서버 요약 보고서

목차

---

0. 기능 요구사항

1. 클라이언트 지원
    - ~~각종 도형 및 메세지 전송 (다각형 함수 사용 X)~~
    - ~~카카오톡 1 기능 지원 (읽음 알림)~~
    - ~~파일 전송~~
2. 서버 지원
    - ~~소켓 옵션 활용~~
    - ~~소켓 입출력 모델 사용~~
    - ~~서버에서 넌블록킹 소켓 사용~~
    - ~~클라이언트 관리 기능 GUI~~
3. 서버-클라이언트 공통 지원
    - ~~채팅 ID 추가~~
    - ~~메세지 전송 방식 : 고정 + 가변 길이~~
    - ~~UDP 프로토콜 지원~~

---

1. 서버 기능 요약(기능서, 최종기능)

1) 서버시작

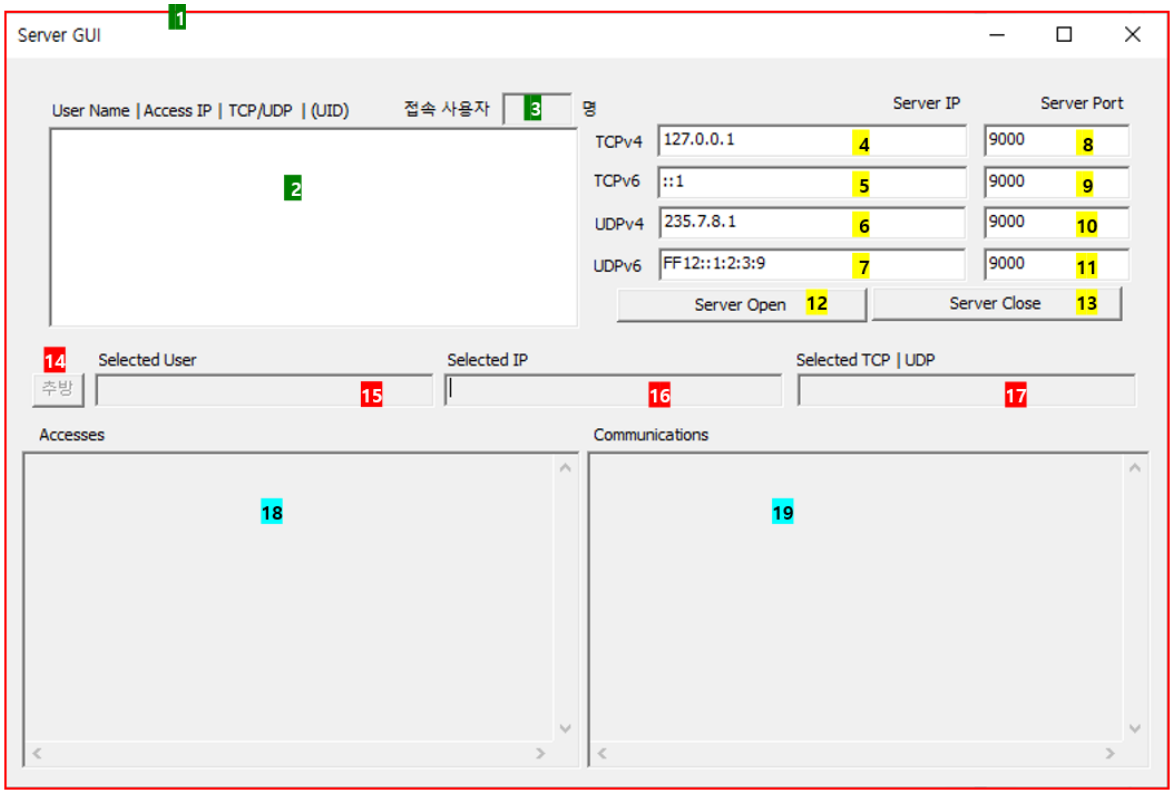
- 서버는 원하는 IP와 Port를 각 프로토콜별(TCP/UDP, IPv4/6)로 정하여서 서버를 시작할 수 있다.
- 클라이언트는 TCP/UDP, IPv4/6 방식과 상관없이 모두 서버에 연결이 가능하다.
- 서버의 TCP 소켓은 소켓 입출력 모델을 사용하고, UDP소켓은 멀티캐스트 방식을 이용한다.

2) 클라이언트 메세지 수신 및 전달

- 서버는 클라이언트의 서버에 대한 접근 메세지(type=ACCESS)를 확인하고 그에 대한 메세지를 출력한다.
- 서버는 서버의 TCP/UDP 소켓으로 클라이언트의 접속을 통해서 메세지를 수신한다.
- 수신한 메세지를 서버의 각 소켓에 접근해 있는 모든 클라이언트에게 메세지를 전달한다.
- 클라이언트는 이를 이용해 서버에 연결되어있는 다른 클라이언트와 채팅, 그림그리기, 파일 전송 등을 할 수 있다.
- 클라이언트에서는 수신 데이터를 구분하여 다른 동작을 취하지만, 서버에서는 클라이언트로부터 들어온 메세지를 그대로 클라이언트에게 전달만 해줄 뿐이다.

3) 클라이언트 관리 GUI

- 서버에서는 현재 클라이언트가 접속한 현황 목록을 확인, 유지할 수 있다.
- 서버는 원할 시에 접속 현황목록에서 한 클라이언트를 지정해 추방 메세지(type=KICKOUT)를 보내 서버와의 연결을 끊도록 할 수 있다.
- 서버는 클라이언트가 전송하는 채팅 메세지, 파일 메세지를 누가 보내는지 확인할 수 있다.



#	ID	Attr	부가설명
1	IDD_DIALOG1	다이얼로그 박스	메인 화면
2	IIDC_USERLIST	리스트 박스	클라이언트 접속 현황을 보여주고, 선택할 수 있다.
3	IDC_COUNT	에디트 컨트롤	현재 클라이언트 접속 인원 수 (서버관점) 를 알려준다.
4	IDC_TCP_IPV4	에디트 컨트롤	서버가 시작할 TCP IPv4주소 를 설정한다.
5	IDC_TCP_IPV6	에디트 컨트롤	서버가 시작할 TCP IPv6주소 를 설정한다.
6	IDC_UDP_IPV4	에디트 컨트롤	서버가 시작할 UDP IPv4주소 (멀티캐스트) 를 설정한다.
7	IDC_UDP_IPV6	에디트 컨트롤	서버가 시작할 UDP IPv6주소 (멀티캐스트) 를 설정한다.
8	IDC_TCP_PORTV4	에디트 컨트롤	서버가 시작할 TCP IPv4 포트번호를 설정한다.
9	IDC_TCP_PORTV6	에디트 컨트롤	서버가 시작할 TCP IPv6 포트번호를 설정한다.
10	IDC_UDP_PORTV4	에디트 컨트롤	서버가 시작할 UDP IPv4 포트번호를 설정한다.
11	IDC_UDP_PORTV6	에디트 컨트롤	서버가 시작할 UDP IPv6 포트번호를 설정한다.
12	IDC_SERVER_OPEN	버튼	정한 주소와 포트 번호로 서버를 시작한다.
13	IDC_SERVER_CLOSE	버튼	서버를 종료하고 모든 스레드를 종료하고 소켓을 닫는다.
14	IDC_BUTTON_OUT1	버튼	클라이언트 접속 현황 리스트에서 선택한 클라이언트에게 추방 메세지를 보내어 서버와의 접속을 끊는다.
15	IDC_USERNAME1	에디트 컨트롤	클라이언트 접속 현황 리스트에서 선택한 클라이언트의 클라이언트 ID를 알려준다.
16	IDC_USERADDR1	에디트 컨트롤	클라이언트 접속 현황 리스트에서 선택한 클라이언트의 접속 IP:PORT를 알려준다.

