

보고서

- 과제 3 -

과목명 | 응용소프트웨어실습

담당교수 | 이강훈

제출일 | 2019년 6월 6일

소속 | 소프트웨어학부

학번 | 2015202065

이름 | 윤홍찬



광운대학교
KwangWoon University

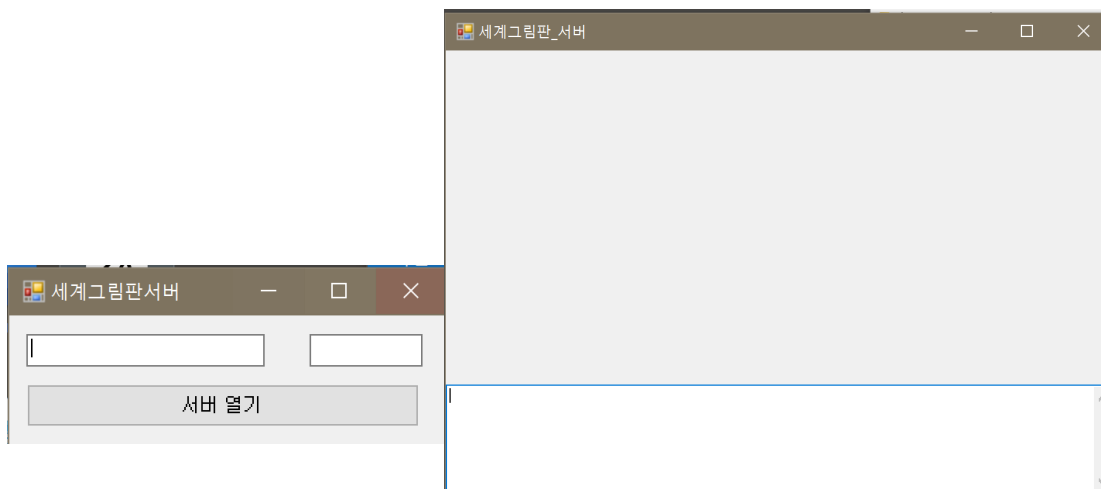
< 목 차 >

I . 체크리스트	2
II . Form 구성	3
III . 그림판	3
IV . 다중 클라이언트 구현	6
V . 서버의 정보 저장	10
VI . 채팅	12
VII . 고찰	13

