REMOTE SYSTEM CONTROLLING
(IT TECHNOS)

P-884

PROJECT REPORT

Submitted By

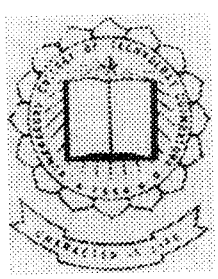
R. JAYAMURUGAN (9927S0053)
V. MUTHU RATHEESH (9927S0059)
S. VIJAY PANDIAN (9927S0083)

Under The Guidance Of

Mrs. V. VANITHA M.E.,
Lecturer, Dept. of CSE.

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

Bachelor of Engineering
In
INFORMATION TECHNOLOGY
Of
Bharathiar University, Coimbatore.



DEPARTMENT OF COMPUTER SCIENCE AND ENGG.
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE-641 006

MARCH – 2003

CERTIFICATE

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE-6.

This is to certify that the project work entitled "**REMOTE SYSTEM CONTROLLING**" done by,

R. JAYAMURUGAN (9927S0053)
V. MUTHU RATHEESH (9927S0059)
S. VIJAY PANDIAN (9927S0083)

Submitted in partial fulfillments of the requirements for the award of the degree of Bachelor of Engineering in Information Technology of the Bharathiar University, Coimbatore, during the academic year 2002-2003.

19/3/03

Guide
Mrs. V. Vanitha, M.E.,
Lecturer, Dept. of CSE.

Project Co-ordinator
Mr. K. R. Baskaran, M.S.,
Asst. Professor, Dept of IT.

Head of the Department

Certified that the candidates were examined by us in the project work & Viva Voce examination held on ..20/03/03

Internal Examiner **External Examiner**

IT TECHNOS

527R , K.R.S Building, D.B.Road, R.S.Puram ,Coimbatore – 641002.
Ph: 98422-06562

March 17, 2003

TO WHOMSOEVER IT MAY CONCERN

This is to certify that the following students,

R. JAYAMURUGAN
V. MUTHU RATHEESH
S. VIJAY PANDIAN

Studying Final year Bachelor Of Engineering in Information Technology at Kumaraguru College of Technology, Coimbatore has completed their project on **"REMOTE SYSTEM CONTROLLING"** during November 2002 – April 2003.

We wish them all success in their future endeavors.

For **IT TECHNOS,**

(R.NARAYANAN)
Proprietor

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

We are thankful to our Principal, **Dr. K.K. Padmanaban, B.Sc(Engg)., M.Tech., Ph.D.,** for his invaluable support in the completion of this project.

We are extremely thankful to **Dr. S. Thangasamy, Ph.D.,** Head of the Department, Department of Computer Science and Engineering for his valuable advice and suggestions throughout this project.

We are indent to express our heartiest thanks to our project guide **Mrs. V. Vanitha, M.E.,** Lecturer, Department of Computer Science and Engineering, who rendered her valuable guidance and support to perform our project work extremely well.

We are indent to express our heartiest thanks to our project coordinator **Mr. K. R. Baskaran, M.S.,** Asst. professor, Department of Information Technology, and to our class advisor **Mrs. N. Chitra Devi M.E.,** Lecturer, Department of Information Technology, who has helped us to perform the project work extremely well.

We are also thankful to all faculty members of the Department of Computer Science and Engineering, Kumaraguru College of Technology, CBE for their valuable guidance, support and encouragement during the course of our project work.

We are also thankful to the company **"IT TECHNOS"** for providing the project and the project guide for his valuable guidance to complete the project successfully.

We express our humble gratitude and thanks to our parents and family members who have supported and helped us to complete the project and our friends, for lending us valuable tips, support and co-operation throughout our project work.

SYNOPSIS

SYNOPSIS

The computer network does not have a specific boundary. It can be spread with in a campus, city or beyond countries. Administrating a network is a tedious process. It must have a constant monitoring to manage the network. Since the network is spread beyond the country, the administrator can't able to view and control all the clients. If he wants to control all the clients he must reach the remote terminal place to control it, which is not at all possible.

Our project "**Remote System Controlling**" is designed to minimize the work of an administrator. It is specially designed to save the valuable time of administrator and to control the remote client which was located anywhere in the network. We can view the remote client screen and control it in the local system itself. We can control the remote system's keyboard and mouse. Each and every activity that happens in the remote system will be reflected in the local system. The administrator can send the message and he can terminate the process running on the remote system by sitting in front of the local system itself. Any person without the technical knowledge can use this system easily.

CONTENTS

CONTENTS

INTRODUCTION

INTRODUCTION

1.1 Current Status of the Problem taken:

In an existing system, if an administrator wants to monitor a remote client he had to go to the remote terminal to watch his activities. In a company, if the administrator wants to know what project is going on and what application is running on the remote client he can view it by reaching the client terminal, which is a time consuming process for an administrator to view all the remote clients. If a message has to be passed, he had to reach the remote client to deliver the message. So the valuable time is wasted for an administrator. In an institution, if any one doing an illegal activity which was not allowed by administrator, then he can identified only by constant viewing of the network, which is a time consuming process in an existing system.

The main drawbacks of the existing system are,

1. Time consuming process
2. It is tiring process to go around and check all the clients
3. To deliver a message the administrator may have to go from one end to another.

1.2. Proposed System:

The proposed system is designed in such a way that the drawbacks of the existing system are removed. It is mainly designed to make the administrator work easy and can be operated easily be any person without any technical knowledge. The proposed system is capable of monitoring all the systems connected in the network. The administrator can control any system connected in the network by sitting in front of the local system itself.

The advantage of the proposed system are,

1. Less time consuming process
2. To monitor all the system for the local system
3. Controlling all the systems connected in the network
4. Message passing from server to clients
5. Terminate the process running in the remote client.

PROPOSED APPROACH TOA PRODUCT

2. PROPOSED APPROACH TO A PRODUCT

Our project "**Remote System Controlling**" is used to monitor and control the systems which are connected in the network.

Get the IP address of the remote client from the user. If the remote client is active then the connection will be established between the remote client and the local system by the help of socket. Once the connection is established the remote client screen is captured and it sent o the server. The server will receive the remote client screen in the file and it will be display on the local system.

After the remote screen is captured on the local system, we can control the remote system with the help of the input devices. If you click on the remote screen which is displayed on the local system, the x and y co-ordinate in which the mouse point is clicked is identified and that information is send to the remote client. In the remote client the corresponding action from the event occurred here will take place. Each and every activity done by the remote user will be reflected on the local system after the remote screen is captured. The administrator can terminate any process which was running in the remote system by sitting in front of the local system itself.

2.1 CLIENT SPECIFICATION:

System	:	Pentium III @ 600 MHz.
Cache	:	128 MB.
RAM	:	128 MB.
Hard Disk	:	20 GB.
Monitor	:	14" Color Monitor.
Keyboard	:	104 Enhanced.
Mouse	:	Logitech three button mouse.
NIC	:	LAN card.

2.2 SERVER SPECIFICATION:

System	:	Pentium III @ 600 MHz.
Cache	:	128 MB.
RAM	:	256 MB.
Hard Disk	:	40 GB.
Monitor	:	14" Color Monitor.
Keyboard	:	104 Enhanced.
Mouse	:	Logitech three button mouse.
NIC	:	LAN card.

2.3 SOFTWARE SPECIFIATIONS:

Operating System	:	Windows 2000/NT Workstation.
Software Required	:	Visual C++.

