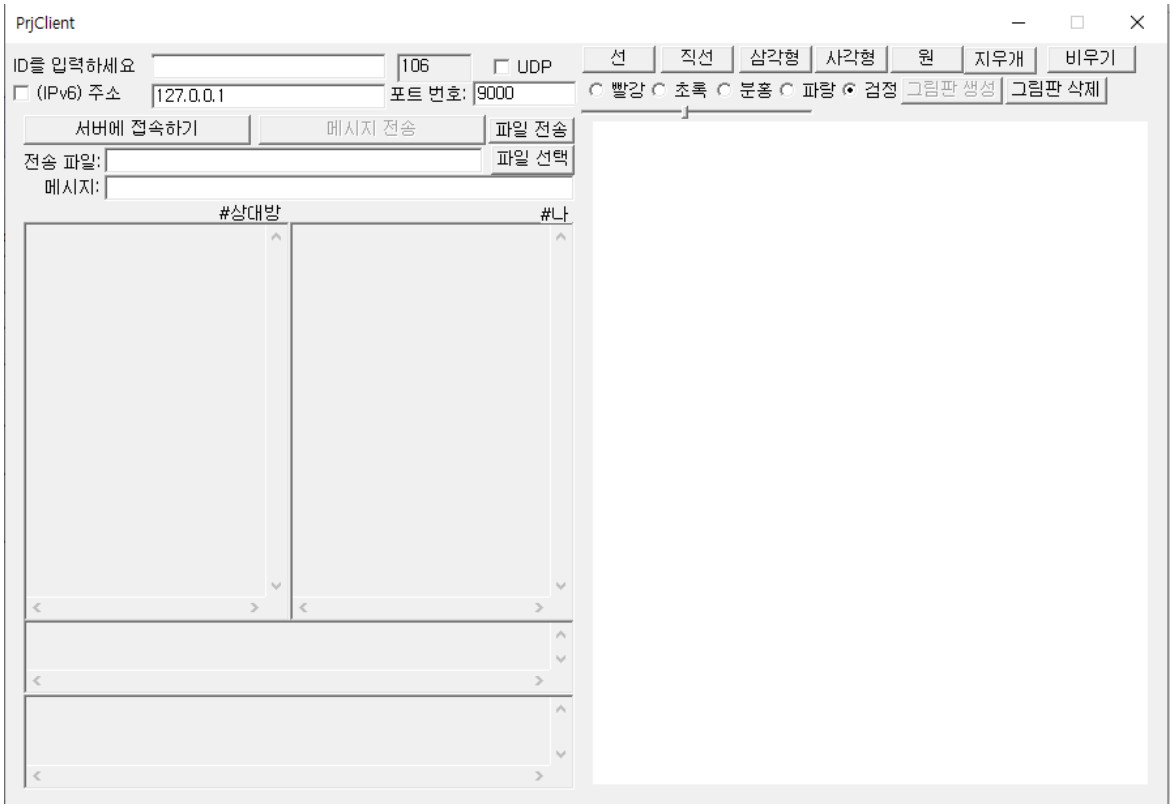
→ 파일 선택시에 절대 경로가 나오기 때문에 \\를 기준으로 마지막에 나오는 문장이 순수 파일명이다.

→ 이 파일명을 이용해서 주 프로시저에서 WM_COMMAND의 파일 전송기능(IDC_SENDFILE)에서 함수를 호출한다.

```
char* getFileName(char* fullPath) {
  char* ptr = strtok(fullPath, "\\");
  char* onlyFileName;
  while (ptr) {
    onlyFileName = ptr;
    ptr = strtok(NULL, "\\");
  }
  return onlyFileName;
}

int SendFile(char *fileName) {
  // 1- 파일 보낼 것 이라고 먼저 알리기
  fileinit_msg.type = FILEINIT;
  char fullPath[MSGSIZE];
  strncpy(fullPath, fileName, MSGSIZE);
  strncpy(fileinit_msg.buf, getFileName(fileName), MSGSIZE);

  SendByProtocol((char*)&fileinit_msg);

  // 2- 파일 읽고 보내기
  FILE_MSG sendfile = { FILEBYTE, }; // 파일 버퍼
  strncpy(sendfile.client_id, fileinit_msg.client_id, ID_SIZE);

  DisplayText_Send("%s파일 전송 시작\r\n", fileinit_msg.buf);
  DisplayText_Recv("\r\n");
  FILE* send_fp = fopen(fullPath, "rb");
  if (send_fp == NULL) {
    DisplayText_Send("%s 파일 전송 실패 입니다\r\n", fileName);
    DisplayText_Recv("\r\n");
    return FALSE;
  }
  while (!feof(send_fp)) {
    fread(sendfile.buf, FILE_SIZE, 1, send_fp);
    SendByProtocol((char*)&sendfile);
  }

  // 3- 파일 내용 모두 전송완료
  fileinit_msg.type = FILEEND;
  SendByProtocol((char*)&fileinit_msg);
  DisplayText_Send("%s 파일 전송 완료\r\n", fileinit_msg.buf);
  DisplayText_Recv("\r\n");

  fclose(send_fp);
  return TRUE;
}
```

