데이터 통신

- Chatting & File Transfer -

제출일	17.04.04
분 반	01
조	4조
조장	김규태
조원	신종욱
	윤동현
	이상인
	정석현
	허원철

-목 차-

1 실습 개요

-실습 일시 및 장소

실습 일시: 4월14일~4월18일

실습 장소: 학생생활관,카페 및 공대5호관

-실습 목적

네트워크를 통한 다른 시스템의 process 간 통신 실제 Network (Ethernet Protocol)을 이용해서 chatting 과 File 전송을 동시에 할 수 있는 Application 구현

-실습 시나리오

두 개의 PC에서 각각 프로그램을 실행

두 대의 PC는 네트워크로 연결

PC 1의 프로그램에서 전송 버튼을 클릭

PC 2의 프로그램에서 Ethernet frame을 수신

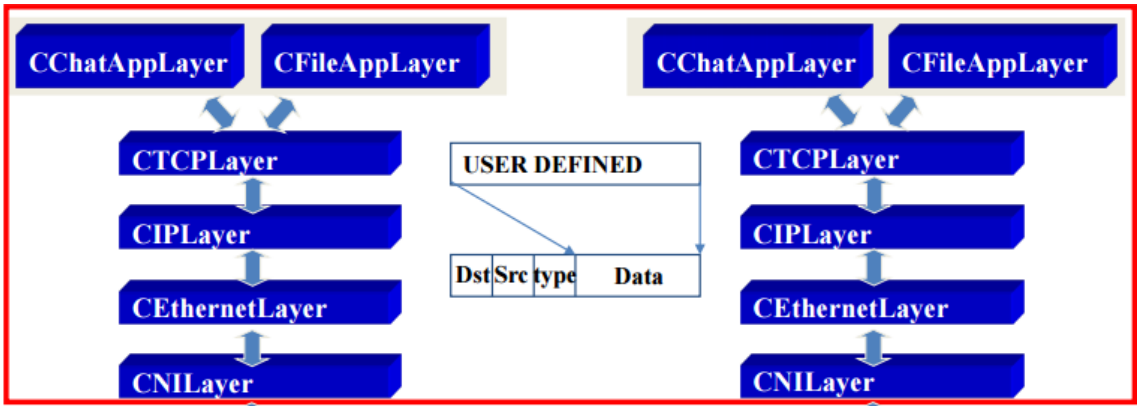
전달된 결과를 화면에 팝업 창으로 출력

채팅 전송 후 파일 전송 버튼을 클릭

전달된 결과를 화면에 팝업 창으로 출력

2 프로토콜 스택

-구조 설명



ChatAppLayer

채팅메시지에 대해서 Fragmentation을 해서 길이에 제한 없이 전송하도록 구현

전송 시 데이터를 일정 크기로 잘라서 하나씩 전송한다.

capp_type에 메시지 데이터가 첫 부분(0x00)인지, 중간 부분 (0x01) 인지, 마지막 부분 (0x02) 인지 구분하여 데이터를 전송 수신 시 데이터가 첫 부분 일 경우 그 크기만큼 버퍼 할당 데이터가 중간 부분인 경우 계속 버퍼에 저장 데이터가 마지막 부분일 경우 버퍼에 저장된 메시지 전체를 Dialog 객체 에게 전달한다. (상위 레이어로 전달)

TCP Layer

TCP 헤더의 destination port number를 확인하여, Chatting Application Layer 나
File Transfer Application Layer로 구분해서 전달 메시지를 보낼 때는 ChatAppLayer에서 header 에 메시지(data)넣어서 IPLayer에게 전달하고 받을 때는 IPLayer로부터 받은 데이터 중에서 TCPLayer의 header를 분리하고 상위 계층인 ChatAppLayer로 데이터를 전달을 한다.

IP Layer

IP 헤더에 필요한 정보를 담아서 송신 수신 된 패킷의 source IP가 자신의 것이면, 버리고 수신 된 패킷의 destination IP가 자신의 것이면 TCP Layer로 데이터를 전달하고 아니면 버린다.

Ethernet Layer

수신 받은 프레임 중에서 destination address가 자신의 주소가 아니면 버린다
헤더를 제외한 데이터 부분은 IPLayer로 전송

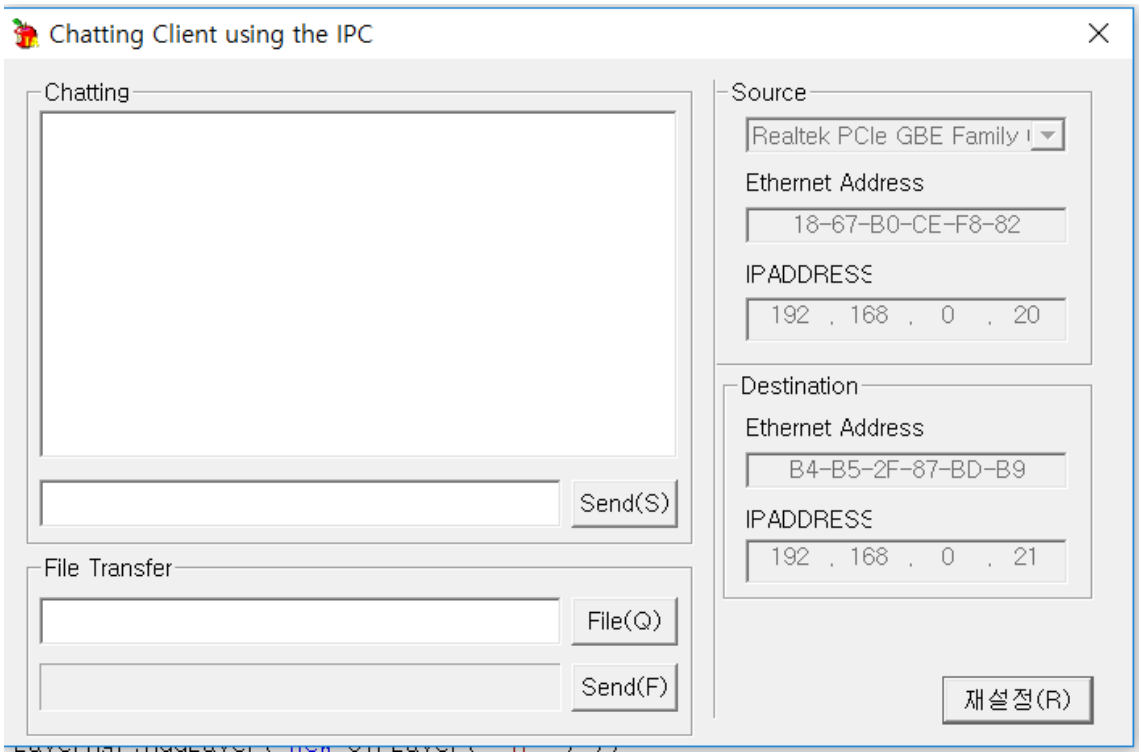
NILayer

WinPcap을 이용하여 기본적인 packet 송수신 operation 구현한 class
Adapter와 상위 Layer간의 데이터 송수신의 중간자 역할을 담당
송신일 때는 다른 PC 에 패킷을 전송하고 수신일 때는 받은 패킷들을 상위계층으로 보낸다.

3 구현설명

IPCAPPDlg.cpp

프로그램 실행 시 보이는 프로그램의 동작과 관련된 부분으로 Mac,ip를 설정하는 입력 창 을 제공하며 입력된 내용을 각 layer의 주소로 설정한다. 또한 layer들을 이용하여 파일 또는 메시지를 전송하며 전송되는 내용들을 사용자에게 화면을 통하여 알리거나 표시해 준다.



IPCAPPdlg는 리소스 파일에 의해 생성된 이러한 화면에 id를 이용하여 내부적으로 설정하고 버튼을 누르거나 text창에 입력된 내용을 받아오는 등의 이벤트 처리에 관여한다.