#	ID	Attr	부가설명
17	IDC_USERTCP	에디트 컨트롤	클라이언트 접속 현황 리스트에서 선택한 클라이언트의 TCP/UDP를 알려준다.
18	IDC_EDIT_ALLMSG	에디트 컨트롤	클라이언트의 최초 접속 시에 접속했다는 알림 메세지를 확인할 수 있다.
19	IDC_EDIT_ALLMSG2	에디트 컨트롤	클라이언트 간에 대화내용과 파일 메세지를 확인할 수 있다.

3. 기능 상세

a) 사용자 정의 상수(서버-클라이언트 공용 시그널)

1. **사용자 정의 윈도우 메세지**

```
#define WM_SOCKET  (WM_USER+1)
```

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

```
#define MULTICAST_RECV_IPv4 "235.7.8.1"
#define MULTICAST_SEND_TO_CLIENT_IPv4 "235.7.8.2"

#define MULTICAST_RECV_IPv6 "FF12::1:2:3:9"
#define MULTICAST_SEND_TO_CLIENT_IPv6 "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)**

```
#define BUFSIZE     284
#define MSGSIZE     (BUFSIZE-(sizeof(int)*2)-ID_SIZE-CHECK_WHO-TIME_SIZE)
#define ID_SIZE     20
#define CHECK_WHO   1
#define TIME_SIZE   23
```

4. **송/수신 메세지 식별 타입 (서버-클라이언트 공용)**

```
#define CHATTING    2000
#define ACCESS      3000
#define KICKOUT     3001

#define FILEINIT    4001
#define FILEBYTE    4002
#define FILEEND     4003
```

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

1. CHAT_MSG : 클라이언트 전달용 채팅 및 각종 알림 메세지 구조체

→ 클라이언트에게 온 메세지는 CHAT_MSG형태로 바로 받아서 그대로 접속 중인 다른 클라이언트들에게 전달만한다.

→ 크기는 클라이언트에서 보내는 메세지 구조체와 동일해서 포인터 형식으로 전달하는 방식이다.

1. CHAT_MSG 구조체

```
//S-1) 클라이언트 Access 메세지 (클라이언트에게 받기만 함)
typedef struct CHAT_MSG_ {
    int  type;                  // 메세지 타입 (CHATTING: 채팅, DRAWERS: 그림, ACCESS: 최초접속)
    char client_id[ID_SIZE];    // 클라이언트 ID
    char buf[MSGSIZE];          // 메세지 버퍼
    char whenSent[TIME_SIZE];
    int whoSent;                // 서버에서 보냈다면 whoSent = -1 그 외에는 NULL
}CHAT_MSG;
```

→ 클라이언트가 송신한 메세지 내용을 포함하는 구조체이다. (클라이언트와 동일)
→ 서버에서 사용하는 메세지타입은 (CHATTING, FILEINIT, ACCESS, KIKCOUT 밖에 없다.)

c) 소켓정보 유지 Array & ArrayList

1. 소켓 입출력 모델 사용시에 필요한 TCP 구조체 배열

```
// Array) TCP select 함수에 사용할 배열
struct SOCKETINFO_ONLY_TCP {
    SOCKET      sock;
    bool        isIPv6;
    CHAT_MSG    chatmsg;
    int         recvbytes;
};
int nTotalTCPSockets = 0;
SOCKETINFO_ONLY_TCP* SocketInfoArray[FD_SETSIZE];
```

2. 서버에 접근한 모든 클라이언트(TCP, UDP 모두)를 유지하기 위한 ArrayList

```
// ArrayList) TCP UDP 연결리스트
struct SOCKETINFO_UDPnTCP {
    // 1) 소켓 관련 정보 (소켓, 소켓 IP주소, IPv6, UDP판별 변수)
    SOCKET          sock;           // TCP 소켓 정보 (isUDP == FALSE)
    char            client_id[ID_SIZE]; // 클라이언트 ID
    int             clientUniqueID;
    CHAT_MSG        chatmsg;            // 수신 받을 데이터
    SOCKADDR_IN* sockaddrv4;        // IPv4 어드레스 (isIPv6 == FALSE)
    SOCKADDR_IN6* sockaddrv6;       // IPv6 어드레스 (isIPv6 == TRUE)
    bool            isIPv6;         // IPv6 판별함수
    bool            isUDP;          // UDP 판별함수
    char            socktype[8];    // 소켓 타입 문자열 ("TCP₩0", "UDP₩0")

    // 2) recv/send bytes
    int             recvbytes;      // 받은 바이트 수
    int             sendbytes;      // 보낼 바이트 수
    BOOL            recvdelayed;
    SOCKETINFO_UDPnTCP* next;       // 다음 노드
};
int nALLSockets = 0;
SOCKETINFO_UDPnTCP* AllSocketInfoArray = NULL;
int curIndexLB, curIndexCB;
```

1- SOCKET, 2- 클라이언트 ID, 3- 클라이언트 식별번호, 4- 메세지, 5- 연결 IP: v4,6둘중하나만 사용
6- 그외 소켓 접속정보 , 7- 송/수신 바이트 수, 8- 리스트 NextNode

d) 전역변수 및 프로시져 지역변수

```
static HINSTANCE     g_hInst;     // 응용 프로그램 인스턴스 핸들
static HANDLE        g_hServerThread;   // 서버 스레드
static HANDLE        g_ServerSendThread;

static HWND          hEdit_User_Send; // 송신 EditControll
static HWND          hEdit_Serv_Send; // 수신 EditControll

// 클라이언트 리스트
static HWND          hUserList;     // IDC_USERLIST : 클라이언트 List Box
static HWND          hUserCount;    // 클라이언트 수 표시 Edit Controll

// 추방할 클라이언트 정보
```

```
static HWND          hUserNames;    // IDC_USERNAME
static HWND          hUserAddrs;    // IDC_USERADDR
static HWND          hUserTCPorUDP;     // IDC_USERTCP


// 서버를 시작할 주소정보 EditControll
static HWND          serverTCPIPv4; static HWND     serverTCPPORTv4;
static HWND          serverTCPIPv6; static HWND     serverTCPPORTv6;
static HWND          serverUDPIPv4; static HWND     serverUDPPORTv4;
static HWND          serverUDPIPv6; static HWND     serverUDPPORTv6;

// 서버를 시작할 주소 정보
static char strTCPv4[60], strTCPv6[60], strUDPv4[60], strUDPv6[60];
static char strTPORTv4[10], strTPORTv6[10], strUPORTv4[10], strUPORTv6[10];

static HWND          serverOpenBtn;
static HWND          serverCloseBtn;
static HANDLE*       handleHandle;
```

e) 사용자 정의 함수 & 스레드

```
// 0) 소켓 관리 함수
BOOL AddSocketInfo(SOCKET sock, bool isIPv6);
void RemoveSocketInfo(int nIndex);
BOOL AddAllSocketInfo(SOCKET sock, char* username, int userUniqudID, int checkIPv6, int checkUDP, SOCKADDR* pee
void RemoveAllSocketInfo(int index);

// 1) 모든 통신스레드의 부모스레드(Server에서 사용할 소켓 정보 초기화 스레드)
DWORD WINAPI ServerMain(LPVOID);
// 2) 통신 스레드
DWORD WINAPI TCP(LPVOID);                       // TCP 통신 스레드(소켓 입출력 모델 사용)
DWORD WINAPI UDPv4_Multicast(LPVOID);           // UDPv4 통신 스레드
DWORD WINAPI UDPv6_Multicast(LPVOID);           // UDPv6 통신 스레드

