4. IMPLEMENTASI SISTEM

Teori-teori dan desain sistem yang telah dibahas pada bab-bab sebelumnya diimplementasikan dalam aplikasi yang siap untuk digunakan. Implementasi yang dibahas mencakup implementasi aplikasi beserta *header* atau *class* yang telah dibuat selama pembuatan sistem. Segmen kode dijelaskan dalam bab ini.

4.1. Implementasi Aplikasi yang Digunakan

Pembuatan aplikasi Skripsi ini menggunakan perangkat lunak bahasa pemrograman Microsoft Visual C# 2005. Pemilihan perangkat lunak ini karena menyediakan *Graphical User Interface* (GUI) untuk memudahkan desain dan pemrograman berorientasi obyek.

4.2. *Library* dan *Header* yang Digunakan

Dalam pembuatan aplikasi Skripsi ini, sebagian besar proses yang ada dibuat menggunakan *library* standar yang sudah disediakan oleh Microsoft Visual C#. Berikut daftar *library* dan fungsi dalam *library* yang digunakan pada Skripsi ini antara lain :

- User32.dll
  - SetWindowsHookEx

    Digunakan untuk instalasi sebuah fungsi *hook* menjadi sebuah *hook chain*. Hal ini dilakukan untuk memonitor sistem sebuah *event*. *Event* ini bekerja dengan *thread* yg spesifik atau dengan semua *thread* di *dekstop* yang sama yang memanggil *thread* tersebut.
  - UnhookWindowsHookEx

    Digunakan untuk menghilangkan fungsi *hook* yang ada di *hook chain* oleh SetWindowsHookEx
  - CallNextHookEx

    Berguna untuk menyalurkan informasi *hook* ke fungsi *hook* berikutnya di dalam *hook chain* sekarang. Sebuah fungsi *hook*

dapat memanggil fungsi ini sebelum atau sesudah memproses informasi *hook*.

- GetCursorInfo

  Berguna untuk mendapatkan informasi tentang global kursor.

- CopyIcon

  Berguna untuk menggandakan sebuah *icon* dari modul yang lain ke modul yang sekarang.

- GetIconInfo

  Berguna untuk mendapatkan informasi tentang sebuah icon atau kursor.

- GetDesktopWindow

  Berguna untuk mengembalikan sebuah *handle* ke *desktop window*. *Desktop window* meliputi seluruh layar. *Desktop window* adalah area paling depan dari semua layar yang digambar.

- GetWindowDC

  Digunakan untuk mendapatkan *Device Context (DC)* untuk semua *window*, termasuk bar judul, menu, dan *scroll bars*.

- ReleaseDC

  Digunakan untuk melepaskan *Device Context (DC)*, membebaskan nya untuk dipakai aplikasi lain. Efek dari ReleaseDC tergantung pada tipe dari *device context*.

- GetWindowRect

  Digunakan untuk mendapatkan dimensi dari seluruh area sebuah *window*. dimensi diberikan dalam bentuk koordinat layar yaitu atas-kiri pojok layar.

- Gdi32.dll
  - BitBlt

    Digunakan untuk transfer *Bit-Block* data warna sesuai dengan sumber *device context* ke tujuan *device context*.

  - CreateCompatibleBitmap

    Digunakan untuk membentuk sebuah *bitmap* yang sesuai dengan *device* yang berhubungan dengan *device context*.

- CreateCompatibleDC

  Digunakan untuk menciptakan sebuah *memory device context* yang sesuai dengan *device* tertentu.
- DeleteDC

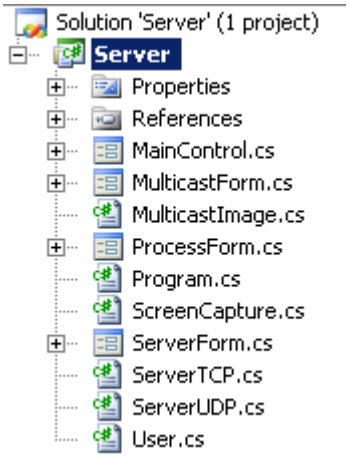
  Digunakan untuk menghapus sebuah *device context*.
- DeleteObject

  Digunakan untuk menghapus *pen*, *brush*, *font*, *bitmap*, *region* atau *pallete*, membebaskan semua sumber sistem yang berhubungan dengan objek. setelah objek di hapus, *handle* tersebut tidak valid.
- SelectObject

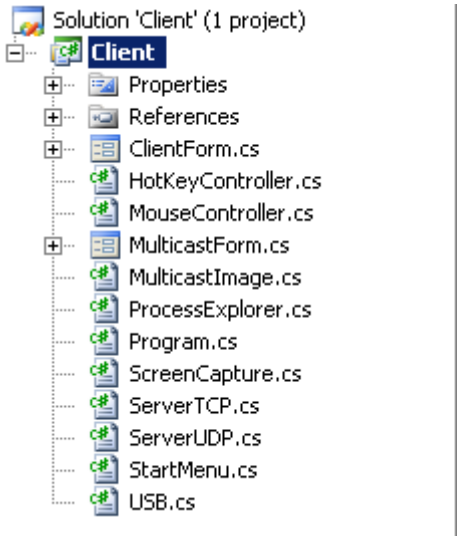
  Memilih sebuah objek menjadi sebuah *device context*. Objek yang baru menggantikan objek sebelumnya dengan tipe yang sama.