세부 구현

```
void CIPCAppDlg::OnSendMessage()
{
   //send버튼을 누를 경우 발생되는 함수로 SendData()를 이용하여 데이터를 전송한다.
   UpdateData( TRUE ) ;
   if ( !m_stMessage.IsEmpty() )
   {
      SendData( ) ;//실제로 data를 전송한다.
      m_stMessage = "" ;//전송후 입력칸을 초기화 한다.
     //커서를 입력칸으로 설정한다.
      (CEdit*) GetDlgItem( IDC_EDIT_MSG )->SetFocus( );
   }
   UpdateData( FALSE ) ;
}
```

Send버튼을 눌렀을 때 실행되는 것으로 m_stMessage즉 입력한 문자가 없을 경우 를 제외하고 실행되며 SendData를 실행하여 메시지를 전송한다. 메시지 전송후 입력된 메시지를 초기화하며

```
void CIPCAppDlg::SendData()
{   //실제로 data를 전송하는 함수
   CString MsgHeader ; //메시지의 앞부분을 설정한다.
   MsgHeader.Format( "[Src:Dst] " ) ;
   //header와 실제 전송하려는 메시지를 합쳐 m_ListChat에 추가한다.
   m_ListChat.AddString( MsgHeader + m_stMessage ) ;
   // 입력한 메시지를 파일로 저장
   int nlength = m_stMessage.GetLength();
   unsigned char* ppayload = new unsigned char[nlength+1];
   memcpy(ppayload,(unsigned char*)(LPCTSTR)m_stMessage,nlength);
   ppayload[nlength] = '₩0';
   // TCPLayer Port를 설정
   m_TCP->SetDestinPort(TCP_PORT_CHAT);
   // 보낼 data와 메시지 길이에 1을 더하여 Send함수로 넘겨준다.
   m_ChatApp->Send(ppayload,m_stMessage.GetLength()+1);
   //ListChat의 가로 스크롤을 추가한다.
   EditScroll(m_ListChat,( MsgHeader + m_stMessage ));
}
```

커서를 메시지 입력 칸으로 설정한다.

입력한 메시지를 보내는 함수로서 사용자가 send 버튼을 눌렀을 때 OnSendMessage() 함수 내부에서 실행된다. MsgHeader를 생성하고 이곳에 받는 곳 과 보내는 곳의 주소를 표현해 주며 실제 보내는 메시지인 m_stMessage와 합쳐 m_ListChat에 저장하며 이 CString의 길이와 payload를 설정하여 chatAppLayer의 send로 넘겨준다.

```
void CIPCAppDlg::OnButtonAddrSet()
{   //설정 버튼을 누를 경우 발생하는 이벤트이다.
   UpdateData( TRUE ) ;
   //ip주소를 저장하기 위한 변수를 생성한다.
   unsigned char src_ip[4];
   unsigned char dst_ip[4];
   if ( !m_unDstEnetAddr |||m_unSrcEnetAddr  )
   {   //잘못된 주소가 입력될경우 경고창을 띄운다.
      MessageBox( "주소를 설정 오류발생","경고",MB_OK | MB_ICONSTOP ) ;
      return ;
   }
   unsigned char srcMAC[12], dstMAC[12];
   if ( m_bSendReady ){
      //기존의 상태가 sendReady상태일경우 주소를 재설정하기 위한 상태로 변경한다.
      SetDlgState( IPC_ADDR_RESET ) ;
      SetDlgState( IPC_INITIALIZING ) ;
   }
   else{
      //입력된 ip주소를 받아와 저장한다.
      m_unSrcIPAddr.GetAddress(src_ip[0],src_ip[1],src_ip[2],src_ip[3]);
      m_unDstIPAddr.GetAddress(dst_ip[0],dst_ip[1],dst_ip[2],dst_ip[3]);
      //입력된 ip를 이용하여 iplayer의 ip를 설정한다.
      m_IP->SetSrcIPAddress(src_ip);
      m_IP->SetDstIPAddress(dst_ip);
      //미리 설정한 이더넷 주소를 얻어온다.
      sscanf(m_unSrcEnetAddr, "%02X-%02X-%02X-%02X-%02X-%02X", &srcMAC[0], &srcMAC[1], &srcMAC[2], &srcMAC[3], &srcMAC[4], &srcMAC[5]);
      sscanf(m_unDstEnetAddr, "%02X-%02X-%02X-%02X-%02X-%02X", &dstMAC[0], &dstMAC[1], &dstMAC[2], &dstMAC[3], &dstMAC[4], &dstMAC[5]);

      ((CEthernetLayer *)m_LayerMgr.GetLayer("Ethernet"))->SetEnetSrcAddress(srcMAC);
      ((CEthernetLayer *)m_LayerMgr.GetLayer("Ethernet"))->SetEnetDstAddress(dstMAC);

      int nIndex = m_ComboEnetName.GetCurSel();
      m_NI->SetAdapterNumber(nIndex);
     //패킷 시작
      m_NI->PacketStartDriver();
     //Dlg의 상태를 변경한다.
      SetDlgState( IPC_ADDR_SET ) ;
      SetDlgState( IPC_READYTOSEND ) ;
   }
   //설정 또는 재설정시 상태를 지금상태와 반대로 한다.
   m_bSendReady = !m_bSendReady ;
}
```

입력된 ip 주소와 미리 설정한 mac주소를 가져와 ip주소와 mac주소를 설정한다. 입력된 주소의 오류 여부를 검사하며 오류가 있을 경우 오류메시지박스를 띄워준다. sendReady여부에 따라 만약 ready상태이면 재설정하기위한 상태로 변경하며 ready상태가 아닐 경우 입력된 ip주소를 받아와 설정하며 자신이 선택한 통신 방법에 따라 NILayer의 adapterNumber를 설정한다. NILayer의 패킷을 실행시키고 상태들을 전송이 가능한 상태로 변화하며 sendReday는 지금과 반대의 상태로 지정한다.

```
CIPCAppDlg::CIPCAppDlg(CWnd* pParent /*=NULL*/)
   : CDialog(CIPCAppDlg::IDD, pParent),
   CBaseLayer( "ChatDlg" ),
   m_bSendReady(FALSE)
{
   //자신의 Mac주소를 직접 설정함
   m_unSrcEnetAddr = "18-67-B0-CE-F8-82";
   m_unDstEnetAddr = "B4-B5-2F-87-BD-B9";
   //보내려는 message내용과 file을 초기화시킴
   m_stMessage = _T("");
   m_filePath = _T("");

   m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
   //LayerManager에 각 layer를 추가 함
   m_LayerMgr.AddLayer( this ) ;
   m_LayerMgr.AddLayer( new CNILayer( "NI" ) ) ;
   m_LayerMgr.AddLayer( new CEthernetLayer( "Ethernet" ) ) ;
   m_LayerMgr.AddLayer( new CIPLayer( "IP" ) );
   m_LayerMgr.AddLayer( new CTCPLayer( "TCP" ) );
   m_LayerMgr.AddLayer( new CChatAppLayer( "ChatApp" ) ) ;
   m_LayerMgr.AddLayer( new CFileAppLayer( "FileApp" ) ) ;
   ////////////////////////// fill the blank ///////////////////////////////
   // 레이어를 연결한다. (레이어 생성) layer의 계층을 연결해준다.
   m_LayerMgr.ConnectLayers("NI ( *Ethernet ( *IP ( *TCP ( *FileApp ( *ChatDlg ) *ChatApp ( *ChatDlg ) ) ) ) )" );
   ////////////////////////////////////////////////////////////////////////
   //생성된 layer를 받아옴
   m_ChatApp = (CChatAppLayer*)mp_UnderLayer ;
   m_TCP = (CTCPLayer *)m_LayerMgr.GetLayer("TCP");
   m_IP = (CIPLayer *)m_LayerMgr.GetLayer("IP");
   m_NI = (CNILayer *)m_LayerMgr.GetLayer("NI");
}
```

Mac주소를 미리 설정하여 통신에 이용하며 메시지와 보내려는 파일의 경로를 초기화 한다. 이후 layerManager에 각 layer들을 추가하며 계층을 설정하여 각 layer를 이어주고 이어진 layer들을 다시 받아온다.

```
void EditScroll(CListBox& listbox, CString message){
    //입력되는 문자를 모두 볼수 있도록 가로 스크롤을 추가한다.
    CSize size;
    CDC *pDC = listbox.GetDC();
     // 현재 리스트박스의 폰트를 얻어와 DC에 적용시킨다.
     CFont* pOld = pDC->SelectObject(listbox.GetFont());
    size = pDC->GetTextExtent(message);
     size.cx += 3;
     pDC->SelectObject(pOld);
     listbox.ReleaseDC(pDC);
     // 구한 문자열의 Pixel 단위를 넘긴다.
    if (listbox.GetHorizontalExtent() < size.cx)
     {   //이전 listbox보다 클경우 재설정한다.
       listbox.SetHorizontalExtent(size.cx);
     }
}
```

입력되는 문자의 길이가 표시할 수 있는 칸을 넘어갈 경우 스크롤 바를 생성하여 보낸/보내는 메시지를 전부 확인 가능하게 스크롤 바를 생성해주는 함수로서 기존의 스크롤 바 크기와 입력된 메시지를 표현하는 스크롤 바의 크기를 비교하여 더 큰 쪽으로 스크롤 바의 크기를 설정한다.