4. 최종 결과 화면

1. **프로그램 시작**
   → 클라이언트 식별 번호 램덤으로 자동 생성
   → 서버에 접속하기 전까지는 아무기능도 사용 못함

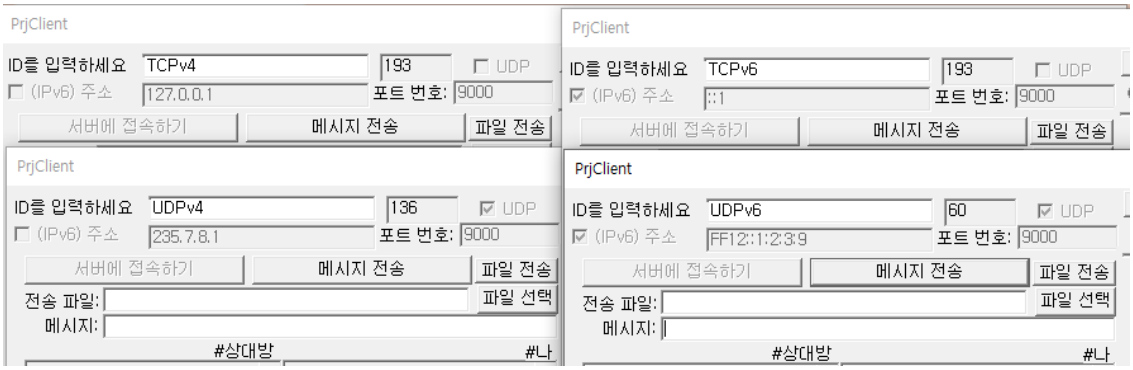


2. **서버에 접속하기**

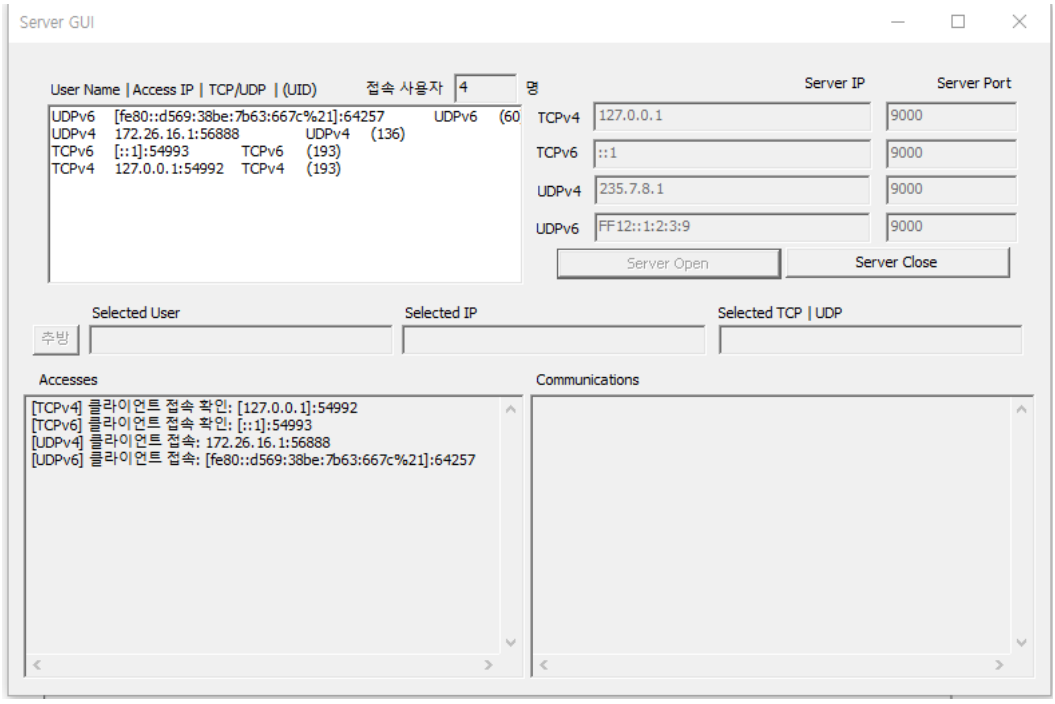
→TCP/UDP, IPv4, IPv6방식 모두 가능

→ 체크박스 UDP, (IPv6)주소 를 클릭하여 조정 가능

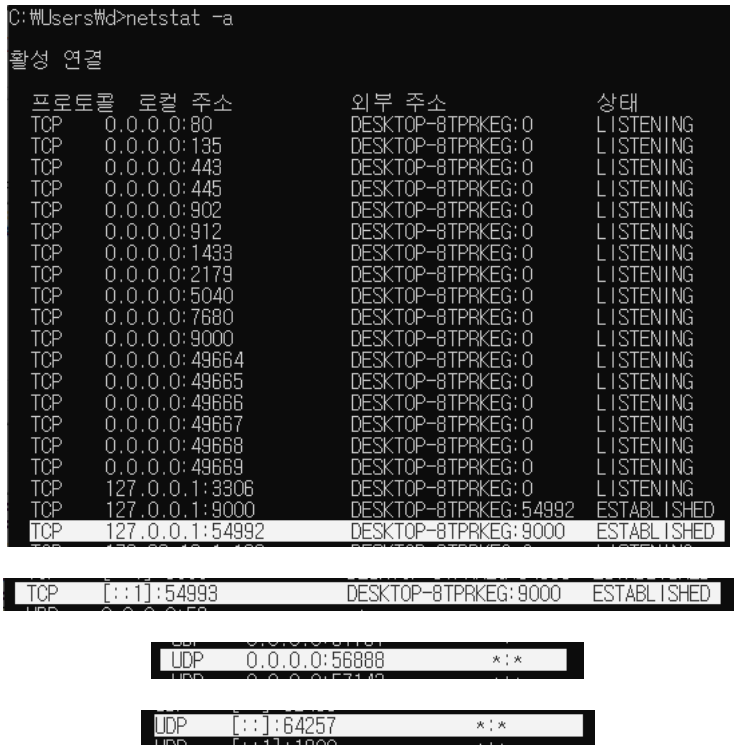
a. 클라이언트에서 접속 후 상태



b. 클라이언트 접속 후 서버 상태



c. cmd에서 netstat -a 명령어를 통해 확인한 소켓 상태 (개인 PC)



3. **클라이언트 간 채팅 기능**

a. TCP/UDP, IPv4, IPv6상관없이 모두와 채팅 가능.

b. 채팅을 본인이 보낸시간이 채팅 메세지 정보에 포함되어서 전송되어 상대방에게 언제 보냈는지를 알려줌.