2.4 INTODUCTION TO SOFTWARE USED:

The software used to develop this system is Visual C++.

VISUAL C++:

Visual C++ has various features for which it is selected. It had very good compiling tools. Some of the features of Visual C++ are

- Supports network communication programs
- Supports ActiveX, ODBC, OLE
- Easy to handle graphics and animation
- Easy to write threading applications

Windows NT:

Nowadays, it is a very powerful GUI based Network Operating System. Some features of Windows 2000/NT are,

- Multitasking
- Multithreading
- Remote Procedure Calls
- Multiprocessor support
- Powerful security.

SYSTEM DESIGN

3. SYSTEM DESIGN

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 design the physical system, prepares input and output specification, makes the screen capture and displaying of the client screens. The physical design maps out the details of the physical system, plans the system implementation.

This project has two applications among which one will be run on the client terminal and other on the server terminal. They are,

1. Transfer Application
2. Receiving Application.

3.1 INPUT DESIGN:

The inputs to Transfer Application are,

- Input from client to start application
- The client screen is also the input which has to be captured

As soon as the client gives the input to start the application the connection is established with the server as well as the capture of the monitor which the other input.

The inputs for the receiving application are,

- IP address of the remote client
- The captured screen sent by the Agent application.

3.2 OUTPUT DESIGN:

The output of the Transfer Application is the captured screen of the remote client. It is stored in a file and it is transmitted to the Receiving application.

In the Receiving application the output is the display of the client screen on to the Receiving application. This file which is obtained as input has first to be stored on to the system and from which the file is being loaded.

IMPLEMENTATION DETAILS

4. IMPLEMENTATION DETAILS

4.1 INTRODUCTION:

In this project the client system is monitored from the server system by the administrator. The protocol used here is TCP/IP.

This project consists of creating two applications as said before,

- Transfer application
- Receiving application

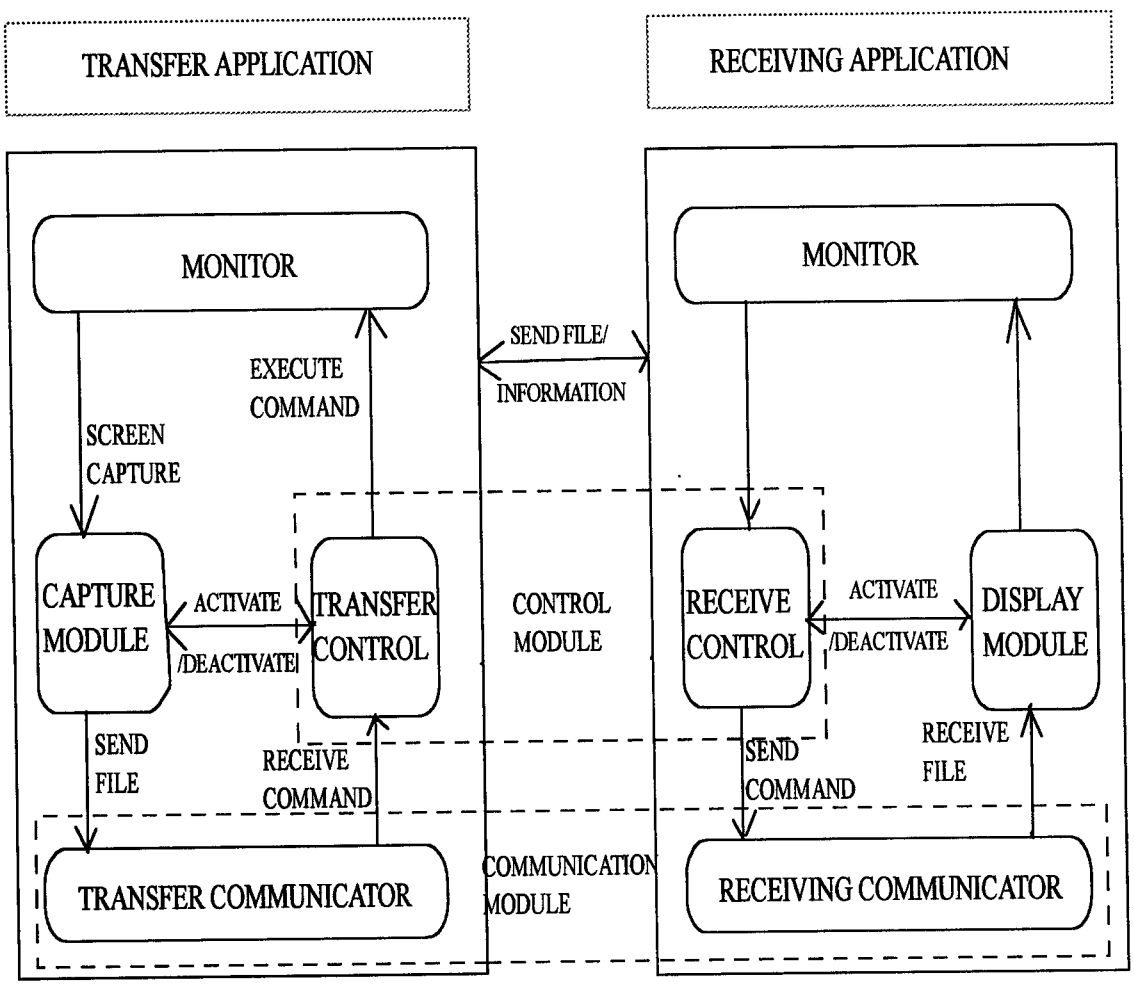
The Transfer application is run on the client system which captures the screen and sends the file to the server. The Transfer application also receives the input form the Receiving application which is the information about the mouse.

The Receiving application is run on the server system. It accepts the input from the client system the client screen as a file. It is used to send the information to the transfer application.

4.2 OVERALL DIAGRAM

The communication between the client and the server is done over LAN network. The overall diagram of the project is given below,

OVERALL DIAGRAM



In the overall diagram you can see that there are two applications. In between the two applications the file and the signals are being transferred over the network via TCP/IP protocol.

The doted rectangle shows that they are a part of the module. The modules in this project are:

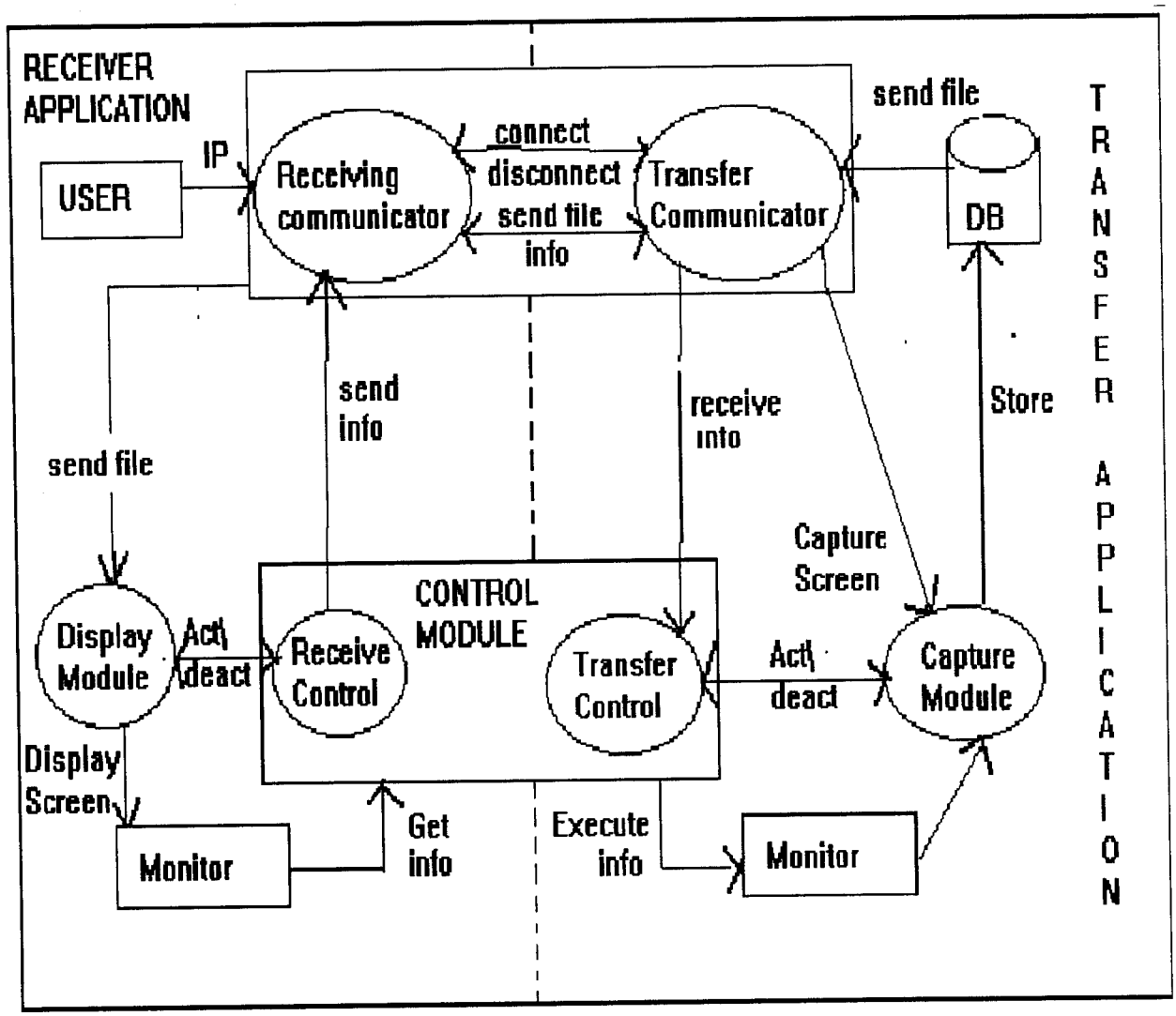
- Display Module
- Capture Module
- Communication Module
- Control Module

4.3 SYSTEM FLOW DIAGRAM:

The system flow diagram for the project is shown in the figure. The process starts with the administrator giving the IP address of the client system. The connection is established between the server communicator and the client communicator in the communication module.

Once the connection had been established the timer in the transfer application is activated and the screen is being captured from the monitor by the Capture Module. This screen is stored in a file. The file stored on the disk is sent to the receiving application for display.

SYSTEM FLOW DIAGRAM



The file is sent to the receiving application via the Communication Module over the LAN. This file is then sent to the Display Module where the file is displayed on to the server application.

When the mouse is moved in the receiving application the mouse information such as the position and the button clicked is sent to the transfer application via the communication module. This information on reaching the transfer application the mouse information of the mouse is executed by the control module of the transfer application.

This module is also responsible for display the messages on the client screen. Thus the control module takes care of the operation of controlling the client system from the server system.

4.4 MODULE DESCRIPITION:

This project consists of four modules. The four modules are,

- Display Module
- Capture Module
- Communication Module
- Control Module

4.4.1 DISPLAY MODULE:

In general the display module is used in the receiving application. The sole purpose of the display module is to display the client screen sent by the client program on the servers screen.

The display module involves the following steps

- Open the Bitmap file
- Creating a Logical Palette
- Selecting the Bitmap Image into Device Context

OPEN THE BITMAP FILE:

The bitmap file that is transferred from the transfer application to the receiving application is to be loaded into the screen. For this first the bitmap file is to be opened. A buffer is created with the size of the file and the bitmap information is loaded into the buffer form the bitmap file. After which the bitmap information apart from the header file information is read and then the next step is performed. The information read is read from the buffer.

CREATING A LOGICAL PALETTE:

Depending upon the bit count of the bitmap the logical palette is created. The logical palette is created for the bitmaps which has the color less than or equal to 256 colors. A logical palette is not created for the colors above 256 colors.

Generally a palette means an array of colors. A logical color palette is like a buffer between color-intensive applications and the system, allowing an application to use as many colors as needed without interfering with its own displayed colors or with colors displayed by other windows. After creation of a palette, "RealizePalette ()" function is used to put the bitmap image colors on to the screen.

When a window has the input focus and calls "RealizePalette ()" function, windows ensures that the window will display all the requested colors, up to the maximum number simultaneously available on the screen. Windows also displays colors not found in the window's palette by matching them to available colors.

SELECTING THE BITMAP IMAGE INTO DEVICE CONTEXT:

A device context is a windows data structure containing information about the drawing attributes of a device such as display or a printer. A device context has to be created in order to select the bitmap image into the servers screen. The bits of the bitmap are to be sent to the screen for display. To perform this operation the function "SetDIBitsToDevice ()" function is used. This function sets the bits of the bitmap to the device context specified.

4.4.2 CAPTURE MODULE:

This module exists in the transfer application. This module is responsible for copying the screen from the client's screen, after which the image is the converted into a bitmap image.

To capture the screen the following steps are followed,

- Creating of Device Context
- Getting the Image from the Device Context
- Converting the Device Dependent Bitmap (DDB) into Device Independent Bitmap (DIB).
- Writing the DIB into a bitmap file

CREATING A DEVICE CONTEXT:

A Device Context is a Windows Data Structure containing the information about the drawing attributes of a device such as printer or a display. A device context is necessary for capturing a screen. A device context is created for the display using the "CreateDC ()" function. Before creating a device context it is first declared and initialized.

In this module two device contexts are created,

- One device context is used to point to the Display device.
- Another device context is used to copy the image from the display device into its device context.

The second device context is used to create the Device Dependent Bitmap and it is send to the next step.

GETTING THE IMAGE FROM THE DEVICE CONTEXT:

The image in the display from the first device context is copied into the second device context. First a bitmap object is created and it is initialized. Then a bitmap is created with the width and the height of the screen using the "CreateCompatibleBitmap ()" function. The Device Context used here is the display device context. Then this bitmap is selected into the second Device Context and the bits of the display are copied into its using the function "BitBlt ()" function. The image obtained is a Device Dependent Bitmap. A Device Dependent Bitmap depends upon the device that is used. To view it in another system it has to be converted into a Device Independent Bitmap which is our next step.

CONVERTING DDB INTO DIB:

In this step the Device Dependent Bitmap is converted into a Device Independent Bitmap. First the DDB information is read. The DDB bitmap contains the height, width, bits, bit count of the bitmap. The information corresponding to the DIB that is BITMAPINFOHEADER is written. Then a buffer is created and the DIB is created and the corresponding information is written into it. The size of the image is calculated and then the information in the BITMAPINFORHEADER is filled. Then the bits of the DDB is obtained and put into the buffer using the "GetDIBits ()" function. Thus the Device Dependent Bitmap is converted into a Device Independent Bitmap. This Device Dependent Bitmap has to be stored in a file for transfer. Thus moving onto the next step.