4.3. Penerapan dan Aplikasi

Pemrograman aplikasi menggunakan prinsip pemrograman berorientasi objek (*Object Oriented Programming*). Aplikasi dibagi ke dalam *class-class*, yang memiliki *member variable* dan *member function*. *Member variable* adalah bagian yang menyimpan data dari *class*, sedangkan *member function* atau *method* adalah fungsi yang mengolah data tersebut menjadi suatu *output* tertentu. Aplikasi dibagi menjadi dua bagian, yaitu *server* dan *client***Error! Reference source not found.** Gambar 4.1 menunjukkan bagan struktur *class* dari aplikasi *server*. Gambar 4.2 menunjukkan bagan struktur *class* dari aplikasi *client*.



Gambar 4.1. Bagan Struktur *Class* Aplikasi *Server*



Gambar 4.2. Bagan Struktur *Class* Aplikasi *Client*

Class-class utama yang digunakan pada aplikasi adalah:

- `ServerTCP`, yang berguna sebagai *socket* protokol TCP.
- `ServerUDP`, yang berguna sebagai *socket* protokol UDP.
- `MulticastImage`, yang berguna sebagai *socket* protokol UDP.
- `ScreenCapture`, yang berguna untuk mengambil gambar dari monitor.
- `HotKeyController`, yang berguna untuk mengendalikan *hot key* dari *keyboard.*
- `MouseController`, yang berguna untuk mengendalikan fungsi *mouse.*
- `ProcessExplorer`, yang berguna untuk mengendalikan proses yang sedang berjalan.
- `USB`, yang berguna untuk menge-*block* USB.

Pada masing-masing *class* terdapat fungsi-fungsi yang digunakan aplikasi untuk menjalankan proses yang diinginkan. Daftar fungsi-fungsi pada implementasi aplikasi yang digunakan dapat dilihat pada Tabel 4.1**Error! Reference source not found.**.

Tabel 4.1. Daftar Fungsi dan Prosedur yang Digunakan

Segmen Program	**Nama Class**	**Nama Fungsi/ Prosedur**	**Keterangan Fungsi**	**Flowchart**
4.1	ServerTCP	Listen	Untuk menerima data dari protokol	*Flowchart* 3.4

			TCP	

Tabel 4.1. Daftar Fungsi dan Prosedur yang Digunakan (sambungan)

4.2	ServerTCP	SendData	Untuk mengirimkan data ke protokol TCP	*Flowchart* 3.4
4.3	ServerUDP	Listen	Untuk menerima data dari protokol UDP	*Flowchart* 3.2
4.4	ServerUDP	SendData	Untuk mengirim data ke protokol UDP	*Flowchart* 3.2
4.5	MulticastImage	Listen	Untuk menerima data gambar	*Flowchart* 3.7
4.6	MulticastImage	SendImage	Untuk mengirim data gambar	*Flowchart* 3.3
4.7	ScreenCapture	CaptureScreen	Untuk mendapatkan gambar monitor	*Flowchart* 3.3
4.8	ScreenCapture	CaptureMouse	Untuk mendapatkan gambar *mouse*	*Flowchart* 3.3
4.9	ScreenCapture	CaptureWindow	Untuk mendapatkan gambar monitor beserta *mouse*	*Flowchart* 3.3
4.10	HotKeyController	SetKey	Untuk menghentikan fungsi *hot key keyboard*	*Flowchart* 3.7
4.10	HotKeyController	ReleaseKey	Untuk melepaskan kembali fungsi *hot key keyboard*	*Flowchart* 3.7
4.13	MouseController	SetMouse	Untuk menghentikan fungsi klik kiri *mouse*	
4.13	MouseController	ReleaseMouse	Untuk melepaskan kembali fungsi klik kiri *mouse*	
4.15	ProcessExplorer	CekNewProcess	Untuk mengenali proses baru	-
4.16	ProcessExplorer	KillProcess	Untuk menutup proses	-

Tabel 4.1. Daftar Fungsi dan Prosedur yang Digunakan (sambungan)

4.16	ProcessExplorer	GetAllProcess	Untuk membaca semua proses yang sedang berjalan	*Flowchart* 3.7
4.17	USB	DisableUSB	Untuk menghentikan fungsi USB	*Flowchart* 3.7
4.17	USB	EnableUSB	Untuk melepaskan kembali fungsi USB	*Flowchart* 3.7

4.3.1. *Class* ServerTCP

Class ServerTCP merupakan *class* yang berfungsi untuk mengirim dan menerima data yang berasal dari protokol TCP. Class ServerTCP merupakan class yang sangat penting karena banyak perintah yang harus dikirimkan melalui prokotol TCP.

Fungsi-fungsi yang ada pada *class* ini adalah ada dua, yaitu fungsi Listen dan SendData. Fungsi Listen digunakan untuk menerima data dari protokol TCP dari manapun asalkan berada pada *port* yang sama. Berikut adalah *listing code* dari kedua fungsi tersebut.