// Window
BOOL CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);  // 컨트롤 핸들러 (서버 GUI 프로시저)

// 3) EditControll 출력 함수
void DisplayText_Acc(char* fmt, ...);
void DisplayText_Send(char* fmt, ...);

// 4) 사용자 정의함수
char* listupText(char* fmt, ...);               // C-1) 리스트 박스 출력 문자열 반환 함수
void resetUserCount();                          // C-2) 유지된 연결리스트로 현재 접속 유저 수 최신화
void selectedUser(char* selectedItem, int index);   // C-3) 선택된 유저 추방 정보 채우기위한 문자열 반환 함수
void updateUserList();                          // C-4) 유지된 연결리스트로 리스트박스 최신화


// 5) 오류 함수
void err_quit(char* msg);
void err_display(char* msg);
void err_display(int errcode);
```

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

1. 윈속 초기화를 하고 다이얼로그 박스를 생성한다.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCm
    // 윈속 초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) return 1;

    // 대화상자 생성
    g_hInst = hInstance;
    DialogBox(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), NULL, WndProc);

    // 윈속 종료
    WSACleanup();
    return 0;
}
```

Window) WndProc: 다이얼로그 박스 프로시져 (메세지 핸들)

- 지역변수를 선언하고 WM_INITDIALOG에서 컨트롤 핸들을 가져오고 초기화한다.
- 윈도우 컨트롤 핸들 처리 (기능 식별)

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

a. Server Open 버튼 클릭
→ 서버에서 지정한 TCP/UDP, IPv4/6 주소로 클라이언트가 접속할 수 있는 서버를 시작한다.
→ 서버 시작시에 주소 입력 EditControll은 비활성화 상태가 된다. (읽기전용, 변경 불가)

```
case IDC_SERVEROPEN:
      g_hServerThread = CreateThread(NULL, 0, ServerMain, NULL, 0, NULL);
      EnableWindow(serverOpenBtn, FALSE);
      EnableWindow(serverTCPIPv4, FALSE); EnableWindow(serverTCPPORTv4, FALSE);
      EnableWindow(serverTCPIPv6, FALSE); EnableWindow(serverTCPPORTv6, FALSE);
      EnableWindow(serverUDPIPv4, FALSE); EnableWindow(serverUDPPORTv4, FALSE);
      EnableWindow(serverUDPIPv6, FALSE); EnableWindow(serverUDPPORTv6, FALSE);
      return TRUE;
```

b. Server Close 버튼 클릭
→ 서버에서 유지하고 있는 TCP, UDPv4, UDPv6 스레드를 모두 강제 종료하여 통신을 모두 끊는다.

```
case IDC_SERVERCLOSE:
      TerminateThread(handleHandle[0], 1);
      TerminateThread(handleHandle[1], 1);
      TerminateThread(handleHandle[2], 1);
      return TRUE;
```

c. 접속 클라이언트 현황 목록에서 아이템 클릭 시
→ 클릭한 아이템의 문자열을 가져와 selectedUser()함수에 파라미터로 넘겨준다.
→ selectedUser()에서 문자열을 정해둔 형식에서 토큰화하여 접속 클라이언트의 클라이언트 ID, 주소, TCP/UDP종류를 추방 필드에 출력시킨다.

```
case IDC_USERLIST:
      switch (HIWORD(wParam)) {
      case LBN_SELCHANGE:
        curIndexLB = SendMessage(hUserList, LB_GETCURSEL, 0, 0);
        char* selectedItem = (char*)malloc(256);
        SendMessage(hUserList, LB_GETTEXT, curIndexLB, (LPARAM)selectedItem);
        selectedUser(selectedItem, curIndexLB);
        EnableWindow(hUserOutBtns, TRUE);
      }
      return TRUE;
```

```
void selectedUser(char* selectedItem, int index) {
  char username[20];
  char ipaddr[65];
  char TorU[8];

  char* ptr = strtok(selectedItem, "\t");
  strncpy(username, ptr, 20);

  ptr = strtok(NULL, "\t");
  strncpy(ipaddr, ptr, 65);
  ipaddr[64] = NULL;

  ptr = strtok(NULL, "\t");
  strncpy(TorU, ptr, 8);

  SendMessage(hUserNames, EM_SETSEL, 0, 20);
  SendMessage(hUserNames, WM_CLEAR, 0, 0);
  SendMessage(hUserNames, EM_REPLACESEL, FALSE, (LPARAM)username);

  SendMessage(hUserAddrs, EM_SETSEL, 0, 65);
  SendMessage(hUserAddrs, WM_CLEAR, 0, 0);
  SendMessage(hUserAddrs, EM_REPLACESEL, FALSE, (LPARAM)ipaddr);

  SendMessage(hUserTCPorUDP, EM_SETSEL, 0, 8);
```

```
  SendMessage(hUserTCPorUDP, WM_CLEAR, 0, 0);
  SendMessage(hUserTCPorUDP, EM_REPLACESEL, FALSE, (LPARAM)TorU);

}
```

d. 추방 버튼 클릭
→ 접속 클라이언트 목록 리스트에서 클릭한 아이템의 인덱스를 이용해 연결리스트에서 해당 클라이언트에게 추방 메세지(type=KICKOUT)를 보내고 노드를 삭제한다.
→ 바뀐 연결리스트를 이용해 목록을 최신화 한다.

```
case IDC_BUTTON_OUT1:
      RemoveAllSocketInfo(curIndexLB);
      EnableWindow(hUserOutBtns, FALSE);
      return TRUE;
```

1) DWORD WINAPI ServerMain(LPVOID arg) → 각 소켓 통신 스레드의 부모 스레드

1. 스레드 내에서 각 소켓통신 스레드에서 사용할 소켓과 주소를 모두 초기화한다.
2. 초기화한 스레드 정보들을 모두 전역변수 구조체에 담아서 파라미터로 넘겨 자식 스레드(TCP, UDPv4, UDPv6)를 모두 시작한다.
3. 스레드종료를 기다리고 스레드가 종료되면(서버가 통신을 종료하려하면) 모든 소켓을 닫는다.

```
// 주요코드
typedef struct ALL_SERVER_SOCKET_ {
  SOCKET tcp_v4;
  SOCKET tcp_v6;
  SOCKET udp_recv_v4;
  SOCKET udp_send_v4;
  SOCKET udp_recv_v6;
  SOCKET udp_send_v6;

  SOCKADDR_IN remoteaddr_v4;
  SOCKADDR_IN6 remoteaddr_v6;
}ALL_SERVER_SOCKET;
ALL_SERVER_SOCKET All_Sock;
-----------------------------------------------------------------------
All_Sock = {
    listen_sockv4, listen_sockv6,
    listen_sock_UDPv4, send_sock_UDPv4,
    listen_sock_UDPv6, send_sock_UDPv6,
    remoteaddr_v4, remoteaddr_v6};

  ALL_SERVER_SOCKET* All_Sock_P = &All_Sock;

  HANDLE hThread[3];
  hThread[0] = CreateThread(NULL, 0, TCP, (LPVOID)All_Sock_P, 0, NULL);
  hThread[1] = CreateThread(NULL, 0, UDPv4_Multicast, (LPVOID)All_Sock_P, 0, NULL);
  hThread[2] = CreateThread(NULL, 0, UDPv6_Multicast, (LPVOID)All_Sock_P, 0, NULL);
  handleHandle = hThread;
  DWORD please = WaitForMultipleObjects(3, hThread, TRUE, INFINITE);
  closesocket(listen_sockv4);
  closesocket(listen_sockv6);
  closesocket(listen_sock_UDPv4);
  closesocket(send_sock_UDPv4);
  closesocket(listen_sock_UDPv6);
  closesocket(send_sock_UDPv6);


  return 0;
```

2) 각 소켓 통신 스레드(ServerMain 자식스레드)

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

종류	type	동작
클라이언트 최초 접속 메세지	ACCESS	클라이언트가 통신스레드 시작시에 서버에게 반드시 보내는 최초 메세지이다. 접속 정보를 Accesses 에디트 컨트롤에 출력한다.