WRITING THE DIB INTO A BITMAP FILE:

In this step the Device Independent Bitmap is written into a file. First an object is created for the CFile class which handles the file operations. Then the file is created with the required filename. First the header file information is entered saying it is a BMP file. After which the size of the file is filled. Then this information is written into the file, followed by the bits of the DIB bitmaps.

4.4.3 COMMUNICATION MODULE:

This module is used for establishing the connection between the client and the server system. The module used the TCP/IP protocol for the transfer of files and information across the network.

This module consists of two parts:

- Transfer Communicator
- Receiving Communicator

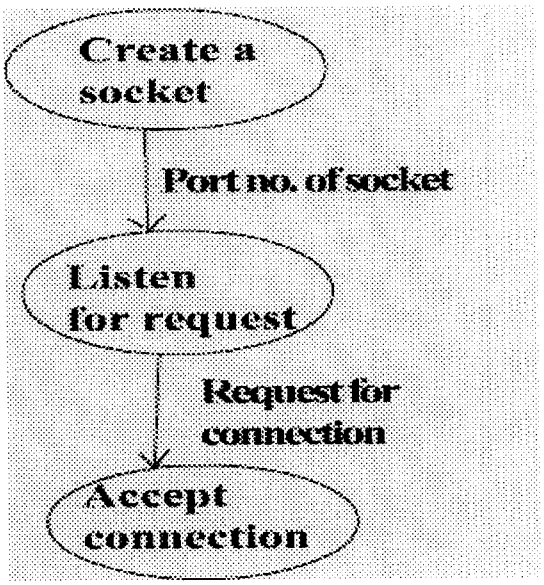
TRANSFER COMMUNICATOR:

This part deals with the transfer of the screen from the transfer application to the receiving application. The entire information passing to and from the receiving application is done handled by this part.

This module deals with the establishment and passing of information across the network. The steps involved in this part is

- Creation of sockets
- Putting it in listen mode
- Accepting the connection

PROCESS DIAGRAM FOR TRANSFER COMMUNICATOR



CREATION OF SOCKET:

In any networking project first a socket has to be developed. A socket is an object that represents an endpoint for communication between processes across a network transport. Without a socket the connection cannot be established.

The class used here is CSocket. In this class a socket is created with the help of "Create()" function. In the creation of the socket the port number has to be specified.

PUTTING IN LISTEN MODE:

The application is put in listening mode after the creation of the socket. In the listening mode the application checks if any request has been obtained for connection. At a time the listen mode is put to listen to only one request. The function used here is "Listen()".

ACCEPTING A CONNECTION:

As soon as there is a request the connection is established with the help of the function "Accept()".

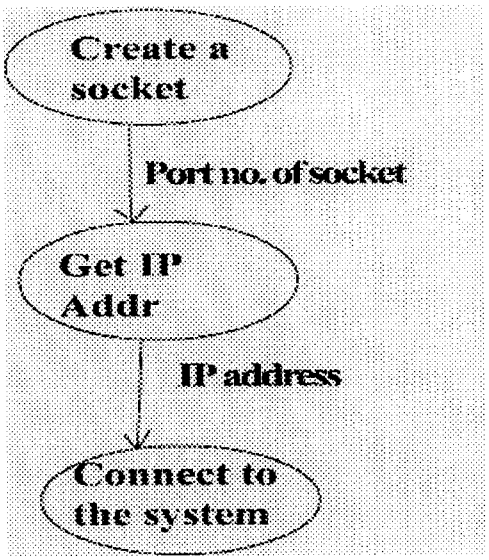
RECEIVING COMMUNICATOR:

This part deals with the reception of the screen from the transfer application and the receiving application. The entire information passing to and from the transfer application is done handled by this part.

This module deals with the establishment and passing of information across the network. The steps involved in this part is

- Creation of Socket
- Getting the IP address
- Connection to system

RECEIVING COMMUNICATOR PROCESS DIAGRAM



CREATION OF SOCKET:

In any networking project first a socket has to be developed. A socket is an object that represents an endpoint for communication between processes across a network transport. Without a socket the connection cannot be established.

The class used here is CSocket. In this class a socket is created with the help of "Create()" function. In the creation of the socket the port number has to be specified.

GETTING THE IP ADDRESS:

The IP address of the system to be connected has to be given by the administrator. The IP address is obtained in the dialog box.

CONNECTING TO SYSTEM:

After the IP address of the system to connected is obtained, the connection is established with the help of the "Connect()" function.

4.4.4 CONTROL MODULE:

This module is responsible for sending files, messages and mouse information across the network between the transfer application and the receiving application.

This module has two parts

- Transfer control
- Receive control

TRANSFER CONTROL:

In the transfer control all the operations after the establishment of connection is performed. The transfer control performs the following operations

- Send file
- Execute Mouse Operations
- Display messages
- Shut down the system

SEND FILE:

Once the connection has been established the capture module captures the screen and stored it in a file. This file is transferred from the transfer application to the receiving application by this module. The files are sent using the function "Send ()".

EXECUTE MOUSE OPERATIONS:

The mouse information is obtained from the receiving application. This information is executed via the function "mouse_event()" function.

DISPLAYING MESSAGE:

The message sent by the receiving application, which is displayed with the help of dialog box.

RECEIVING CONTROL:

The receiving control resides in the receive application. It is responsible for receiving the file that is sent by the transfer application, extracting the mouse information, sending messages.

The functions of the receive control is

- Receive File
- Extract Mouse Information and send it
- Send the messages to the transfer application.
- Sending shut down signal to transfer application.

SYSTEM TESTING

5. SYSTEM TESTING

System testing is one of the important phases in the software development. Each and every module is tested to check whether it satisfies the user requirements or not. If the system testing is not conducted, then the software developed can become a non-conforming product.

In our project the system testing is conducted as follows. Get the IP address of the remote client from the user. If the remote client is active then the connection will be established else display the corresponding message using dialog box. After the connection is established, the remote client screen will display on the local system. The screen is captured and store in the file for transmission. If any screen is modified in the remote client, then the modified screen is captured and then transmitted to the server for obtaining live updation.

After the remote client screen is captured we can control all the activities of the remote system. We can view all the processes running on the remote system. Those processes will be display in the list box of the local system. By clicking that process name we can view all its memory details like page fault, page size, starting address of the process, process ID, etc. These details will be displayed on the edit box which is placed just below the list box that contains the remote processes name.

If we click a button on the remote screen which is displayed on the local system, the x and y co-ordinates of the click are identified and that information is sent to the remote client to perform its action. Similarly, when the keyboard key are pressed the virtual key code and ascii code of the key that pressed are sent to the remote client and the corresponding action will be performed on the remote client. Each and every activity on the remote client will be reflected in the local system.

CONCLUSION AND FUTURE LOOK

6. CONCLUSION AND FUTURE LOOK

6.1 CONCLUSION:

The complete design and development of the system 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 connecting with many systems 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 modifications. As a further enhancement this system can be integrated with any other system.

This system has been developed using Visual C++. The main aim behind the development of this system is to provide a solution for the administrator to watch over the client from the server.

6.2 LIMITATIONS:

This project is very simple and easy to implement. This project is working properly. This project is successfully one and this can be implemented easily. The system developed can monitor only one system at a time. It can be run only in windows platform. The system can monitor only those systems connected in one server and not many servers. The future enhancement can monitor multiple systems at a particular time.