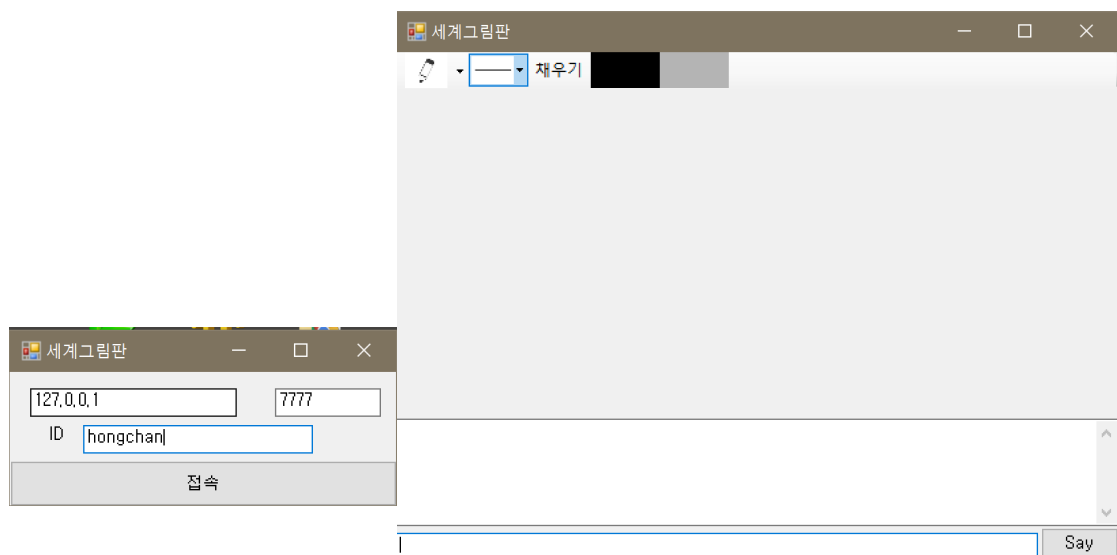
I. 체크리스트

항목	구현 내용	구현 여부	점수
Form 구성	Server Form - Modal 창 (0.2점)	O	0.2
	Server Form - 그림판 + 채팅로그 구성 (0.3점)	O	0.3
	Client Form - Modal 창 (0.2점)	O	0.2
	Client Form - ToolStrip Panel (0.5점)	O	0.5
	Client Form - 그림판 + 채팅로그 구성 (0.3점)	O	0.3
그림판	도구 (Pencil, Line, Rect, Circle) (0.4점)	O	0.4
	선 두께 (0.2점)	O	0.2
	채우기 (0.2점)	O	0.2
	선택 (0.1점)	O	0.1
	면색 (0.1점)	O	0.1
	더블 버퍼링 (0.5점)	O	0.5
화면 조작	확대 (0.5점)	X	0
	축소 (0.5점)	X	0
	이동 (0.5점)	X	0
	상대적 마우스 위치 고정 (0.5점)	X	0
	클라이언트 독립 수행 (0.5점)	X	0
다중 클라이언트 구현	최대 10개의 클라이언트가 서버와 통신하여 그림판을 공유 (1점)	O	1
	늦게 들어온 클라이언트도 서버에 그려진 모든 그림 데이터를 공유 (1점)	O	1
	클라이언트 중 일부가 접속이 종료될 시에도 정상적으로 작동함 (1점)	O	1
서버 정보 저장	서버 정보 저장 (0.5점)	O	0.5
채팅	서버와 1:1 채팅 - 닉네임 (0.2점)	O	0.2
	서버와 1:1 채팅 - 메시지 (0.3점)	O	0.3
	다중 채팅 (0.5점)	O	0.5
총점			7.5

Ⅱ. Form 구성



[Server Form]



[Client Form (로드될 때 채팅창이 활성화 됨)]

Ⅲ. 그림판

그림판을 구현하는데 강의 자료를 참고하였다. 여기서는 강의 자료에 있는 코드는 생략하고, 직접 구현한 코드에 대해서만 설명하겠다. 또한 서버와 통신하여 그림 객체를 보내거나 받는 부분은 다음 장에서 설명하고, 여기서는 각 클라이언트와 서버가 Panel에 어떻게 그림을 그리는지에 대해서만 설명하겠다.

1. 채우기

채우기 버튼을 누르면 실행되는 이벤트 핸들러를 다음과 같이 작성하였다.

```
218 private void fillToggle_Click(object sender, EventArgs e)
219 {
220     if (fillTog)
221         fillTog = false;
222     else
223         fillTog = true;
224 }
```

이후, 그리기 부분에서 fillTog 가 true, false 임에 따라 다르게 작동하게 된다. 이는 그리는 부분에서 설명하겠다.

2. 선색 및 면색

```
226 private void lineColor_Click(object sender, EventArgs e)
227 {
228     if (colorDlg.ShowDialog() == DialogResult.OK)
229     {
230         lineClr = colorDlg.Color;
231     }
232 }
233
234 private void fillColor_Click(object sender, EventArgs e)
235 {
236     if (colorDlg.ShowDialog() == DialogResult.OK)
237     {
238         fillClr = colorDlg.Color;
239     }
240 }
```

각각 선, 면 설정 버튼에 대한 이벤트 핸들러이다. 각각 color dialog를 띄워 색을 입력받은 후, 변수에 색상을 저장한다. 바로 다음 그리는 부분에서 이에 대한 설명을 추가하겠다.

3. 그리기

```
389 int pIndex = 0;
390 int lIndex = 0;
391 int rIndex = 0;
392 int cIndex = 0;
393
394 for (int i = 0; i <= index; i++)
395 {
396     if (pIndex <= npencil)
397     {
398         if (mypencil[pIndex].getIndex() == i)
399         {
400             // Draw Pencil
401             pen.Width = mypencil[pIndex].getThick();
402             pen.DashStyle = DashStyle.Solid;
403             pen.Color = mypencil[pIndex].getLineColor();
404
405             if (mypencil[pIndex].getPoints().Count() == 0)
406             { // Do NOTHING
407             }
408             else
409             {
410                 e.Graphics.DrawCurve(pen, mypencil[pIndex].getPoints().ToArray());
411                 pIndex++;
412             }
413         }
414     }
```