EthernetLayer.CPP

```
void CEthernetLayer::ResetHeader( )
{
    memset( m_sHeader.enet_dstaddr.S_un.s_ether_addr, 0, 6 ) ;
    m_sHeader.enet_dstaddr.S_un.s_un_byte.e0=0x00;
    m_sHeader.enet_dstaddr.S_un.s_un_byte.e1=0x00;
    m_sHeader.enet_dstaddr.S_un.s_un_byte.e2=0x00;
    m_sHeader.enet_dstaddr.S_un.s_un_byte.e3=0x00;
    m_sHeader.enet_dstaddr.S_un.s_un_byte.e4=0x00;
    m_sHeader.enet_dstaddr.S_un.s_un_byte.e5=0x00;

    memset( m_sHeader.enet_srcaddr.S_un.s_ether_addr, 0, 6 ) ;
    m_sHeader.enet_srcaddr.S_un.s_un_byte.e0=0x00;
    m_sHeader.enet_srcaddr.S_un.s_un_byte.e1=0x00;
    m_sHeader.enet_srcaddr.S_un.s_un_byte.e2=0x00;
    m_sHeader.enet_srcaddr.S_un.s_un_byte.e3=0x00;
    m_sHeader.enet_srcaddr.S_un.s_un_byte.e4=0x00;
    m_sHeader.enet_srcaddr.S_un.s_un_byte.e5=0x00;

    memset( m_sHeader.enet_data, 0, ETHER_MAX_DATA_SIZE ) ;
    m_sHeader.enet_type = 0x3412 ;
}
```

Header 부분을 초기화 하는 부분이다.

```
BOOL CEthernetLayer::Send(unsigned char *ppayload, int nlength)
{
    memcpy( m_sHeader.enet_data, ppayload, nlength ) ;

    BOOL bSuccess = FALSE ;
    bSuccess = mp_UnderLayer->Send((unsigned char*) &m_sHeader,nlength+ETHER_HEADER_SIZE);

    return bSuccess ;
}

BOOL CEthernetLayer::Receive( unsigned char* ppayload )
{

    LPETHERNET pFrame = (LPETHERNET) ppayload ;

    BOOL bSuccess = FALSE ;

    if(ntohs(pFrame->enet_type) == 0x1234&&(memcmp((const char*)pFrame->enet_dstaddr.S_un.s_ether_addr,(const char*)m_sHeader.enet_srcaddr.S_un.s_ether_addr,6) == 0)){
        bSuccess = mp_aUpperLayer[0]->Receive((unsigned char*) pFrame->enet_data);
    }
    return bSuccess ;
}
```

Send함수는 ppayload를 하위 레이어로 보내고

Receive함수는 enet_type를 ntohs함수를 이용해서 네트워크 바이트오더를 호스트 바이트오더로 바꾼다음 확인하고 받은 도착주소의 맥주소와 현재 헤더의 맥주소를 비교해서 같다면 UpperLayer로 Recevie해준다.

IPLayer.cpp

```
void CIPLayer::ResetHeader( )
{
     m_sHeader.ip_verlen = 0x00;
     m_sHeader.ip_tos = 0x00;
     m_sHeader.ip_len = 0x0000;
     m_sHeader.ip_id = 0x0000;
     m_sHeader.ip_fragoff = 0x0000;
     m_sHeader.ip_ttl = 0x00;
     m_sHeader.ip_proto = 0x00;
     m_sHeader.ip_cksum = 0x00;
     memset( m_sHeader.ip_src, 0, 4);
     memset( m_sHeader.ip_dst, 0, 4);
     memset( m_sHeader.ip_data, 0, IP_DATA_SIZE);
}
```

IP의 헤더정보를 초기화 한다.

```
BOOL CIPLayer::Send(unsigned char* ppayload, int nlength)
{
     memcpy( m_sHeader.ip_data, ppayload, nlength ) ;

     BOOL bSuccess = FALSE ;
     bSuccess = mp_UnderLayer->Send((unsigned char*)&m_sHeader,nlength+IP_HEADER_SIZE);

     return bSuccess;
}

BOOL CIPLayer::Receive(unsigned char* ppayload)
{
     PIPLayer_HEADER pFrame = (PIPLayer_HEADER) ppayload ;

     BOOL bSuccess = FALSE ;


     if(memcmp((char *)pFrame->ip_dst,(char *)m_sHeader.ip_src,4) ==0 &&
          memcmp((char *)pFrame->ip_src,(char *)m_sHeader.ip_src,4) !=0 &&
          memcmp((char *)pFrame->ip_src,(char *)m_sHeader.ip_dst,4) ==0 )
     {
          bSuccess = mp_aUpperLayer[0]->Receive((unsigned char*)pFrame->ip_data);
     }
     return bSuccess ;
}
```

Send함수는 하위레이어로 현재 정보를 넘겨주고

Receive함수는 받은 IP주소의 정보와 내가가진 IP주소를 비교하여 조건에 알맞은지 확인후에 맞는지 확인후 UpperLayer의 receive를 불러온다

TCPLayer.cpp

```
void CTCPLayer::ResetHeader( )
{
    m_sHeader.tcp_sport = 0x0000;
    m_sHeader.tcp_dport = 0x0000;
    m_sHeader.tcp_seq = 0x00000000;
    m_sHeader.tcp_ack = 0x00000000;
    m_sHeader.tcp_offset = 0x00;
    m_sHeader.tcp_flag = 0x00;
    m_sHeader.tcp_window = 0x0000;
    m_sHeader.tcp_cksum = 0x0000;
    m_sHeader.tcp_urgptr = 0x0000;
    memset(m_sHeader.tcp_data,0,TCP_DATA_SIZE);
}
```

헤더 부분을 초기화한다

여기서 dport는 도착해야할 FIle인지 Chat인지 확인하는 변수이다

```
BOOL CTCPLayer::Send(unsigned char* ppayload, int nlength)
{
    memcpy( m_sHeader.tcp_data, ppayload, nlength ) ;

    BOOL bSuccess = FALSE ;
    bSuccess = mp_UnderLayer->Send((unsigned char*)&m_sHeader,nlength+TCP_HEADER_SIZE);

    return bSuccess;
}

BOOL CTCPLayer::Receive(unsigned char* ppayload)
{
    PTCPLayer_HEADER pFrame = (PTCPLayer_HEADER) ppayload ;

    BOOL bSuccess = FALSE;

    if(pFrame->tcp_dport == TCP_PORT_CHAT){ // 채팅 응용 프로토콜로부터 받은 데이터일 경우
        bSuccess = mp_aUpperLayer[1]->Receive((unsigned char*)pFrame->tcp_data);
    }
    else if(pFrame->tcp_dport == TCP_PORT_FILE){ // 파일 전송 응용 프로토콜로부터 받은 데이터인 경우
        bSuccess = mp_aUpperLayer[0]->Receive((unsigned char*)pFrame->tcp_data);
    }

    return bSuccess ;
}
```

send함수는 하위레이어로 send함수를 호출한다

Receive함수는 받은 tcp.dport의 값을 확인한 다음 적절한 상위 레이어로 보낸다

chatappLayer.cpp

```
BOOL CChatAppLayer::Send(unsigned char *ppayload, int nlength)
{
    // ppayload : 입력한 문자열 , nlength : 문자의 길이
    m_ppayload = ppayload;
    m_length = nlength;

    if(nlength <= APP_DATA_SIZE){ //문자의 길이가 app_data_size 보다 작을경우
        m_sHeader.capp_totlen = nlength;
        memcpy(m_sHeader.capp_data,ppayload,nlength); // GetBuff에 data 복사
        mp_UnderLayer->Send((unsigned char*) &m_sHeader,nlength+APP_HEADER_SIZE);//하위 계층으로 넘긴다
    }else{
        AfxBeginThread(ChatThread,this);//클경우 쓰레드 이용한다.
    }

    return TRUE;
}
```

