VIRTUAL NETWORK COMPUTING

Project Report P-1197

Submitted in partial fulfillment of the
Requirement for the award of the degree of the

**Bachelor of Computer Science and Engineering of
Bharathiar University, Coimbatore.**

Submitted by

**P.Saravana kumar
0027K0199**

**S.Sasidharan
0027K0200**

Under the guidance of

**Mrs. V. Vanitha M.E.,
Senior Lecturer**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**KUMARAGURU COLLEGE OF TECHNOLOGY,
COIMBATORE – 641006.**

MARCH 2004.

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to Bharathiar University, Coimbatore)

CERTIFICATE

This is to certify that the project entitled

VIRTUAL NETWORK COMPUTING

Is done by

P.Saravana Kumar
0027K0199

S.Sasidharan
0027K0200

And submitted in partial fulfillment of the
Requirement for the award of the degree of the

Bachelor of Computer Science and Engineering of
Bharathiar University, Coimbatore.

S. Thangasamy

Professor & Head of the department
(Dr.S.THANGASAMY)

[illegible] 17/3/04

Project Guide
(Mrs.V.VANITHA)

Certified that the candidates were examined by us in the project work
Viva voce examination held on 23-3-2004.

[illegible] 23/3/'04

Internal Examiner

[illegible] 23/3

External Examiner

DECLARATION

Declaration

We,

P.Saravana kumar **0027K0199**

S.Sasisharan **0027K0200**

declare that the project entitled "Virtual Network Computing" is done by us and to the best of our knowledge, a similar work has not been submitted earlier to the Bharathiar University or any other institution, for fulfillment of the requirement of the course study.

This project report is submitted on the partial fulfillment of the requirement for all awards of the degree of Bachelor of Computer Science and Engineering of Bharathiar University.

NAME	REGISTER NUMBER	SIGNATURE
P.Saravana kumar	0027k0199	[illegible]
S.Sasidharan	0027k0200	S. Sasidharan

Countersigned: [illegible] 17/3/04

Staff in charge: Mrs.V.Vanitha M.E,
Senior Lecturer,
Department of Computer Science & Engineering,
Kumaraguru college of Technology,
Coimbatore.

Place: Coimbatore.

Date: 23-3-2004

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

The exhilaration achieved upon the successful completion of any task should be definitely shared with the people behind the venture. This project is an amalgam of study and experience of many people without whose help this project would not have taken shape.

At the onset, I take this opportunity to thank the management of my college for having provided us excellent facilities to work with. I express my deep gratitude to our Principal Dr.K.K.Padmanabhan B.Sc (Engg), M.Tech. for ushering us in the path of triumph.

I am always thankful to our beloved Professor and the Head of the Department, Prof.S.Thangasamy B.E.(HONS)., whose consistent support and enthusiastic involvement helped us a great deal.

I am greatly indebted to my beloved guide Mrs.V.Vanitha M.E., Senior Lecturer, Department of Computer Science and Engineering for her excellent guidance and timely support during the course of this project. As a token of my esteem and gratitude, I honor her for her assistance towards this cause.

I also thank my project coordinator Mrs.D.Chandrakala M.E., Senior Lecturer and my beloved class advisor Mrs.M.S.Hema B.E., for their invaluable assistance.

I also feel elated in manifesting my deep sense of gratitude to all the staff and lab technicians in the Department of Computer Science and Engineering.

I feel proud to pay my respectful thanks to my Parents for their enthusiasm and encouragement and also I thank my friends who have associated themselves to bring out this project successfully.

SYNOPSIS

SYNOPSIS

Virtual Network Computing is a remote display system which allows you to view a computing 'desktop' environment not only on the machine it is running, but from anywhere on the network. It provides cross-platform remote control over other computer sessions.

Features of the system include the ability for several users to remotely access the same desktop simultaneously - if one of them moves the mouse, starts a program or presses a key, the displays of all the other viewers will also change. You can set your desktop to read-only (i.e., the remote user can view it but not do anything with the mouse or keyboard).

Two main modules involved in the systems are

1. Server
2. Viewer

Server runs on a system whose desktop has to be captured. Viewer runs on a system from where remote system has to be controlled.

Function of the Server is to accept the request from clients and after authentication, it captures the desktop image of the system where server is running, convert it to array of pixels and transfer it to viewer. Whereas viewer reproduce the desktop image from the received pixels. When any key or mouse event occurs on the frame where desktop image is reproduced, it is captured and sent to server. Server handles these events and reflects it on the desktop. Then server sends updated desktop image to the client.

RFB (remote frame buffer) protocol is used which is a simple protocol for remote access to graphical user interfaces. Because it works at the framebuffer level it is applicable to all windowing systems and applications.

CONTENTS

CONTENTS

INTRODUCTION

1. INTRODUCTION

Our project deals with the capturing of remote system desktop in a network and performs operations on the remote desktop. Here RFB protocol is used which is a simple protocol for remote access to graphical user interfaces.

Encoding process is used to improve the performance. Authentication process, server and client initialization are carried out. It involves handshaking messages like pixel format to be used and frame buffer size.

1.1 Existing system and limitations:

There are number of VNC Systems which provide remote desktop access in a network but provides it only on machines running in same platform.

1.2 Proposed system and advantages:

The ultimate solution for the above problems is to code the entire image processing functions using high-level languages as individual modules and to use them as and when needed. This will provide the following advantages.