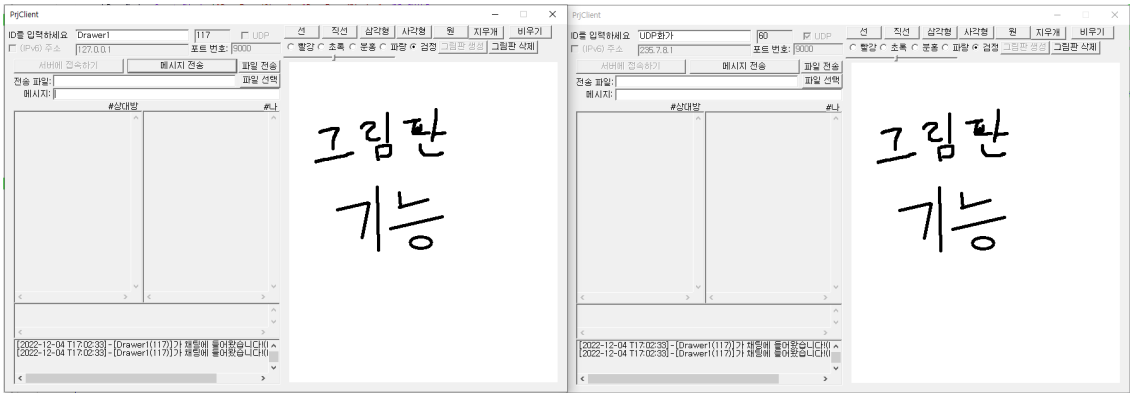
c. 각 메세지를 보낸 클라이언트의 ID와 식별 번호를 동시에 표기해 메세지 송신자를 확인 가능함.