App_data_size에 따라 그냥보낼지 쓰레드를 보낼지 선택한다

```
BOOL CChatAppLayer::Receive( unsigned char* ppayload )
{
    // ppayload를 ChatApp 헤더 구조체로 넣는다.
    LPCHAT_APP capp_hdr = (LPCHAT_APP) ppayload ;
    int length = capp_hdr->capp_totlen-1;
    static unsigned char *GetBuff; // 데이터를 쌓을 GetBuff를 선언한다.

    if(length <= APP_DATA_SIZE){
        GetBuff = (unsigned char *)malloc(length);//버퍼 생성
        memset(GetBuff,0,length);  // GetBuff를 초기화해준다.
        memcpy(GetBuff,capp_hdr->capp_data,length); // GetBuff에 data 복사
        GetBuff[length] = '₩0';

        mp_aUpperLayer[0]->Receive((unsigned char*) GetBuff); // 상위 계층으로 데이터 올림

        return TRUE;
    }
    if(capp_hdr->capp_type == DATA_TYPE_BEGIN)//데이터 첫부분일때
    {
        GetBuff = (unsigned char *)malloc((length)); //버퍼를 생성한다.
        memset(GetBuff,0,(length)); //버퍼에 데이터를 쓴다.
    }
    else if(capp_hdr->capp_type == DATA_TYPE_CONT) // 데이터 중간부분일때

    {
        strncat((char *)GetBuff,(char *)capp_hdr->capp_data,strlen((char *)capp_hdr->capp_data));
        GetBuff[strlen((char *)GetBuff)] = '₩0';
    }
    else if(capp_hdr->capp_type == DATA_TYPE_END) //데이터 마지막 부분
    {
        memcpy(GetBuff,GetBuff,(length)); //버퍼에 length 길이만큼의 버퍼를 복사한다
        GetBuff[(length)] = '₩0';

        mp_aUpperLayer[0]->Receive((unsigned char*) GetBuff);// 상위 계층으로 데이터 올림
        free(GetBuff);// 버퍼 해제
    }
    else
        return FALSE;
```

receive함수는 쓰레드로 받을경우에는 데이터의 첫부분과 중간부분,마지막부분으로 나눠서 받는데 마지막 부분을 받을땐 상위계층으로 recevie함수를 호출한다

```
UINT CChatAppLayer::ChatThread( LPVOID pParam )
{

   BOOL bSuccess = FALSE;
   CChatAppLayer *Capp = (CChatAppLayer *)pParam;
   int data_length = APP_DATA_SIZE;
   int seq_tot_num;
   int data_index;
   int temp = 0;

   if( Capp->m_length < APP_DATA_SIZE ) //capp 길이가 app_data_size 보다 작을경우
      seq_tot_num = 1;
   else //클경우
      seq_tot_num = (Capp->m_length/APP_DATA_SIZE)+1;

   for(int i=0;i<=seq_tot_num+1;i++)
   {
      if(seq_tot_num==1){ //작을경우
         data_length = Capp->m_length;
      }
      else{
         if(i==seq_tot_num)
            data_length = Capp->m_length%APP_DATA_SIZE;
         else
            data_length = APP_DATA_SIZE;
      }

     memset(Capp->m_sHeader.capp_data, 0, data_length);
```

작을경우에는 한 개로 크기가 클경우에는 쪼개서 보낸다.

```
    memset(Capp->m_sHeader.capp_data, 0, data_length);

     if(i==0) // 처음부분
     {
        Capp->m_sHeader.capp_totlen = Capp->m_length;
        Capp->m_sHeader.capp_type = DATA_TYPE_BEGIN;
        memset(Capp->m_sHeader.capp_data,0,data_length);
        data_length = 0;
     }
     else if(i!=0 && i<=seq_tot_num)  // 중간부분
     {
        data_index = data_length;
        Capp->m_sHeader.capp_type = DATA_TYPE_CONT;
        Capp->m_sHeader.capp_seq_num = i-1;

        CString str = Capp->m_ppayload;
        str = str.Mid(temp,temp+data_index);

        memcpy(Capp->m_sHeader.capp_data,str,data_length);
        temp += data_index;
     }
     else // 마지막부분
     {
        Capp->m_sHeader.capp_type = DATA_TYPE_END;
        memset(Capp->m_ppayload,0,data_length);
        data_length = 0;
     }
     ((CTCPLayer*) (Capp->GetUnderLayer()))->SetDestinPort(TCP_PORT_CHAT);//tcp layer 로 내려보내 TCPLayer Port를 설정
      bSuccess = Capp->mp_UnderLayer->Send((unsigned char*) &Capp->m_sHeader,data_length+APP_HEADER_SIZE);
  }

  return bSuccess;
}
```

쓰레드로 받으면서 처음,중간,마지막받는 과정에서 중간부분일경우는 계속 받다가 마지막 부분을 받을 때 tcp port를 설정해서 하위 레이어에 send함수를 호출한다

FileAppLayer

FileThread :

변수 :

dwFileSize : 파일 전체의 크기

tot_seq_num : 파일이 MAX_FILE_SIZE보다 클 경우 나누어지는 파일 조각의 개수

fapp_hdr : CFileAppLayer전체에 해당하는 객체

m_sHeader : FileAppLayer.h의 구조체에서 선언한 헤더

```
if(bFILE==TRUE)//receive
{
    // 핸들러에 파일정보 저장
    hFile = CreateFile((char *)fapp_hdr->m_FilePath,GENERIC_READ,0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_READONLY,NULL);
    if(hFile == INVALID_HANDLE_VALUE) // 핸들러에 제대로 등록 되었는지 확인
    {
        AfxMessageBox("올바른 파일 경로가 아닙니다.");
        ((CIPCAppDlg *)fapp_hdr->mp_aUpperLayer[0])->OnOffFileButton(TRUE);
        return FALSE;
    }

    // 파일경로에서 파일명만 가져옴
    fileName = fapp_hdr->m_FilePath;
    fileName = fileName.Right(fileName.GetLength()-fileName.ReverseFind('₩₩')-1);

    // 파일 전체 크기(byte)를
    dwFileSize = GetFileSize(hFile,0);

    // 파일 전체 크기만큼 버퍼 공간 생성
    pszBuf = (char *)malloc(dwFileSize);

    if(dwFileSize < FILE_READ_SIZE) // 파일 크기가 한 번에 읽어들이는 크기보다 작으면 seq_num = 1
        tot_seq_num = 1;
    else // 파일 크기가 한 번에 읽어들이는 크기보다 크면, 파일 전체 사이즈를 READ_SIZE로 나눔.
        tot_seq_num = (dwFileSize/FILE_READ_SIZE)+1; // Sequential number 총 개수
}
```

CString fileName

```
,
else//send
{
    if(send_fileTotlen < FILE_READ_SIZE) // 파일 크기가 한 번에 읽어들이는 크기보다 작으면 seq_num = 1
        tot_seq_num = 1;
    else // 파일 크기가 한 번에 읽어들이는 크기보다 크면, 파일 전체 사이즈를 READ_SIZE로 나눔.
        tot_seq_num = (send_fileTotlen/FILE_READ_SIZE)+1; // Sequential number 총 개수
}
```

FileThread는 우선 send할 경우와 Recieve할 경우로 나누어진다.

이 두 경우 모두 보내는 파일 또는 읽어 들인 파일의 크기를 보고 tot_seq_num을 구한다.

```
while(i<=tot_seq_num+1)//
{
    if(bMSGCheck == TRUE)   // 수신 완료 메시지 전송
    {
        fapp_hdr->m_sHeader.fapp_totlen = 0;
        fapp_hdr->m_sHeader.fapp_msg_type = MSG_TYPE_CHECK;
        // TCP PORT 설정
        ((CTCPLayer*)(fapp_hdr->GetUnderLayer()))->SetDestinPort(TCP_PORT_FILE);

        bSuccess = fapp_hdr->mp_UnderLayer->Send((unsigned char*) &fapp_hdr->m_sHeader,FILE_HEADER_SIZE);

        bGoLoop = FALSE;
        bMSGCheck = FALSE;
    }
    if(bMSGFail == TRUE)    /
    {
        bGoLoop = FALSE;
        bMSGFail = FALSE;
    }

    if(bGoLoop == TRUE)
    {
        if(i==0) //  처음부분
        {
            fapp_hdr->m_sHeader.fapp_type = DATA_TYPE_BEGIN;// 단편화된 데이터
```

CBaseLayer *CBaseLayer::mp_UnderLayer

while루프를 사용하여 **수신이 완료되었을 경우(bMSGCheck루프)**와 **메시지를 보내는 데에 실패하였을 경우(bMSGFail루프)** 그리고 **실제 송수신을 하는 경우(bGoLoop)**로 나누어진다.