- It is small and simple. Server or a Viewer program can be run directly from a floppy. There is no installation needed.
- It is truly platform-independent. A desktop running on a Linux machine may be displayed on a PC running in windows. The simplicity of the protocol makes it easy to port to new platforms. We have a Java viewer, which will run in any Java-capable browser.
- It is sharable. One desktop can be displayed and used by several viewers at once.

SOFTWARE REQUIREMENT ANALYSIS

2. SOFTWARE REQUIREMENT ANALYSIS

System study is an activity that encompasses most of the tasks that we have collectively called computer system engineering. System study is conducted with the following objectives.

- Identify the needs.
- Evaluate the system concept for feasibility.
- Perform economic and technical analysis.
- Allocate function to hardware, software, people and other system elements.
- Create a system definition that forms the foundation for all subsequent engineering works.

2.1 Product Definition:

VNC is a remote display system used to view remote desktop on the network. It provides cross-platform remote desktop access. It consist of two sections namely server and viewer. Once viewer is connected to server, it is possible to open any application, save files and any operation using viewer on the server. More than one client can view same desktop simultaneously.

2.2 Project plan:

The project mainly aims at developing system to provide cross-platform remote desktop access in a network. A request is made from viewer for connection which includes authentication process. Then desktop image is captured, converted to pixels and sent to the viewer. In the same time, any mouse or key event occurred in client side must be captured and sent to the server along with its state. These events are appropriately reflected in server side.

SOFTWARE REQUIREMENT SPECIFICATION

3. SOFTWARE REQUIREMENT SPECIFICATION

3.1 Purpose:

The primary purpose of the Software Requirements Specification (SRS) is to document the previously agreed to functionality, external interfaces, attributes, and the performance of the **VIRTUAL NETWORK COMPUTING.** This specification is the primary document upon which all of the subsequent design source code, and test plan will be based.

3.2 Scope:

The scope of this document, Software Requirement Specification (SRS) is to describe the requirements definition effort. The SRS documentation for Virtual Network Computing describes the functions, external interfaces, attributes, and performance issues specified in the product definition.

3.3 Product overview and summary:

My product provides a reliable, robust and efficient means of capturing remote desktop. The project basically involves server program which captures desktop image, transfers it to viewer where it is reproduced. Viewer also has to send key and mouse events to server where it is reflected. Viewer always request updated desktop image.

3.4 Development and Operating Environment:

The development environment gives the minimum hardware and software requirements.

Hardware Specification:

Processor – Pentium III

RAM – 64 MB

Hard Disk – 10 GB

Floppy Drive – 1.44 MB

Monitor – 14" Monitor

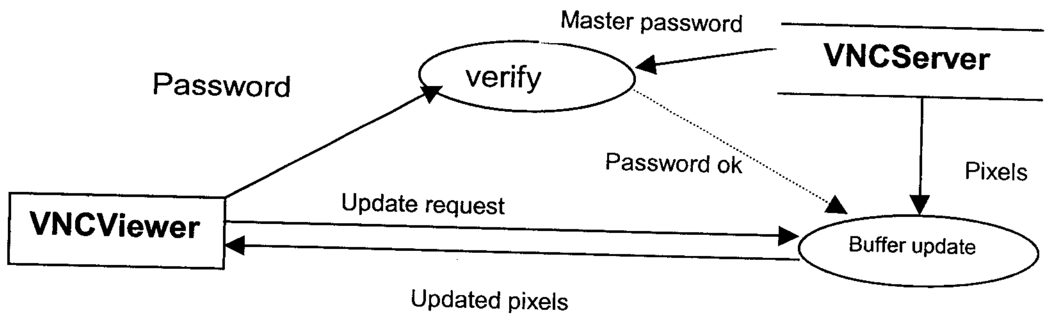
Software Specification:

Operating System – Windows 98, Linux.

Platform – Java

3.5 External interfaces and Dataflow:

The external interface includes frame to receive password in Viewer part where as in server side a user input is obtained for the master password and a option to give the desktop a read only or full access.



3.6 Functional specifications:

The description of the modules is as follows:

3.6.1. Authentication – password received from viewer is checked against master password for security purpose and to avoid any malicious act. If the password provided is correct, then client request are processed or else the connection is disconnected.

3.6.2. Encoding process – Encoding handled is Raw encoding type. In Raw encoding, pixel values in left-to-right scan line order are send to the viewer.

3.6.3. Client messages:

FrameBufferUpdate - Notifies the server that the client is interested in the area of the frame buffer specified by x-position, y-position, width and height. The server usually responds to a FramebufferUpdateRequest by sending a FramebufferUpdate.

KeyEvent - when any key is pressed, 'keysym' values and state of the key is send to the server where appropriate event is generated. Modifier keys such as Control and Alt should be taken as modifying the interpretation of other keysyms. Note that there are no keysyms for ASCII control characters such as ctrl-a - these should be generated by viewers sending a Control press followed by an 'a' press.

Pointer Event - Indicates either pointer movement or a pointer button press or release. The pointer location and button mask is send to the server where appropriate action is generated.

3.6.4. Desktop capturing – whenever framebufferupdate request is received from viewer, this module is invoked. Here desktop image is captured and its pixel values are retrieved and selected encoding process is used. Then the pixel values are sent to the viewer.

3.6.5. Event Handling - This function handles mouse and key events occurred in the client side. In case of key event, key code and state of the key is received, according to which appropriate action is performed. Similarly mouse actions are performed according to state of the mouse button received.

3.7 Exception handling:

When wrong password is provided then appropriate warning message will be displayed during authentication. During connection, host name and port number send as a arguments in a viewer section must be a valid one. Else connection refused message will be displayed.