Segmen Program 4.1. *Source Code* Fungsi Listen TCP

```
public void Listen(ClientForm cf)
{
      byte[] DataReceivedByte;
      EndPoint AddressFrom = (EndPoint)TcpIpEndPoint;

      while (true)
      {
            try
            {
                  SocClient = Listener.AcceptSocket();
                  DataReceivedByte              =              new
byte[SocClient.ReceiveBufferSize];

      SocClient.ReceiveFrom(DataReceivedByte, ref AddressFrom);

                  DataReceivedText                                  =
System.Text.Encoding.ASCII.GetString(DataReceivedByte);
                  DataReceivedText = DataReceivedText.Substring(0,
DataReceivedText.IndexOf("\0"));

                  if (DataReceivedText == "…")
                  {
```

```
                }
                SocClient.Close();
                Array.Clear(DataReceivedByte,                    0,
DataReceivedByte.Length);

            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
      }
}
```

Pertama kali yang dilakukan adalah membuat sebuah *buffer* bertipe data *byte* yang digunakan untuk menerima data dari jaringan dan sebuah *EndPoint* yang digunakan untuk mengetahui alamat pengirim. *Socket* kemudian akan terus melakukan *listen* dan apabila ada data yang datang, maka akan langsung disimpan dalam *buffer* yang telah disediakan. Data yang masuk diubah menjadi *string* kemudian baru dijalankan sesuai perintah yang telah diproses sebelumnya. Terakhir adalah menutup *socket* dan membersihkan *memory* dari *buffer*. *Output* yang dihasilkan tergantung dari perintah yang diterima. Contoh apabila mendapat perintah untuk *multicast* maka akan menjalankan *multicast*.

Segmen Program 4.2. Fungsi SendData TCP

```
public void SendData(string Data, string IpDest)
{
      try
      {
            SocClient  =  new  Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
            TcpEndPoint  =  new  IPEndPoint(IPAddress.Parse(IpDest),
TcpPortDest);

            try
            {
                SocClient.Connect(TcpEndPoint);
      SocClient.Send(System.Text.Encoding.ASCII.GetBytes(Data));
                SocClient.Close();
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
      }
      catch (Exception e)
      {
            MessageBox.Show(e.Message);
      }
}
```

Pertama membuat objek *socket* untuk TCP dan membuat sebuah *End Point* sesuai dengan IP yang dituju sesuai *parameter*. Kemudian menyambungkan *socket* sesuai dengan *End Point* yang telah dibuat dan mengirim data bertipe *byte* dari *string* yang dikirimkan sebagai *parameter* dan menutup *socket. Input* yang dibutuhkan ada dua, yaitu data yang akan dikirim dalam tipe *string* dan IP tujuan dalam tipe *string.*

Tabel 4.2. Daftar perintah yang digunakan pada protokol TCP

Perintah	Keterangan
Enable Hot Key Keyboard	Mengembalikan fungsi *hot key keyboard* agar dapat digunakan kembali
Disable Hot Key Keyboard	Membatasi fungsi *hot key keyboard* agar tidak dapat digunakan
Enable Mouse	Mengembalikan fungsi *left click mouse* agar dapat digunakan kembali
Disable Mouse	Membatasi fungsi *left click mouse* agar dapat digunakan kembali
Start Multicast	Untuk menandakan bahwa sedang terjadi *multicast*
End Multicast	Untuk menandakan bahwa *multicast* sudah selesai
Process List	Mengirimkan semua proses yang terjadi kepada *server*
Close Process	Menutup proses sesuai dengan *input* yang dikirim dari *server*
Block Application	Membatasi aplikasi agar tidak dapat digunakan sesuai *input* dari *server*
Release Application	Membuat semua aplikasi bisa digunakan
Enable USB	Membatasi USB agar tidak dikenali
Disable USB	USB dapat dikenali kembali

Tabel 4.2. Daftar perintah yang digunakan pada protokol TCP (sambungan)

View Client Start	*Client* mengirimkan gambar untuk diliat *server*
View Client Stop	*Client* menghentikan pengiriman gambar
Client Multicast Start	*Client* melakukan *multicast*
Client Multicast Stop	*Client* menghentikan *multicast*
Request Join	*Client* melakukan *request* ke dalam *class*
Process List	Mendapatkan semua proses yang dilakukan *client*
Open Application	*Client* membuka sebuah aplikasi
Out Class	*Client* keluar dari *class*

4.3.2. *Class* ServerUDP

Class ServerUDP merupakan *class* yang berfungsi untuk mengirim dan menerima data yang berasal dari protokol UDP. Class ServerUDP merupakan class yang sangat penting karena banyak perintah yang harus dikirimkan melalui prokotol UDP.

Fungsi-fungsi yang ada pada *class* ini adalah ada dua, yaitu fungsi Listen dan SendData. Fungsi Listen digunakan untuk menerima data dari protokol UDP dari manapun asalkan berada pada *port* dan IP yang sama. Berikut adalah *listing code* dari kedua fungsi tersebut.

Segmen Program 4.3. Source Code Fungsi Listen UDP

```
public void Listen(ClientForm cf)
{
  int pos;
  string Code = "";
  EndPoint AddressFrom = (EndPoint)UdpIpEndPoint;
  DataReceivedByte = new Byte[100000];

  while (true)
  {
   try
   {
      UdpSocket.ReceiveFrom(DataReceivedByte, ref AddressFrom);
      DataReceivedText                                          =
```

```
System.Text.Encoding.ASCII.GetString(DataReceivedByte);

      DataReceivedText       =       DataReceivedText.Substring(0,
DataReceivedText.IndexOf("\0"));

      pos = DataReceivedText.IndexOf("#");

      if (pos > 0)
      {
        Code = DataReceivedText.Substring(0, pos);
        DataReceivedText = DataReceivedText.Remove(0, pos + 1);
      }
      if (Code == "…")
      {

      }
      Array.Clear(DataReceivedByte, 0, DataReceivedByte.Length);
   }
   catch (Exception e)
   {
      MessageBox.Show(e.Message);
   }
  }
}
```

Pertama kali yang dilakukan adalah membuat *buffer* untuk menerima data dan membuat sebuah *End Point* untuk mengetahui alamat pengirim. *Socket* kemudian akan menunggu data dari *End Point* yang telah ditentukan. Data yang diterima diubah menjadi tipe *string* dan menjalankan sesuai dengan perintah yang didapatkan. Terakhir adalah membersihkan *buffer*. *Output* yang dihasilkan tergantung dari perintah yang dikirimkan. Contoh bila mendapat perintah untuk membuat sebuah *class* maka *client* akan langsung menampilkan data *class* tersebut.