종류	type	동작
클라이언트간 채팅 메세지	CHATTING	클라이언트간 송신 채팅 메세지를 나타내고 Communications 에디트 컨트롤에 출력한다.
클라이언트 파일전송 시작 메세지	FILEINIT	클라이언트가 보낼 파일명과 누가보냈는지를 Communications 에디트 컨트롤에 출력한다.

2-1) DWORD WINAPI TCP(LPVOID arg)

1. 스레드 시작 후에 ServerMain에서 넘겨준 파라미터중 TCP소켓을 미리 정해둔 주소로 bind()한다.
2. TCP 소켓 입출력 모델에 사용할 소켓 셋을 선언한다.
3. 소켓 셋을 이용해 사용하는 모든 TCP 소켓을 관리하여 accept(), recv()를 동작시킨다.
4. 만일 수신 데이터가 발생할 시에 타입에 따라 동작 처리를 한뒤 메세지를 그대로 유지한 채 현재 연결 중인 TCP클라이언트 소켓과 파라미터로 넘겨받은 UDP 멀티캐스트 소켓에 모두 메세지를 송신한다.

```
// 소켓 입출력 모델 Select() 사용 코드 일부
FD_SET rset;
SOCKET client_sock;
int addrlen, i, j;
int retval, retvalUDP;
SOCKADDR_IN clientaddrv4;
SOCKADDR_IN6 clientaddrv6;

while (1) {
    // 소켓 셋 초기화
    FD_ZERO(&rset);
    FD_SET(listen_sockv4, &rset);
    FD_SET(listen_sockv6, &rset);
    for (i = 0; i < nTotalTCPSockets; i++) {
      FD_SET(SocketInfoArray[i]->sock, &rset);
    }

    // select()
    retval = select(0, &rset, NULL, NULL, NULL);
    if (retval == SOCKET_ERROR) {
      err_display("select()");
      break;
    }

    // 소켓 셋 검사(1): 클라이언트 접속 수용
    if (FD_ISSET(listen_sockv4, &rset)) {
      addrlen = sizeof(clientaddrv4);
      client_sock = accept(listen_sockv4, (SOCKADDR*)&clientaddrv4, &addrlen);
      if (client_sock == INVALID_SOCKET) {
        err_display("accept()");
        break;
      }
      else {
        // 접속한 클라이언트 정보 출력
        DisplayText_Acc("[TCPv4] 클라이언트 접속 확인: [%s]:%d \r\n",
          inet_ntoa(clientaddrv4.sin_addr), ntohs(clientaddrv4.sin_port));
        // 소켓 정보 추가
        AddSocketInfo(client_sock, false);
      }
    }

    // 소켓 셋 검사(2): 데이터 통신
    for (i = 0; i < nTotalTCPSockets; i++) {
      SOCKETINFO_ONLY_TCP* ptr = SocketInfoArray[i];
      if (FD_ISSET(ptr->sock, &rset)) {
        // 데이터 받기
        retval = recv(ptr->sock, (char*)&(ptr->chatmsg) + ptr->recvbytes,
          BUFSIZE - ptr->recvbytes, 0);
        if (retval == 0 || retval == SOCKET_ERROR) {
          RemoveSocketInfo(i);
          continue;
        }
        if (ptr->chatmsg.type == ACCESS) {
          AddAllSocketInfo(ptr->sock, ptr->chatmsg.client_id, ptr->chatmsg.whoSent, ptr->isIPv6, FALSE, NULL);
          continue;
        }

        if (ptr->chatmsg.type == CHATTING)
          DisplayText_Send("[%s]-[%s(%d)]: %s\r\n", ptr->chatmsg.whenSent, ptr->chatmsg.client_id, ptr->chatmsg.whoSent, ptr->chatr

        if(ptr->chatmsg.type == FILEINIT){
          DisplayText_Send("%s(%d) sended File: %s\r\n", ptr->chatmsg.client_id, ptr->chatmsg.whoSent, ptr->chatmsg.buf);
        }

        // 받은 바이트 수 누적
```

```
        ptr->recvbytes += retval;

        if (ptr->recvbytes == BUFSIZE) {
          // 받은 바이트 수 리셋
          ptr->recvbytes = 0;

          // 현재 접속한 모든 클라이언트에게 데이터를 보냄!
          for (j = 0; j < nTotalTCPSockets; j++) {
            SOCKETINFO_ONLY_TCP* ptr2 = SocketInfoArray[j];
            retval = send(ptr2->sock, (char*)&(ptr->chatmsg), BUFSIZE, 0);

            if (retval == SOCKET_ERROR) {
              err_display("send()");
              RemoveSocketInfo(j);
              --j; // 루프 인덱스 보정
              continue;
            }
          }

          // UDP 에게도 보내기
          retvalUDP = sendto(send_sock_UDPv4, (char*)&(ptr->chatmsg), BUFSIZE, 0,
            (SOCKADDR*)&remoteaddr_v4, sizeof(remoteaddr_v4));
          // UDP 에게도 보내기
          retvalUDP = sendto(send_sock_UDPv6, (char*)&(ptr->chatmsg), BUFSIZE, 0,
            (SOCKADDR*)&remoteaddr_v6, sizeof(remoteaddr_v6));
        }
      }
    }
  }
```

2-2) DWORD WINAPI UDPv4_Multicast(LPVOID arg)

1. 스레드 시작 후에 ServerMain에서 넘겨준 파라미터 중 UDPv4소켓(송/수신) 을 미리 정해둔 주소로 bind()한다.
2. 수신 소켓(listen_sock_UDPv4)을 소켓옵션을 이용해 서버 시작 시에 설정해둔 주소로 멀티캐스트에 가입한다.
3. 송신 소켓(send_sock_UDPv4)을 소켓옵션을 이용해 TTL을 설정한다.
4. TCP에서 사용하는 소켓 입출력 모델을 사용하지 않지만, 클라이언트로부터 메세지를 받고 타입에 따라 동작 처리를 진행한뒤에 접속해있는 모든 클라이언트들에게 그대로 다시 전달(송신)해주는 기능은 동일하다.

```
while (1) {
    CHAT_MSG* chatmsg = (CHAT_MSG*) malloc(sizeof(CHAT_MSG));
    addrlen_UDP = sizeof(peeraddr_v4);

    retvalUDP = recvfrom(listen_sock_UDPv4, (char*)(chatmsg), BUFSIZE, 0,
      (SOCKADDR*)&peeraddr_v4, &addrlen_UDP); // peeraddr NULL 로 해도 ㄱㅊ

    // 서버에서는 BUFSIZE+1 만큼 더 보냄
    // 클라이언트는 buf에 저장되어있는 글자수만큼만 보냄
    if (retvalUDP == SOCKET_ERROR) {
      err_display("recvfrom()");
      continue;
    }

    if (chatmsg->type == ACCESS) {
      //SendMessage(hUserList, LB_ADDSTRING, 0, (LPARAM)(char*)&(ptr->buf.client_id));
      WSAAddressToString((SOCKADDR*)&peeraddr_v4, sizeof(peeraddr_v4), NULL, ipaddr, &ipaddrlen);
      DisplayText_Acc("[UDPv4] 클라이언트 접속: %s\n", ipaddr); // 서버가 보낸게 아니라면, 서버는 맨끝 바이트를 -1로 초기화
      AddAllSocketInfo(NULL, chatmsg->client_id, chatmsg->whoSent, FALSE, TRUE, (SOCKADDR*)&peeraddr_v4);
      continue;
    }

    if (chatmsg->type == CHATTING)
          DisplayText_Send("[%s]-[%s(%d)]: %s\r\n", chatmsg->whenSent, chatmsg->client_id, chatmsg->whoSent, chatmsg->buf);

    if (chatmsg->type == FILEINIT) {
      DisplayText_Send("%s(%d) sended File: %s\r\n", chatmsg->client_id, chatmsg->whoSent, chatmsg->buf);
    }

    // UDP v4에게 보냄
    retvalUDP = sendto(send_sock_UDPv4, (char*)(chatmsg), BUFSIZE, 0,
      (SOCKADDR*)&remoteaddr_v4, sizeof(remoteaddr_v4));

    if (retvalUDP == SOCKET_ERROR) {
      err_display("sendto()");
      continue;
    }

    // UDP v6 에게도 보냄
    retvalUDP = sendto(send_sock_UDPv6, (char*)(chatmsg), BUFSIZE, 0,
      (SOCKADDR*)&remoteaddr_v6, sizeof(remoteaddr_v6));
    if (retvalUDP == SOCKET_ERROR) {
```

```
      err_display("sendto()");
      continue;
    }

    // to TCP
    for (int j = 0; j < nTotalTCPSockets; j++) {
      SOCKETINFO_ONLY_TCP* ptr2 = SocketInfoArray[j];
      retvalTCP = send(ptr2->sock, (char*)(chatmsg), BUFSIZE, 0);
      if (retvalTCP == SOCKET_ERROR) {
        err_display("send()");
        RemoveSocketInfo(j);
        --j; // 루프 인덱스 보정
        continue;
      }
    }
    free(chatmsg);
  }
```