d. 메세지를 보낸 송신자에 따라
본인이 보낸 것이라면 오른쪽 EditControll에 상대가 보냈다면 왼쪽 EditControll에 출력

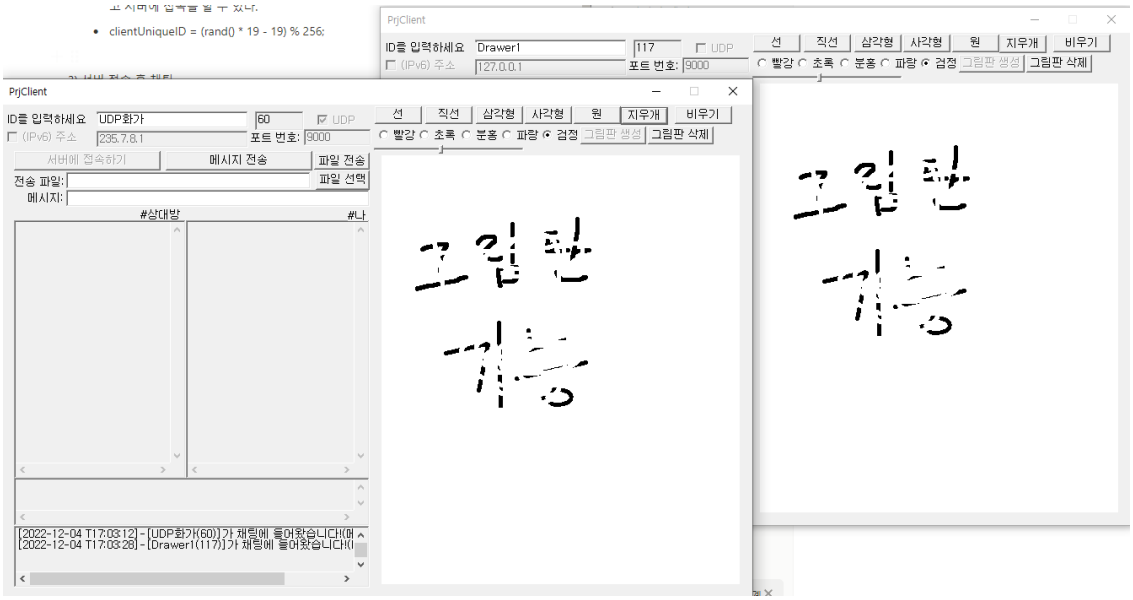
4. **클라이언트 간 그림판 사용**

   → TCP/UDP, IPv4, IPv6상관없이 모두와 그림판 그림 사용 가능

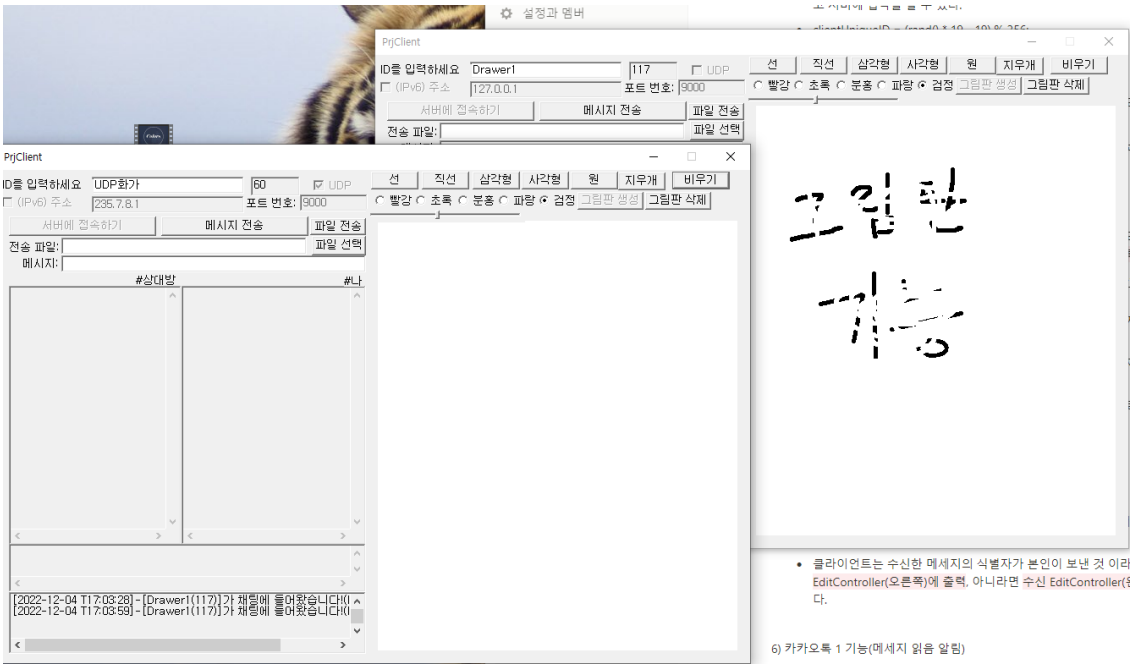
   1. 선 기능(펜으로 그냥 그리기)



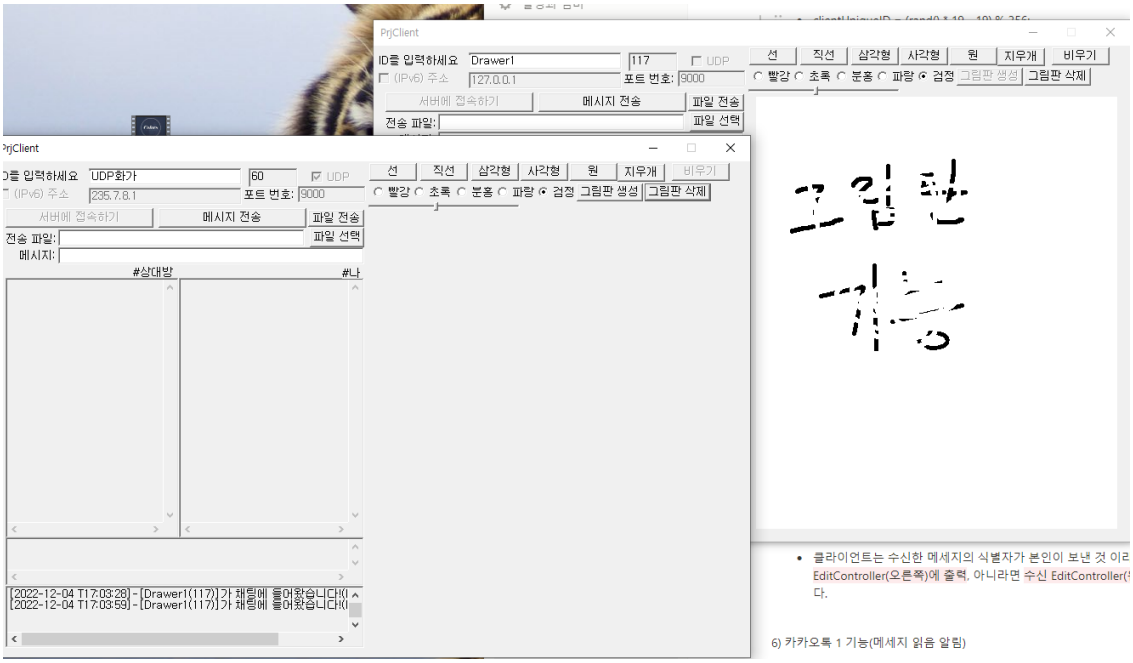
b. 지우개 기능



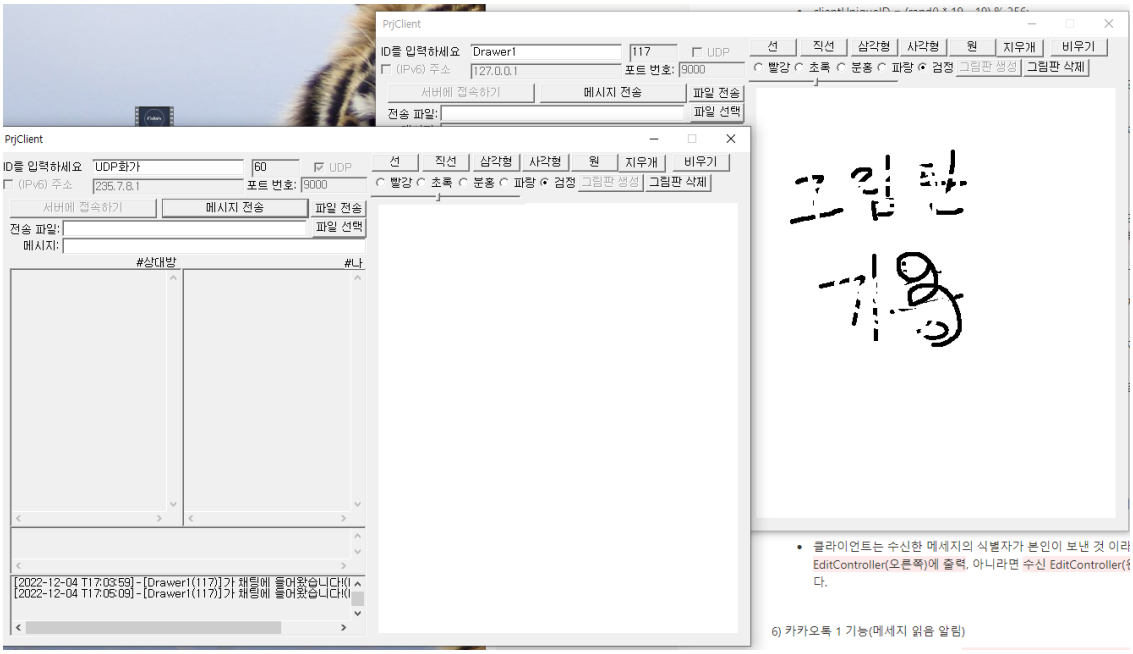
c. 그림판 비우기 기능(클라이언트 본인만)