6.3 FUTURE ENHANCEMENTS:

There are many features that can be added to the system. Due to the insufficient time it could not be incorporated in this system.

The future enhancements that can be provided are:

- Access to multiple clients at a particular time.
- Monitoring the system of another server from a server.

REFERENCES

7. REFERENCES

1. David J. Kruglinski, George Shepherd, Scot Wingo,
"Programming Microsoft Visual C++".
Microsoft Press; Fifth Edition.

2. Richard C. Leinecker and Archer Tom,
"Visual C++ 6 Programming Bible",
IDG Books India (P) Ltd; Fifth Edition.

3. Roger S. Pressman
"Software Engineering and Application",
Mc Graw Hill; Fourth Edition.

4. John Paul Mueller
"Visual C++ 6 from the Ground up",
Tata Mc Graw Hill.

5. www.programmerheaven.com (Dec 15th).

6. www.remoteadmin.com (Jan 10th).

7. www.remotecontrol.com (Feb 7th).

APPENDIX

APPENDIX

8.1 FUNCTION REFFERENCE:

Function Name	Uses	Category
Listen()	To listen connection through Socket.	Predefined Function. Member of Socket class
Accept()	To accept a connection.	Predefined Function. Member of Socket class
Connect()	To request a connection	Predefined Function. Member of Socket class
Send()	To send bytes through Socket.	Predefined Function. Member of Socket class.
Receive()	To receive bytes through Socket.	Predefined Function. Member of Socket Class.
HdcToJpeg()	Convert the captured image into a jpeg file	User defined Function.
GetProcessMemoryInfo()	To get the memory details of the specified process	Predefined Function.

8.2 SAMPLE CODING:

TRANSFER APPLICATION:

CONNECTION ESTABLISHMENT:

```
#include "stdafx.h"
#include "Agent.h"
#include "CtrlListenSock.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CCtrlListenSock::CCtrlListenSock()
{
}

CCtrlListenSock::~CCtrlListenSock()
{
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CCtrlListenSock, CSocket)
```

```
        //{{AFX_MSG_MAP(CCtrlListenSock)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

//////////////////////////////////////////////////////////////////////
// CCtrlListenSock member functions

void CCtrlListenSock::OnAccept(int nErrorCode)
{
        //Accept the connection of the manager's control socket
        int res=Accept(((CMainFrame *)AfxGetMainWnd())->ProcessCtrlSock);
                ASSERT(res!=SOCKET_ERROR);

        CSocket::OnAccept(nErrorCode);
}
```

MOUSE EVENT CONTROL

```
#include "stdafx.h"
#include "Agent.h"
#include "AgentCtrlSock.h"
#include "MainFrm.h"
void CAgentCtrlSock::OnReceive(int nErrorCode)
{       rsvMouseEventMsg managermsg;
        intres=((CMainFrame*)AfxGetMainWnd())>AgentCtrlSock.Receive(&manager
msg,sizeof(rsvMouseEventMsg));
  switch(managermsg.type)
        {
        case Mouse_Move:
        {mouuse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_MOVE,man
agermsg.x,managermsg.y,NULL,NULL);              break;          }


case Mouse_LDown:
{
mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTDOWN,manager
msg.x,managermsg.y,NULL,NULL);                  break;          }
  case Mouse_LUp:
        {
mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTUP,managermsg.
x,managermsg.y,NULL,NULL);                      break;          }
  case Mouse_RDown:
{
mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_RIGHTDOWN,manag
ermsg.x,managermsg.y,NULL,NULL);                break;          }
```

```
  case Mouse_RUp:
        {
mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_RIGHTUP,managerms
g.x,managermsg.y,NULL,NULL);                    break;          }
        case Mouse_LDouble:
        {
mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTDOWN,manager
msg.x,managermsg.y,NULL,NULL);
        mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTUP,mana
germsg.x,managermsg.y,NULL,NULL);
        mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTDOWN,
managermsg.x,managermsg.y,NULL,NULL);
        mouse_event(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTUP,mana
germsg.x,managermsg.y,NULL,NULL);
                        break;          }
        }
        CSocket::OnReceive(nErrorCode);
}

void CAgentCtrlSock::OnClose(int nErrorCode)
{

        Close();
        CSocket::OnClose(nErrorCode);
}
```

REMOTE PROCESSCONTROL

```
#include "stdafx.h"
#include "Agent.h"
#include "ProcessCtrlSock.h"
#include "string.h"

void CProcessCtrlSock::OnReceive(int nErrorCode)
{
static uint8 type=0;
 static BOOL CheckType =TRUE;

if ( !EnumProcesses( aProcesses, sizeof(aProcesses), &cbNeeded ) )
{
cProcesses = cbNeeded / sizeof(DWORD);

for ( i = 0; i < cProcesses; i++)
{
 pszProcessName[MAX_PATH] = "unknown";
HANDLE hProcess;
PROCESS_MEMORY_COUNTERS pmc;
hProcess=OpenProcess(PROCESS_ALL_ACCESS,FALSE,aProcesses[i] );
 }

if ( hProcess )
{
 HMODULE hMod;

if ( EnumProcessModules( hProcess, &hMod, sizeof(hMod), &cbNeeded) )
{
```

```
GetModuleBaseName(hProcess,hMod,szProcessName,sizeof(szProcessName) );
 }
GetProcessMemoryInfo( hProcess, &pmc, sizeof(pmc));
CloseHandle( hProcess );
strcpy(pd[ProcessCt].name,szProcessName);
pd[ProcessCt].ProcessID=aProcesses[i];
pd[ProcessCt].pmc=pmc;
ProcessCt++;
 }
}

Msg.type=rsvProcessDetailUpdate;
PDUMsg.nProcessCt=ProcessCt;
res=Send(&PDUMsg.type,sizeof(PDUMsg.type));
CSocket::OnReceive(nErrorCode);
 }

void CProcessCtrlSock::OnClose(int nErrorCode)
{
        Close();

        CSocket::OnClose(nErrorCode);
}
```