2-3) DWORD WINAPI UDPv6_Multicast(LPVOID arg)

1. 스레드 시작후에 ServerMain 에서 넘겨준 파라미터 중 UDPv6소켓(송/수신) 을 미리 정해둔 주소로 bind()한다.
2. 그외에는 UDPv4에서의 방법과 IPv6주소를 사용하는 것 빼고 모든 동작이 동일하다.

0) 소켓 정보 Array & ArrayList 관리 함수

0-1) BOOL AddSocketInfo(SOCKET sock, bool isIPv6)

1. TCP스레드에서 입출력 모델로 aceept() 하는 클라이언트 소켓 발생시
   SOCKETINFO_ONLY_TCP 구조체 배열에 정보를 추가한다.

```
BOOL AddSocketInfo(SOCKET sock, bool isIPv6)
{
  if (nTotalTCPSockets >= FD_SETSIZE) {
    DisplayText_Acc("[오류] 소켓 정보를 추가할 수 없습니다! \r\n");
    return FALSE;
  }

  SOCKETINFO_ONLY_TCP* ptr = new SOCKETINFO_ONLY_TCP;
  if (ptr == NULL) {
    DisplayText_Acc("[오류] 메모리가 부족합니다! \r\n");
    return FALSE;
  }

  ptr->sock = sock;
  ptr->isIPv6 = isIPv6;
  ptr->recvbytes = 0;
  SocketInfoArray[nTotalTCPSockets++] = ptr;

  return TRUE;
}
```

0-2) void RemoveSocketInfo(int nIndex)

1. TCP스레드에서 입출력 모델로 소켓을 사용하던 중 더이상 연결이 유지되지 않으면
   SOCKETINFO_ONLY_TCP 구조체 배열에서 해당 소켓정보를 삭제하고 삭제 알림을 출력한다.

```
void RemoveSocketInfo(int nIndex)
{
  SOCKETINFO_ONLY_TCP* ptr = SocketInfoArray[nIndex];

  // 종료한 클라이언트 정보 출력
  if (ptr->isIPv6 == false) {
    SOCKADDR_IN clientaddrv4;
    int addrlen = sizeof(clientaddrv4);
    getpeername(ptr->sock, (SOCKADDR*)&clientaddrv4, &addrlen);
    DisplayText_Acc("[TCPv4] 클라이언트 종료: [%s]:%d \r\n",
      inet_ntoa(clientaddrv4.sin_addr), ntohs(clientaddrv4.sin_port));
  }
  else {
    SOCKADDR_IN6 clientaddrv6;
    int addrlen = sizeof(clientaddrv6);
    getpeername(ptr->sock, (SOCKADDR*)&clientaddrv6, &addrlen);

    char ipaddr[50];
    DWORD ipaddrlen = sizeof(ipaddr);
    WSAAddressToString((SOCKADDR*)&clientaddrv6, sizeof(clientaddrv6),
      NULL, ipaddr, &ipaddrlen);
```

```
    DisplayText_Acc("[TCPv6] 클라이언트 종료: %s \r\n", ipaddr);
  }

  closesocket(ptr->sock);
  delete ptr;

  if (nIndex != (nTotalTCPSockets - 1))
    SocketInfoArray[nIndex] = SocketInfoArray[nTotalTCPSockets - 1];

  --nTotalTCPSockets;

  resetUserCount();
  updateUserList();
}
```

0-3) BOOL AddAllSocketInfo(SOCKET sock, char *username, int userUniqudID, int isIPv6, int isUDP, SOCKADDR peeraddr)

1. TCP, UDP 소켓에서 서버에 접속하려 할때 최초로 보내는 ACCESS 타입의 메세지를 받을때 이 함수를 호출한다.
2. 클라이언트의 정보를 파라미터로 넘겨받아서 구조체 노드를 생성하고 정해 기준에 따라값을 할당한다.
   1- 소켓 구조체, 2- IPv4/6접속여부, 3- TCP/UDP접속 여부, 4- 클라이언트 식별번호, 5- 클라이언트 ID
3. 변경된 연결리스트를 이용해 현재 접속하고 있는 클라이언트 현황 리스트를 최신화 한다. (ListBox, 접속 인원수)

```
// 소켓 TCP UDP 통합 ArrayList : AllSocketInfoArray
BOOL AddAllSocketInfo(SOCKET sock, char *username, int userUniqudID, int isIPv6, int isUDP, SOCKADDR* peeraddr) {
  SOCKETINFO_UDPnTCP* ptr = new SOCKETINFO_UDPnTCP;
  SOCKADDR_IN* sockaddrv4 = new SOCKADDR_IN;
  SOCKADDR_IN6* sockaddrv6 = new SOCKADDR_IN6;

  int addrlen;
  char* listupMsg = (char*)malloc(256);
  char* ipaddrv4 = (char*)malloc(INET_ADDRSTRLEN);
  char* ipaddrv6 = (char*)malloc(INET6_ADDRSTRLEN);
  DWORD ipaddr6len = INET6_ADDRSTRLEN;
  DWORD ipaddr4len = INET_ADDRSTRLEN;

  if (ptr == NULL) {
    err_display("wrong socket info");
    return false;
  }

  // 1. 소켓 설정 및 정보 모아서 리스트박스에 넣을준비
  ptr->sock = sock;
  ptr->isIPv6 = isIPv6;
  ptr->isUDP = isUDP;
  ptr->clientUniqueID = userUniqudID;

  // 2. 소켓들어온 클라이언트 이름
  int len_username = strlen(username);
  int dummy_username = ID_SIZE - len_username;

  strncpy(ptr->client_id, username, len_username);
  memset(ptr->client_id + len_username, 0, dummy_username);
  ptr->client_id[ID_SIZE - 1] = NULL;

  // 3. UDP 판별 및 프로토콜 가져오기
  if (isUDP == FALSE) {
    // 3. IPv6 판별 및 주소 가져오기
    if (isIPv6 == false) {
      strncpy(ptr->socktype, "TCPv4", 6);
      addrlen = sizeof(SOCKADDR_IN);
      getpeername(sock, (SOCKADDR*)sockaddrv4, &addrlen);
      ptr->sockaddrv4 = sockaddrv4;
    }
    else {
      strncpy(ptr->socktype, "TCPv6", 6);
      addrlen = sizeof(SOCKADDR_IN6);
      getpeername(sock, (SOCKADDR*)sockaddrv6, &addrlen);
      ptr->sockaddrv6 = sockaddrv6;
    }
  }
  else {
    if (isIPv6 == FALSE) {
      strncpy(ptr->socktype, "UDPv4", 6);
      ptr->sockaddrv4 = (SOCKADDR_IN*)peeraddr;
    }
    else {
      strncpy(ptr->socktype, "UDPv6", 6);
      ptr->sockaddrv6 = (SOCKADDR_IN6*)peeraddr;
    }
  }
```

```
  ptr->next = AllSocketInfoArray;
  AllSocketInfoArray = ptr;
  nALLSockets++;

  resetUserCount();
  updateUserList();

  return TRUE;
}
```