이는 Panel이 그려질 때 호출되는 Panel_Paint 함수의 일부분이다. 위의 사진은 Pencil을 그리는 부분이다. 제일 먼저 바깥쪽 for문은 전체 index(도형 개수)만큼 돌면서 그림을 그리게 된다. 이처럼 전체 for문을 사용한 이유는 그려진 그림 순서에 따라 그려지도록 하기 위함이다. 396행 if문은 pencil이 아직 다 안그려졌는지 확인하고 398행의 if문은 현재 도형의 순서와 같은지를 체크한다. 만약 같다면 설정된 색상, 굵기로 곡선을 그리게 된다.

```

428 | if(rIndex <= nrect)
429 | {
430 |     if (myrect[rIndex].getIndex() == i)
431 |     {
432 |         // Draw Rectangle
433 |         pen.Width = myrect[rIndex].getThick();
434 |         pen.DashStyle = DashStyle.Solid;
435 |         pen.Color = myrect[rIndex].getLineColor();
436 |
437 |         e.Graphics.DrawRectangle(pen, myrect[rIndex].getRect());
438 |
439 |         if (myrect[rIndex].getFill())
440 |         {
441 |             // Fill Rectangle
442 |             brush.Color = myrect[rIndex].getFillColor();
443 |             e.Graphics.FillRectangle(brush, myrect[rIndex].getRect());
444 |         }
445 |         rIndex++;
446 |     }
447 | }

```

위의 사진은 직사각형을 그리는 부분이다. 위의 pencil과 다른 점은 439행 ~ 444행인데, 이 부분에서 채우기 버튼이 활성화 된 채 그려졌으면, FillRectangle()을 호출해 내부 색상을 채운다. Line, Circle은 이 둘과 유사한 방식으로 그리게 되므로, 설명은 생략하겠다.

4. 더블 버퍼링

```

public class DoubleBufferPanel : Panel
{
    public DoubleBufferPanel()
    {
        this.SetStyle(ControlStyles.DoubleBuffer |
            ControlStyles.UserPaint |
            ControlStyles.AllPaintingInWmPaint,
            true);
        this.UpdateStyles();
    }
}

```

먼저 위와 같이 Panel을 상속받는 클래스를 작성하였다.

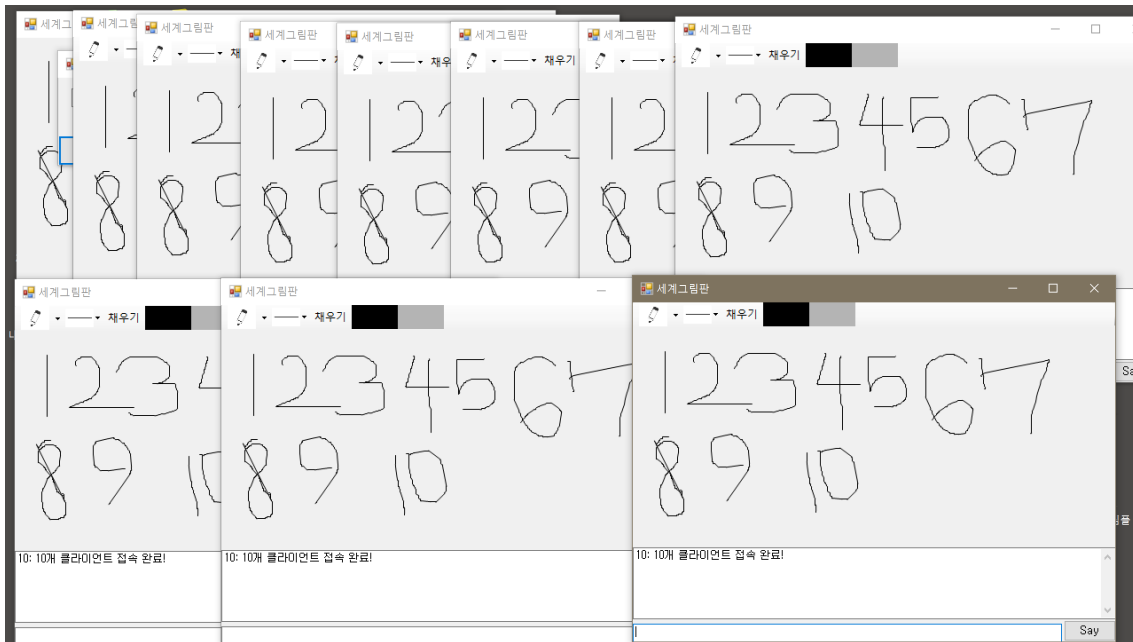
```

this.panel = new GlobalPaintClient.DoubleBufferPanel();

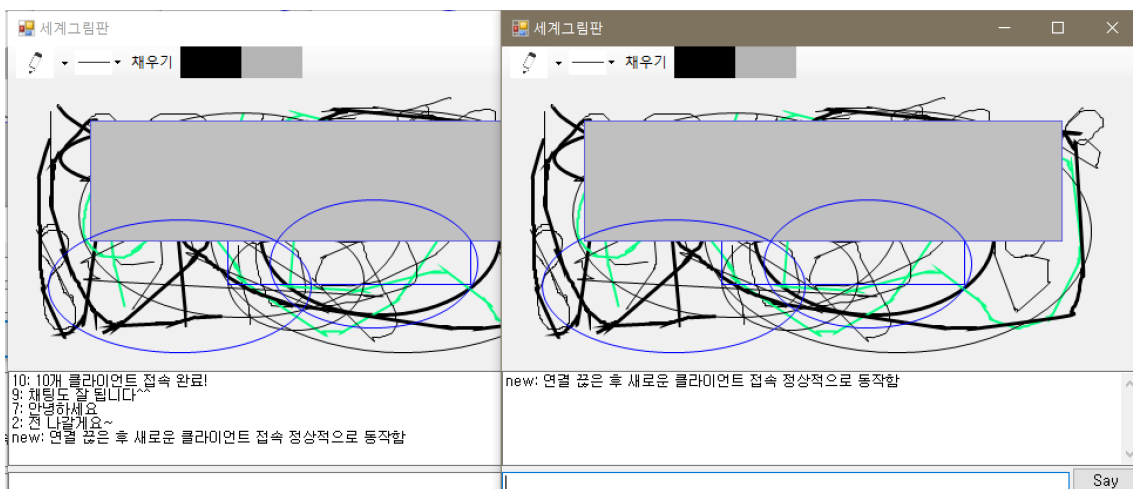
```