IMPLEMENTATION OF PROPOSED SYSTEM

4. IMPLEMENTATION OF PROPOSED SYSTEM

4.1 Initialization:

Initial interaction between the RFB client and server involves a negotiation of the pixel format with which pixel data will be sent.

4.2 Pixel Format:

Pixel format refers to the representation of individual colours by pixel values. The most common pixel formats are 24-bit or 16-bit “true colour”, where bit-fields within the pixel value translate directly to red, green and blue intensities and 8-bit “colour map” where an arbitrary mapping can be used to translate from pixel values to the RGB intensities.

Server-pixel-format specifies the server’s natural pixel format. This pixel format will be used unless the client requests a different format using the SetPixelFormat message.

Bits-per-pixel is the number of bits used for each pixel value on the wire. This must be greater than or equal to the depth which is the number of useful bits in the pixel value. Big-endian-flag is non-zero (true) if multi-byte pixels are interpreted as big endian.

If true-colour-flag is non-zero (true) then the last six items specify how to extract the red, green and blue intensities from the pixel value. Red-max is the maximum red value. Red-shift is the number of shifts needed to get the red value in a pixel to the least significant bit. Green-max, green-shift and blue-max, blue-shift are similar for green and blue.

Setpixelformat sets the format in which pixel values should be sent in FramebufferUpdate messages. If the client does not send a SetPixelFormat message then the server sends pixel values in its natural format as specified in the ServerInitialisation message.

4.3 Framebufferupdate:

The server usually responds to a FramebufferUpdateRequest by sending a FramebufferUpdate. Here server captures desktop image using createscreencapture () and pixel values are retrieved using PixelGrabber class. Then these pixel values are transferred to viewer where color component of pixel values are obtained. These are displayed on the frame to show desktop image.

4.4 Event Handling:

A key press or release events are captured. Down-flag variable is non-zero (true) if the key is now pressed, zero (false) if it is now released. The key itself is specified using the "keysym" values. For most ordinary keys, the "keysym" is the same as the corresponding ASCII value. In the server side, according to state of the key keypress () or keyrelease () functions are invoked to perform appropriate key event. "keysym" values of some common keys are

Backspace – 0xff08

Tab – 0xff09

Escape – 0xff1b

Page Up – 0xff55

Page Down – 0xff56

F1 – 0xffbe

Shift (left) – 0xffe1

Alt (left) – 0xffe9

Control (left) – 0xffe3

A pointer event indicates either pointer movement or a pointer button press or release. The pointer location (x-position, y-position), and the current state of button are passed to server where mousepress () or mouserelease () are invoked.

SYSTEM DESIGN

5. SYSTEM DESIGN

The system design phase is an interesting phase. It provides the way the information is to be fed and how the output is to be obtained. The design goes through logical and physical stages of development. Logical stage involves preparing the input and output preparation of the system. The physical stage details the hardware and the system implementations.

5.1 Input Design:

The input in the viewer module are host name and port number as a command line argument, to which the client should be connected. Then in the authentication part, password must be provided in the frame displayed.

5.2 Output Design:

The output of this project is an image of the desktop displayed in a frame. When any key or mouse event happens in frame, it is send to the server where these actions are performed and the updated image is send back to the viewer. Output frame consist of a button named “disconnect” using which client gets disconnected.

SYSTEM TESTING

6. SYSTEM TESTING

Testing is an activity to verify that a correct system is being built and is performed with the intent of finding faults in the system. Testing is an activity, however not restricted to being performed after the development phase is complete. But this is to be carried out in parallel with all stages of system development, starting with requirement specification. Testing results once gathered and evaluated, provide a qualitative indication of software quality and reliability and serve as a basis for design modification if required.

System Testing is a process of checking whether the development system is working according to the original objectives and requirements. The system should be tested experimentally with test data so as to ensure that the system works according to the required specification. When the system is found working, test it with actual data and check performance.

Here, initially working of Authentication process is checked by giving valid and invalid password and looking for warning messages when invalid password is given.

After successful authentication, desktop image is captured from server. To check whether updated desktop image keep coming from server for each update request, various activities like minimizing windows, opening any application are done on the server. These changes must be reflected on viewer side.

Any key or mouse action is performed on the desktop captured on viewer and look for those actions to be reflected on the server system, in order to verify that ensure that each key or mouse event message from the client side is correctly transferred and handled by server.

FUTURE ENHANCEMENTS

7. FUTURE ENHANCEMENTS

Virtual Network Computing provides various features like using remote system's resource and acts as a monitoring system. It provides cross-platform remote desktop access in a network.

In this project, encryption can be used in the authentication process. In order to increase the security further, all the data transferred between server and client can be encrypted using suitable encryption techniques.

CopyRect and RRE encoding can be used to improve the performance. Performance can also be improved by compressing the desktop image data. It reduces time consumption in the transfer.

CONCLUSION

8. CONCLUSION

The complete design and development of the system **"VIRTUAL NETWORK COMPUTING"** is presented in this dissertation. The system has user-friendly features. It is possible for any user to use this system.

The programming techniques used in the design of the system provide a scope for further expansion and implementation of any changes, which may occur in the future. The system has been tested by connection with many system and they provide satisfactory performance.

This system is developed with the specifications and abiding by the existing rules and regulations of the company.

Since the requirements of any organization and their standards are changing day to day the system has been designed in such a way that its scope and boundaries could be expanded in future with little modification. As a further enhancement this system can be integrated with any other system.

This system has been developed using JAVA. The main aim behind the development of this system is to provide cross-platform remote desktop access in a network.