bMSGCheck루프의 경우 fapp_hdr을 초기화(TCP Port설정) 하여 underlayer로 보내고 bGoLoop와 bMSGFail두개를 FALSE로 만들어 두 루프를 무시한다.(수신이 완료됨)

bMSGFail루프에서는 bGoLoop와 bMSGFail을 FALSE로 만들어서 두 루프를 무시한다.(재전송 메시지 전송)

그리고 bGoLoop에서는 다시 한번 파일을 나누는 세 부분으로 나뉘어 지는데 그 세 가지는 파일의 첫 부분, 중간부분, 마지막 부분이다.

```
if(bGoLoop == TRUE)
{
    if(i==0) //  처음부분
    {
        fapp_hdr->m_sHeader.fapp_type = DATA_TYPE_BEGIN;// 단편화된 데이터
        fapp_hdr->m_sHeader.fapp_msg_type = MSG_TYPE_FRAG; // 메시지의 종류
        fapp_hdr->m_sHeader.fapp_totlen = dwFileSize; // 파일 전체 크기

        dwRead = fileName.GetLength(); // 파일명 길이
        memcpy(fapp_hdr->m_sHeader.fapp_data,fileName,dwRead); // 파일명 저장
        fapp_hdr->m_sHeader.fapp_data[dwRead] = '₩0';
    }
```

파일의 첫 부분을 처리하는 if(i == 1)에서는 fapp_hdr의 m_sHeader에 자른 데이터와 메시지 종류, 파일명, 크기를 저장한다.

```
    else if(i!=0 && i<=tot_seq_num) // 중간부분
    {
        // 파일을 순차적으로 읽어옴
        bResult = ReadFile(hFile,pszBuf,FILE_READ_SIZE,&dwRead,NULL);
                        /* ReadFile(인자 순서)
                        * hFile :           File handler
                        * pszBuf :          파일의 내용을 읽어들일 버퍼
                        * FILE_READ_SIZE :  한 번 읽어들일 파일의 byte단위의 수
                        * dwRead :          읽어들인 파일의 데이터 양에 대한 출력용 인수
                        * Overlapped 설정 (보통 NULL)
                        */
        dwState += dwRead;
        pszBuf[dwRead] = '₩0';

        fapp_hdr->m_sHeader.fapp_totlen = dwRead; // 읽어온 데이터 크기
        fapp_hdr->m_sHeader.fapp_type = DATA_TYPE_CONT; // 단편화된 데이터 타입
        fapp_hdr->m_sHeader.fapp_msg_type = MSG_TYPE_FRAG; // 메시지의 종류

        fapp_hdr->m_sHeader.fapp_seq_num = i-1; // sequential number
        ((CIPLayer*)(fapp_hdr->GetUnderLayer()->GetUnderLayer()))->SetFragOff(fapp_hdr->m_sHeader.fapp_seq_num);

        memcpy(fapp_hdr->m_sHeader.fapp_data,pszBuf,dwRead); // 읽어온 데이터
        memset(pszBuf,0,dwRead);

        //  현재 진행상황을 프로그래스바에 표시해줌.
        progress_value = (100 * dwState) / dwFileSize ;
        ((CIPCAppDlg *)fapp_hdr->mp_aUpperLayer[0])->m_ProgressCtrl.SetPos(progress_value);
    }
```

파일의 중간부분을 처리하는 if(i != 0 && I < tot_seq_num)은 ReadFile을 통해 파일을 순차적으로 읽어온다. fapp_hdr의 m_sHeader에 자른 데이터와 메시지의 종류, 읽어온 만큼의 크기, 시퀀스넘버를 저장한다. dwState변수와 dwFileSize변수를 이용하여 진행상황을 프로그래스 바에 표시한다.

```
    ,
    else // 마지막 부분
    {
        CloseHandle(hFile); // 파일 Handler를 닫는다.
        fapp_hdr->m_sHeader.fapp_totlen = 0;
        fapp_hdr->m_sHeader.fapp_msg_type = MSG_TYPE_FRAG;//
        fapp_hdr->m_sHeader.fapp_type = DATA_TYPE_END;
        memset(fapp_hdr->m_sHeader.fapp_data,0,APP_DATA_SIZE);
        dwRead = 0;
    }

    //bGoLoop = FALSE;

    bSuccess = fapp_hdr->mp_UnderLayer->Send((unsigned char*) &fapp_hdr->m_sHeader,FILE_HEADER_SIZE+dwRead);
    memset(fapp_hdr->m_sHeader.fapp_data,0,dwRead);//헤더의 fapp_data를 초기화한다.
    i++;
}
```

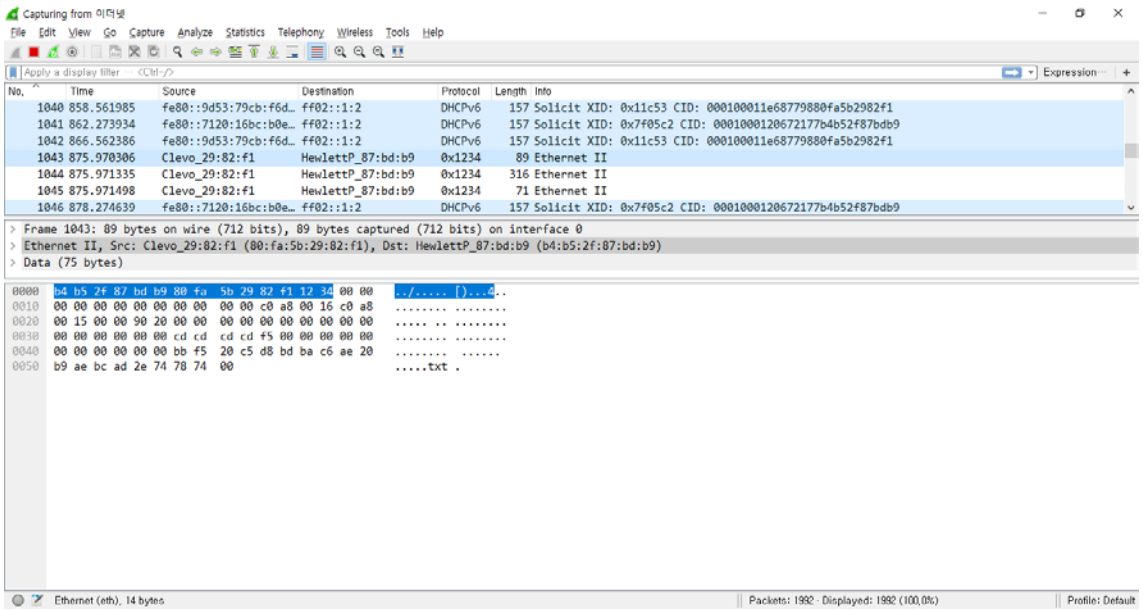
파일의 마지막 부분을 처리하는 else는 filehandler를 닫고 fapp_hdr의 m_sHeader에 데이터 길이를 0으로 만들고 메시지 타입을 FRAG로 apptype은 END타입으로 만든다. dwRead를 0으로 만들어 더 이상 읽을 파일이 없음을 알리고 헤더를 초기화하고 다음 조각의 파일을 읽기 위해 전체 while문에서 tot_seq_num과 비교하는 I값을 ++한다.

```
if(bSuccess==TRUE)
    AfxMessageBox("파일 전송 완료"); // 전송완료 메시지 창 출력
else
    AfxMessageBox("파일 전송 실패"); // 전송실패 메시지 창 출력

((CIPCAppDlg *)fapp_hdr->mp_aUpperLayer[0])->m_ProgressCtrl.SetPos(0);//progress bar를 초기화한다.
((CIPCAppDlg *)fapp_hdr->mp_aUpperLayer[0])->OnOffFileButton(TRUE);//파일전송버튼을 다시 활성화시킨다.

if(tot_seq_num!=1) // 나눌 필요 없는 1400바이트 이하의 파일일 경우 버퍼를 저장하려 했던 pszBuf변수에 할당했던 메모리를 반환시킨다.
{
    free(pszBuf);
}

bGoLoop = TRUE; // 다음 송신에서 반복문에 진입하기 위해서 bGoLoop를 TRUE로 초기화

return bSuccess ;
```