그리고 Server, Client Form InitializeComponent() 메소드 내의 Panel을 생성하는 부분을 위의 정의한 클래스의 객체를 생성하도록 하였다.

IV. 다중 클라이언트 구현



[10개의 클라이언트가 동시 접속한 모습]



[늦게 들어온 클라이언트도 서버에 그려진 모든 그림 데이터를 공유하는 모습 & 클라이언트 중 일부가 접속이 종료될 시에도 정상적으로 작동하는 모습]

1. Packet 정의

```
public static class Constant
{
    public const int PACKET_SIZE = 1024 * 40;
    public const int INIT_PACKET_SIZE = 1024 * 200;
}
```

서버와 클라이언트 사이에 전송될 패킷 사이즈는 위와 같이 설정하였다.

```

74 [Serializable]
75 public class PanelInit : Packet
76 {
77     public MyPencil[] mypencil;
78     public MyLines[] mylines;
79     public MyRect[] myrect;
80     public MyCircle[] mycircle;
81
82     public int npencil;
83     public int nline;
84     public int nrect;
85     public int ncircle;
86     public int index;
87 }
88
89 [Serializable]
90 public class PanelInfo : Packet
91 {
92     public MyPencil mypencil;
93     public MyLines mylines;
94     public MyRect myrect;
95     public MyCircle mycircle;
96
97     public int npencil;
98     public int nline;
99     public int nrect;
100    public int ncircle;
101    public int index;
102 }

```

[원 : 맨 처음 클라이언트가 서버에 접속할 때 보낼 패킷
오 : 클라이언트가 그림을 그릴 때 서버에 전송될 패킷]

```

112 [Serializable]
113 public class IndexInfo : Packet
114 {
115     public int npencil;
116     public int nline;
117     public int nrect;
118     public int ncircle;
119     public int index;
120 }

```

[클라이언트가 마우스를 떼어 그림을 다 그렸을 때 전송될 패킷]

2. Server 동작 과정

```

82 while (m_bStop)
83 {
84     m_client = m_server.AcceptTcpClient();
85
86     if (m_client.Connected)
87     {
88         // Make Thread - Client Access
89         CreateThread();
90     }
91 }

```

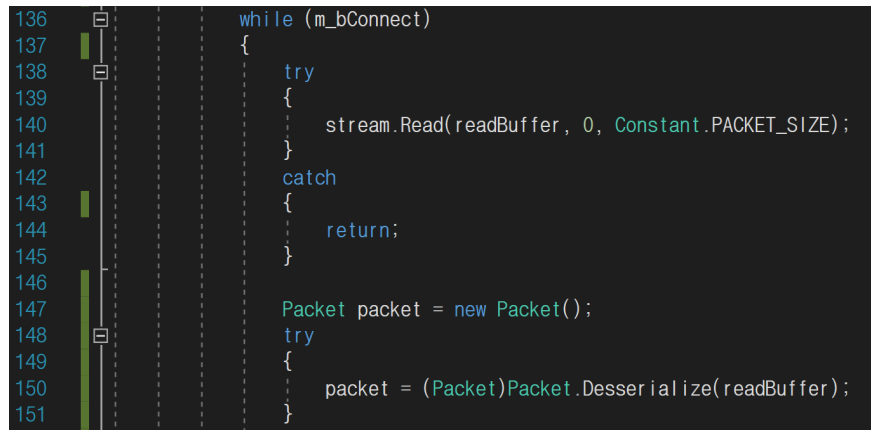
서버가 클라이언트를 받을 때마다 위와 같이 스레드를 생성한다.

```

128 // Add client
129 client = m_client;
130 m_bConnect = true;
131 stream = client.GetStream();
132 streams.Add(stream);
133
134 clientID = ClientInit(stream);