Segmen Program 4.4. *Source Code* Fungsi SendData UDP

```
public void SendData(string Data)
{
 try
 {
   byte[] tempData = System.Text.Encoding.ASCII.GetBytes(Data);
   UdpSocket.SendTo(tempData, MulticastEndPoint);

   Array.Clear(tempData, 0, tempData.Length);
 }
 catch (Exception e)
 {
   MessageBox.Show(e.Message);
 }
}
```

Pertama kali yang dilakukan adalah membuat *buffer* untuk data yang akan dikirimkan. Kemudian mengubah data tipe *string* yang akan dikirimkan

kedalam tipe *byte* dan langsung mengirimkan ke *End Point* yang telah ditentukan. *Input* yang dibutuhkan ada satu yaitu data bertipe *string* untuk dikirimkan.

Tabel 4.3. Daftar perintah yang digunakan pada protokol UDP

Perintah	Keterangan
Refresh Server	*Client* melakukan *request* kepada semua *class*
Create Class	*Server* mengirim data *class* yang telah dibuat
Close Class	*Server* menutup *class*

4.3.3. *Class* MulticastImage

Class MulticastImage merupakan *class* yang berfungsi untuk mengirim dan menerima data berupa gambar yang berasal dari protokol UDP. Class MulticastImage *class* utama yang mengambil gambar dari *server* dan mengirimkannya kepada semua *client* untuk *broadcast learning*. *Class* MulticastImage menggunakan konsep *composition* karena di dalam *class* ini menggunakan obyek dari *class* ScreenCapture.

Fungsi-fungsi yang ada pada *class* ini adalah ada dua, yaitu fungsi Listen dan SendImage. Fungsi Listen digunakan untuk menerima data berupa gambar dari protokol UDP dari *server* manapun asalkan berada pada *port* dan IP yang sama. Berikut adalah *listing code* dari kedua fungsi tersebut.

Segmen Program 4.5. *Source Code* Fungsi Listen MulticastImage

```
public void Listen(MulticastForm mf)
{
     AddressFrom = (EndPoint)UdpIpEndPoint;
     DataReceivedByte = new byte[8192];
     Image ImageReceived;
     MemoryStream ImageStream = new MemoryStream();

     while (true)
     {
          if (ClientForm.MulticastStatus == true)
          {
              try
              {
                 UdpSocket.ReceiveFrom(DataReceivedByte,       ref
AddressFrom);
                 DataReceivedText                                =
```

```
System.Text.Encoding.ASCII.GetString(DataReceivedByte);
                DataReceivedText = DataReceivedText.Substring(0,
DataReceivedText.IndexOf("\0"));

                if (DataReceivedText == "…")
                {
                }


                Array.Clear(DataReceivedByte,                    0,
DataReceivedByte.Length);
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
        }
     }
}
```

Fungsi listen pada *class* MulticastImage sebagian besar sama dengan fungsi listen pada *class* UDP. Yang membedakan adalah perintah yang digunakan di *class* MulticastImage khusus untuk data berbentuk gambar. Pada *class* ini diperiksa apakah sedang terjadi *multicast* dan apabila apabila sedang terjadi *multicast* maka fungsi di dalamnya baru dijalankan. Hal ini dilakukan untuk menghemat kinerja program yang tidak perlu. *Output* dari fungsi ini adalah menerima data dalam bentuk gambar dan menampilkannya pada *form.*

Segmen Program 4.6. *Source Code* Fungsi SendImage MulticastImage

```
public void SendImage()
{
    int position = 0;
    MemoryStream ImageStream = new MemoryStream();
    Image ImageData;

    try
    {

UdpSocket.SendTo(System.Text.Encoding.ASCII.GetBytes("Begin
Send"), MulticastEndPoint);

        ScreenCapture data = new ScreenCapture();
        ImageData = data.CaptureWindow();

        ImageData.Save(ImageStream, ImageFormat.Jpeg);

        byte[] tempData = new byte[8192];
        ImageStream.Seek(0, SeekOrigin.Begin);

        while (position < ImageStream.Length)
        {
            ImageStream.Read(tempData, 0, 8192);
```

```
            UdpSocket.SendTo(tempData, MulticastEndPoint);

            position += tempData.Length;
            System.Threading.Thread.Sleep(1);
        }


UdpSocket.SendTo(System.Text.Encoding.ASCII.GetBytes("End  Send"),
MulticastEndPoint);
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
    }
}
```

Fungsi SendImage pada *class* MulticastImage ini sebagian besar sama dengan fungsi SendData pada *class* ServerUDP. Yang membedakan adalah data yang dikirimkan dalam bentuk gambar. Data gambar didapatkan dari *class* ScreenCapture yang dibahas setelah ini dan data gambar tersebut akan dikirimkan secara terpisah karena protokol UDP tidak mampu untuk mengirim data dalam jumlah yang besar sekaligus sehingga harus dipotong-potong terlebih dahulu.