0-4) void RemoveAllSocketInfo(int index)

1. ListBox에서 선택한 아이템의 인덱스를 파라미터로 넘겨 추방 시킬 시에 이 함수를 호출한다.
2. 인덱스로 연결리스트를 순회해서 추방할 클라이언트 정보를 가져온다.
3. TCP는 해당 소켓에 바로 추방 메세지를 보낸다.
4. UDP는 스레드나 함수에서 AddAllSocketInfo()에서 구조체에 저장한 peeraddress로 메세지가 보내지지 않아서 UDP 소켓에 멀티캐스트 주소로 추방 메세지를 보낸다. (왜 이런지 이유는 찾지 못했습니다)
   이때 UDP로 접속한 클라이언트에게 추방 메세지가 모두 전달 되지만 메세지 안의 클라이언트ID와 식별번호가 같은 클라이언트만 서버와의 접속을 종료한다.
5. 클라이언트에게 추방 메세지를 보내고 연결이 종료된 후에 연결리스트와 현재 접속 클라이언트 리스트를 최신화한다.

```
void RemoveAllSocketInfo(int index) {
  SOCKETINFO_UDPnTCP* ptr = AllSocketInfoArray;
  SOCKETINFO_UDPnTCP* prev = NULL;

  char ipaddr[50];
  DWORD ipaddrlen = sizeof(ipaddr);

  int i = 0, retval;
  while (ptr != NULL) {

    if (index == i) {
      CHAT_MSG endMsg;
      endMsg.type = KICKOUT;
      strncpy(endMsg.client_id, ptr->client_id, ID_SIZE);
      endMsg.whoSent = ptr->clientUniqueID;
      if (ptr->isUDP == false)
        retval = send(ptr->sock, (char*)&endMsg , BUFSIZE , 0);

      else {
        if (ptr->isIPv6 == false) {
          WSAAddressToString((SOCKADDR*)ptr->sockaddrv4, sizeof(SOCKADDR_IN),
            NULL, ipaddr, &ipaddrlen);

          retval = sendto(All_Sock.udp_send_v4, (char*)&endMsg , BUFSIZE, 0,
            (SOCKADDR*)&(All_Sock.remoteaddr_v4), sizeof(SOCKADDR_IN));
          DisplayText_Acc("[UDPv4] 클라이언트 종료: %s\n", ipaddr);
        }
        else {
          WSAAddressToString((SOCKADDR*)ptr->sockaddrv6, sizeof(SOCKADDR_IN6),
            NULL, ipaddr, &ipaddrlen);

          retval = sendto(All_Sock.udp_send_v6, (char*)&endMsg, BUFSIZE, 0,
            (SOCKADDR*)&(All_Sock.remoteaddr_v6), sizeof(SOCKADDR_IN6));
          DisplayText_Acc("[UDPv6] 클라이언트 종료: %s\n", ipaddr);
        }

      }

      if (prev)
        prev->next = ptr->next;
      else
        AllSocketInfoArray = ptr->next;
      delete ptr;
      break;

    }
    prev = ptr;
    ptr = ptr->next;
    i++;
  }

  nALLSockets--;
  resetUserCount();
  updateUserList();

}
```

3) EditControll 출력 함수

1. 타입에 따라서 출력되는 EditControll 위치가 다르다
   → type = ACCESS 일 경우 DisplayText_ACC
   → type = CHATTING | FILEINIT일 경우 DIsplayText_Send

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

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

  int nLength = GetWindowTextLength(hEdit_User_Send);
  SendMessage(hEdit_User_Send, EM_SETSEL, nLength, nLength);
  SendMessage(hEdit_User_Send, 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(hEdit_Serv_Send);
  SendMessage(hEdit_Serv_Send, EM_SETSEL, nLength, nLength);
  SendMessage(hEdit_Serv_Send, EM_REPLACESEL, FALSE, (LPARAM)cbuf);
  va_end(arg);
}
```

4) 사용자 정의함수

4) void resetUserCount()

1. 현재 접속하고 있는 유저의 리스트 수(현재 유지하고 있는 연결리스트 노드수)로 최신화 시키는 함수

4) char* listupText(char* fmt, ...)

1. ListBox에 추가할 아이템의 문자열을 반환 하는 함수 (updateUserList()에서 호출함)

```
void resetUserCount() {
  char count[4];
  ZeroMemory(count, 4);
  itoa(nALLSockets, count, 10);
  SendMessage(hUserCount, EM_SETSEL, 0, 4);
  SendMessage(hUserCount, WM_CLEAR, 0, 0);
  SendMessage(hUserCount, EM_REPLACESEL, FALSE, (LPARAM)count);
}

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

  char cbuf[256];
  vsprintf(cbuf, fmt, arg);

  return (char*)cbuf;
}
```

4) void selectedUser(char* selectedItem, int index)

1. ListBox에서 아이템을 선택하고 해당 아이템의 값은 인덱스와 문자열로 이루어져있다.
2. 해당 아이템에서 문자열을 가져와서 정해둔 기준으로 토큰화를 시켜 값을 나누고 추방 란에 유저의 정보를 출력한다.

```
void selectedUser(char* selectedItem, int index) {
  char username[20];
  char ipaddr[65];
  char TorU[8];

  char* ptr = strtok(selectedItem, "\t");
  strncpy(username, ptr, 20);

  ptr = strtok(NULL, "\t");
  strncpy(ipaddr, ptr, 65);
  ipaddr[64] = NULL;

  ptr = strtok(NULL, "\t");
  strncpy(TorU, ptr, 8);

  SendMessage(hUserNames, EM_SETSEL, 0, 20);
  SendMessage(hUserNames, WM_CLEAR, 0, 0);
  SendMessage(hUserNames, EM_REPLACESEL, FALSE, (LPARAM)username);

  SendMessage(hUserAddrs, EM_SETSEL, 0, 65);
  SendMessage(hUserAddrs, WM_CLEAR, 0, 0);
  SendMessage(hUserAddrs, EM_REPLACESEL, FALSE, (LPARAM)ipaddr);

  SendMessage(hUserTCPorUDP, EM_SETSEL, 0, 8);
  SendMessage(hUserTCPorUDP, WM_CLEAR, 0, 0);
  SendMessage(hUserTCPorUDP, EM_REPLACESEL, FALSE, (LPARAM)TorU);

}
```

4) void updateUserList()

1. ListBox안의 내용을 모두 초기화 시키고 현재 유지되고 있는 연결리스트를 차례로 순회하면서 들어온 순으로 ListBox를 최신화한다.
2. 이때 ListBox에 추가되는 아이템들은 모두 문자열이어야 하기 때문에 정해둔 기준으로 문자열을 만들어서 아이템을 추가한다.