```

위는 스레드에서 수행되는 코드이다. 131행에서 stream을 열고, 132행에서는 List 자료구조에 저장하고 있다. 이후 134행에서는 ClientInit() 메소드를 호출하는데, 이 메소드는 위에서 설명한 그림 객체 패킷을 클라이언트에게 보내고, 아이디를 return한다.

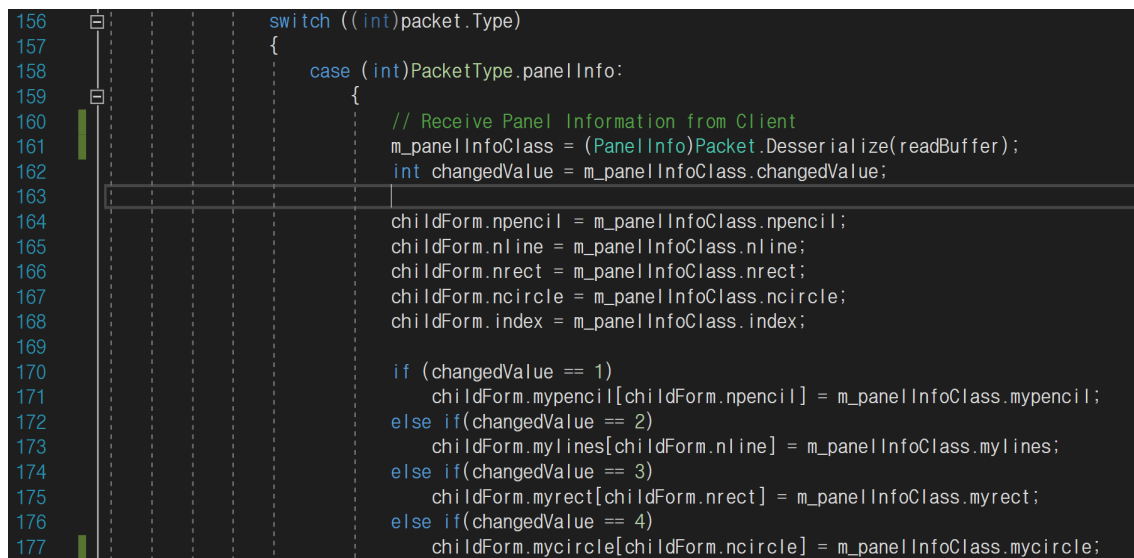


```

136 while (m_bConnect)
137 {
138     try
139     {
140         stream.Read(readBuffer, 0, Constant.PACKET_SIZE);
141     }
142     catch
143     {
144         return;
145     }
146
147     Packet packet = new Packet();
148     try
149     {
150         packet = (Packet)Packet.Desserialize(readBuffer);
151     }

```

초기 정보를 클라이언트에게 전송하면, 각 스레드는 140행에서 클라이언트의 데이터 전송을 기다리게 된다.

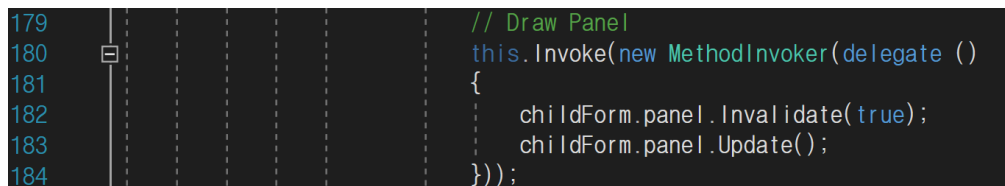


```

156 switch ((int)packet.Type)
157 {
158     case (int)PacketType.panelInfo:
159     {
160         // Receive Panel Information from Client
161         m_panelInfoClass = (PanelInfo)Packet.Desserialize(readBuffer);
162         int changedValue = m_panelInfoClass.changedValue;
163
164         childForm.npencil = m_panelInfoClass.npencil;
165         childForm.nline = m_panelInfoClass.nline;
166         childForm.nrect = m_panelInfoClass.nrect;
167         childForm.ncircle = m_panelInfoClass.ncircle;
168         childForm.index = m_panelInfoClass.index;
169
170         if (changedValue == 1)
171             childForm.mypencil[childForm.npencil] = m_panelInfoClass.mypencil;
172         else if (changedValue == 2)
173             childForm.mylines[childForm.nline] = m_panelInfoClass.mylines;
174         else if (changedValue == 3)
175             childForm.myrect[childForm.nrect] = m_panelInfoClass.myrect;
176         else if (changedValue == 4)
177             childForm.mycircle[childForm.ncircle] = m_panelInfoClass.mycircle;