4.3.4. *Class* ScreenCapture

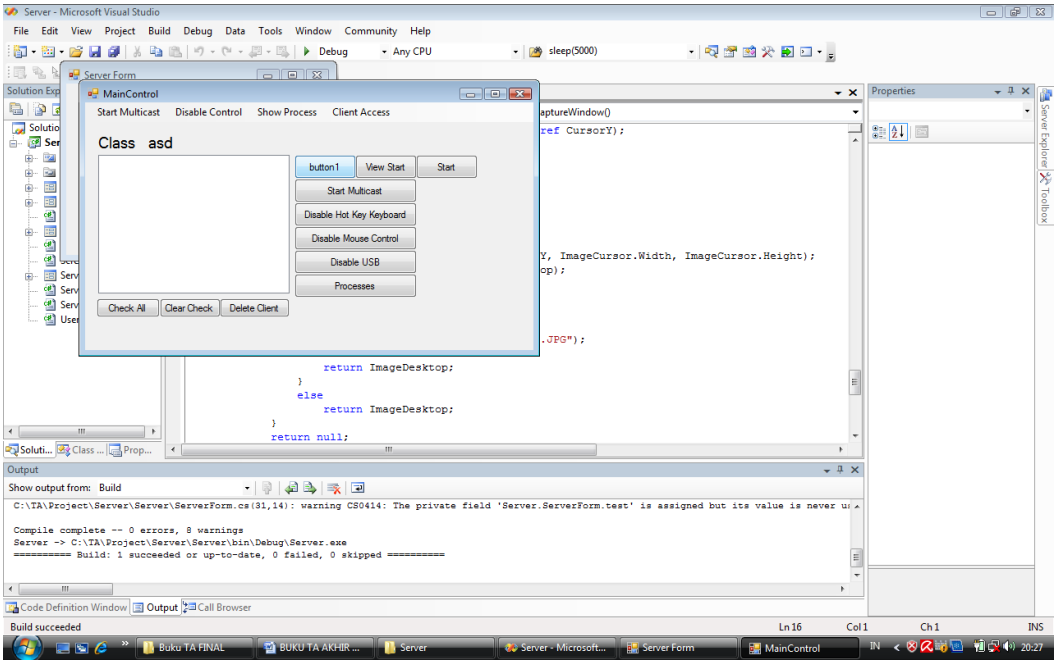
Class ScreenCapture merupakan *class* yang digunakan untuk mengambil gambar dari monitor dan menyimpannya dalam bentuk data gambar. Pada *class* ini, beberapa fungsi yang dipakai adalah CaptureScreen, CaptureMouse dan CaptureWindow. Fungsi CaptureScreen digunakan untuk mengambil gambar dari layar monitor kemudian disimpan dalam data bertipe *Image*. Berikut adalah *listing* program dari fungsi ScreenCapture.

Segmen Program 4.7. *Source Code* CaptureScreen

```
public Image CaptureScreen()
{
    IntPtr handle = User32.GetDesktopWindow();

    //get the hDC ot the target window
    IntPtr hdcSrc = User32.GetWindowDC(handle);

    //get the size
    User32.RECT WindowsRect = new User32.RECT();
    User32.GetWindowRect(handle, ref WindowsRect);
    int width = WindowsRect.right - WindowsRect.left;
    int heigth = WindowsRect.bottom - WindowsRect.top;

    //create device context we can copy to
    IntPtr hdcDest = GDI32.CreateCompatibleDC(hdcSrc);
```

```
    //create a bitmap we can copy to
    //using GetDeviceCaps to get the width/heigth
    IntPtr hBitmap = GDI32.CreateCompatibleBitmap(hdcSrc, width,
heigth);

    //select the bitmap object
    IntPtr hold = GDI32.SelectObject(hdcDest, hBitmap);

    //bitblt over
    GDI32.BitBlt(hdcDest, 0, 0, width, heigth, hdcSrc, 0, 0,
GDI32.SRCCOPY);

    //restore selection
    GDI32.SelectObject(hdcDest, hold);

    //clean up
    GDI32.DeleteDC(hdcDest);
    User32.ReleaseDC(handle, hdcSrc);

    //get .NET image object of it
    Image img = Image.FromHbitmap(hBitmap);

    //free up the bitmap object
    GDI32.DeleteObject(hBitmap);

    return img;

}
```

Pertama kali yang dilakukan adalah mendapatkan *handle* dari seluruh layar dan kemudian membuat *device context* (DC) sumber dari *handle*. Kedua adalah mendapatkan ukuran layar yang digunakan dan membuat DC tujuan yang bertipe sama dengan DC sumber. Selanjutnya membuat *bitmap* yang sesuai dengan DC yang telah disiapkan dan mem-*backup* DC sumber karena akan diubah menjadi *bitmap*. Setelah diubah, maka dilakukan proses menggambar dan mengembalikan DC sumber ke semula dan kemudian membuat data bertipe *image* dan menghapus semua DC yang telah digunakan. *Output* yang dihasilkan adalah sebuah gambar yang memiliki ukuran sama dengan layar. Untuk lebih jelas dapat dilihat pada Gambar 4.3.