BIBLIOGRAPHY

9. BIBLIOGRAPHY

- Herbert Schildt, "**Java 2: The Complete Reference**", Tata McGraw Hill Edition, 2002, 5th Edition.

Websites visited :

- www.search.cpan.org visited on 30th November, 2003.
- www.realvnc.org visited on 19th December, 2003
- www.developvnc.org visited on 13th January, 2004.
- www.rfbproto.org visited on 19th January, 2004.

ANNEXURE

10. ANNEXURE

10.1 Sample Code:

10.1.1 viewer code:

```
import java.awt.*;
import java.io.*;

public class vncviewer extends java.applet.Applet
                        implements java.lang.Runnable
{
 boolean inAnApplet = true;
 public static void main(String[] argv) {
  vncviewer v = new vncviewer();
  v.mainArgs = argv;
  v.inAnApplet = false;
  v.f = new Frame("VNC");
  v.f.add("Center", v);
  v.init();
  v.start();
 }
public void init() {
  readParameters();
  options = new optionsFrame(this);
  rfbThread = new Thread(this);
  rfbThread.start();
}
public void run() {

 gridbag = new GridBagLayout();
 setLayout(gridbag);

  disconnectButton = new Button("Disconnect");
 disconnectButton.disable();
 buttonPanel.add(disconnectButton);
    try {
  doProtocolInitialisation();
  vncCanvas vc = new vncCanvas(this);
  gbc.weightx = 1.0;
  gbc.weighty = 1.0;
  gridbag.setConstraints(vc,gbc);
  add(vc);
  disconnectButton.enable();
  vc.processNormalProtocol();
  } catch (Exception e) {
    e.printStackTrace();
    fatalError(e.toString());  }

 void doProtocolInitialisation() throws IOException {
   System.out.println("sending client init");
   rfb.writeClientInit();
   rfb.readServerInit();
   System.out.println("Desktop name is " + rfb.desktopName);
   System.out.println("Desktop size is " + rfb.framebufferWidth + " x " +
                      rfb.framebufferHeight);
   setEncodings();
 }
```

Thread to reproduce image and handle I/O events:

```
  public void processNormalProtocol() throws IOException {

  rfb.writeFramebufferUpdateRequest(0, 0, rfb.framebufferWidth,
                                    rfb.framebufferHeight, false);

  sg = getGraphics();

  needToResetClip = false;
  while (true) {
    int msgType = rfb.readServerMessageType();

    switch (msgType) {
    case rfbProto.FramebufferUpdate:
        rfb.readFramebufferUpdate();

        for (int i = 0; i < rfb.updateNRects; i++) {
          rfb.readFramebufferUpdateRectHdr();

          if (needToResetClip &&
              (rfb.updateRectEncoding != rfbProto.EncodingRaw)) {
            try {
              sg.setClip(0, 0, rfb.framebufferWidth, rfb.framebufferHeight);
              pig.setClip(0, 0, rfb.framebufferWidth, rfb.framebufferHeight);
            } catch (NoSuchMethodError e) {
            }
            needToResetClip = false;
          }
        case rfbProto.EncodingRaw:
          drawRawRect(rfb.updateRectX, rfb.updateRectY,
                      rfb.updateRectW, rfb.updateRectH);
          break;


                      rfb.updateRectEncoding);

      }
      rfb.writeFramebufferUpdateRequest(0, 0, rfb.framebufferWidth,
                                        rfb.framebufferHeight, true);
      break;


   default:
      throw new IOException("Unknown RFB message type " + msgType);
   }
  }
}


void drawRawRect(int x, int y, int w, int h) throws IOException {
 if (v.options.drawEachPixelForRawRects) {
   for (int j = y; j < (y + h); j++) {
      for (int k = x; k < (x + w); k++) {
        int pixel = rfb.is.read();
        sg.setColor(colors[pixel]);
        sg.fillRect(k, j, 1, 1);
        pig.setColor(colors[pixel]);
        pig.fillRect(k, j, 1, 1);
      }
   }
   return;
 }

 for (int j = y; j < (y + h); j++) {
   rfb.is.readFully(pixels, j * rfb.framebufferWidth + x, w);
 }

 amis.newPixels(x, y, w, h);

 try {
   sg.setClip(x, y, w, h);
   pig.setClip(x, y, w, h);
   needToResetClip = true;
  } catch (NoSuchMethodError e) {
    sg2 = sg.create();
    sg.clipRect(x, y, w, h);
    pig2 = pig.create();
    pig.clipRect(x, y, w, h);
  }

  sg.drawImage(rawPixelsImage, 0, 0, this);
  pig.drawImage(rawPixelsImage, 0, 0, this);

  if (sg2 != null) {
    sg.dispose();   // reclaims resources more quickly
    sg = sg2;
    sg2 = null;
    pig.dispose();
    pig = pig2;
    pig2 = null;
  }
}


public boolean handleEvent(Event evt) {
  if ((rfb != null) && rfb.inNormalProtocol) {
    try {
        switch (evt.id) {
        case Event.MOUSE_MOVE:
        case Event.MOUSE_DOWN:
        case Event.MOUSE_DRAG:
        case Event.MOUSE_UP:
          if (v.gotFocus) {
            requestFocus();
          }
          rfb.writePointerEvent(evt);
          break;
        case Event.KEY_PRESS:
        case Event.KEY_RELEASE:
        case Event.KEY_ACTION:
        case Event.KEY_ACTION_RELEASE:
          rfb.writeKeyEvent(evt);
          break;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return true;
  }
```