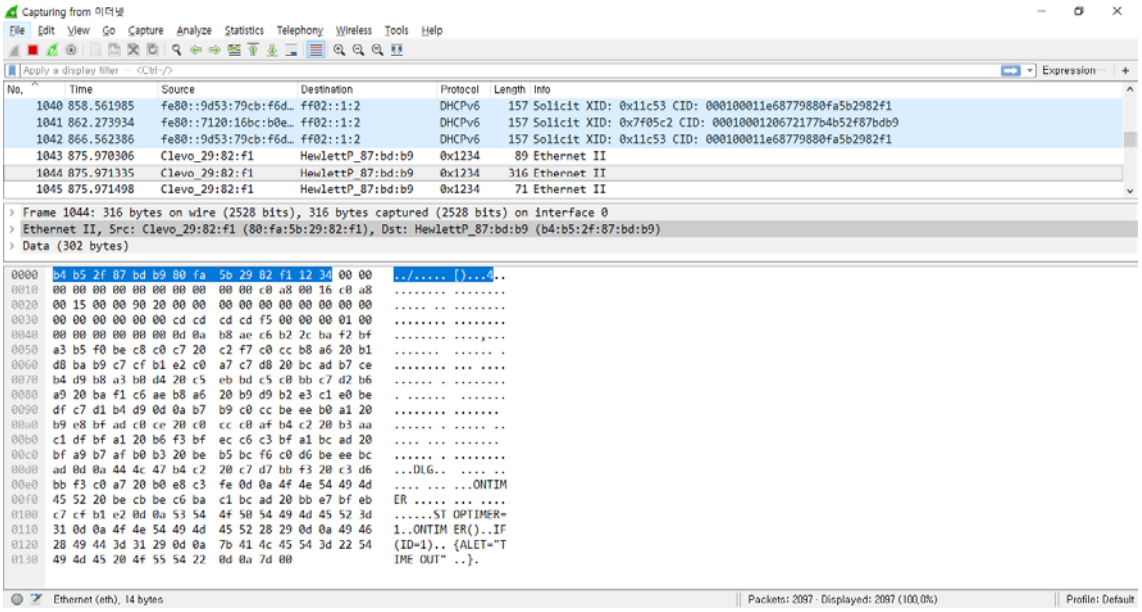
반복문을 빠져나왔다면 파일 전송에 성공하였을 경우 전송완료 메시지 창을 출력하고 프로세스바를 초기화하고 파일전송버튼을 다시 활성화시킨다.

반복문에 들어가지 않는 나눌 필요 없는 1400바이트 이하의 파일일 경우 버퍼를 저장하려 했던 pszBuf변수에 할당했던 메모리를 반환시킨다.

그리고 다음 송신에서 반복문에 진입하기 위해 bGoLoop를 TRUE로 초기화한다.

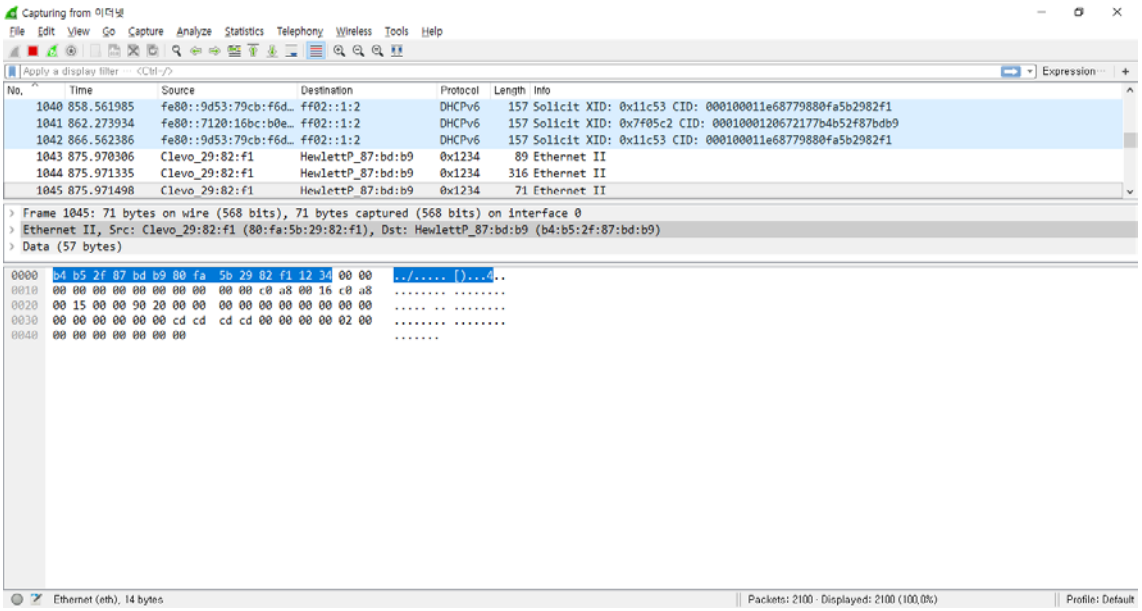


처음 전송 부분에는 파일의 이름(변수 : fileName)을 보낸다.(bGoLoop의 첫 번째 if문)



두 번째 전송 부분에는 실제 데이터를 전송한다.(bGoLoop의 else if문)

데이터의 크기에 따라 여러 번 전송되는 부분



마지막 전송에는 마지막 정보임을 알리고 초기화 된 데이터를 전송한다.

FileAppLayer

CFileAppLayer::Receive

```
BOOL CFileAppLayer::Receive(unsigned char* ppayload)
{	static HANDLE hFile = NULL;
	DWORD dwWrite=0, dwState=0;

	int progress_value;
	BOOL bResult;
	BOOL bSuccess = FALSE;

	LPFILE_APP fapp_hdr = (LPFILE_APP) ppayload ;
	unsigned char *GetBuff;
```

```
	if(hFile == INVALID_HANDLE_VALUE) // 파일 핸들러가 제대로 등록 됐는지 확인
	{	return FALSE;
	}
	if(fapp_hdr->fapp_msg_type == MSG_TYPE_FRAG) // 단편화된 데이터 메시지
	{	// 밑 계층에서 넘겨받은 ppayload를 분석하여 ChatDlg 계층으로 넘겨준다.
		if(fapp_hdr->fapp_type == DATA_TYPE_BEGIN) // 데이터 첫 부분
		{
		((CIPCAppDlg *)mp_aUpperLayer[0])->OnOffFileButton(FALSE);
		send_fileTotlen = fapp_hdr->fapp_totlen; //수신받을 데이터의 전체길이
```

```
hFile=CreateFile((char
*)fapp_hdr>fapp_data,GENERIC_WRITE,FILE_SHARE_READ,NULL,CREATE_ALWAYS,FILE
_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN,NULL);
                        if(hFile == INVALID_HANDLE_VALUE)
                        {       AfxMessageBox("파일생성오류");
                                RETURN FALSE;
                        }
                        bMSGCheck = TRUE; // 데이터를 잘받았다고 메시지 보내기위한 스
위치 역할의 BOOL 변수값을 TRUE
                        bGoLoop = FALSE;
```

---> 수신시에 받은 데이터의 첫 부분을 통해 단편화 여부를 검사한다. 전체 길이가 1448bytes 초과라면 단편화가 이루어진다.

만약 1448bytes 이하파일로 단편화가 되어있지 않다면, 그대로 상위 레이어로 올라간다.

1448byte를 초과한 파일이라면 단편화가 이루어져, 해당 파일 정보를 받아 파일을 생성하여, 파일명으로 파일크기 만큼의 파일을 저장할 공간을 생성한다.

그런데 파일이 제대로 핸들러 등록이 되지 않았다면 "파일 생성오류" 라는 메시지창을 출력한다.