Gambar 4.3. *Output* gambar dari fungsi CaptureScreen

Setelah proses pengambilan gambar monitor selesai, maka langkah berikut yang dilakukan adalah mengambil gambar dan posisi dari *pointer mouse*. Proses ini dilakukan oleh fungsi CaptureMouse. Sama seperti fungsi CaptureScreen, hasil dari fungsi ini disimpan dalam data bertipe *Image* yang berbentuk kursor sekaligus dengan posisinya. Berikut adalah listing program dari fungsi CaptureMouse.

Segmen Program 4.8. *Source Code* Fungsi CaptureMouse

```
static Image CaptureCursor(ref int x, ref int y)
{
    Image bmp;
    IntPtr hicon;
    Win32Stuff.CURSORINFO ci = new Win32Stuff.CURSORINFO();
    Win32Stuff.ICONINFO icInfo;
    ci.cbSize = Marshal.SizeOf(ci);
    if (Win32Stuff.GetCursorInfo(out ci))
    {
        if (ci.flags == Win32Stuff.CURSOR_SHOWING)
        {
            hicon = Win32Stuff.CopyIcon(ci.hCursor);
            if (Win32Stuff.GetIconInfo(hicon, out icInfo))
            {
                x = ci.ptScreenPos.x - ((int)icInfo.xHotspot);
                y = ci.ptScreenPos.y - ((int)icInfo.yHotspot);

                Icon ic = Icon.FromHandle(hicon);
                bmp = ic.ToBitmap();

                return bmp;
```

```
            }
        }
    }

    return null;
}
```

Output dari fungsi CaptureCursor adalah sebuah *icon* kursor. Untuk lebih jelasnya dapat dilihat pada Gambar 4.4.

Gambar 4.4. Output icon kursor

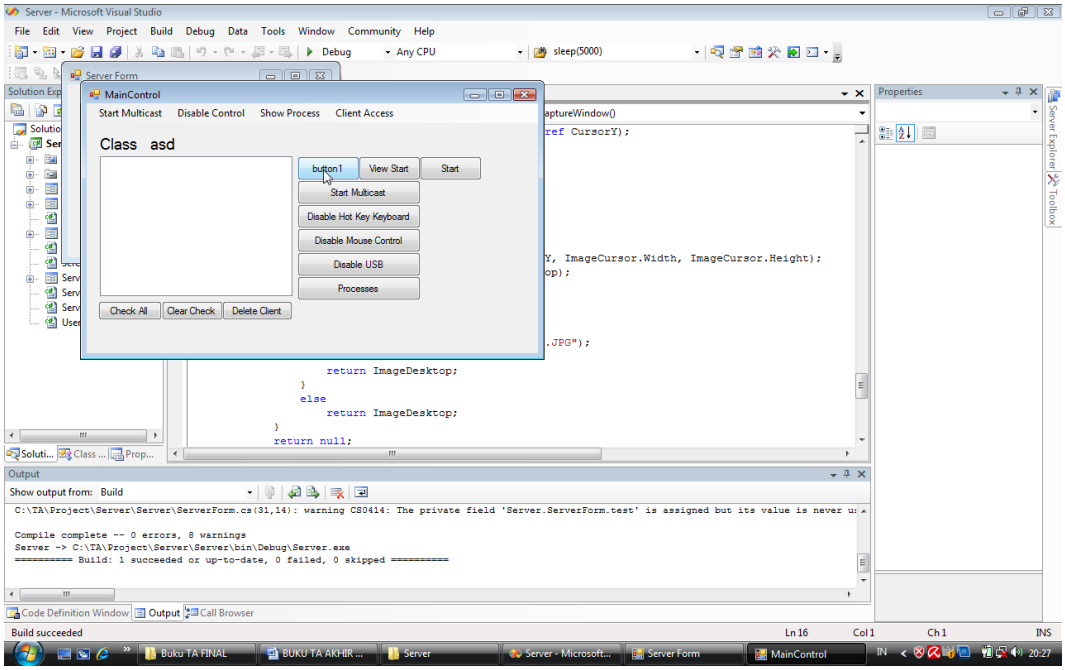
Setelah mengambil gambar monitor dan *mouse* beserta posisinya, proses selajutnya yang dilakukan adalah menggabungkan kedua data tersebut menjadi satu sehingga data *Image* menjadi tampilan monitor dengan *mouse*. Fungsi yang digunakan untuk menggabungkan dua gambar tersebut adalah CaptureWindow. Berikut adalah listing dari program CaptureWindow.

Segmen Program 4.9. *Source Code* Fungsi CaptureWindow

```
public Image CaptureWindow()
{
    int CursorX = 0;
    int CursorY = 0;
    Image ImageDesktop;
    Image ImageCursor;

    Graphics g;
    Rectangle r;

    ImageDesktop = CaptureScreen();
    ImageCursor = CaptureCursor(ref CursorX, ref CursorY);

    if (ImageDesktop != null)
    {
        if (ImageCursor != null)
        {
            r = new Rectangle(CursorX, CursorY, ImageCursor.Width,
ImageCursor.Height);
            g = Graphics.FromImage(ImageDesktop);

            g.DrawImage(ImageCursor, r);
            g.Flush();

            return ImageDesktop;
        }
        else
            return ImageDesktop;
    }
```

```
    return null;
}
```

Output gambar dari CaptureScreen akan digabung dengan CaptureCursor sehingga gambar layar akan memiliki gambar kursor. Untuk lebih jelasnya dapat dilihat pada Gambar 4.5.