10.1.2 server code :

```
package vnc;

import rfb.server.*;

import java.util.*;

public class VNCHost
{

        public static void main( String args[] )
        {
                String key, serverClassName, displayName, password = null,
                restrictedTo = null, noPasswordFor = null;
                Class serverClass;
                int display, width, height;
                display=50;
                height=600;
                width=800;
                displayName=new String("Robot");
                serverClassName=new String("vnc.awt.VNCRobot");
                try
                {
                        serverClass = Class.forName( serverClassName );
                        new RFBHost( display, displayName, serverClass, width,
                        height, new DefaultRFBAuthenticator( password,
                        restrictedTo, noPasswordFor ) );
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Thread to handle Client Messages

```
public void run()
{
        try
        {
                writeProtocolVersionMsg();
                readProtocolVersionMsg();
                if( !authenticator.authenticate( this ) )
```

```
                        throw new Throwable();
                readClientInit();
                initServer();
                writeServerInit();

                while( true )
                {
                        int b = input.readUnsignedByte();
                        switch( b )
                        {

                        case rfb.FrameBufferUpdateRequest:
                                readFrameBufferUpdateRequest();
                                if( isLocal )
                                        Thread.sleep( 200 );
                                break;
                        case rfb.KeyEvent:
                                readKeyEvent();
                                break;
                        case rfb.PointerEvent:
                                readPointerEvent();
                                break;
                        default:
                                System.err.println(b);
                        }
                }
        }
        catch( Throwable x )
        {
        }
}
```

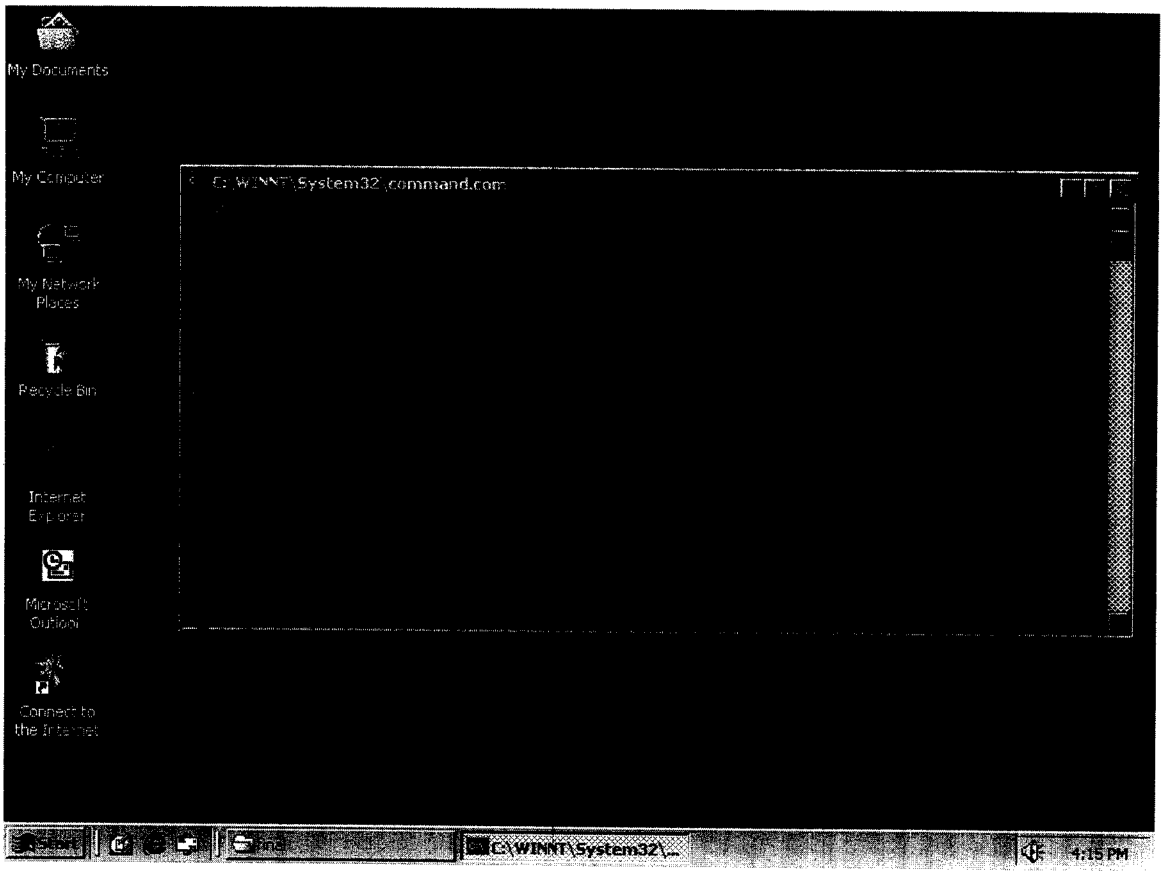
Desktop Capturing

```
package vnc.awt;

import rfb.*;
import rfb.server.*;
import java.io.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;

public class VNCRobot extends Component implements RFBServer
{
        public VNCRobot( int display, String displayName, int width, int height )
        {
                this.displayName = displayName;
                device = GraphicsEnvironment .getLocalGraphicsEnvironment().
                getDefaultScreenDevice();
                try
                {
                        robot = new Robot( device );
                }
                catch( AWTException x )
                {
                }
        }

        public int getFrameBufferWidth( RFBClient client )
        {
                return device.getDefaultConfiguration().getBounds().width;
        }

        public int getFrameBufferHeight( RFBClient client )
        {
                return device.getDefaultConfiguration().getBounds().height;
        }


public void setPixelFormat( RFBClient client, PixelFormat pixelFormat ) throws
IOException
{
        pixelFormat.setDirectColorModel( (DirectColorModel)
        Toolkit.getDefaultToolkit().getColorModel() );
}


public void frameBufferUpdateRequest( RFBClient client, boolean incremental,
int x, int y, int w, int h ) throws IOException
{
                if( incremental )
                        return;