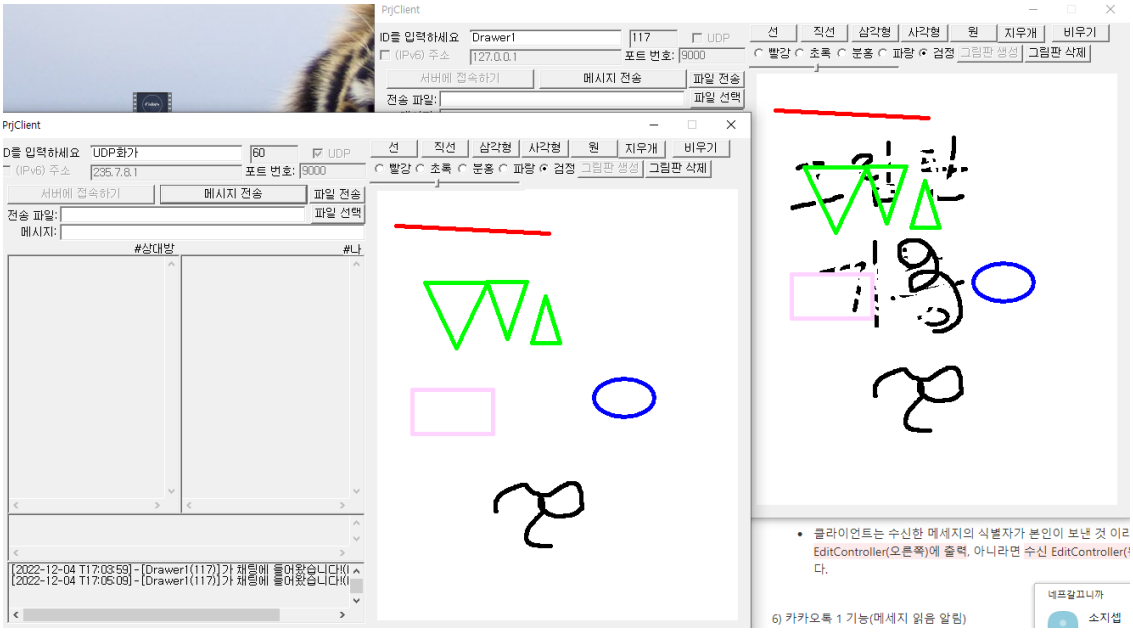
d. 그림판 삭제 기능(클라이언트 본인만)
→ 자식 윈도우 프로시저 삭제(DestroyWindow)
→ 비우기 버튼 비활성화



e. 그림판 생성 기능(클라이언트 본인만)
→ 자식 윈도우 프로시저를 다시 생성
→ 비우기버튼 활성화



f. 그림판 색 변경 및 도형 기능



5. **카카오톡 1 기능 유사 구현**

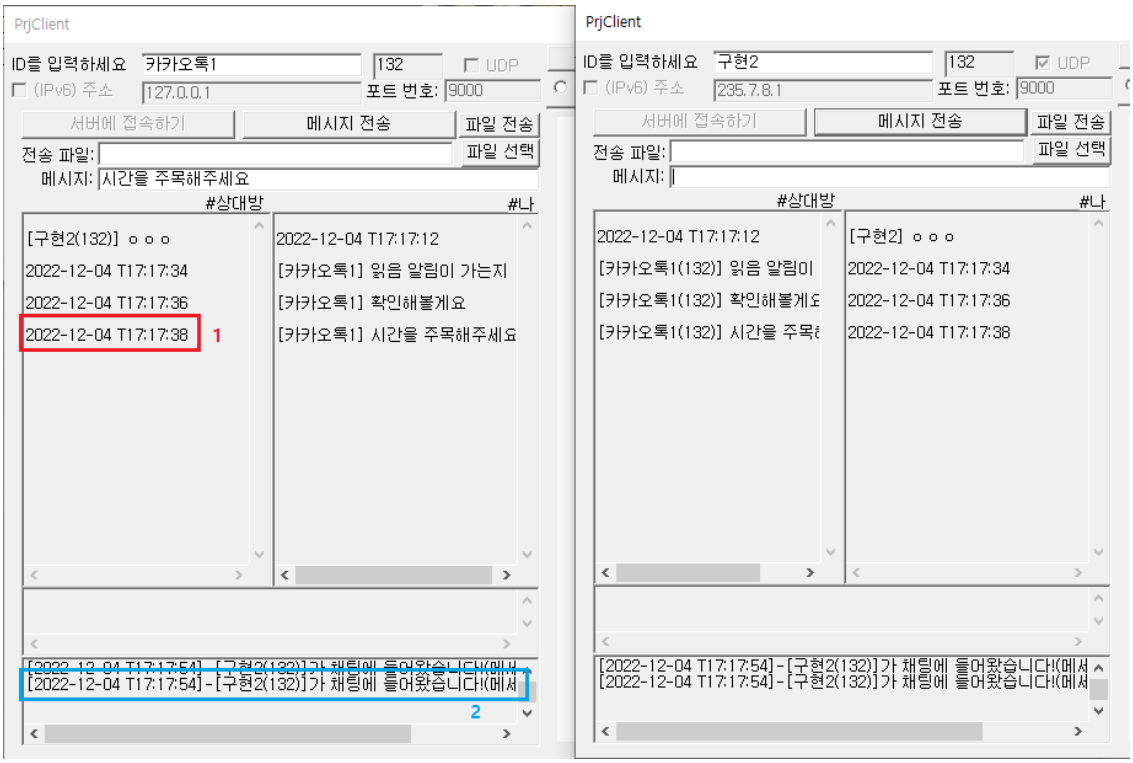
   a. 카카오톡에서 톡방에 들어가면 1기능이 사라진다는 점에서 채팅 프로그램을 활성화 시키면 알림메세지가 가면 되겠다는 생각으로 구현하였습니다.