Gambar 4.5. *Output* fungsi CaptureWindow

4.3.5. *Class* HotKeyController

Class HotKeyController adalah *class* yang berfungsi untuk membatasi fungsi *keyboard* dari *client*. Fungsi *keyboard* yang dihentikan adalah Alt + Tab, *Windows Key*, Ctrl + Esc, *Task Manager*, Ctrl + Shift + Esc. Fungsi yang digunakan adalah SetKey dan ReleaseKey. Fungsi SetKey digunakan untuk menghentikan dengan cara menginstalasi *hook* sedangkan fungsi ReleaseKey digunakan untuk melepaskan kembali. Berikut adalah listing dari *class* HotKeyController.

Segmen Program 4.10. *Source Code class* HotKeyController

```
public void SetKey()
{
    if (hKeyboardHook == 0)
    {
        KeyboardHookProcedure = new HookProc(KeyboardHookProc);

        hKeyboardHook = SetWindowsHookEx(WH_KEYBOARD_LL,
```

```
                    KeyboardHookProcedure,
Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly().GetModules()[
0]).ToInt32(),0);
    }
}

public void ReleaseKey()
{
    UnhookWindowsHookEx(hKeyboardHook);
    hKeyboardHook = 0;
}
```

Fungsi SetKey memanggil fungsi KeyboardHookProc. Fungsi KeyboardHookProc digunakan untuk memeriksa tombol apakah yang sedang ditekan oleh *client* dan tombol yang diperiksa adalah Alt + Tab, Ctrl + Esc, dan Window Key. Fungsi KeyboardHookProc akan mengenali segala jenis *input* dari *keyboard* dan memeriksa apakah *input* yang terjadi perlu diabaikan. Berikut adalah listing program fungsi KeyboardHookProc.

Segmen Program 4.11. *Source Code* Fungsi KeyboardHookProc

```
public int KeyboardHookProc(int nCode, int wParam, ref KeyboardStr
lParam)
{
    bool cek = false;

    switch (wParam)
    {
        case 256:
        case 257:
        case 260:
        case 261:
            if ( (lParam.vkCode == 9) || (lParam.vkCode == 27) ||
(lParam.vkCode == 91))
            {
                cek = true;
            }
        break;
    }

    if (cek == true)
    {
        return 1;
    }
    else
    {
        return  CallNextHookEx(hKeyboardHook,  nCode,  wParam,  ref
lParam);
    }
}
```

Untuk *Task Manager*, digunakan Fungsi lain, yaitu fungsi EnableTaskManager dan DisableTaskManager. Sesuai namanya, fungsi EnableTaskManager digunakan untuk memperbolehkan *Task Manager* sedangkan DisableTaskManager digunakan untuk menghentikan *Task Manager.* Untuk *Task Manager*, cara yang digunakan adalah mengubah nilai *registry* yang ada di dalam sistem operasi Window. Berikut adalah listing dari fungsi EnableTaskManager dan DisableTaskManager.

Segmen Program 4.12. *Source Code* Fungsi EnableTaskManager dan DisableTaskManager

```
public void DisableTaskManager()
{
    RegistryKey regKey;

    string keyValue = "1";
    string                    registryPolicies                    =
"Software\\Microsoft\\Windows\\Currentversion\\Policies\\System";

    try
    {
        regKey                                                        =
Registry.CurrentUser.CreateSubKey(registryPolicies);
        regKey.SetValue("DisableTaskMgr", keyValue);
        regKey.Close();
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
    }
}

public void EnableTaskManager()
{
    string                    registryPolicies                    =
"Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System";
    RegistryKey regKey;
    RegistryKey regTemp;

    try
    {
        regKey = Registry.CurrentUser;
        regTemp = regKey.OpenSubKey(registryPolicies);


        if (regTemp != null)
        {
            regKey.DeleteSubKeyTree(registryPolicies);
        }
    }
    catch (Exception e)
    {
```

```
            MessageBox.Show(e.Message);
        }
    }
```

4.3.6. *Class* MouseController

Class MouseController adalah *class* yang berfungsi untuk menghentikan fungsi klik kiri pada *mouse*. Fungsi yang digunakan hampir mirip dengan *keyboard* yaitu SetMouse dan ReleaseMouse. Fungsi SetMouse digunakan untuk menghentikan dan ReleaseMouse digunakan untuk melepaskan kembali. Berikut adalah listing dari *class* MouseController.

Segmen Program 4.13. *Source Code* SetMouse dan ReleaseMouse

```
public void SetMouse()
{
    if (hMouseHook == 0)
    {
        MouseHookProcedure = new HookProc(MouseHookProc);

        hMouseHook = SetWindowsHookEx(WH_MOUSE_LL,
                     MouseHookProcedure,
Marshal.GetHINSTANCE(Assembly.GetExecutingAssembly().GetModules()[
0]),0);
    }
}

public void ReleaseMouse()
{
    UnhookWindowsHookEx(hMouseHook);
    hMouseHook = 0;
}
```

Fungsi SetMouse memanggil fungsi MouseHookProc. Fungsi MouseHookProc berguna untuk menerima *input* dari *mouse* dan memeriksa apakah terjadi klik kiri atau tidak. Apabila terjadi klik kiri maka langsung diabaikan. Berikut adalah *listing program* fungsi MouseHookProc.