```
void updateUserList() {
  SOCKETINFO_UDPnTCP* ptr = AllSocketInfoArray;
  char* ipaddrv4 = (char*)malloc(INET_ADDRSTRLEN);
  char* ipaddrv6 = (char*)malloc(INET6_ADDRSTRLEN);
  DWORD ipaddr6len = INET6_ADDRSTRLEN, ipaddr4len = INET_ADDRSTRLEN;

  SendMessage(hUserList, LB_RESETCONTENT, 0, 0);

  char* listupMsg = (char*)malloc(256);
  int i = 1;
  while (ptr != NULL) {
    if (ptr->isIPv6 == false) {
      WSAAddressToString((SOCKADDR*)ptr->sockaddrv4, sizeof(*ptr->sockaddrv4), NULL, ipaddrv4, &ipaddr4len);
      strncpy(listupMsg,
        listupText("%s\t%s \t%s\t(%d)\r", ptr->client_id, ipaddrv4, ptr->socktype, ptr->clientUniqueID),
        256);
    }
    else {
      WSAAddressToString((SOCKADDR*)ptr->sockaddrv6, sizeof(*ptr->sockaddrv6), NULL, ipaddrv6, &ipaddr6len);
      strncpy(listupMsg,
        listupText("%s\t%s \t%s\t(%d)\r", ptr->client_id, ipaddrv6, ptr->socktype, ptr->clientUniqueID), // socckaddr_port,
        256);
    }
    SendMessage(hUserList, LB_ADDSTRING, 0, (LPARAM)listupMsg);
    ptr = ptr->next;
  }
}
```

5) 오류 함수

1.

```
// 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);
```

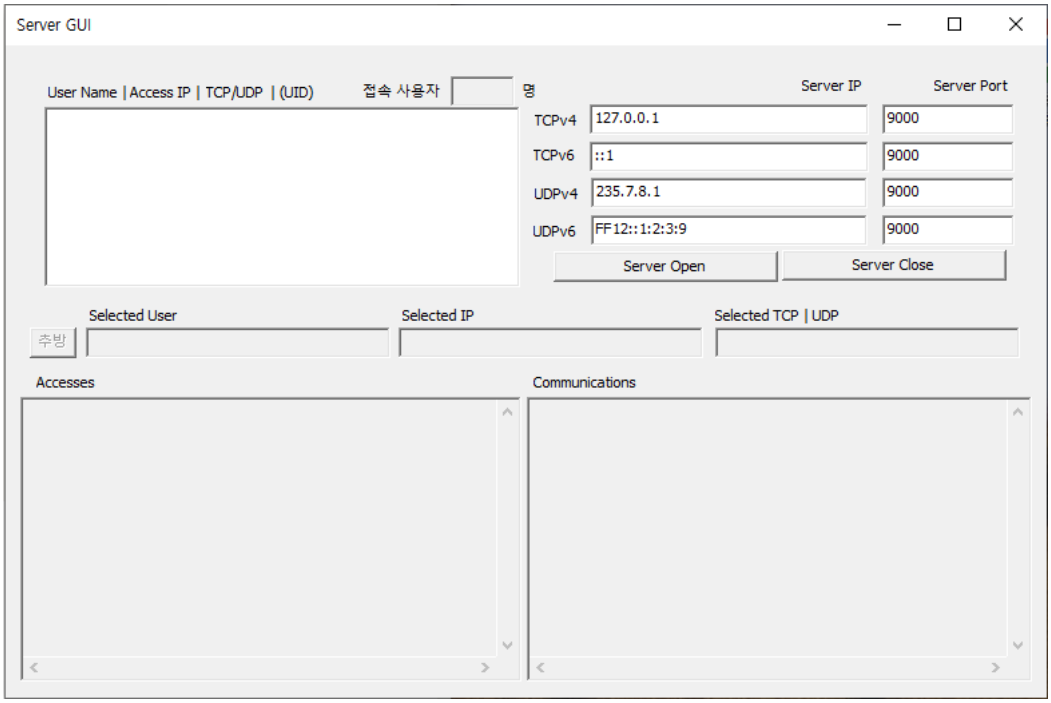
```
  exit(1);
}