   b. 클라이언트가 메세지 입력 EditControll에 커서를 올리면 (Focus를 가지게되면) 서버에 읽음 알림 메세지 (type=READCHECK)를 보냄

   c. 보내고 난 후에는 서버에서는 모든 클라이언트에게 그대로 전달

   d. 누가, 언제 읽었는지를 알 수 있다!

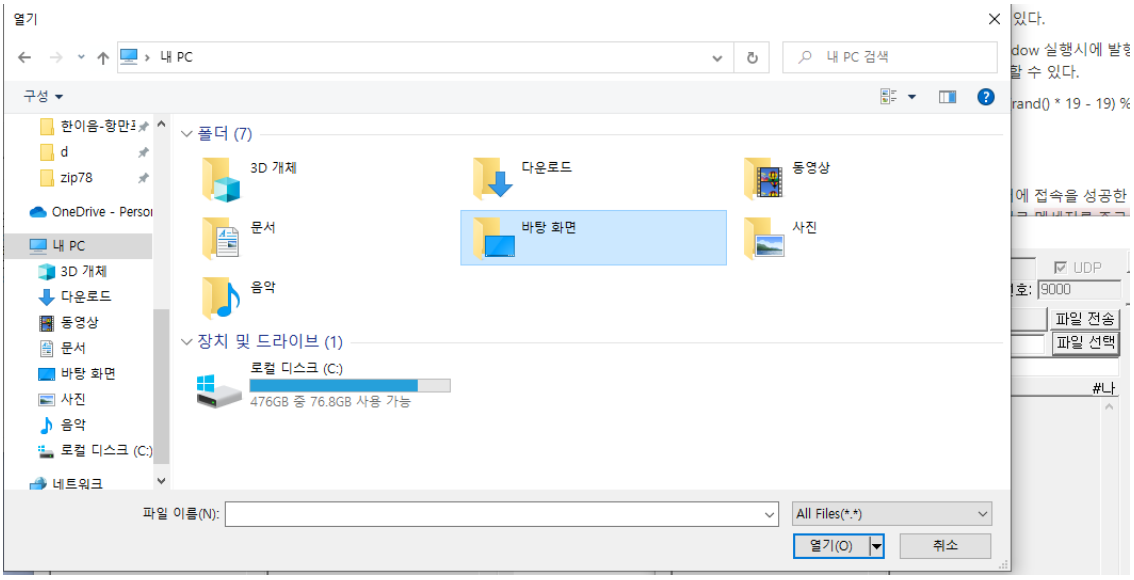
   e. 아래 캡처 사진은 “구현2” ID를 클라이언트 프로그램이 활성화 된 상태입니다.



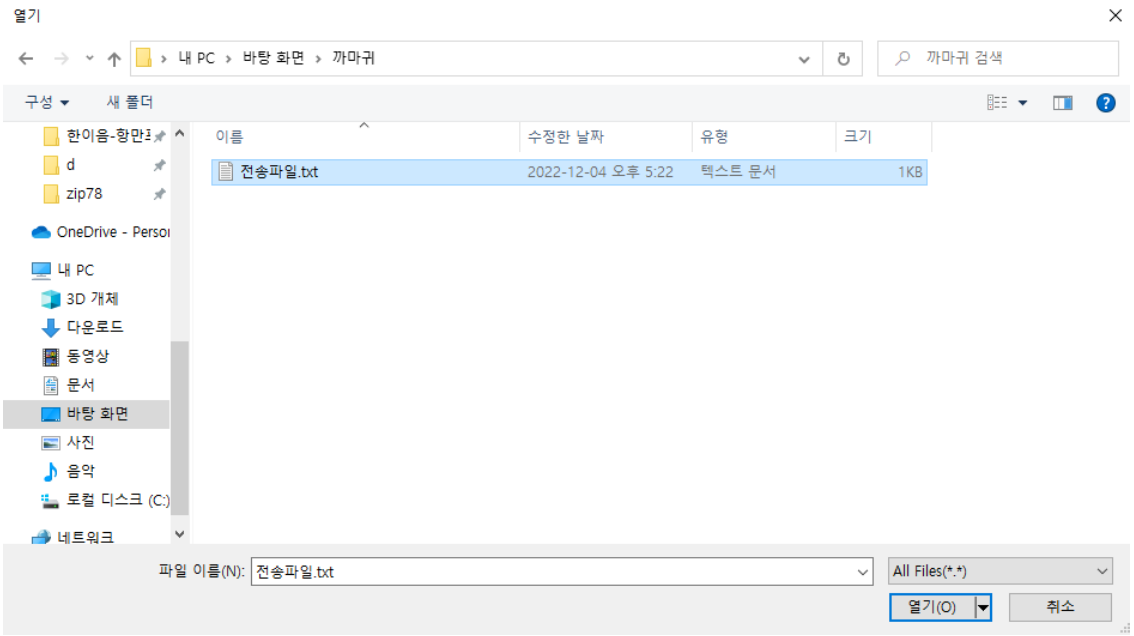
6. **클라이언트 간 파일 송/수신 기능**

→ 어느 클라이언트가 어떤 파일을 보냈는지 확인하고 받은 파일을 저장할 수 있다.