Segmen Program 4.14. *Source Code* Fungsi MouseHookProc

```
public int MouseHookProc(int nCode, int wParam, IntPtr lParam)
{
    bool cek = false;

    switch (wParam)
    {
        case WM_LBUTTONDOWN:
            cek = true;
            break;
    }
```

```
    if (cek == true)
    {
        return 1;
    }
    else
    {
        return CallNextHookEx(hMouseHook, nCode, wParam, lParam);
    }
}
```

4.3.7. *Class* ProcessExplorer

Class ProcessExplorer adalah *class* yang berguna untuk memantau proses yang sedang terjadi di *client*. Beberapa fungsi yang ada di dalam *class* antara lain CekNewProcess yang berguna untuk memeriksa apakah ada proses yang baru saja berjalan. Ketika ada proses baru, maka *client* secara otomatis akan mengirimkan pemberitahuan kepada *server* bahwa *client* telah membuka proses. Berikut adalah *listing* dari GetNewProcess.

Segmen Program 4.15. *Source Code* CekNewProcess

```
public void CekNewProcess()
{
    while (true)
    {
        if (ClientForm.BlockStatus == false)
        {
            System.Threading.Thread.Sleep(1);

            ProcList                                                =
Process.GetProcesses(Environment.MachineName);

            foreach (Process proc in ProcList)
            {
                if
(ClientForm.BlockList.Contains(proc.ProcessName) == true)
                {
                    proc.Kill();
                    MessageBox.Show(proc.ProcessName   +   "   is
Blocked");
                }
            }

            ProcList                                                =
Process.GetProcesses(Environment.MachineName);
            CountProcNow = ProcList.Length;

            if (CountProcNow > CountProcTemp)
            {
                CountProcTemp = CountProcNow;

                foreach (Process pl in ProcList)
                {
```

```
                    bool Cek = false;
                    foreach (Process pt in ProcTemp)
                    {
                        if (pl.Id == pt.Id)
                        {
                            Cek = true;
                        }
                    }

                    if (Cek == false)
                    {
                        if (ClientForm.IpServerDest != "")
                        {

ClientForm.TcpSocket.SendData("Open         Application#"         +
pl.ProcessName, ClientForm.IpServerDest);
                        }
                    }
                }

                ProcTemp = ProcList;
            }
            else
            {
                CountProcTemp = CountProcNow;
                ProcTemp = ProcList;
            }
        }
    }
}
```

Proses yang dilakukan oleh fungsi CekNewProcess adalah membandingkan setiap saat apakah ada proses yang baru dijalankan dengan cara menyimpan proses yang sebelumnya dilakukan dan membandingkan dengan proses yang sekarang dan memeriksa proses apa yang tidak ada dengan sebelumnya yang berarti adalah proses yang baru dijalankan dan langsung mengirimkannya kepada *server.* Pada fungsi ini juga dilakukan pemeriksaan apakah proses yang baru dijalankan ada dalam daftar proses yang tidak diperbolehkan. Apabila tidak diperbolehkan maka proses tersebut akan langsung ditutup dan tidak bisa digunakan sampa *server* menghendaki untuk dapat digunakan kembali.

Fungsi lainnya adalah GetAllProcess dan KillProcess. Fungsi GetAllProcess berguna untuk mendapatkan semua proses yang sedang berjalan pada *client* dan akan dikirimkan kepada *server.* Fungsi KillProcess berguna untuk mematikan proses sesuai dengan *input* yang telah diterima. Berikut adalah *listing program* fungsi GetAllProcess dan KillProcess :

Segmen Program 4.16. *Source Code* GetAllProcess dan KillProcess

```
public string GetAllProcess()
{
    ProcList = Process.GetProcesses(Environment.MachineName);
    string temp="Process List";

    foreach (Process proc in ProcList)
    {
        temp = temp + "#" +proc.ProcessName;
    }

    return temp;
}

public void KillProcess(string ProcName)
{
    ProcList = Process.GetProcesses(Environment.MachineName);

    foreach (Process proc in ProcList)
    {
         if (proc.ProcessName.Equals(ProcName))
         {
             proc.Kill();
         }
     }
}
```

Fungsi GetAllProcess akan mengembalikan semua proses yang dilakukan oleh *client* dalam bentuk *string* dan dikirimkan kepada *server* untuk diperhatikan lebih lanjut dan fungsi KillProcess digunakan untuk menutup suatu proses.

4.3.8. *Class* USB

Class USB berguna untuk menghentikan fungsi USB dan mengembalikan fungsi USB. Fungsi yang digunakan dalam *class* ini adalah EnableUSB dan DisableUSB. Berikut adalah *listing program* dari kedua fungsi tersebut :

Segmen Program 4.17. *Source Code* DisabelUSB dan EnableUSB

```
public void DisableUSB()
{
     RegKey = Registry.LocalMachine.OpenSubKey(UsbDest, true);
     RegKey.SetValue("Start", 4);
}

public void EnableUSB()
{
    RegKey = Registry.LocalMachine.OpenSubKey(UsbDest, true);
    RegKey.SetValue("Start", 3);
}
```

Fungsi DisableUSB dan EnableUSB sama-sama menggunakan *registry* untuk memperbolehkan atau melarang USB untuk dikenali. Nilai yang digunakan ada dua yaitu angka 4 untuk *disable* dan 3 untuk *enable*. Untuk USB, ada dua kondisi, yang pertama apabila fungsi DisableUSB sudah diaktifkan maka USB tidak bisa digunakan dan kondisi yang kedua apabila sudah ada USB yang terdeteksi dan kemudian fungsi DisableUSB dipanggil, maka USB tersebut tetap dikenali kecuali USB tersebut dilepas dan ditancapkan kembali.