```

위는 클라이언트로부터 PanelInfo 패킷을 받았을 때의 처리이다. 먼저 패킷으로부터 변수들을 최신화하고, 어떤 것이 변경되었는지 확인한 다음 변경된 도형을 최신화한다.



```

179 // Draw Panel
180 this.Invoke(new MethodInvoker(delegate ()
181 {
182     childForm.panel.Invalidate(true);
183     childForm.panel.Update();
184 }));

```

스레드를 사용하였기 때문에 대리자를 사용하여 Panel을 다시 그린다.

```

186 // Send All Client
187 for(int i = 0; i < streams.Count; i++)
188 {
189     NetworkStream s = streams[i];
190
191     if (s == stream)
192     {
193         // Pass My Stream
194         continue;
195     }
196     Packet.Serialize(m_panelInfoClass).CopyTo(sendBuffer, 0);
197     try
198     {
199         Send(s);
200     }
201     catch (Exception e)
202     {
203         streams.Remove(s);
204     }
205 }

```

이후 다른 클라이언트에게 최신화 된 도형 정보를 보내는 코드이다. stream배열에는 이미 패킷을 보냈던 클라이언트도 속해있으므로 194행을 보면 넘어가고, 203행은 스트림이 없어 예외가 발생하는 경우 해당 스트림을 List로부터 삭제하고 있다.

3. Client 동작 과정

```

140 parentForm.Init();
141
142 // Make Thread to process Panel Receive
143 thread = new Thread(new ThreadStart(parentForm.ReceiveFromServer));
144 thread.Start();

```

클라이언트가 서버와 연결되면 140행의 메소드가 호출되어, 서버로부터 도형들의 정보를 받아 Panel을 그리고, 아이디를 서버에 전송한다. 이후 서버로부터 오는 응답을 처리하기 위해 143행에서 스레드를 생성한다.

```

181 public void ReceiveFromServer()
182 {
183     while (true)
184     {
185         try
186         {
187             stream.Read(readBuffer, 0, Constant.PACKET_SIZE);
188         }
189         catch
190         {
191             return;
192         }
193
194         Packet packet = new Packet();
195         try
196         {
197             packet = (Packet)Packet.Desserialize(readBuffer);
198         }
199         catch { }
200
201         switch ((int)packet.Type)

```

스레드에 의해 실행되는 부분이다, 이는 서버로부터 read를 하면서, PanelInfo 타입이면 클라이언트의 도형 정보를 업데이트하고 새롭게 그린다. 이에 대한 부분은 서

버와 유사하므로 생략하겠다.

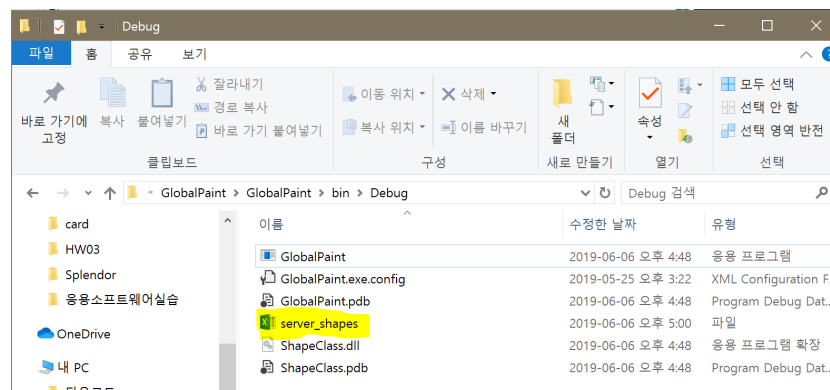
```
246 // Send Panel to Server
247 if (!serverSend)
248 {
249     // Call by Client (Not Call by Server)
250     int changedShape = 1;
251     if (pencil)
252         changedShape = 1;
253     else if (line)
254         changedShape = 2;
255     else if (rect)
256         changedShape = 3;
257     else if (circle)
258         changedShape = 4;
259     if (pencil || line || rect || circle)
260         parentForm.SendPanelInfo(changedShape);
261 }
```