제대로 파일을 받았다면 받았다는 메시지를 보내기위한 스위치 역할하는 bMSGCheck 값을 True 로 선언한다.

```
} AfxBeginThread(FileThread,this); }
                else if(fapp_hdr->fapp_type == DATA_TYPE_CONT) // 데이터 중간 부
분
                { receive_fileTotlen+= fapp_hdr->fapp_totlen; // 받은 파일의 길이를
누적 합산
                        GetBuff = (unsigned char*)malloc(fapp_hdr->fapp_totlen); //
                버퍼 생성
                        memset(GetBuff,0,fapp_hdr->fapp_totlen); // 버퍼초기화
```

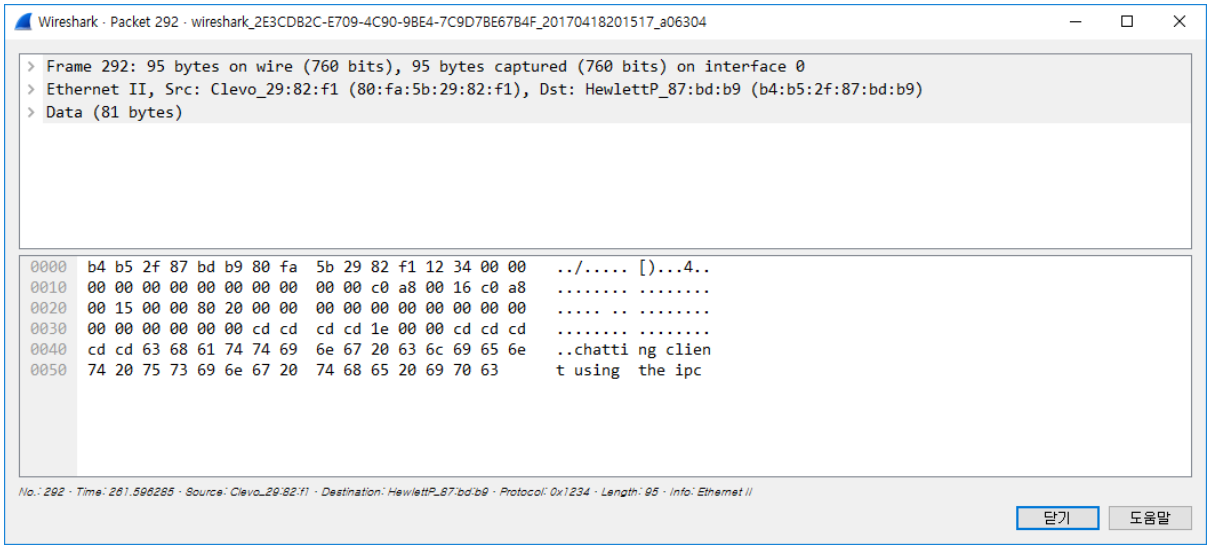
---> 파일의 중간부분으로 넘어가서 받은 파일의 길이를 누적합산 한 다음 버퍼를 이용하여 받은 파일을 덮어씌운다. 단편화된 조각을 받아 받은 조각의 개수를 단편화한 조각의 총 개수로 나누어 progress bar를 수정한다.

```
}
                else if(fapp_hdr->fapp_type == DATA_TYPE_END) // 데이터 끝 부분
                {       if(send_fileTotlen == receive_fileTotlen)
                                bSuccess = TRUE;
                        else    bSuccess = FALSE;
                        CloseHandle(hFile);
                        bMSGCheck = FALSE;
                        bGoLoop = FALSE;
                        bFILE = FALSE;
                        if(bSuccess==TRUE)
                        AfxMessageBox("파일 수신 완료"); // 수신완료 메시지 창
                        else
                        AfxMessageBox(""파일 수신 실패); // 수신실패 메시지 창
```

```
receive_fileTotlen = 0;      // 수신 완료 후 다음 파일 수신확인을 위해 받은 파일의 길이를 초기
화
                        ((CIPCAppDlg *)mp_aUpperLayer[0])->m_ProgressCtrl.SetPos(0);
                        ((CIPCAppDlg *)mp_aUpperLayer[0])->OnOffFileButton(TRUE);
                }
        }
        else if(fapp_hdr->fapp_msg_type == MSG_TYPE_CHECK) // 수신받은 데이터 확인 메
시지{
                bGoLoop = TRUE;
        }
        else if(fapp_hdr->fapp_msg_type == MSG_TYPE_FAIL) // 잘못된 데이터 수신시 보내
는 메시지
        {       bMSGFail = TRUE;
        }
        return bSuccess ;
}
```

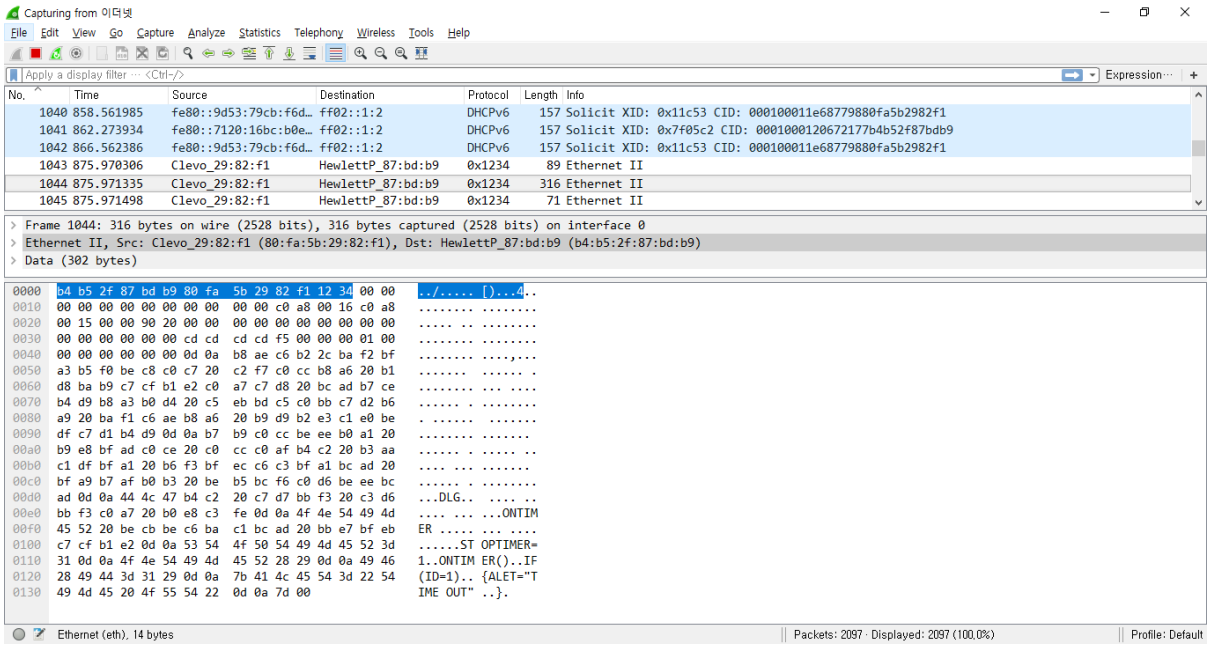
--->파일의 끝 부분에서는 파일 총길이와 현재 누적 길이를 비교하여 값이 맞으면 bSuccess를 TRUE로, 핸들러를 닫는다. bSuccess가 TRUE면 수신완료 메세지를 띄우고 다음 전송을 위해 받은 파일길이를 초기화시킨다. 그러고는 수신받은 데이터를 확인하는 bGoLoop를 TRUE로 설정하고, 잘못받은 메세지 타입이라면 bMSGFail을 TRUE로 설정함.