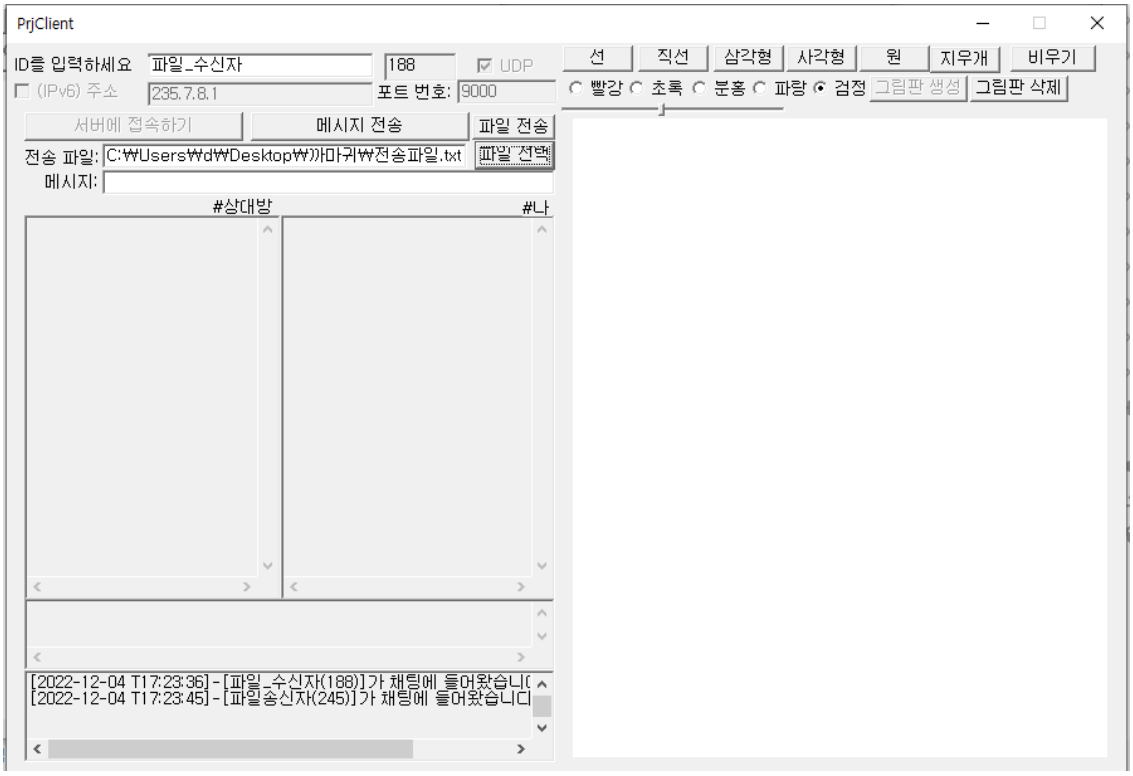
1. 클라이언트 프로그램 화면에서 파일선택 버튼 클릭



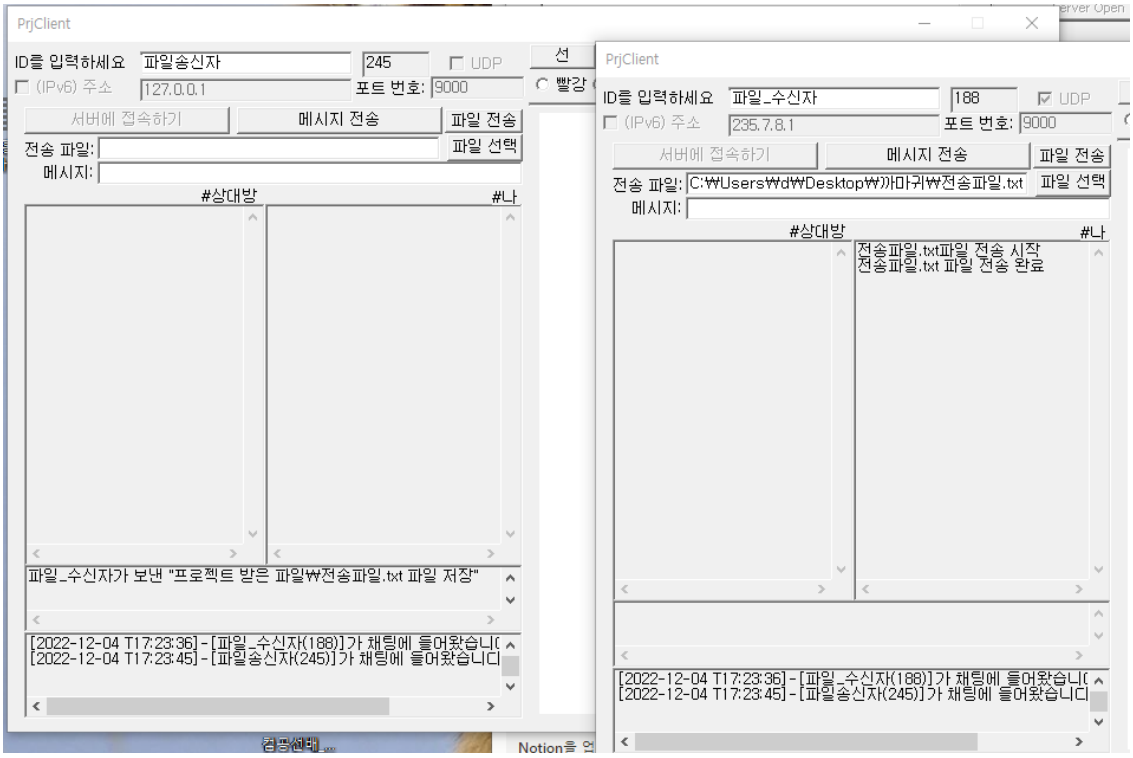
2. 파일선택 후 열기 클릭



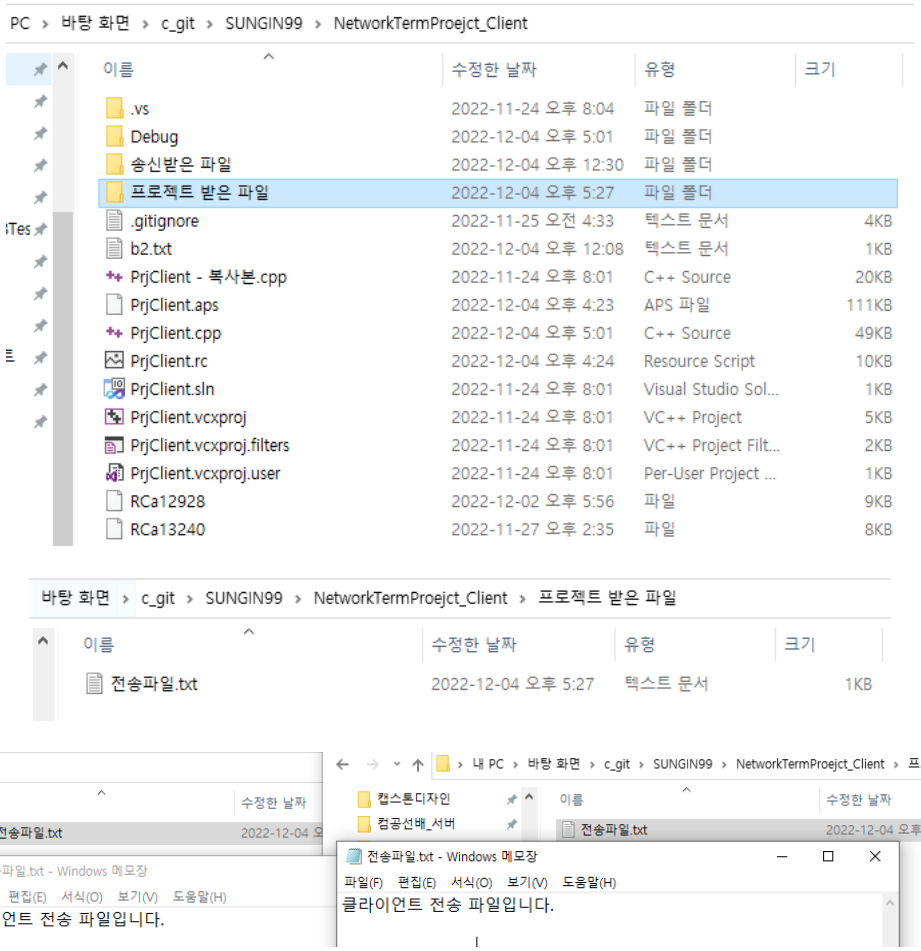
c. 전송파일 EditControll 에 절대 경로로 파일명 문자열 출력



d. “파일전송” 클릭 후 클라이언트 간 화면 상태



e. 프로젝트가 있는 지정된 디렉터리 “프로젝트 받은 파일”에 파일 저장 됨


PC › 바탕 화면 › c_git › SUNGIN99 › NetworkTermProejct_Client
이름 수정한 날짜 유형 크기
.vs 2022-11-24 오후 8:04 파일 폴더
Debug 2022-12-04 오후 5:01 파일 폴더
송신받은 파일 2022-12-04 오후 12:30 파일 폴더
프로젝트 받은 파일 2022-12-04 오후 5:27 파일 폴더
.gitignore 2022-11-25 오전 4:33 텍스트 문서 4KB
b2.txt 2022-12-04 오후 12:08 텍스트 문서 1KB
PrjClient - 복사본.cpp 2022-11-24 오후 8:01 C++ Source 20KB
PrjClient.aps 2022-12-04 오후 4:23 APS 파일 111KB
PrjClient.cpp 2022-12-04 오후 5:01 C++ Source 49KB
PrjClient.rc 2022-12-04 오후 4:24 Resource Script 10KB
PrjClient.sln 2022-11-24 오후 8:01 Visual Studio Sol... 1KB
PrjClient.vcxproj 2022-11-24 오후 8:01 VC++ Project 5KB
PrjClient.vcxproj.filters 2022-11-24 오후 8:01 VC++ Project Filt... 2KB
PrjClient.vcxproj.user 2022-11-24 오후 8:01 Per-User Project ... 1KB
RCa12928 2022-12-02 오후 5:56 파일 9KB
RCa13240 2022-11-27 오후 2:35 파일 8KB
바탕 화면 › c_git › SUNGIN99 › NetworkTermProejct_Client › 프로젝트 받은 파일
이름 수정한 날짜 유형 크기
전송파일.txt 2022-12-04 오후 5:27 텍스트 문서 1KB

내 PC › 바탕 화면 › c_git › SUNGIN99 › NetworkTermProejct_Client › 프로젝트 받은 파일
캡스톤디자인
컴공선배_서버
이름 수정한 날짜 유형
전송파일.txt 2022-12-04 오후 5:27 텍스트 문서
전송파일.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
클라이언트 전송 파일입니다.
전송파일.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
클라이언트 전송 파일입니다.