// 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);
  DisplayText_Acc("[%s] %s", msg, (char*)lpMsgBuf);
  LocalFree(lpMsgBuf);
}
```

4. 최종 결과 화면

1. **프로그램 시작**

   → 현재 서버는 소켓은 시작되지 않은 상태

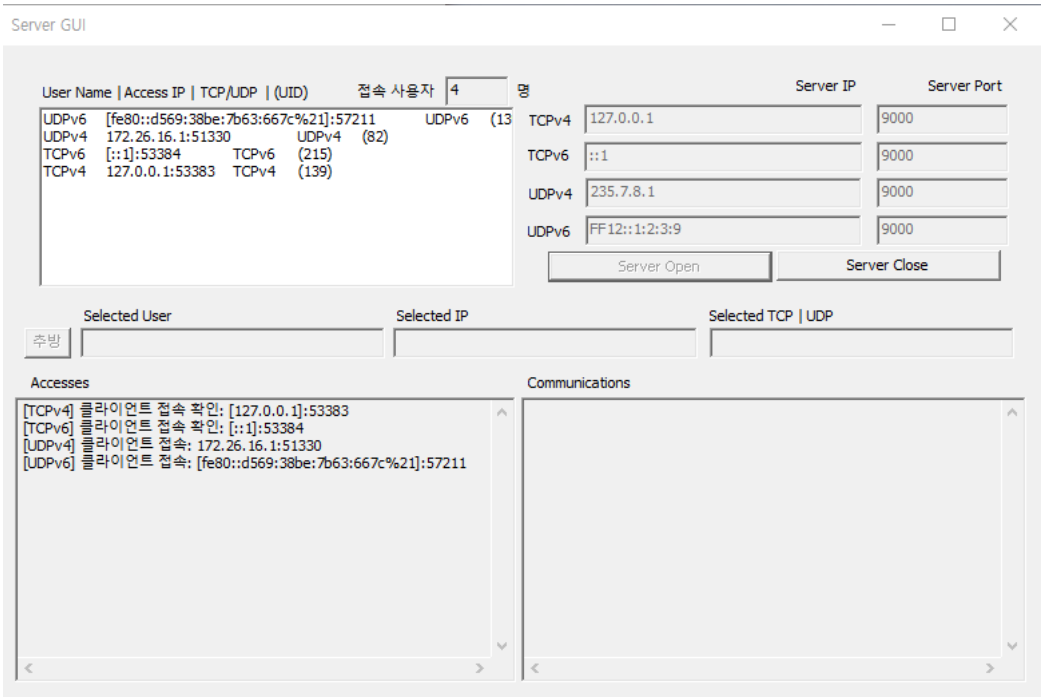


2. **서버 시작**

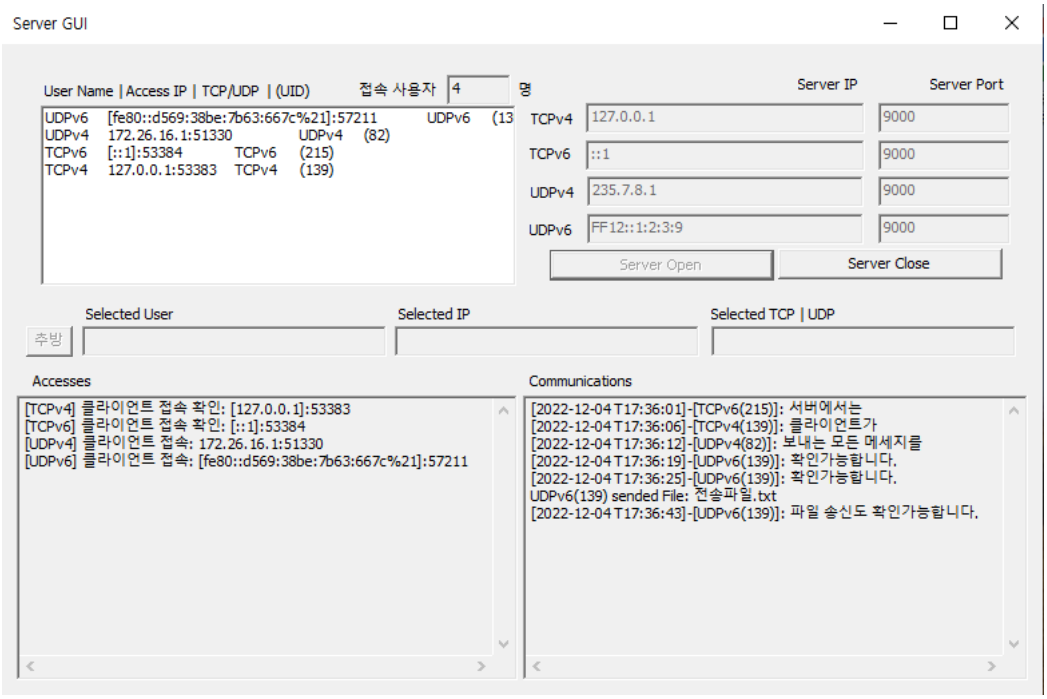
   → 서버 Open버튼 클릭

   a. 주소 EditControll 및 ServerOpen 버튼 비활성화

3. **클라이언트 접속 시 상태**
   → 들어온 클라이언트의 접속 정보를 ListBox에 유지시켜주고 현재 인원 수 또한 유지한다.

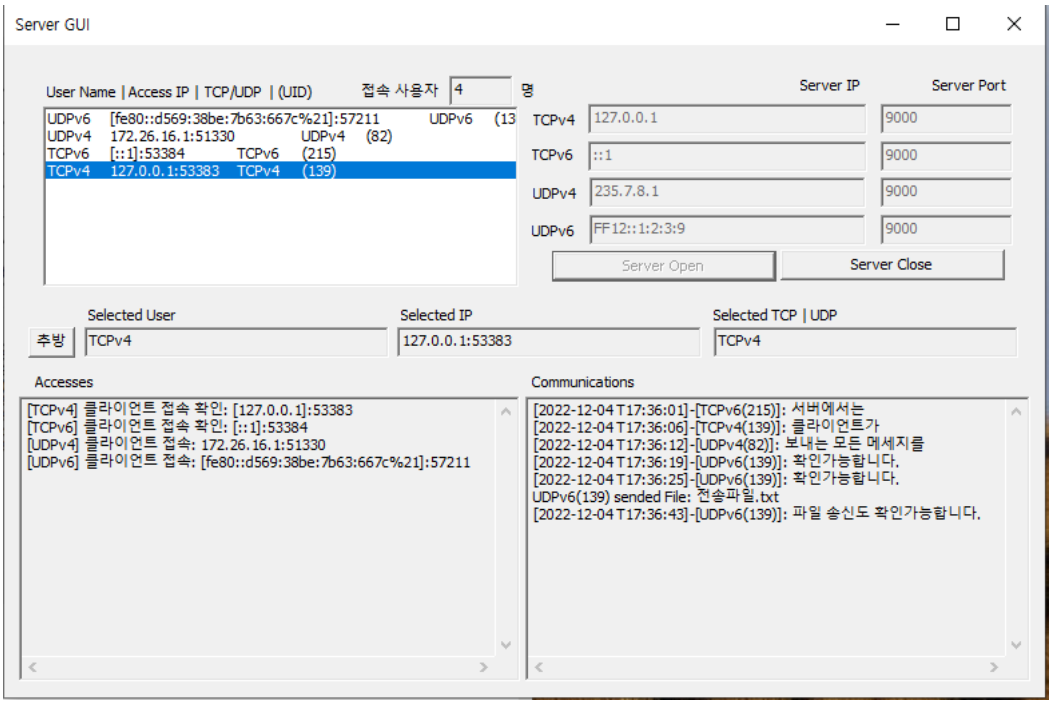


4. **클라이언트간 채팅 | 파일 메세지 송신 확인**
   → 클라이언트가 메세지를 보낸시간,
   → 채팅 메세지내용, 파일 이름까지 모두 확인이 가능합니다.

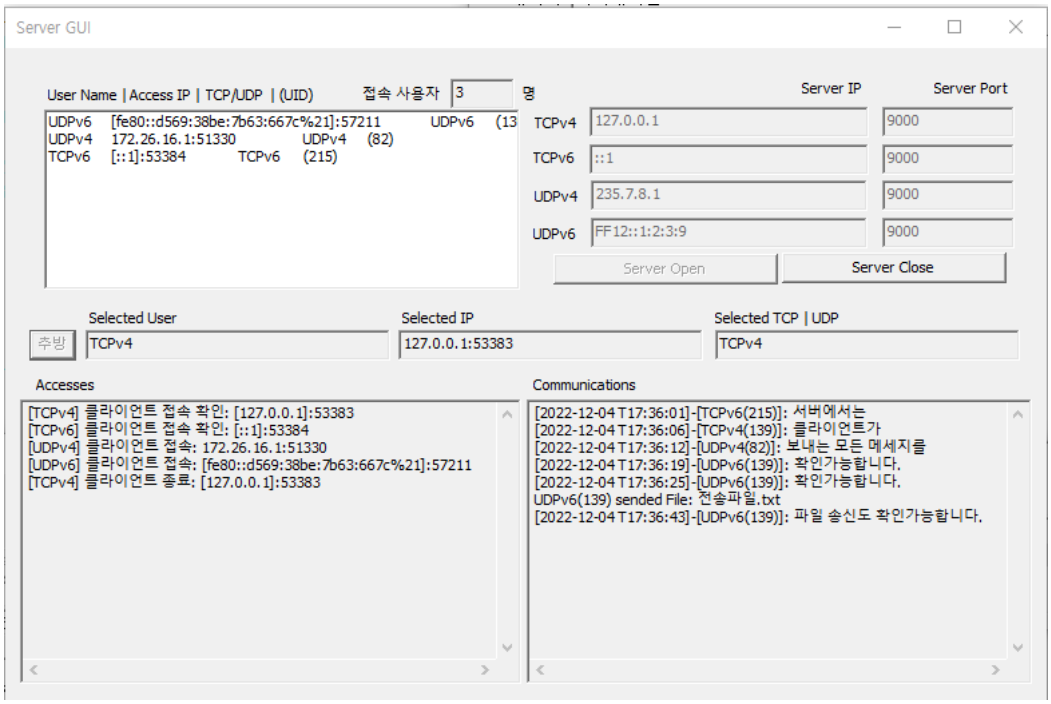


5. **클라이언트 추방 기능**

a. 추방을 원하는 클라이언트 ListBox에서 선택
→ 선택한 클라이언트의 정보가 추방 란에 출력되는 것을 볼 수 있습니다.



b. 추방버튼 클릭
→ 클라이언트 유저 목록에서 해당 클라이언트의 정보가 사라지고, Accesses 란에 종료되었다는 메세지도 출력됩니다.



c. 추방 당한 클라이언트 화면

→ 클라이언트는 해당 알림메세지가 나오고 더이상 채팅을 할수 없는 상태가 됩니다.