위 코드는 Panel_Paint 메소드의 일부분으로, Panel이 업데이트 될 때 서버에 업데이트 정보를 전달하는 코드이다. 247행에서 if문은 서버에서 전달되어 Panel이 업데이트 되는지를 판단한다. 만약 서버에서 최신 도형 정보를 얻어와 그릴 때는 이 부분이 호출되면 안되므로, if 조건문을 삽입하였다.

```
522 // Painting Order
523 index++;
524
525 // Send Changed number;
526 parentForm.SendChangedIndexInfo();
```

Panel에서 마우스를 땀을 때(도형을 다 그렸을 때) 실행되는 부분의 일부분이다. 523행의 index는 전체 도형의 개수를 추적하고, 526행에서 호출하는 메소드는 서버에게 최신 도형의 개수를 보낸다.

V. 서버의 정보 저장



[server_shapes 라는 이름으로 저장되는 정보들]

```

18 [Serializable]
19 public class StoreShape
20 {
21     public MyPencil[] mypencil;
22     public MyLines[] mylines;
23     public MyRect[] myrect;
24     public MyCircle[] mycircle;
25
26     public int npencil;
27     public int nline;
28     public int nrect;
29     public int ncircle;
30     public int index;

```

[파일에 저장할 정보들을 클래스 안에 넣음]

```

229 private void serverForm2_FormClosing(object sender, FormClosingEventArgs e)
230 {
231     StoreShape sh = new StoreShape();
232     sh.mypencil = mypencil;
233     sh.mylines = mylines;
234     sh.mycircle = mycircle;
235     sh.myrect = myrect;
236     sh.npencil = npencil;
237     sh.nline = nline;
238     sh.ncircle = ncircle;
239     sh.nrect = nrect;
240     sh.index = index;
241
242     Stream stm = File.Open(filePath, FileMode.Create, FileAccess.Write);
243     BinaryFormatter bf = new BinaryFormatter();
244     bf.Serialize(stm, sh);
245     stm.Close();
246 }
247

```

위 코드는 server form이 종료되기 직전에 호출되는 이벤트 핸들러이다. 231행 ~ 240행은 현재 도형의 정보를 객체에 저장하고, 242행 ~ 244행은 파일에 저장하는 과정이다.

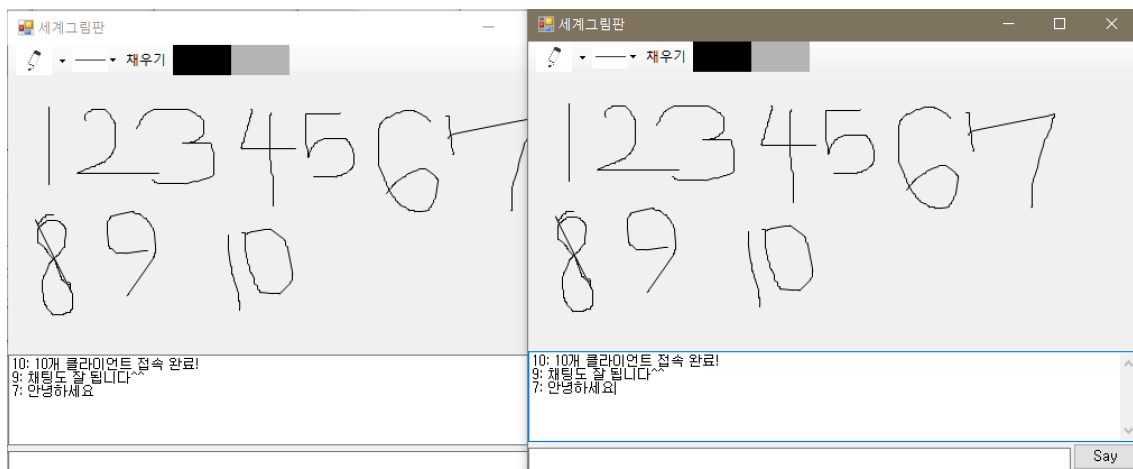
```

205 private void serverForm2_Load(object sender, EventArgs e)
206 {
207
208     if (File.Exists(filePath))
209     {
210         // file exist
211         Stream stm = File.Open(filePath, FileMode.Open, FileAccess.Read);
212         BinaryFormatter bf = new BinaryFormatter();
213
214         StoreShape sh = (StoreShape)bf.Deserialize(stm);
215         stm.Close();
216
217         mypencil = sh.mypencil;
218         mylines = sh.mylines;
219         mycircle = sh.mycircle;
220         myrect = sh.myrect;
221         npencil = sh.npencil;
222         nline = sh.nline;
223         ncircle = sh.ncircle;
224         nrect = sh.nrect;
225         index = sh.index;
226     }
227