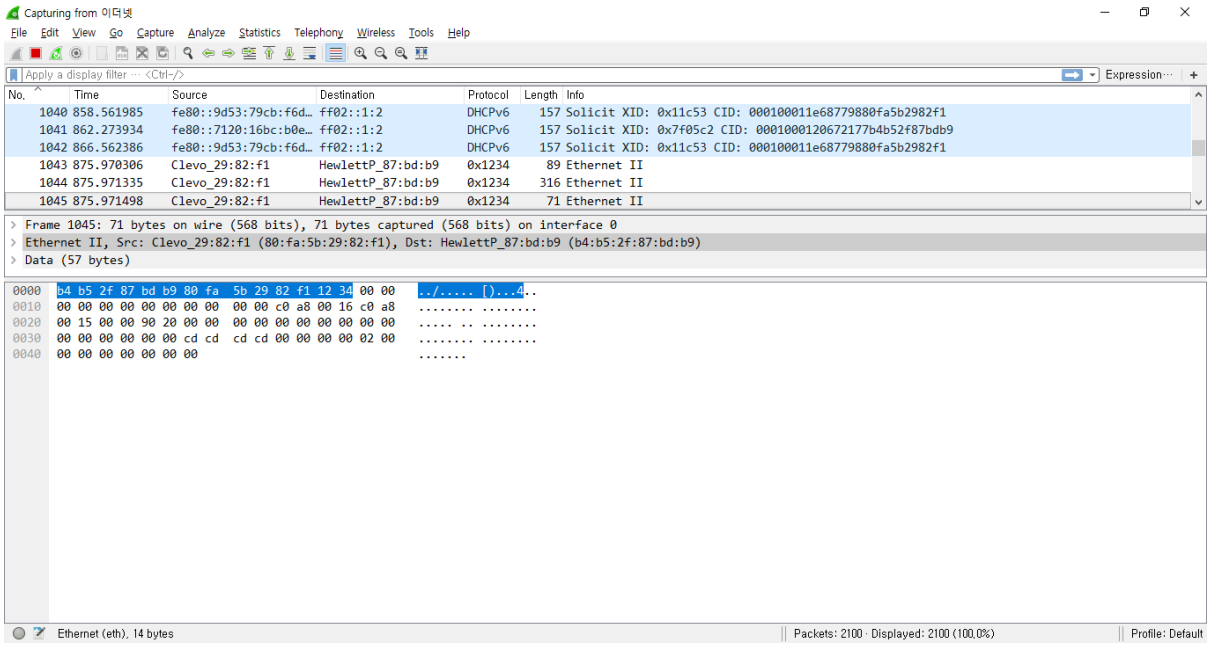
4. 결과 분석



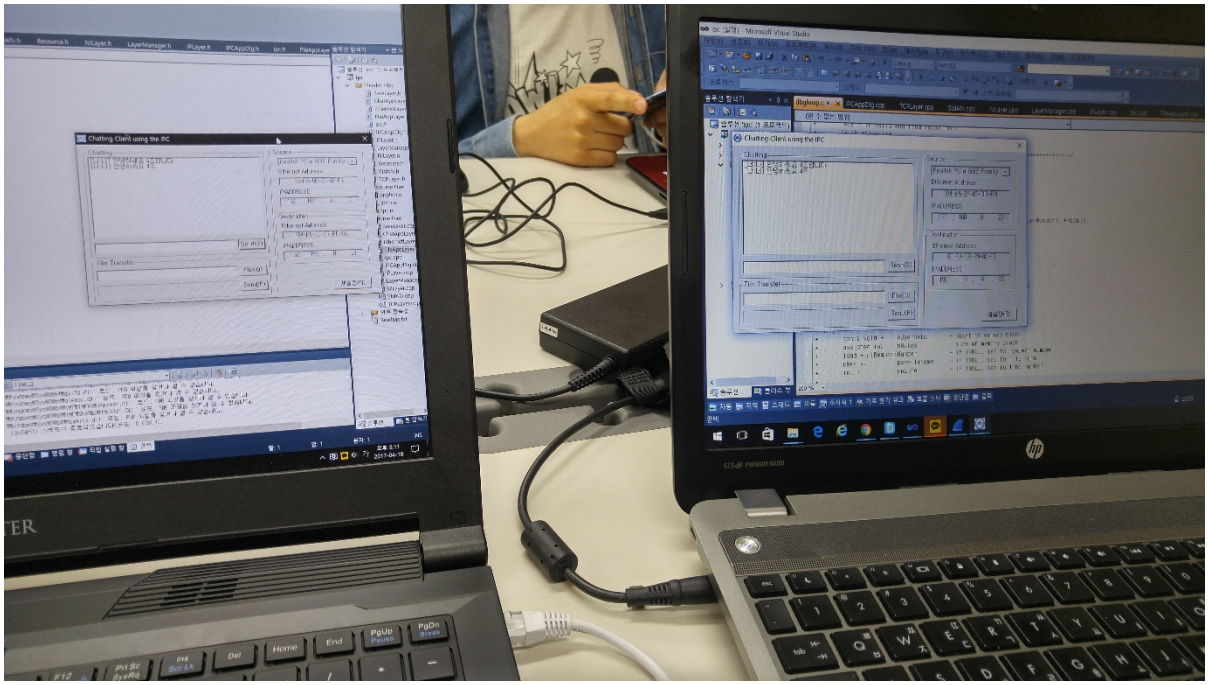
- **채팅을 보냈을 때의 패킷 캡쳐**

-

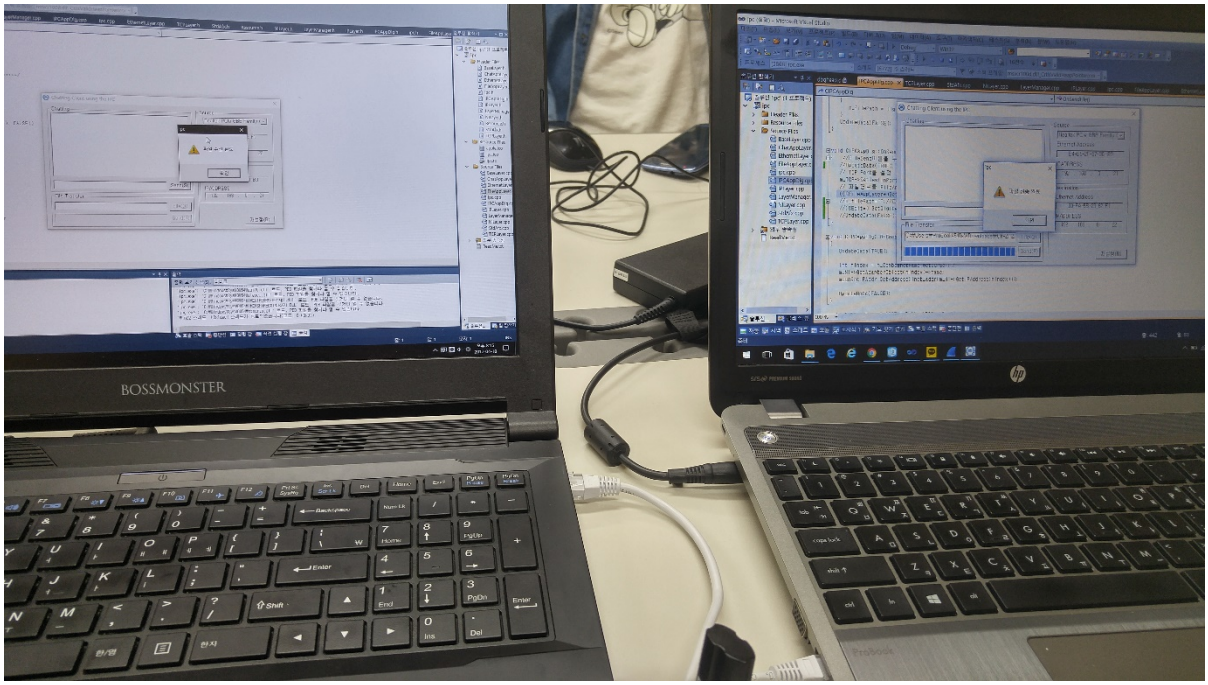




5 실행결과

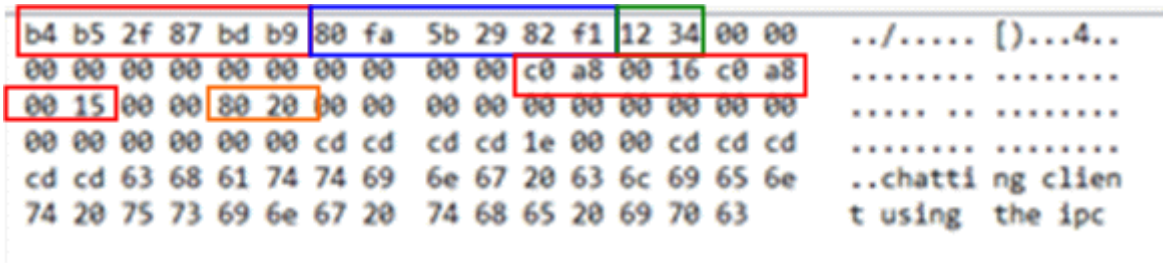


- 채팅 전송 결과 화면



- **파일 전송 결과 화면**

- **WIreShark를 이용한 패킷분석**



- 첫번째 줄의 빨간색 사각형은 목적지 MAC주소, 파란색 사각형은 출발지 MAC주소, 초록색 사각형은 프레임타입을 의미한다. 그리고 두번째 줄의 숫자는 ip주소를 의미하고 주황색 사각형은 포트번호를 의미한다.

6 느낀점

- 구현하기가 매우 힘들었다.
- 네트워크통신과정에서 각 계층이 하는 역할에 대해 알게 되었다.
- 우리가 쉽게 접하는 파일 전송에 대해 깊게 알아보니 생각보다 복잡하다는 것을 알게 되었고 처음 만든 사람에게 경의를 표한다.