KEYBOARD EVENT CONTROL

```
#include "stdafx.h"
#include "Agent.h"
#include "AgentKCtrlSock.h"
#include "MainFrm.h"
void CAgentKCtrlSock::OnReceive(int nErrorCode)
{
rsvKeyEventMsg msg;
res=((CMainFrame*)AfxGetMainWnd())-
>AgentKeyCtrlSock.Receive(&msg,sizeof(rsvKeyEventMsg));
if(res==SOCKET_ERROR)
msg.type=0;
switch(msg.type)
{
case Key_Down :
{
 keybd_event((BYTE)msg.vkey ,(BYTE)msg.scancode,NULL,NULL);
break;
        }
case Key_Up :
{
keybd_event((BYTE)msg.vkey ,(BYTE)msg.scancode,KEYEVENTF_KEYUP,NULL);
break;
        }

case Key_AltTab:
{
keybd_event(VK_MENU,56,NULL,NULL);
keybd_event(VK_TAB,15,NULL,NULL);
```

```
keybd_event(VK_TAB,15,KEYEVENTF_KEYUP,NULL);
keybd_event(VK_MENU,56,KEYEVENTF_KEYUP,NULL);
break;
        }


case Key_CtrlEsc:
{
keybd_event(VK_CONTROL,29,NULL,NULL);
keybd_event(VK_ESCAPE,1,NULL,NULL);
keybd_event(VK_CONTROL,29,KEYEVENTF_KEYUP,NULL);
keybd_event(VK_ESCAPE,1,KEYEVENTF_KEYUP,NULL);
break;
}
case Key_WinD:
{
keybd_event(VK_LWIN,91,NULL,NULL);
keybd_event(68,32,NULL,NULL);
keybd_event(VK_LWIN,91,KEYEVENTF_KEYUP,NULL);
keybd_event(68,32,KEYEVENTF_KEYUP,NULL);
break;
        }
case Key_WinE:
{
keybd_event(VK_LWIN,91,NULL,NULL);
keybd_event(69,18,NULL,NULL);
keybd_event(VK_LWIN,91,KEYEVENTF_KEYUP,NULL);
keybd_event(69,18,KEYEVENTF_KEYUP,NULL);
break;          }
        case Key_WinF:
                {
```

```
                keybd_event(VK_LWIN,91,NULL,NULL);
                keybd_event(70,33,NULL,NULL);
                keybd_event(VK_LWIN,91,KEYEVENTF_KEYUP,NULL);
                keybd_event(70,33,KEYEVENTF_KEYUP,NULL);
                break;          }

        case Key_WinM:
        {
        keybd_event(VK_LWIN,91,NULL,NULL);
        keybd_event(77,50,NULL,NULL);
        keybd_event(VK_LWIN,91,KEYEVENTF_KEYUP,NULL);
        keybd_event(77,50,KEYEVENTF_KEYUP,NULL);
        break;          }
        case Key_WinR:
        {
        keybd_event(VK_LWIN,91,NULL,NULL);
        keybd_event(82,19,NULL,NULL);
        keybd_event(VK_LWIN,91,KEYEVENTF_KEYUP,NULL);
        keybd_event(82,19,KEYEVENTF_KEYUP,NULL);
        break;          }
        case Key_WinOnly:
        {
        keybd_event(VK_LWIN,91,NULL,NULL);
        keybd_event(VK_LWIN,91,KEYEVENTF_KEYUP,NULL);
        break;          }
        }
        CSocket::OnReceive(nErrorCode);
}
```

RECEIVING APPLICATION:

CONNECTION ESTABLISHMENT

```
#include "stdafx.h"
#include "Manager.h"
#include "ControlSock.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#if 0
BEGIN_MESSAGE_MAP(CControlSock, CSocket)
        //{{AFX_MSG_MAP(CControlSock)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

// CControlSock

CControlSock::CControlSock()
{
}
```

```
CControlSock::~CControlSock()
{
        Close();
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CControlSock, CSocket)
        //{{AFX_MSG_MAP(CControlSock)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif // 0

//////////////////////////////////////////////////////////////////////
// CControlSock member functions


void CControlSock::OnClose(int nErrorCode)
{
        Close();
        CSocket::OnClose(nErrorCode);
}
```

REMOTE PROCESS CONTROL

```
#include "stdafx.h"
#include "Manager.h"
#include "ProcessDlg.h"
#include "MainFrm.h"
#include "ProcessCtrlSock.h"
#include "afxcmn.h"

void CProcessDlg::OnRefresh()
{
int item=pNMListView->iItem;

if(item>=0)
{
CString str1;
str1.Format("PageFaultCount : 0x%08X (%lu)\r\n\r\nPeakWorkingSetSize : 0x%08X
(%lu)\r\n\r\nWorkingSetSize : 0x%08X (%lu)\r\n\r\nQuotaPeakPagedPoolUsage :
0x%08X (%lu)\r\n\r\nQuotaPagedPoolUsage : 0x%08X
(%lu)\r\n\r\nQuotaPeakNonPagedPoolUsage : 0x%08X
(%lu)\r\n\r\nQuotaNonPagedPoolUsage : 0x%08X (%lu)\r\n\r\nPagefileUsage : 0x%08X
(%lu)\r\n\r\nPeakPagefileUsage : 0x%08X (%lu)"
pt->PDetails[item].pmc.PageFaultCount,
pt->PDetails[item].pmc.PageFaultCount,
 pt->PDetails[item].pmc.PeakWorkingSetSize,
 pt->PDetails[item].pmc.PeakWorkingSetSize,
 pt->PDetails[item].pmc.WorkingSetSize,
 pt->PDetails[item].pmc.QuotaNonPagedPoolUsage,
 }
```

```
else
m_MemVal.SetWindowText(" ");
}



void CProcessDlg::OnKillprocess()
{
 rsvKillProcessMsg KillProc;
KillProc.type=rsvKillProcess;
int item=m_ProcessList.GetSelectionMark();
CString str=m_ProcessList.GetItemText(item,1);

if(!str.IsEmpty())
 {
 long id=atol((char*)str.operator LPCTSTR());
KillProc.KillProcessID=id;
m_ProcessList.DeleteItem(item);
 res=((CMainFrame*)AfxGetMainWnd())-
ProcesessCtrlSock.Send(&KillProc.type,sizeof(KillProc.type));
ASSERT(res!=SOCKET_ERROR);
 }
 }
```

SCREEN CAPTURING

```
#include "Manager.h"
#include "ClientSock.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

void CClientSock::OnReceive(int nErrorCode)
{
  static uint8 type=0;
  static BOOL CheckType =TRUE;
  int res=0,ret=0;

if(res!=0)
{
file.Write(buf,size);
file.Close();
CmanagerView*view=(CManagerView*)((CMainFrame*)AfxGetApp()-
>m_pMainWnd)->GetActiveView();
CClientDC dc(view);
view->OnPrepareDC(&dc);
CRect rect(0,0,horizontal,vertical);
CPicture AgentScr;  AgentScr.Load(name);
AgentScr.Show(&dc,&rect);  AgentScr.FreePictureData();
rsvManagerToAgentMsg msg;
msg.FrameRequest.type=rsvFrameBufferUpdateRequest;
msg.FrameRequest.Incremental=FALSE;
```