```

이는 server form이 로드되면 실행되는 부분이다. 208행에서 우선 백업 파일이 있는지 검사하고, 파일이 있으면 파일로부터 정보를 읽어 들여 저장한다.

VI. 채팅



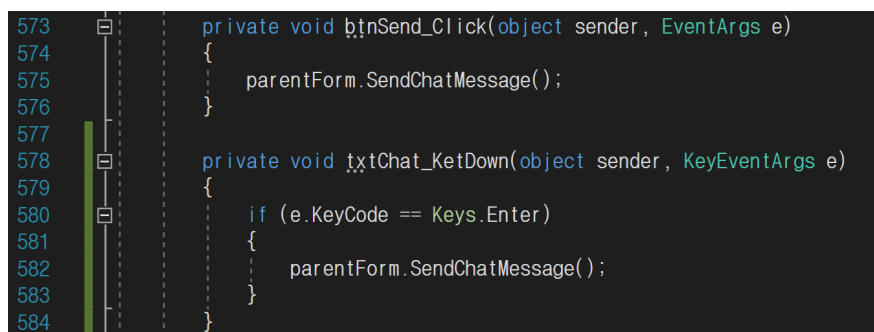
[여러 클라이언트 간 채팅하는 모습]

1. server 동작 과정



이 부분은 서버가 클라이언트로부터 ChatInfo 패킷을 받았을 때이다. 250행에서 받은 정보들로 서버에 로그를 출력하고 이후 각 클라이언트에게 이 정보를 전달한다. 전달하는 부분은 위의 도형정보를 전달하는 부분과 유사하므로 생략하겠다.

2. client 동작 과정



Say 버튼을 누르거나 엔터를 누르게 되면 호출되는 이벤트 핸들러이다. 이들은 SendChatMessage() 메소드를 호출한다.

```

161 public void SendChatMessage()
162 {
163     // Send Chat Message to Server
164     ChatInfo chat = new ChatInfo();
165     chat.Type = (int)PacketType.chatInfo;
166
167     chat.clientID = txtID.Text;
168     chat.msg = childForm.txtMessage.Text;
169
170     Packet.Serialize(chat).CopyTo(sendBuffer, 0);
171     Send();
172
173     // Print Chat Log
174     string msg = txtID.Text + ": " + childForm.txtMessage.Text + "WrWn";
175     childForm.txtChat.AppendText(msg);
176
177     // Clear Chat
178     childForm.txtMessage.Text = "";
179 }

```

위에서 호출되는 메소드이다. 164행 ~ 171행은 서버에게 채팅 정보를 보낸다. 이후 174행 ~ 175행은 클라이언트 채팅 로그에 표시를 하고, 178행은 메시지 부분을 초기화해준다.

```

248 case (int)PacketType.chatInfo:
249 {
250     // Receive Chat Message from Server
251     m_chatInfoClass = (ChatInfo)Packet.Desserialize(readBuffer);
252
253     string receiveID = m_chatInfoClass.clientID;
254     string receiveMsg = m_chatInfoClass.msg;
255     string msg = receiveID + ": " + receiveMsg + "WrWn";
256
257     // Print Chat Log
258     this.Invoke((MethodInvoker)() => {
259         childForm.txtChat.AppendText(msg);
260     });
261     break;
262 }

```

이 부분은 클라이언트가 서버에게 채팅 정보를 받았을 때의 코드이다. 받은 정보를 가지고 채팅 화면에 출력을 하게 된다.

VII. 고찰

과제를 수행하는데 있어 확대, 축소, 이동을 구현하지 못한 채 과제를 제출하게 되었다. 특히 확대, 축소를 구현을 시도하는데, 마우스의 상대적인 좌표에 따라 도형들을 어떻게 그릴지 큰 어려움이 있었다. 비록 이번 과제는 그냥 제출하지만, 구현을 하다 만 주석처리 된 부분을 참고하여 꼭 다시 해결할 것이다. 이 부분을 제외하면 전체적으로 만족스럽게 과제를 수행했다고 자신한다.