        BufferedImage image = robot.createScreenCapture( new Rectangle( x, y,
        w, h ) );

        Rect r = Rect.encode( client.getPreferredEncoding(),
        client.getPixelFormat(), image, x, y, w, h );

                Rect[] rects = { r };
                try
                {
                        client.writeFrameBufferUpdate( rects );
                }
                catch( IOException xx )
                {
                        xx.printStackTrace();
                }
        }

public void keyEvent( RFBClient client, boolean down, int key ) throws
IOException
{
                int[] vk = new int[2];
                keysym.toVKall( key, vk );
                if( vk[0] != KeyEvent.VK_UNDEFINED )
                {
                        if( down )
                                robot.keyPress( vk[0] );
                        else
                                robot.keyRelease( vk[0] );
                }
}

public void pointerEvent( RFBClient client, int buttonMask, int x, int y ) throws
IOException
{
                int newMouseModifiers = 0;
                if( ( buttonMask & rfb.Button1Mask ) != 0 )
                        newMouseModifiers |= MouseEvent.BUTTON1_MASK;
                if( ( buttonMask & rfb.Button2Mask ) != 0 )
                        newMouseModifiers |= MouseEvent.BUTTON2_MASK;
                if( ( buttonMask & rfb.Button3Mask ) != 0 )
                        newMouseModifiers |= MouseEvent.BUTTON3_MASK;
                if( newMouseModifiers != mouseModifiers )
                {
                        if( mouseModifiers == 0 )
                        {
                                robot.keyPress( newMouseModifiers );
                        }
                        else
                        {
                                robot.keyRelease( newMouseModifiers );
                        }

                        mouseModifiers = newMouseModifiers;
```

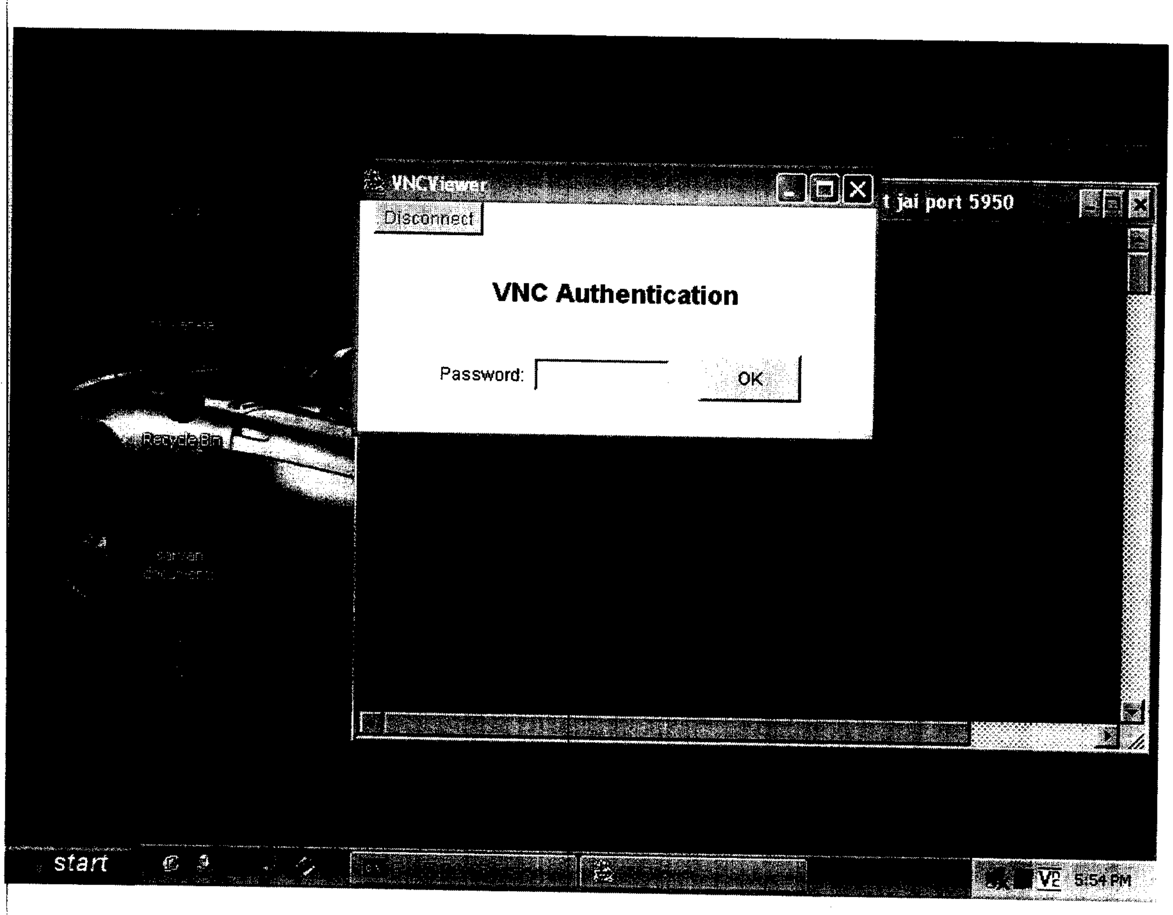
```
            }