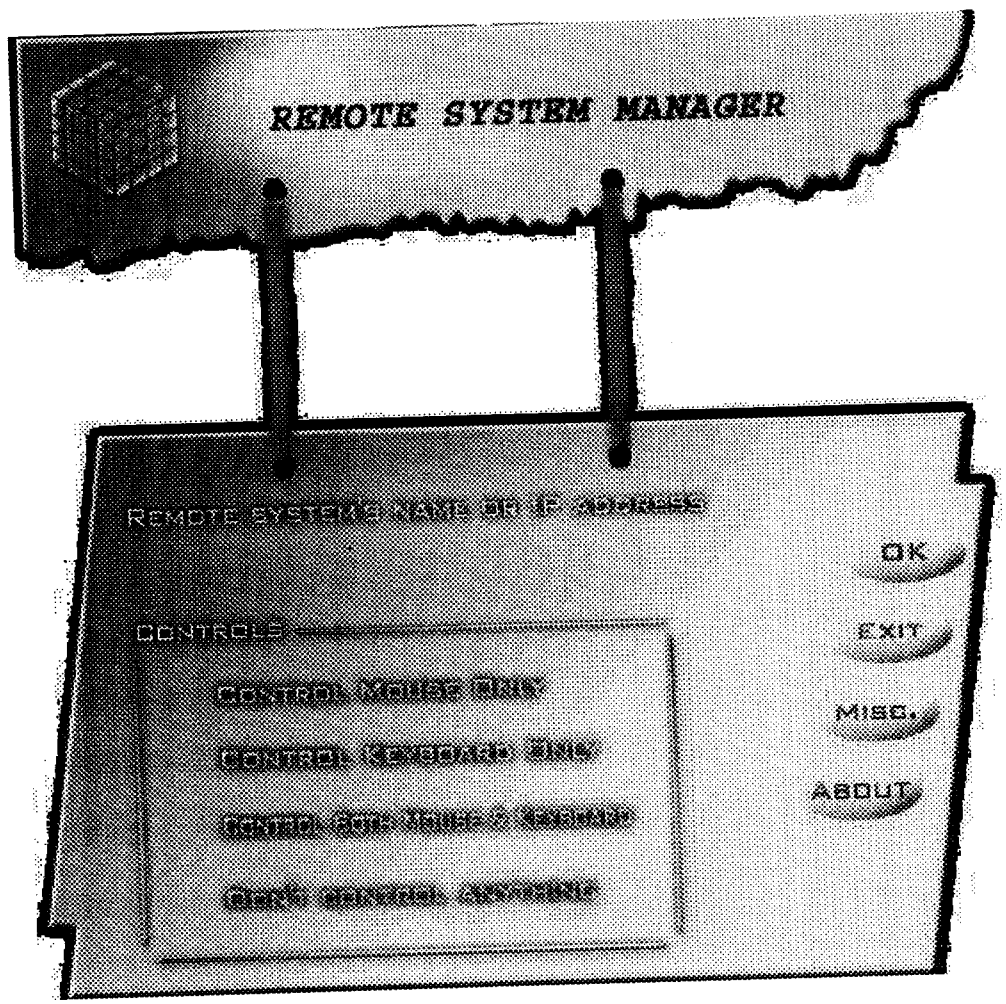
```
msg.FrameRequest.width=horizontal;
msg.FrameRequest.height=vertical;
ret=Send(&msg.FrameRequest.type,sizeof(msg.FrameRequest.type));

if(ret==SOCKET_ERROR)
{
AfxMessageBox("Error in sending data");
Close(); goto end;     }
ret=Send(&msg,sizeof(msg));
if(ret==SOCKET_ERROR)
{
AfxMessageBox("Error in sending data");
Close(); goto end;   }
 delete []buf;
Type=TRUE;
break; }
}       }
end:

CSocket::OnReceive(nErrorCode);
}
        InvokeHelper(0x1, DISPATCH_METHOD, VT_I4, (void*)&result, parms,
                lngHDC, nTLX, nTLY, nBRX, nBRY, nQuality, bstrPath, pbstrMsg);
        return result;
}
```

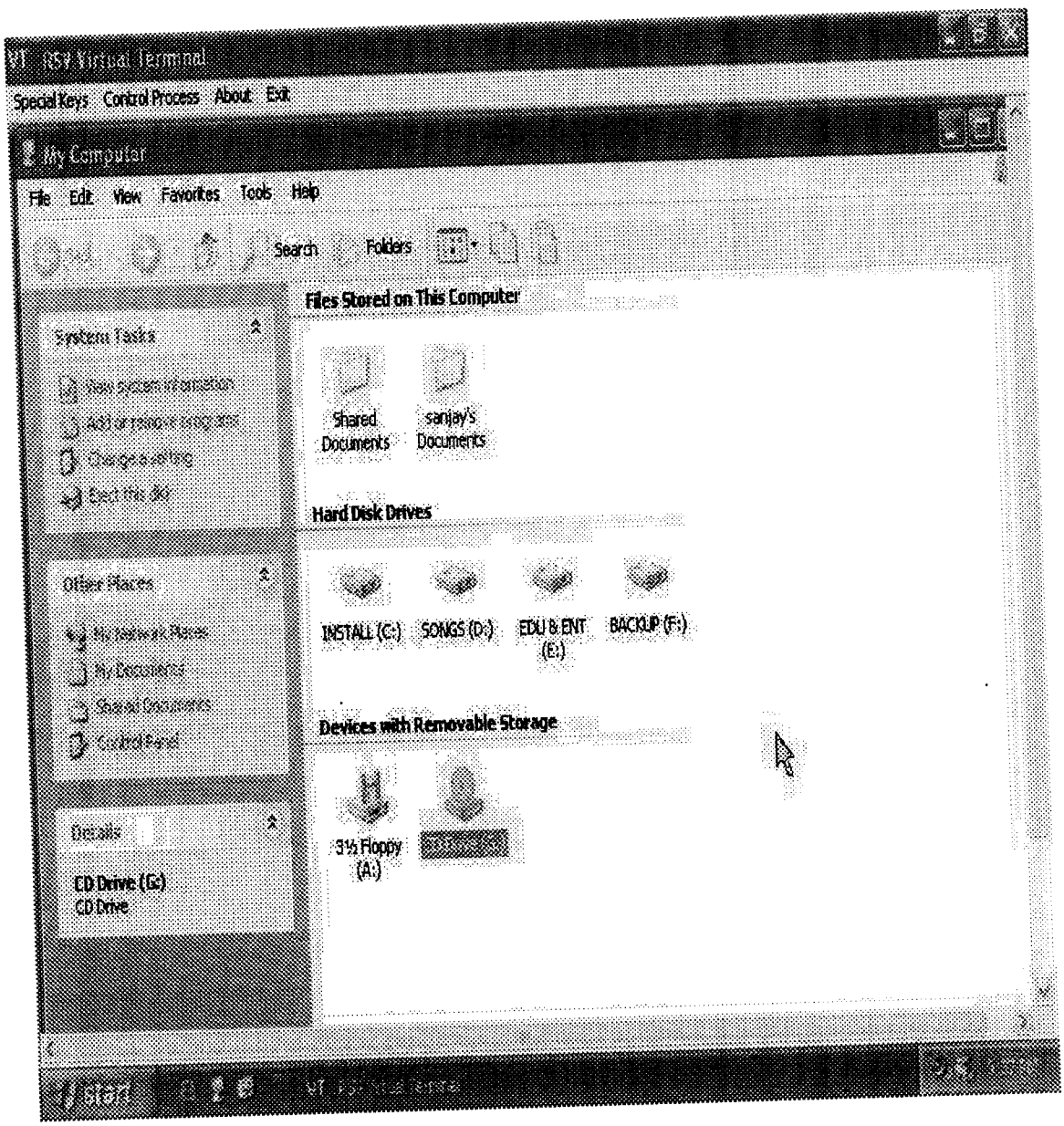
8.3 SAMPLE OUTPUT SCREENS:

MAIN SCREEN



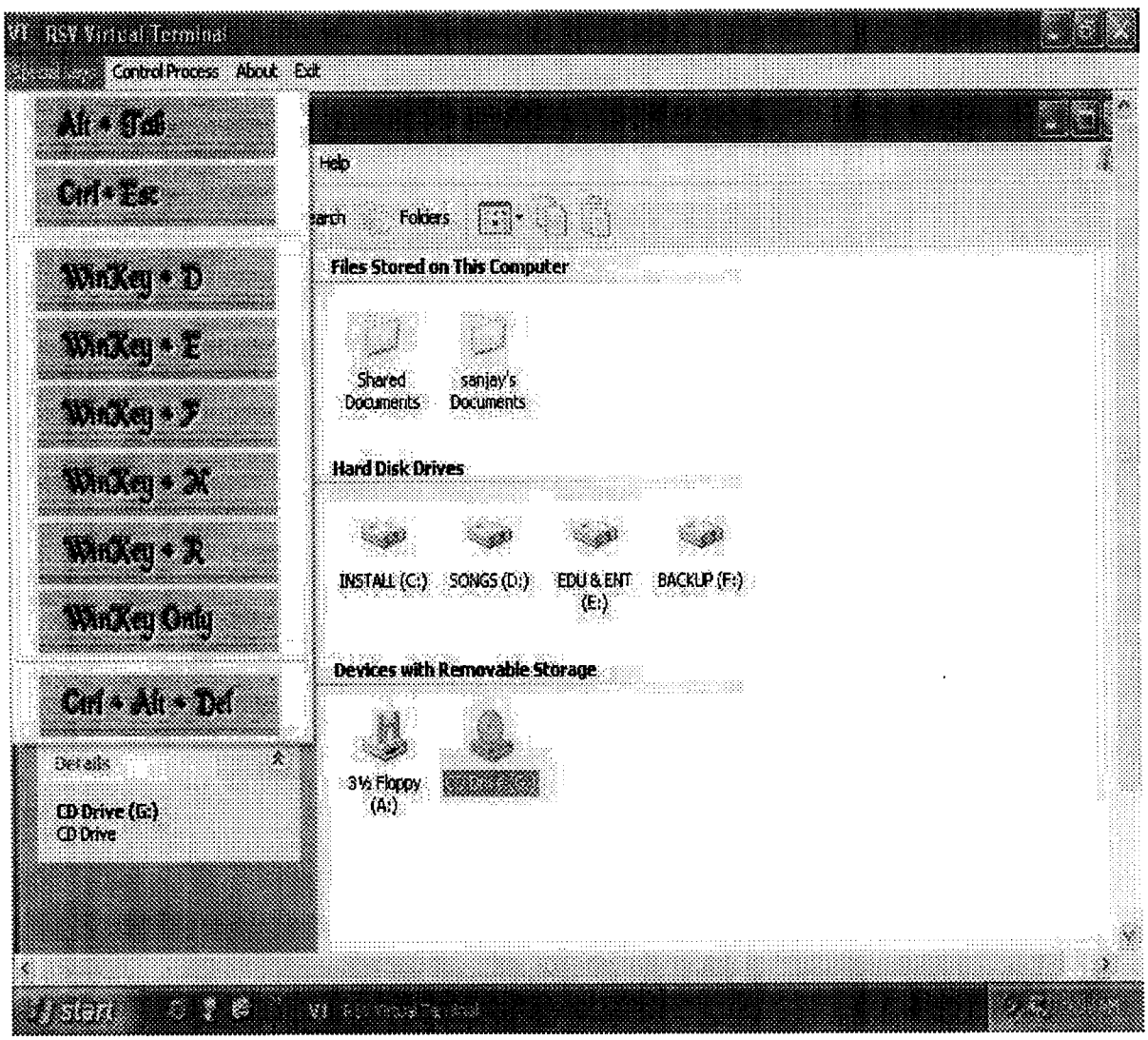
Enter the IP address of the remote client and select any one of the above option to view and control the remote screen.

WINDOW WITH CAPTURED SCREEN



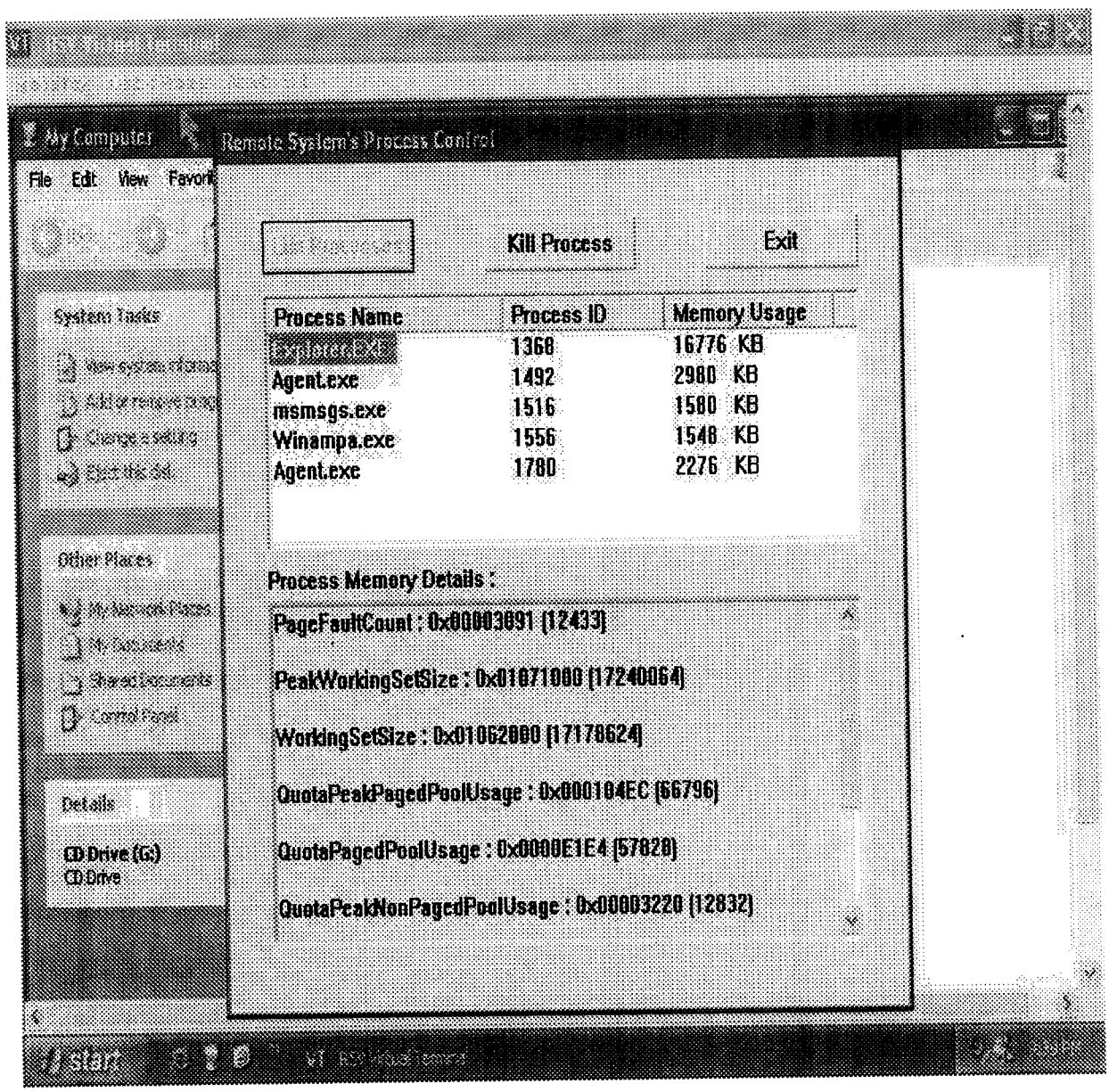
The remote client screen was captured and it displayed in our local machines applications.

USER INTERFACE SCREEN



The remote client was controlled by the special key like ctrl, alt, windows which was displayed as menu item in our local machine.

REMOTE PROCESS CONTROLLING SCREEN



The processes running on the remote client can be viewed and terminated in our local system with the help of above dialog box.