            robot.mouseMove( x, y );
        }

        private String displayName;
        private GraphicsDevice device;
        private Robot robot;
        private int mouseModifiers = 0;
}
```

10.2 OUTPUT SCREENS :

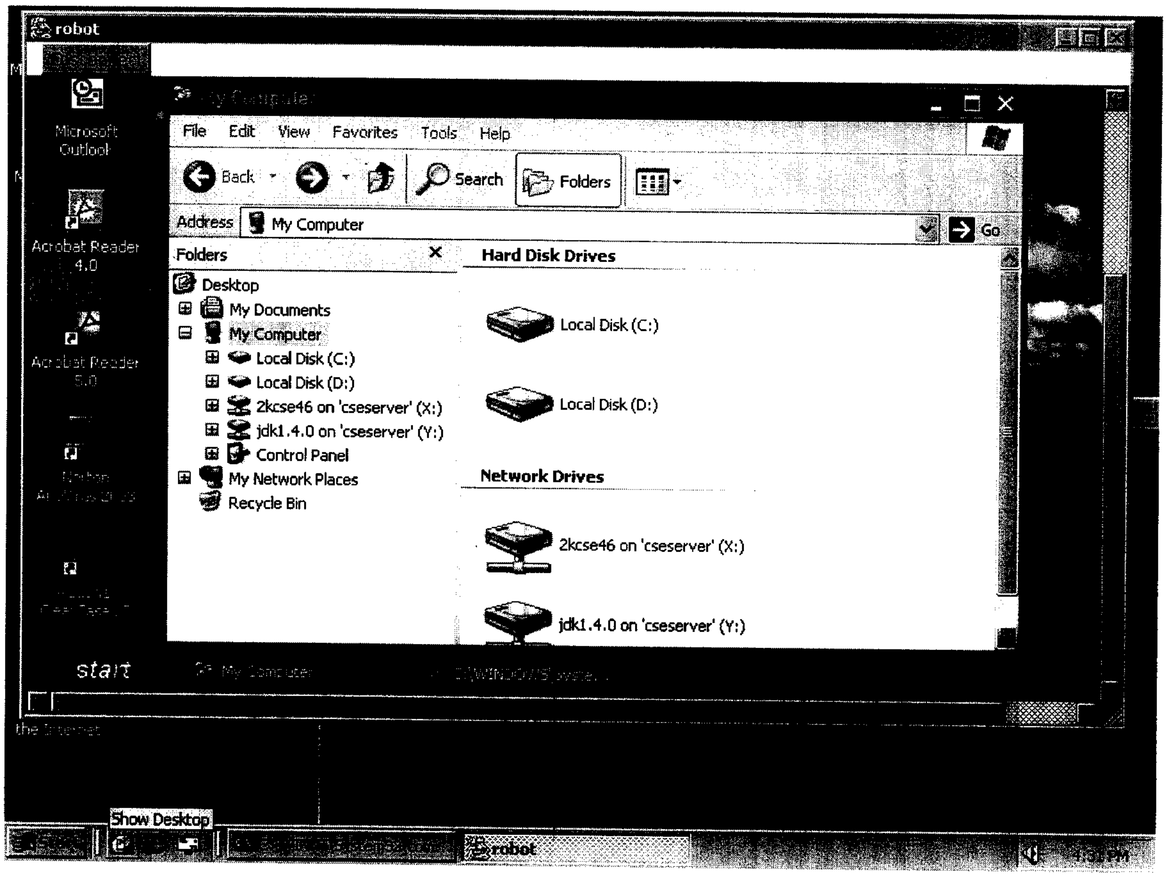
10.2.1 VNCSERVER :



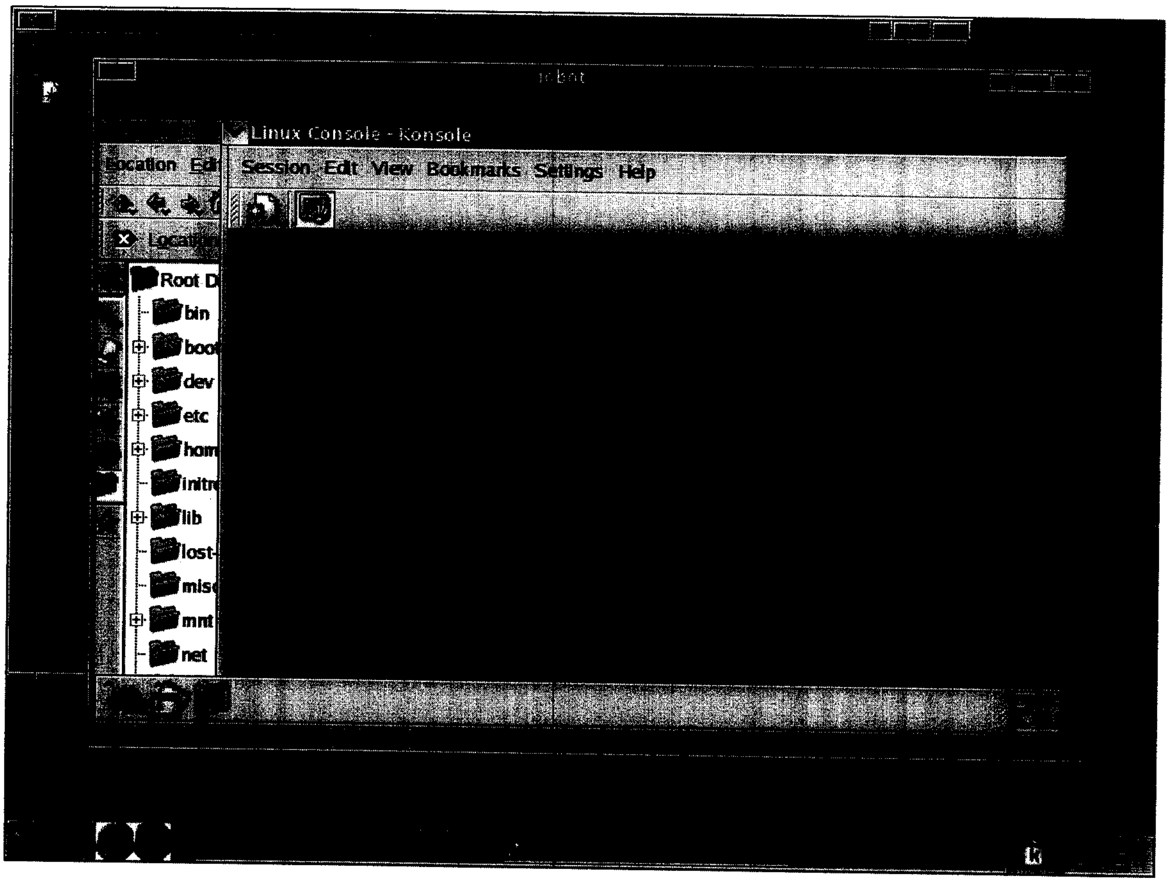
10.2.2 VNCVIEWER :



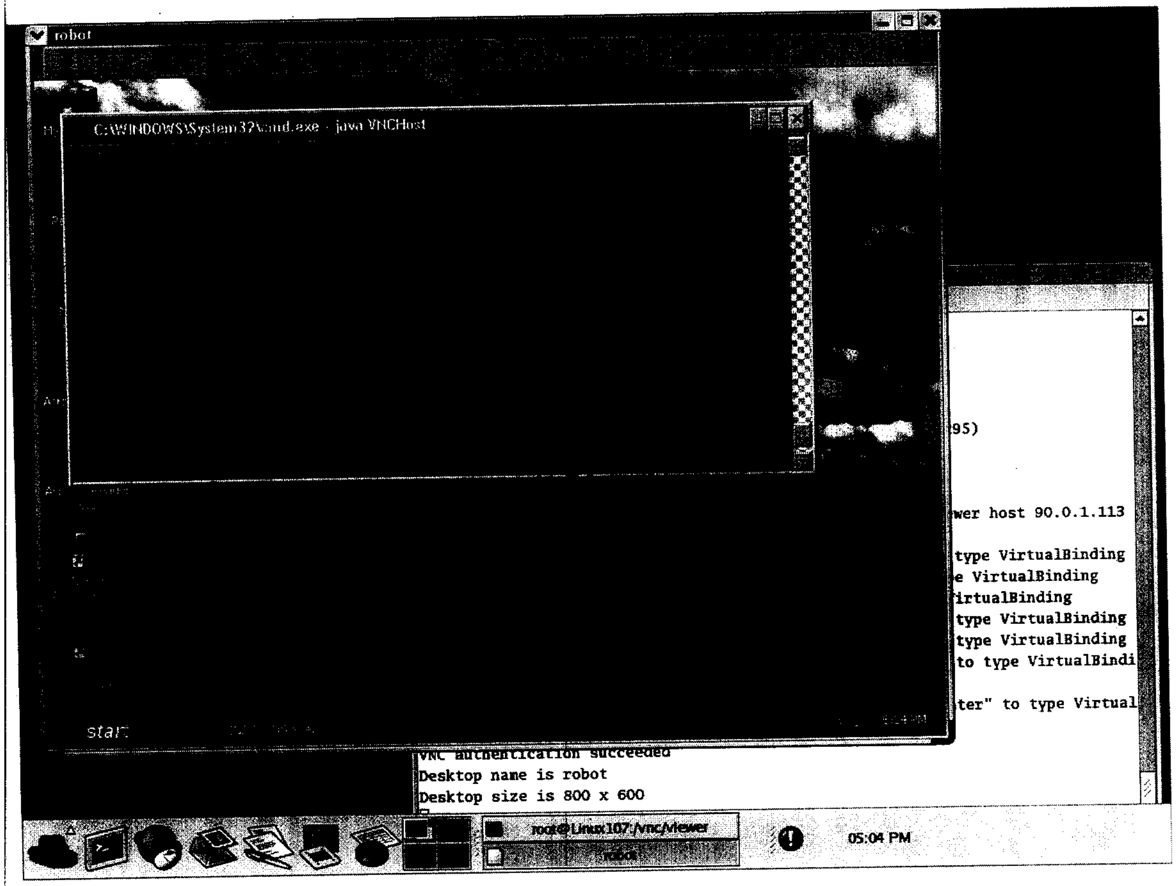
10.2.3 WINDOWS TO WINDOWS DESKTOP CAPTURING :



10.2.4 LINUX TO LINUX DESKTOP CAPTURING :



10.2.6 WINDOWS DESKTOP ON LINUX :


robot
C:\WINDOWS\System32\cmd.exe - java VNCHost
start
root@Linux107:/vnc/viewer
robot
05:04 PM

```
95)

wer host 90.0.1.113

type VirtualBinding
e VirtualBinding
irtualBinding
type VirtualBinding
type VirtualBinding
to type VirtualBindi

ter" to type Virtual

VNC authentication succeeded
Desktop name is robot
Desktop size is 800 x 600
```